LATVIJAS UNIVERSITĀTE
FIZIKAS, MATEMĀTIKAS UN OPTOMETRIJAS FAKULTĀTE
MATEMĀTIKAS NODAĻA

REĀLLAIKA LAIKRINDU ANALĪZE PROGNOZĒŠANAI UN ANOMĀLIJU DETEKTĒŠANAI

MAĢISTRA DARBS

Autors: Artis Alksnis
Studentes apliecības nr. aa171213
Darba vadītājs: profesors Jānis Valeinis

RĪGA 2021

ANOTĀCIJA

Šajā darbā tiek aprakstīts laikrindu anomāliju noteikšanas modeļa izstrādes process un tā realizācija. Darbs tiek balstīts uz temperatūras mērījumu sensoru datiem. Anomāliju noteikšanas modeļa izstrādes ietvaros tiek apskatītas sekojošas tēmas - simulāciju veidošanda, laikrindu analīze, laikrindu priekšapstrāde, laikrindu klasterēšana, laikrindu prognozējošo modeļu izveide, anomāliju noteikšana un modeļu ansambļa izveide. Darba mērķis ir apskatīt dažāda tipa modeļus, metodes un to apvienojumus, lai izveidotu robustu anomāliju noteikšanas modeļu ansambli.

Darba rezultātā tika izveidots laikrindu anomāliju noteikšanas modeļu ansamblis, kura pamatā ir četri modeļi - LightGBM, LSTM, Holt-Winters un laikrindu apakšperiodu klasterēšana. Modelēšana un analīze tika veikta programmēšanas valodā Python un darba pielikums satur JupyterLab darba grāmatas.

Atslēgas vārdi: anomāliju noteikšana laikrindās, Holt-Winters, LightGBM, LSTM, DTW, Python, temperatūras mērījumu laikrindas, laikrindu prognozēšana, laikrindu klasterēšana, laikrindu priekšapstrāde, laikrindu analīze, laikrindu sacensības, simulācijas, modeļa izstrādes process, modeļu ansamblis.

ABSTRACT

This thesis describes time series anomaly detection process and servs as a guild for time series anomaly detection model creation. Analysis and models created in this thesis are created for temperature measurement time series data. In this thesis anomaly detection development process covers following topics - time series simulations, time series analysis, time series preprocessing, time series clustering, time series forecasting, anomaly detection and ensemble model creation. The main goal of this thesis was to reseach and compare different time series models, methods and different model and method combinations to create ensemble model for robust real-time anomaly detection.

This thesis resulted in creation of anomaly detection ensemble model that contains following four models - LightGBM, LSTM, Holt-Wintersa and time series subperiod clustering. Modeling and analysis was done in Python programming language and all JupyterLab notebooks are available in attachment.

Atslēgas vārdi: time series anomaly detection, Holt-Winters, LightGBM, LSTM, DTW, Python, temperature measurement time series, time series forecasting, time series clustering, time series preprocessing, time series analysis, time series competition, simulations, model development process, ensemble model.

SATURA RĀDĪTĀJS

APZĪMĒJUMU SARAKSTS

Darbā kā skaitļu decimāldaļas atdalītājs tiek izmantots punkts.

NN - Neironu tīkli modelis.

AR - Autoregresīvais modelis.

MAPE - Vidējās absolūtās kļūdas procents.

APE - Absolūtais kļūdas procents.

REA - Relatīvā absolūtā kļūda.

RNN - Periodisko/atkārtojošo neironu tīklu modeļi.

LOESS - Lokāli svērtās regresijas metode.

ACF - Autokorelācijas funkcija.

PACF - Parciālā autokorelācijas funkcijas.

ARIMA - Autoregresīvais integrētais slīdošā vidējā modelis

SARIMA - Sezonālais autoregresīvais integrētais slīdošā vidējā modelis

STD - Sezonālā trenda dekompozīcija metode.

SampEn - Modeļa entropijas novērtējums.

KNN - k tuvāko kaimiņu metode.

MAD - Vidējā absolūtā kļūda.

DTW - Dinamiskās deformācijas laikā metode.

Soft-DTW - Gludināta dinamiskās deformācijas laikā metode.

GAK - Globālā kodola izlīdzināšanas metodes attālums.

ML - Mašīnmācīšanās modeļi.

GBT - Gradientu pastiprināšanas algoritms.

SVM - Atbalsta vektoru mašīnas modelis.

IEVADS

Laikrindu analīzes metodes un to prognozēšana modeļi tiek pētīti un izmantoti praksē jau kopš 1920-tajiem gadiem, it īpaši finanšu sektorā. Palielinoties datu apjomam, kas tiek iegūts no dažādām aplikācijām, iekārtām, sensoriem un citiem avotiem, palielinās iespēja un rodas nepieciešamība izmantot lainkrindu analīzes un prognozēšanas metodes prognozēšanas un klasificēšanas nolūkos, lai analizētu šos datus un veiktu prognozes par nākotnes notikumiem. Laikrindu analīze tiek izmantota statistikā, ekonometrijā, signālu apstrādē un dažādās citās nozarēs, kur tiek iegūti, apstrādāti un analizēti dati laikā. Laikrindu modeļu attīstību un izaugsmi ir veicinājusi vispārējā tehnoloģijas attīstība, kas ļauj izmantot algoritmus, kurus iepriekš nav bijis iespējams efektīvi izmantot, dēļ tehniskiem ierobežojumiem. Pēdējo 15 gadu laikā ar vien vairāk tiek attīstīti dažādi neironu tīklu modeļu variācijas un to hibrīdmodeļi. Par hibrīdmodeļiem tiek uzskatīti modeļi, kas apvieno vairāk kā vienu modeļa veidu, piemēram, laikrindu kontekstā populārs un efektīvs hibrīd modelis ir autoregresijas neironu tīklu modelis, kas apvieno autoregresijas (turpmāk tekstā AR) un neironu tīklu (turpmāk tekstā NN) modeļus.

Darba mērķis ir apskatīt laikrindu klāsterēšanas metodes, prognozēšanā modeļus un izstrādāt modeļa izstrādes gaitu priekš laikrindu reāllaika monitorēšanas modeļu ansambļa izveides anomāliju noteikšanai. Darbs ir balstīts uz projektu, kura ietvaros tika izpētītas piemērotākās metodes un izstrādāts modelis ar nolūku konceptuāli pierādīt, ka ir iespējams izveidot mērogojamus modeļus, kas spētu kalpot kā palīgs reāllaika anomāliju noteikšanai. Darba uzdevumā tika dots uzdevums izstrādāt anomāliju noteikšanas modeli priekš temperatūras mērījumu sensoriem saldētavās un aukstajās telpās, lai identificētu un laicīgi informētu par anomālām temperatūras izmaiņām/svārstībām. Īstos sensora datus nebija iespējams izmantot darbā, tāpēc darba ietvaros visa analīze tiks veikta balstoties uz simulētiem datiem, bet dati tiks simulēti izmantojot iepriekšēju analīzi par šāda tipa datiem. Ņemot vērā projekta pamata uzdevumu, kur bija nepieciešams strādāt ar viena mainīgā laikrindām, tad pamatā arī šajā darbā uzsvars tiks likts uz viena mainīgā modeļiem, bet tas neizslēdz iespēju papildināt modeļus, kuri to pieļauj, ar papildus mainīgajiem vai citām laikrindām.

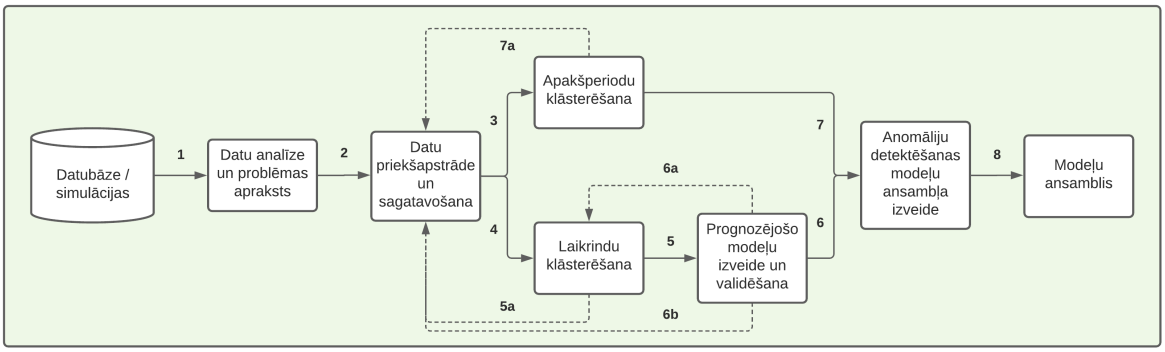
Lai gan darbā izmantotie dati tiks simulēti imitējot reālu sensoru datu uzvedību, bet tas neierobežo iegūtos rezultātus izmantot arī cita veida vai uzvedības laikrindu datu klāsterizēšanai, prognozēšanai un anomāliju noteikšanai. Aprakstot analīzi tiek minētas arī metodes, kas varētu būt noderīgas cita tipa vai struktūras laikrindām. Prognozējošie modeļi tiks veidoti ne tik daudz, lai prognozētu nākotnes vērtības un prognozes rezultātā varētu gūt no tā kādu ieguvumu, bet vairāk anomāliju kontekstā, tas ir, lai novērtētu vai attiecīgais novērojums iekļaujas prognozētajā intervālā vai arī ne. Gadījumā, kad novērojums neiekļaujas sagaidāmajā prognozētajā intervālā, tas tiek uzskatīts par anomāliju.

Spēja detektēt anomālijas var palīdzēt identificēt problēmas, piemēram, ar pašu sensoru vai ar iekārtu, kurā atrodas attiecīgais sensors, lai laicīgi veikt preventīvas darbības pirms ir radušies kādi zaudējumi vai vismaz samazināt radīto zaudējumu apjomu. Sensoru datu mērījumu bie-

žums ir mainīgs, bet tas parasti ir pietiekami liels un sensoru skaits mēdz būt samērā liels, kas var liegt anomālijas noteikt izmantojot cilvēka resursus. Apskatīties modeļi spēj identificēt pat šķietami mazas izmaiņas, kuras, pat gadījumā ja cilvēks veic sensora monitoringu, nav iespējams tik viegli identificēt un līdz ar to izstrādātie modeļi var kalpot kā palīgrīks gadījumos, kad dati tiek novērtēti izmantojot cilvēka resursus. Lai nodrošinātu modeļa izstrādi arī liela apjoma laikrindu gadījumā, kas liedz apskatīt un izanalizēt visas laikrindas, darbā tiek likts uzsvars uz metodēm, kas ļauj vispārināt prognozes veikšanu uz kādu sensoru grupu ar līdzīgu uzvedību. Tas tiks darīts, lai samazinātu gan modeļu izstrādes, gan arī modeļu trenēšanas laiku.

Darbā arī tiek aplūkotas laikrindu sacensības, kur pētnieki no vairākām pasaules valstīm vairāku gadu garumā cenšas uzlabot un izstrādāt tādus modeļus, kas pēc iespējas labāk strādātu uz dažāda veida laikrindām, un kā to rezultāti ir mainījušies gadiem ejot no sacensības uz sacensību.

Modeļa ansambļa izveides procesā tiks apskatītas un izmantotas populārākās laikrindu analīzes un modelēšanas metodes. Darba izpētes procesā tika izstrādāta modeļa izstrādes gaitas diagramma, ko var aplūkot 1. attēlā.



1. att. **Datu analīzes un modeļa izstrādes diagramma**

Darbs ir strukturēts tā, lai tas būtu pēc iespējas tuvāks laikrindu datu analīzes un modeļa izstrādes soļu secībai, ko var aplūkot 1. attēlā. Attēlā tiek secīgi attēlots katrs modeļa izstrādes posms, ar neraustītas taisnes bultu tiek norādīta normāls datu plūsmas vai darbības secības virzies. Savukārt, ar raustītas taisnes bultu tiek norādīts datu plūsmas vai darbību atkārtošanas virzies, ko nepieciešams ievērot pēc nepieciešamības. Katrs no diagrammā redzamajiem blokiem tiks aprakstīts tam paredzētajā nodaļā.
Darbā tiks apskatītas sekojošas tēmas:

- **Datu ieguve** - tā kā orģināli analizētos datus nav iespējams izmantot darba ietvaros, tad darbā tiks veiktas datu simulācijas un tas būs pamata datu iegūšanas veids. Bet dati tik pat labi varētu nākt no kāda cita avota, piemēram, no datu noliktavas.

- **Laikrindu vispārējā analīze un izpēte** - tiks apskatītas, lielākoties ekonometrijas, metodes, kas pirms modelēšanas palīdzēs iepazīt datus un to struktūru.

- **Datu sagatavošana un priekšapstrāde** - šajā tēmā ietilpst viss, kas ir saistīts ar datu tīrīšanu, iztrūkstošu datu identificēšanu un citu datu problēmu risināšanu, kā arī datu sagatavošanu priekš katra konkrētā modeļa specifikas un vajadzībām.

- **Klāsterēšana** - pie klāsterēšanas tiks aprakstīti laikrindu klāsterēšanas algoritmi, metodes un dažādas klāsterēšanas pieejas.

- **Prognozēšana** - prognozēšanas nodaļa tiks sadalīta 3 daļās, kur katrā no sadaļām tiks apskatīti modeļi no attiecīgās modeļu grupas - statistiskie, mašīnmācīšanās (neskaitot dziļās mācīšanās modeļus) un dziļās mācīšanās modeļi. Ir vērts piezīmēt, ka šis grupējums nav oficiāls grupējums, bet tāds grupējums, lai darba ietvaros būtu vieglāk nošķirt dažāda tipa modeļus.

- **Anomāliju noteikšana** - nodaļā tiks izmantoti iepriekš iegūtie rezultāti un modeļi, no klāsterēšanas un prognozēšanas nodaļām, lai veidotu modeļus, kas klasificēs novērojumus sekojošās divās klasēs - normāls vai anomālija.

- **Modeļu ansambļa izveide** - šeit tiks apskatīts, kā apvienot modeļus modeļu ansamblī, kur katrs no atsevišķajiem modeļiem kalpo kā balsotājs, tādējādi padarot gala prognozi robustāku.

Ir vērts piezīmēt, ka darba ietvaros netiek apskatīta modeļu atkārtotas trenēšanas metodes, optimālo klāsteru skaita noteikšanas tehnikas un vispārēja modeļu uzturēšana prakse produkcijas vidē.

Analīzei, modelēšanai, grafiku zīmēšanai un tabulu veidošanai tiks izmantota Python programmēšanas valodu. JupyterLab darba grāmatu (*JupyterLab notebook*) saturs ir pieejams pielikumā, kā arī B.7. pielikumā ir iekļaut YAML faila saturs ar izstrādes vides iestatījumiem.

1. LAIKRINDU SACENSĪBAS

Makridakis sacensības, jeb M-sacensības, ir laikrindu prognozēšanas sacensības, kur pētnieki no visas pasaules sacenšas, lai izstrādātu modeļus/metodes, kas spētu visprecīzāk prognozēt dažāda veida laikrindas. Šīs sacensības ir aizsācis *Spyros Makridakis* un ar šo sacensību palīdzību viņa mērķis ir bijis dziļāk izpētīt laikrindu prognozēšanas modeļus un metodes tieši no praktiskā pielietojuma puses. Sacensību galvenais mērķis ir attīstīt un popularizēt laikrindu prognozēšanas metodes. Un tieši empīriski pētījumi palīdz saprast metožu labās un sliktās puses, jo nepastāv universāli labākā metode un galvenais mērķis risinot kādu problēmu ir piemeklēt piemērotāko metodi tieši attiecīgajai problēmai. Tāpēc sacensību uzsvars ir ne tik daudz kā atrast uzvarētāju vai zaudētāju, bet saprast kuri modeļi un metodes strādā labāk kurās situācijās. [15]

1.1. M1 sacensības

Sacensību rezultāti tika publicēti **1982**. gadā. [10] Sacensībās modeļi tika novērtēti uz **1001** dažādām laikrindām. Laikrindas atšķirās gan ar to tipiem/veidiem - makro, mikro, industrijas un demogrāfijas, gan ar periodu garumu starp novērojumiem - mēnesis, ceturksnis un gads. Prognozēto periodu skaits atšķīrās, atkarībā no periodu garuma starp novērojumiem - priekš mēnešu datiem prognoze tika veikta 18 mēnešiem uz priekšu, ceturkšņa datiem prognoze tika veikta 8 ceturkšņiem uz priekšu, savukārt gada datiem prognoze tika veikta 6 gadiem uz priekšu.

Kopā tika apskatītas **15** metodes/modeļi, ar papildus 9 šo metožu/modeļu variācijām. Sacensībās tika izmantoti tādi modeļi kā:

- Slīdošā vidējā (MA) modelis
- Dažādas eksponenciāli nogludinošā modeļa variācijas
- Autoregresijas slīdošā vidējā (ARMA) modelis
- ARARMA modelis
- Lineārās regresijas modelis
- Dažādas iepriekš uzskaitīto modeļu kombinācijas un variācijas.

Ir vērts pieminēt, ka šajās sacensībās, metode, kas prasīja visvairāk datorresursu un kopējo aprēķina laiku bija ARMA. Vidēji uz katru laikrindu ARMA modeļu trenēšana aizņēma aptuveni vienu stundu, kas iekļāva gan manuālu cilvēka analīzes darbu, gan arī modeļa trenēšanu.

Galvenās modeļu precizitātes novērtēšanas metodes, lai salīdzinātu metodes savā starpā - vidējā kvadrātiskā kļūda (Mean Square Error - MSE), vidējais rangs (Average Ranking - AR) un absolūto kļūdu procentu mediānas (Medians of absolute percentage errors - Md)

Secinājumos darba autors nav uzsvēris nevienu no metodēm kā labāku vai sliktāku. Balstoties uz rezultātiem ir redzams, ka ir metodes, kas ir spēcīgākas prognozējot viena tipa un/vai

perioda garuma laikrindas un ir metodes, kas ir spēcīgākas prognozējot cita tipa un/vai perioda garuma laikrindas. Tā kā galvenais sacensību mērķis ir apkopot un apskatīt populārākās laikrindu prognozējošās metodes, lasītājam tiek dota iespēja izvērtēt un izvēlēties piemērotāko metodi, kas devusi labākos rezultātus priekš laikrindas struktūras, kas ir vislīdzīgākā lasītāja aktuālajai problēmai.

Četras galvenās atziņas, ko autors minējis darba noslēgumā:

- Uz spriedumiem balstīta pieeja nav viennozīmīgai precīzāka kā objektīvas metodes.
- Uz cēloņsakarībām balstītas metodes nav viennozīmīgi precīzākas kā ekstrapolācijas metodes.
- Sarežģītākas un statistiski attīstītākas metodes nav viennozīmīgi precīzākas kā vienkāršas metodes.
- Prognozes precizitāti, līdz ar to arī modeļa izvēli, ietekmē prognozes perioda garums un laikrindas detalizācija

1.2. M2 sacensības

Otro sacensību rezultāti [11] tika publicēt **1993**.gadā. Otrajās Makridakis sacensībās analizēto laikrindu daudzums bija ievērojami mazāks kā pirmajās sacensībās, tās bija 29 laikrindas. No šīm **29** laikrindām - 23 laikrindas nāca no četrām sadarbības kompānijām un 6 bija makroekonomikas rādītāju laikrindas. Šīs sacensības atšķīrās no iepriekšējām, ne tikai ar laikrindu apjomu, bet arī ar to ka tika iesaistīti 5 individuāli prognozētāji, kas veica prognozes izmantojot savu pieredzi un nozares zināšanām. Laikrindu perioda garums bija mēnesis un prognozes tika veiktas 15 mēnešiem uz priekšu un pēc 12 mēnešiem tika veikta atkārtota prognožu veikšana.

Kopā tika apskatītas **16** prognozēšanas metodes/modeļi, iekļaujot piecu ekspertu veiktās prognozes. Papildus tika novērtētas arī 2 kombinētie modeļi un visu modeļu prognožu vidējās vērtības prognoze. Šie 16 modeļi reprezentē 4 dažādas modeļu/metožu kategorijas:

- Naivie jeb bāzes modeļi (2 modeļi).
- Metodes, kas balstās uz slīdošo logu simulācijām (4 modeļi).
- Populāras nogludināšanas metodes (4 modeļi).
- *Box-Jenkins* metodoloģijas modelis (1 modelis).
- Ekspertu veikta prognoze (5 individuāli prognozētāji).

Šo sacensību rezultāti tika vērtēti izmantojot absolūto kļūdas procentu vidējais aritmētiskais (*Mean absolute percentage error* - MAPE). Kopumā rezultāti un līdz ar to secinājumi ir līdzīgi kā iepriekšējās sacensībās, bet ir vērt pieminēt, ka šajās sacensībās bija divas metodes, kas īpaši

izcēlās uz pārējo metožu fona. Šajās sacensībās ar precizitāti izvēlās sekojošas metodes - **Ierobežotā eksponenciālās gludināšanas** un **Vienkāršā eksponenciālā gludināšanas** metodes.

Darba galvenās atziņas:

- Cilvēku veiktās prognozes, kas tika balstītas pamatā uz ekspertu zināšanām, nebija precīzākas kā prognozes, ko deva izmantotie algoritmi.
- Labākās metodes izrādījās vienkāršākās metodes - Ierobežotā un vienkāršā eksponenciālā gludināšanas metodes, kā arī eksponenciāli nogludinošo metožu kombinācija.
- Vienkāršākās metodes ir ne tikai precīzākas, bet arī vieglāk saprotamas un interpretējamākas.
- Šīs vienkāršās prognozēšanas metodes var tikt izmantotas priekš budžeta prognozēšanas.

1.3. M3 sacensības

Trešo Makridakis laikrindu sacensību rezultāti [12] tika publicēti **2000**.gadā un šajās sacensībās prognozēšanas metodes tika testētas uz **3003** dažādām laikrindām. Starp šīm 3003 laikrindām bija sekojošu tipu/veidu laikrindas - mikro, makro, industrijas, finanšu, demogrāfijas un cita tipa laikrindas. Laikrindu novērojumu periodu garums bija mēnesis, ceturksnis, gads un mazai daļai no laikrindām perioda garums nebija konkrēti definēts.
Šajās sacensībās tika izmantotas **24** prognozēšanas metodes no sekojošām kategorijām:

- Naivās jeb bāzes metodes (2 metodes)
- Trenda modeļi (7 modeļi)
- Dekompozīcijas metode (1 metode)
- ARIMA/ARARMA modeļu variācijas (7 modeļi)
- Eksportu sistēmas (6 sistēmas)
- Neironu tīkli (1 modelis)

Metožu prognozes tika salīdzinātas izmantojot sekojošas piecas precizitātes metrikas - simetriska MAPE, vidējais rangs, simetriska APE (absolūtais kļūdas procents) mediāna, procentuālais uzlabojums (*percentage better*) un REA (relatīvā absolūtā kļūda). Metodes tika salīdzinātas uz dažādu prognozes periodu garumu prognožu precizitāti. Spriežot pēc rezultātiem - tie atšķīrās atkarībā no laikrindas veida un perioda garuma. Darba autors nav nosaucis vienu metodi, kas būtu labāka visās situācijās, bet ir pāris metodes, kas ir izteikti labākas par pārējām metodēm konkrēta veida vai perioda garuma laikrindu prognozēšanā.
Darba galvenās atziņas:

- Jaunā metode - Theta, lai gan ļoti vienkārša, bet parāda ļoti labus rezultātus.
- Makridakis sacensībās iepriekš neizmantotā ekspertu sistēma - ForecastPro, arī parādījusi labus rezultātus uz tādu statistisku modeļu fona kā, piemēram, ARIMA.
- *Robust-Trend* metode pārspēj visas metodes, tieši laikrindu prognozēšanā, kur laikrindu perioda garums ir gads.

1.4. M4 sacensības

Ceturtās Makridakis sacensību pirmie rezultāti [13] tika publicēti **2018**.gadā un pēc tam atjaunotie rezultāti tika publicēti **2019**.gadā. Šajās sacensībās metodes tika testētas uz **100 000** laikrindām, kas ir ievērojami vairāk kā iepriekšējās sacensībās. Šie dati bija dažāda tipa un šo datu periodu garumi bija - stunda, diena, nedēļa, mēnesis, ceturksnis un gads. Arī izmantoto metožu skaits šajās sacensībās ir krietni lielāks, nekā iepriekšējās - **61** metodes, no tām 49 metodes izstrādāja sacensību dalībnieki un 12 bija iepriekš zināmas metodes no iepriekšējām sacensībām, kas kalpoja kā atskaites punkts, lai novērtētu dalībnieku izveidoto modeļu jaudu.

Lielākā daļa metožu, kas ir labāko metožu vidū, ir nosauktas izstrādātāju vārdos un ir hibrīdmodeļi no zināmām metodēm vai izmanto kāda jau zināma mašīnmācīšanās algoritma principus. Vairāk par izmantotajām metodēm var apskatīt sacensību rakstā [13].

Papildus precizitātes novērtēšanai, šajās sacensībās tika vērtēti arī metožu skaitlisko aprēķinu izpildes laiki. Atšķirībā no iepriekšējām sacensībām, šajās sacensībās tika novērtēts uzvarētājs un šajās sacensībās uzvarēja Slawek Smyl no ”Uber Technologies”, kas izmantoja eksponenciālās gludināšanas un RNN modeļu apvienojumu.
Darba galvenās atziņas:

- Nepieciešams attīstīt prognožu veikšanas teoriju un papildināt to ar jaunām metodēm.
- Attīstīt jaunu prognozējošo modeļu izstrādi.
- Parastas mašīnmācīšanās metodes nespēj konkurēt ar vienkāršām ekonometrijas metodēm vai hibrīdmodeļiem, ko pierāda šo sacensību rezultāti.
- Apvienojot vairāku modeļu prognozes, veidojot tā sauktos modeļu ansambļus, uzlabojas prognozes precizitāte.

1.5. M5 sacensības

Piekto Makridakis sacensību rezultāti [14] tika pulicēti, salīdzinoši nesen, **2020**.gada oktobrī. Atšķirībā no iepriekšējām sacensībām, šīs sacensības organizēja Kaggle, populāra datu zinātnes vietne, kas organizē dažādas, ar datu zinātni saistītas, sacensības. Šo sacensību kopējais balvu fonds bija **$100 000**, kas tika sadalīts starp pirmajām 5 vietām. Sacensības norisinājās vairākās fāzēs un tajās varēja piedalīties jebkurš, kurš vēlējās izmēģināt savu roku jaunu metožu/modeļu

izstrādē. Sacensībās izmantotās metodes tika testētas uz aptuveni **42 000** dažādām laikrindām ar hierarhisku struktūru. Metodes, kas tikušas izmantotas šajās sacensībās ir dažādas sākot ar vienkāršām metodēm, līdz pat dziļās mācīšanās metodēm un to hibrīdmodeļiem.

Modeļu precizitāte tiek vērtēta izmantojot vidējo kvadrātisko mērogoto kļūdas sakni (RMSSE). Līdzīgi, kā M4 sacensībās, labākie modeļi bija tieši hibrīdmodeļi, kas sastāvēja no vairāku modeļu kombinācijas. Un arī šajās sacensībās, tāpat kā M4 sacensībās, tika izmantoti bāzes modeļi, lai vērtētu dalībnieku modeļu jaudu pret bāzes modeļiem. Sacensībās uzvarēja komanda ar nosaukumu - YJSTU, kas izmantoja vairāku LightGBM modeļu kombināciju.
Darba galvenās atziņas:

- Pirmo 50 labāko modeļu precizitāte, salīdzinājumā ar precīzāko bāzes modeli, bija par 14% augstāka. Savukārt, pirmo piecu labāko modeļu precizitāte bija par 20% augstāka par bāzes modeļu precizitāti un labākajam modelim precizitāti bija pat par 22% augstāka kā tuvākā bāzes/naivā modeļa precizitāte.

- Interesanti, ka labākais modelis nav bijis vislabākais visos agregācijas līmeņos, bet tas ir bijis precīzākais lielākajā daļu agregācijas līmeņu.

- Precīzāko modeļu precizitāte krītas zemāku agregācijas līmeņu novērojumu prognozēšanā.

- Šo sacensību labākais un efektīvākais modelis ir **LightGBM**, kas tika izmantots, kā komponente, 4 no 5 precīzākajiem modeļiem.

1.6. Secinājumi

Ir skaidrs, ka šādas sacensības rada pozitīvu ietekmi jaunu metožu izpētē un radīšanā. Balsoties uz pēdējo sacensību rezultātiem ir redzams, ka praktiskos pielietojumos vienkāršas metodes vai parastas statistikas metodes vai arī vienkāršas mašīnmācīšanās metodes nespēj konkurēt ar hibrīd modeļiem, kas kā pamatu izmanto jaunākās metodes, kā piemēram, LightGBM un neironu tīklus. Bet ir vērts piezīmēt, ka ir situācijas, kur tomēr ir iespējams efektīvi un precīzi veikt prognozes ar vienkāršākām metodēm un to priekšrocība, ja precizitāte ir pietiekami līdzīga, ir tāda, ka tie ir vieglāk saprotami un tos ir vieglāk interpretēt. Šo sacensību rezultāti var tikt izmantoti praksē - sākumā izstrādājot vienkāršākus modeļus, kas kalpotu kā bāzes modeļi un iedvesmoties no labākajiem šo sacensību modeļiem, lai censtos uzlabot bāzes modeļu precizitāti. Visu piecu sacensību apkopojumu var aplūkot 1. tabulā.

1. tabula

Laikrindu sacensību rezultātu apkopojums

Sacensības	Gads	Laikrindu skaits	Laikrindu tipi	Laikrindu periodi	Modeļu/metožu skaits	Labākie modeļi/metodes
M1	1982	1001	Makro, mikro, industrijas un demogrāfijas	Mēnesis, ceturksnis un gads	15	-
M2	1993	29	Makroekonomikas	-	16	Vienkāršā un ierobežotā eksponenciālās gludinās...
M3	2000	3003	Mikro, makro, industrijas, finanšu, demogrāfija...	Mēnesis, ceturksnis un gads	24	Theta, ForecastPro un Robust-Trend
M4	2018	100000	Dažāda veida	Stunda, diena, nedēļa, mēnesis, ceturksnis un gads	61	Eksponenciālās gludināšanas un LSTM modeļu apvi...
M5	2020	42000	Dažāda veida	Hierarskiska laikrindu datu struktūra	-	LightGBM un LSTM

2. LAIKRINDU GALVENĀS PAMATDEFINĪCIJAS

Šajā nodaļā tiek definētas pamatdefinīcijas galvenajiem laikrindu jēdzieniem un metodēm. Atsevišķi jēdzieni un metodes, kas attiecas uz konkrētu nodaļu, tiek definētas un aprakstītas attiecīgajā nodaļā kurā tās tiek izmantotas. Šajā nodaļā definētās definīcijas pamatā ir ņemtas no ekonometrijas kursa konspekta.

2.1. Definīcija. Laikrinda ir novērojumu x_t kopa, kur katrs novērojums ir iegūts kādā fiksētā laikā $t \in T$. Diskrēta laika laikrindas gadījumā T ir diskrēta kopa, kur, piemēram, laikrindu novērojumi savākti pa stundām, dienām vai gadiem (parasti $T = \{0\} \cup \mathbb{N}$ vai $T = \mathbb{Z}$). Šajā kursā aplūkosim tieši šādas laikrindas. Nepārtraukta laika laikrinda tiek iegūta, kad novērojumi tiek iegūti nepārtraukti kādā intervālā, tas ir, $T = [a,b]$ vai $T = \mathbb{R}$.

2.2. Definīcija. Laikrindu modelis novērotiem datiem $\{x_t\}$ ir gadījuma procesa $\{X_t\}$ kopējo sadalījumu (vai dažkārt tikai vidējās vērtības un kovariāciju) specifikācija. Šajā gadījuma laikrinda $\{x_t\}$ ir uzskatāma ka $\{X_t\}$ procesa realizācija.

2.3. Definīcija. Pieņemsim, ka $\{X_t\}$ ir laikrinda ar $E(X_t) < \infty$. Tad $\{X_t\}$ vidējās vērtības funkcija ir

$$\mu_X(t) = E(X_t).$$

Savukārt $\{X_t\}$ kovariāciju funkcija tiek uzdota ar

$$\gamma_X(r,s) = \text{Cov}(X_r, X_s) = E[(X_r - \mu_X(r))(X_s - \mu_X(s))]$$

visiem $r \in \mathbb{Z}$ un $s \in \mathbb{Z}$.

2.4. Definīcija. $\{X_t\}$ ir vāji stacionārs, ja

i $\mu_X(t)$ ir neatkarīgs no t;

ii $\gamma_X(t+h,t)$ ir neatkarīgs no t katram $h \in \mathbb{Z}$

2.5. Definīcija. Pieņemsim, ka $\{X_t\}$ ir stacionāra laikrinda. **Autokovariāciju funkcija** (ACVF) laika nobīdei jeb **lagam** h ir

$$\gamma_X(h) = \text{Cov}(X_{t+h},X_t).$$

Savukārt **autokorelāciju funkcija** (ACF) ir

$$\rho_X(h) = \frac{\gamma_X(h)}{\gamma_X(0)} = \text{Cor}(X_{t+h},X_t).$$

2.6. Definīcija. Pieņemsim, ka $x_1, \ldots, x_n$ ir laikrindas novērojumi. Tad izlases vidējā vērtība ir

$$\bar{x} = \frac{1}{n} \sum_{t=1}^{n} x_t.$$

Izlases autokovariāciju funkcija ir

$$\hat{\gamma}(h) = \frac{1}{n} \sum_{t=1}^{n-|h|} (x_{t+|h|} - \bar{x})(x_t - \bar{x}),\ -n < h < n.$$

Izlases autokorelāciju funkcija ir

$$\hat{\rho}(h) = \frac{\hat{\gamma}(h)}{\hat{\gamma}(0)},\ -n < h < n.$$

2.7. Definīcija. Nesezonālais modelis ar trendu.

$$X_t = m_t + Y_t,\ t = 1, \ldots, n,$$

kur $EY_t = 0$.

Ja $EY_t \neq 0$, tad var aizvietot m_t un Y_t ar $m_t + EY_t$ un $Y_t - EY_t$.

2.8. Definīcija. Definēsim tā saukto viena laga jeb vienas laika vienības **diferenču operatoru** ∇,

$$\nabla X_t = X_t - X_{t-1} = (1 - B)X_t,$$

kur B ir **nobīdes operators**

$$BX_t = X_{t-1}.$$

Šajā gadījumā definēsim attiecīgās operatoru pakāpes $B^j(X_t) = X_{t-j}$ un $\nabla^j(X_t) = \nabla(\nabla^{j-1}(X_t))$ visiem $j \geq 1$, kur $\nabla^0(X_t) = X_t$.

2.9. Definīcija. Klasiskais dekompozīcijas modelis

$$X_t = m_t + s_t + Y_t,\ t = 1, \ldots, n,$$

kur $EY_t = 0$, $s_{t+d} = s_t$ un $\sum_{j=1}^{d} s_j = 0$.

2.10. Definīcija. Definēsim tā saukto d-laga jeb laika nobīdes d diferenču operatoru ∇_d

$$\nabla_d X_t = X_t - X_{t-d} = (1 - B^d)X_t,$$

kur laika nobīdes operators B^d ir definēts agrāk.

2.11. Definīcija. Pieņemsim, ka $\{X_t\}$ ir stacionāra laikrinda. **Autokovariāciju funkcija** (ACVF) laika nobīdei jeb **lagam** h ir

$$\gamma_X(h) = \text{Cov}(X_{t+h},X_t).$$

Savukārt **autokorelāciju funkcija** (ACF) ir

$$\rho_X(h) = \frac{\gamma_X(h)}{\gamma_X(0)} = \text{Cor}(X_{t+h},X_t).$$

2.12. Definīcija. Reālvērtīga funkcija κ definēta uz veseliem skaitļiem ir nenegatīvi definīta, ja

$$\sum_{i,j=1}^{n} a_i\kappa(i-j)a_j \geq 0$$

visiem pozitīviem skaitļiem n un vektoriem a = $(a_1,\ldots,a_n)'$, kur $a_i \in \mathbb{R}$.

2.13. Definīcija. q-tās kārtas slīdošā vidējā MA(q) procesu $\{X_t\}$ uzdod ar

$$X_t = Z_t + \theta_1 Z_{t-1} + \ldots + \theta_q Z_{t-q},$$

kur $\{Z_t \sim WN(0,\sigma^2)\}$ un procesa koeficienti $\theta_1,\ldots,\theta_q$ ir reālvērtīgas konstantes. Pie tam $\theta_q \neq 0$.

2.14. Definīcija. Procesu $\{X_t\}$ sauc par lineāru, ja tam ir sekojoša reprezentācija

$$X_t = \sum_{j=-\infty}^{\infty} \psi_j Z_{t-j} \tag{2.1}$$

visiem t, kur $\{Z_t\} \sim WN(0,\sigma^2)$ un $\{\psi\}$ ir reālvērtīgu konstanšu virkne ar $\sum_{j=-\infty}^{\infty} |\psi_j| < \infty$. Izmantojot nobīdes operatoru var pierakstīt

$$X_t = \psi(B)Z_t,$$

kur $\psi(B) = \sum_{j=-\infty}^{\infty} \psi_j B^j$. Lineāru procesu sauc par slīdošā vidējo procesu $MA(\infty)$, ja $\psi_j = 0$ visiem $j < 0$, tas ir, ja

$$X_t = \sum_{j=0}^{\infty} \psi_j Z_{t-j}.$$

2.15. Definīcija. Laikrindu $\{X_t\}$ sauc par **ARMA(1,1) procesu**, ja tā ir stacionāra un izpildās katram t,

$$X_t - \phi X_{t-1} = Z_t + \theta Z_{t-1},$$

kur $\{Z_t\} \sim WN(0,\sigma^2)$ un $\phi + \theta \neq 0$. Izmantojot nobīdes operatoru B, varam rakstīt

$$\phi(B)X_t = \theta(B)Z_t, \tag{2.2}$$

kur $\phi(B) = 1 - \phi B$ un $\theta(B) = 1 + \theta Z_t$ ir lineāri filtri.

2.16. Definīcija. $\{X_t\}$ sauc par **ARMA(p,q)** procesu, ja $\{X_t\}$ ir stacionārs un katram t

$$X_t - \phi_1 X_{t-1} - \ldots - \phi_p X_{t-p} = Z_t + \theta_1 Z_{t-1} + \ldots + \theta_q Z_{t-q}, \tag{2.3}$$

kur $\{Z_t\} \sim WN(0,\sigma^2)$ un polinomiem $(1 - \phi_1 z - \ldots - \phi_p z^p)$ un $(1 + \theta_1 z + \ldots \theta_q z^q)$ nav kopīgu dalītāju.

Procesu $\{X_t\}$ sauc par **ARMA(p,q)** procesu ar vidējo vērtību μ, ja $\{X_t - \mu\}$ ir ARMA(p,q) process.

2.17. Definīcija. Laikrindu $\{X_t\}$ sauc par p**-tās kārtas autoregresīvo procesu** (AR(p)), ja $\theta(z) = 1$. Savukārt, ja $\phi(z) = 1$, tad to sauc par q**-tās kārtas slīdošā vidējo procesu** (MA(q)).

2.18. Definīcija. Par **parciālo autokorelācijas funkciju (PACF)** ARMA procesam $\{X_t\}$ sauc funkciju $\alpha(\cdot)$, kas definēta ar

$$\alpha(0) = 1$$

un

$$\alpha(h) = \phi_{hh},\ h \geq 1,$$

kur ϕ_{hh} iegūstams no

$$\boldsymbol{\phi}_h = \Gamma_h^{-1} \boldsymbol{\gamma}_h, \tag{2.4}$$

$\Gamma_h = [\gamma(i-j)]_{i,j=1}^h$ un $\gamma_h = [\gamma(1), \gamma(2), \ldots, \gamma(h)]'$. Savukārt dotiem novērojumiem $\{x_1, \ldots x_n\}$ ar $x_i \neq x_j$ visiem i un j **izlases PACF** $\hat{\alpha}(h)$ ir uzdota ar vienādojumiem

$$\hat{\alpha}(0) = 1$$

$$\hat{\alpha}(h) = \hat{\phi}_{hh},\ h \geq 1,$$

kur $\hat{\phi}_{hh}$ ir pēdējā komponente

$$\hat{\boldsymbol{\phi}}_h = \hat{\Gamma}_h^{-1} \hat{\boldsymbol{\gamma}}_h$$

ARIMA modelis tiek definēts sekojoši:

2.19. Definīcija. Ja d ir nenegatīvs vesels skaitlis, tad $\{X_t\}$ sauc par ARIMA(p,d,q) procesu, ja $Y_t := (1-B)^d X_t$ ir kauzāls ARMA(p,q) process un $\{X_t\}$ pieraksta kā

$$\phi^*(B) X_t := \phi(B)(1-B)^d X_t = \theta(B) Z_t,\ \{Z_t\} \sim WN(0,\sigma^2),$$

kur $\phi(z)$ un $\theta(z)$ ir p un q-tās kārtas polinomi un $\phi(z) \neq 0$ visiem $|z| \leq 1$. Savukārt polinomam $\phi^*(z)$ ir sakne, kad $z = 1$. Process $\{X_t\}$ ir stacionārs tad un tikai tad ja $d = 0$, kad tas reducējas uz ARMA(p,q) procesu. ARIMA modeļi ir piemēroti modeļiem ar lineāru trendu, tomēr tie var būt piemēroti arī modeļiem bez trenda.

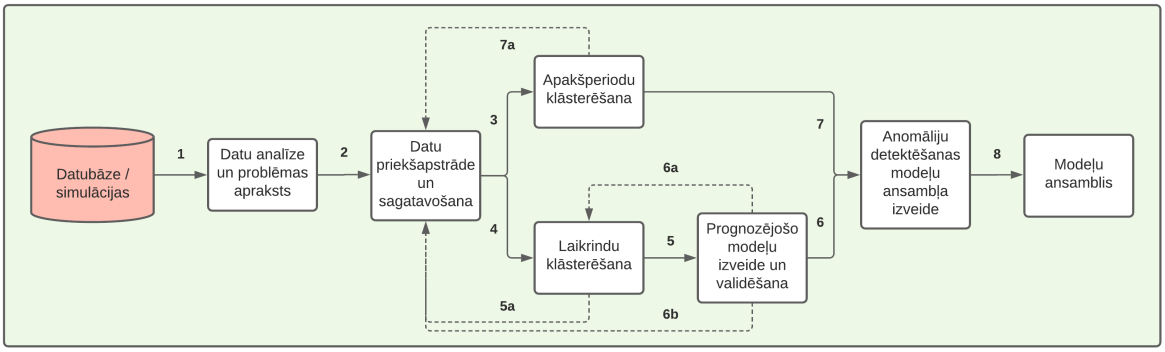
SARIMA modelis tiek definēts sekojoši:

2.20. Definīcija. Ja d un D ir nenegatīvi veseli skaitļi, tad $\{X_t\}$ sauc par sezonālo ARIMA(p,d,q) $\times$ (P,D,Q)$_s$ procesu ar periodu s, ja diferencētais laikrindu process $Y_t = (1-B)^d(1-B^s)^D X_t$ ir kauzāls ARMA process, kuru definē

$$\phi(B)\Phi(B^s)Y_t = \theta(B)\Theta(B^s)Z_t,\ \{Z_t\} \sim WN(0,\sigma^2), \tag{2.5}$$

kur $\phi(z) = 1 - \phi_1 z - \cdots - \phi_p z^p$, $\Phi(z) = 1 - \Phi_1 z - \cdots - \Phi_p z^P$, $\theta(z) = 1 + \theta_1 z + \cdots + \theta_q z^q$ un $\Theta(z) = 1 + \Theta_1 z + \cdots + \Theta_Q z^Q$.

3. DATU APRAKSTS UN SIMULĀCIJU IZVEIDE



2. att. **Datu analīzes un modeļa izstrādes diagramma, simulācijas fāze**

Sākot darbu ar jebkāda veida datiem vispirms ir nepieciešams tos iegūt, lai to izdarītu ir pamatā divi veidi - izmantot reālus datus vai veikt simulācijas (skatīt 2. attēlu). Šī darba ietvaros tiks apskatīti dati, kas tiks iegūti veicot simulēti. Lai gan darbā izmantotie dati būs simulēti, turpmākā analīze un modeļu izstrādi būtiski neatšķiras, vienīgā būtiskākā atšķirība ir tāda, ka simulētiem datiem ir pieejami visi datu punkti, ar to ir domāts, ka šajos datos nav iztrūkstošu datu un mums ir zināmas līdzīgo laikrindu grupas, jo to ir iespējams kontrolēt simulāciju procesā. Simulāciju darba grāmatu var aplūkot B.1. pielikumā.

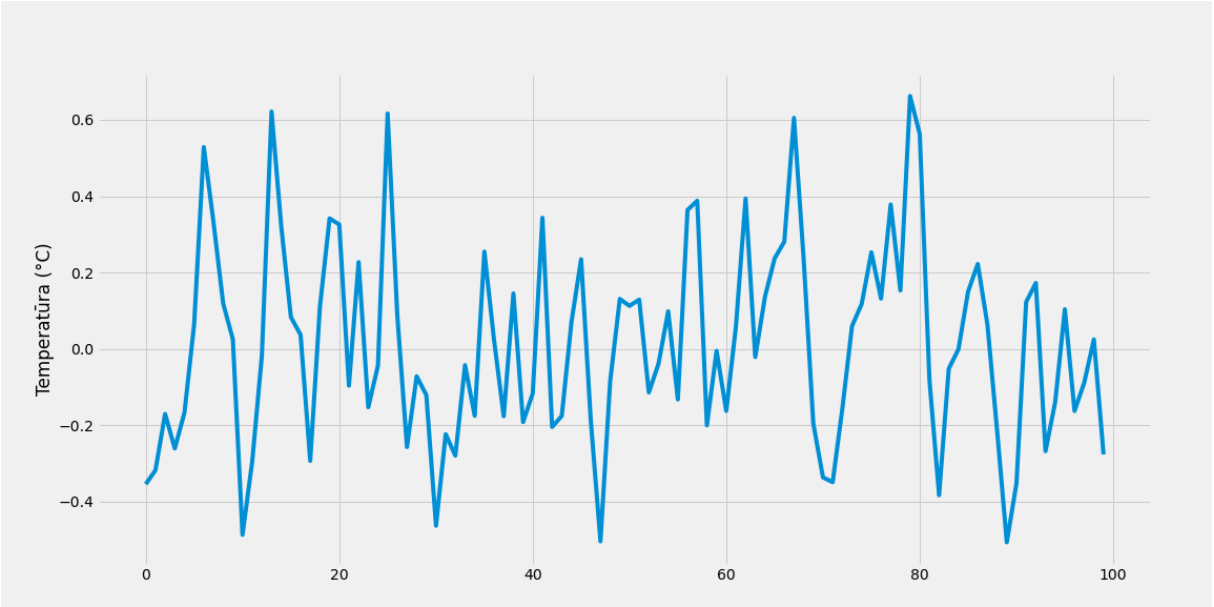
Darbā kā piemērs tiek apskatīti temperatūras sensoru mērījumu dati ar konkrētām īpašībām un uzvedību. Simulēto sensoru laikrindu uzvedība tiek konstruēta balstoties uz projektā pētītajām laikrindām. Izvēlēto sensoru mērījumi tiek veikti iekārtām, kurām, dēļ atkausēšanas periodiem, ir izteikta sezonāla temperatūras svārstība. Tāpēc darbā apskatāmajām laikrindām ir divi stāvokļi jeb periodi - **normālas darbības** un **atkausēšanas**. Simulācijas tiek veidotas tā, lai spētu pēc iespējas tuvāk replicēt laikrindas, kas satur abus iepriekš minētos periodus, un spētu veidot dažādas šo laikrindu variācijas. Lai to panāktu ir izveidota funkcija (funkciju *ts_sim* skatīt pielikumā pievienotajā darba grāmatā B.1.), kas pieņem sekojošu parametrus:

- n_regular - normālas darbības perioda garums (laika vienībās).
- regular_scale - normālas darbības perioda temperatūras svārstības izkliede/standartnovirze.
- regular_noise_sigma - normālas darbības perioda pievienotā baltā trokšņa izkliede/standartnovirze
- n_defrost - atkausēšanas perioda garums.
- n_defrost_skew - atkausēšanas perioda asimptotikas/nobīdes koeficients.
- defrost_scaler - atkausēšanas perioda pieaugums.

- defrost_noise_sigma - atkausēšanas perioda pievienotā trokšņa izkliede/standartnovirze.
- n_ts - simulējamās laikrindas kopējais garums.
- ts_mean - simulējamās laikrindas normālas darbības perioda vidējā vērtība.

3.1. Normālas darbības perioda simulācija

Simulējot normālas darbības temperatūras svārstību periodu tiek pieņemts, ka laikrinda šajā periodā temperatūra uzvedās kā autoregresīvs slīdošās vidējās vērtības (ARMA) process (definīciju skatīt 2. nodaļā. Visu variāciju un grupu laikrindu simulācijām tiek izmantots ARMA(1,1) process ar nejauši izvēlētiem koeficientiem (robežās no 0.1 līdz 0.5) pie AR un MA komponentēm. ARMA(1,1) procesa simulācija 100 laika soļiem un standartnovirzi 0.25 var aplūkot 3. attēlā. ARMA procesa simulācijas tika veiktas izmantojot *statsmodels* python pakotnes palīdzību [21].



3. att. **ARMA(1,1) procesa simulācija pirmie 100 novērojumi**

3.2. Atkausēšanas perioda simulācija

Atkausēšanas perioda mērījumu simulēšanā ir vairāk soļi nekā normālas darbības perioda simulācijā. Lai iegūtu atkausēšanas perioda mērījumus kā pamats tiek izvēlēts lietot sinusa funkciju $y = \sin(x)$, kur $x \in [0, \pi]$ 4a.. Lai kontrolētu atkausēšanas perioda formu sinusa funkcija tiek modificēta, lai ļautu mainīt būtu iespējams koriģēt sinusa asimetriju/nobīdi. Šim nolūkam tiek izvēlēts lietot sekojošu funkciju $y = \sin(x - \frac{y}{n})$, kur $x \in [0, \pi]$ un $n \in [-\infty, 0) \cup (0, \infty]$. Gadījumā kad $n \to \infty$, $\sin(x - \frac{y}{n}) = sin(x)$. Piemēru ar n = 1 skatīt 4b. attēlā.

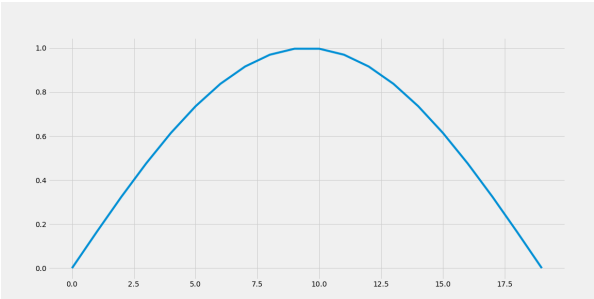
Papildus iegūtajai atkausēšanas perioda temperatūras izmaiņu, ko raksturo sinusa funkcija, nepieciešams pievienot balto troksni 4c., kas raksturo nejaušas temperatūras izmaiņas atkausē-

šanas periodā. Baltais troksnis tiek pievienots veidojot laikrindu, kas ir tādā pašā garumā, kā atkausēšanas periods un to veidojot secīgi tiek nejauši izvēlētas vērtības no normālā sadalījuma

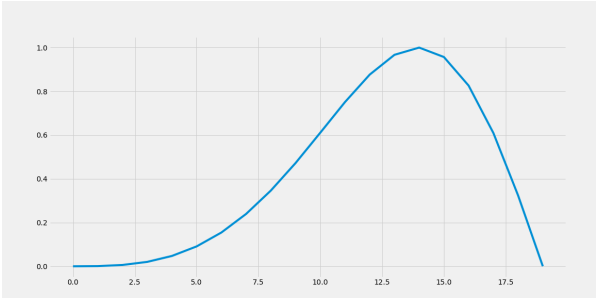
$$X \sim \mathcal{N}(\mu, \sigma^2)\,,$$

ar $\mu = 0$ un σ definētu pie simulāciju veikšanas.

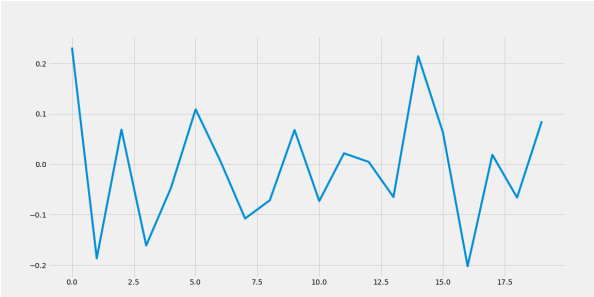
Kad ir izveidota vēlamā atkausēšanas period temperatūras vēlamā uzvedība un pievienots troksnis, rezultātā tiek iegūta atkausēšanas perioda simulācija 4d..



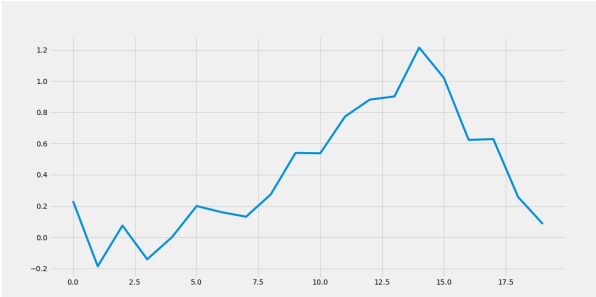
(a) $y = sin(x)$**, kur** $x \in [0, \pi]$



(b) $y = \sin(x - \frac{y}{n})$**, kur** $x \in [0, \pi]$ **un** $n = 1$



(c) **Baltais troksnis**



(d) **Atkausēšanas perioda simulācija**

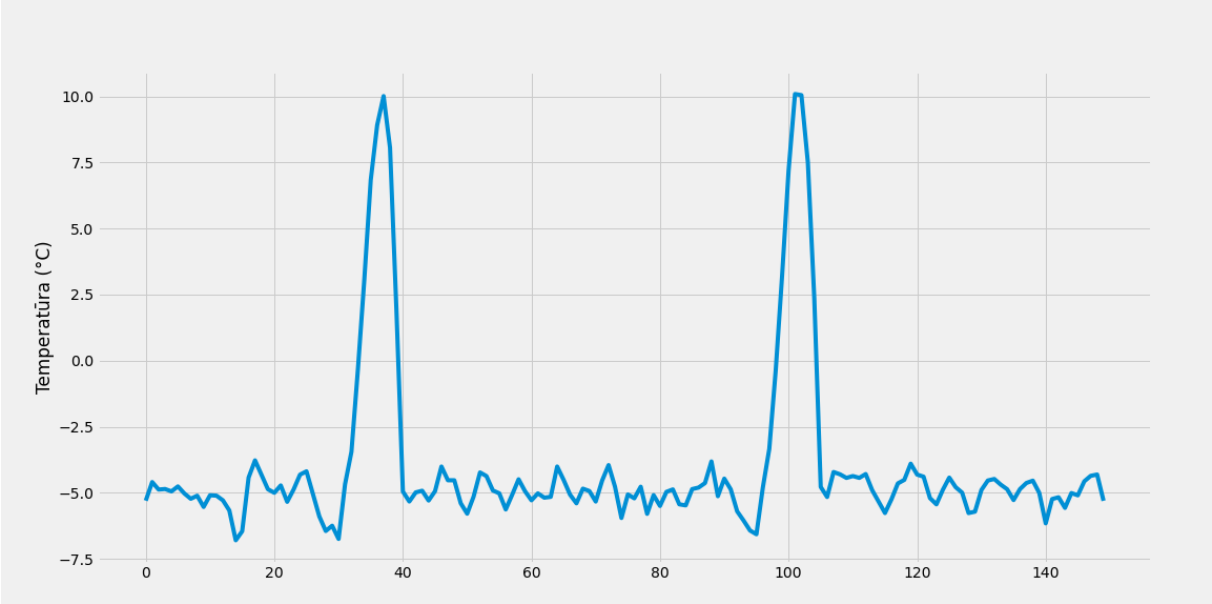
4. att. **Atkausēšanas perioda simulācijas process**

Visas četras atkausēšanas perioda simulācijas izstrādes posmus var aplūkot 4. attēlā. Attēlotajā piemērā atkausēšanas periods ir 20 laika vienību garš, bet veicot simulāciju ir iespējams mainīt tā garumu, kā arī ir iespēja mainīt gan sinusoīda nobīdi un baltā trokšņa standartnovirzi.

3.3. Simulētās laikrindas konstruēšana

Kad ir konstruētas laikrindas atsevišķu periodu simulācijas, tās var apvienot veidojot gala simulāciju. Tas tiek darīts vispirms izvēloties simulējamās laikrindas garumu (*n_ts*), kāda būs normālas darbības perioda vidējā vērtība (*ts_mean*) un temperatūras vērtību par cik temperatūra pieaugs atkausēšanas periodā (*defrost_scaler*). Izmantojot visus parametrus var aprēķināt cik sezonām būs nepieciešams veikt simulācijas un cikliski veidojot normālas darbības temperatūras uzvedības periodu un atkausēšanas periodu nosimulēt laikrindu (pareizinātu ar vēlamo temperatūras pieauguma vērtību) vajadzīgajā garumā un izvēloties nejaušu sākumpunktu no pirmās sezonas normālās darbības temperatūras mērījumu perioda un pēc tam pieskaitot laikrindas

normālās darbības perioda vidējās vērtības norādīto vērtību. Laikrindas sākumpunkts tiek izvēlēts nejauši, lai nodrošinātu to, ka laikrindas nesākas vienā un tajā pašā laika solī. Piemēru šādai simulētai laikrindai skatīt 5. attēlā, kur ir attēloti pirmie 150 novērojumi laikrindai ar vidējo normālās darbības perioda temperatūras vērtību - -5 grādi, normālās darbības perioda garumu - 50 laika vienības, normālās darbības perioda standartnovirzi - 0.5, atkausēšanas perioda garumu - 15 laika vienības, atkausēšanas perioda baltā trokšņa standartnovirzi - -0.25 un atkausēšanas perioda nobīdes koeficient *n* vienādu ar 0.8.



5. att. **Simulētas laikrindas piemērs**

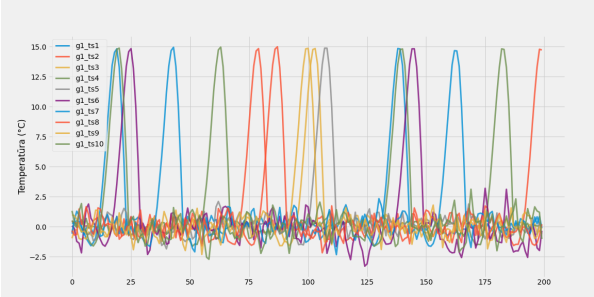
Kā var redzēt 5. attēlā, laikrinda nesākas no normālas darbības perioda pirmās laika vienības, laikrindā ir 3 pīķi, kas norāda atkausēšanās periodus un starp trim atkausēšanas periodiem ir divi pilni normālas darbības temperatūras mērījumu periodi.

3.4. Simulācijas

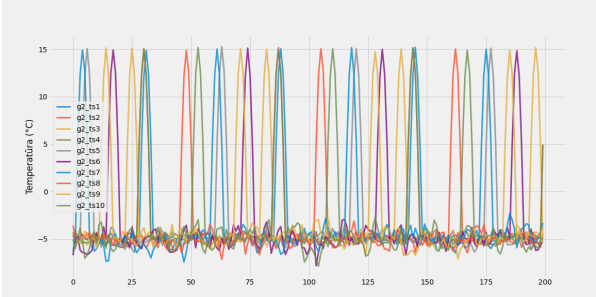
Līdz ko ir nodefinēts simulācijas process un funkcija, tiek veiktas simulācijas ar mērķi izveidot 3 dažādu grupu laikrindas, kur katrā grupā ir pa 10 laikrindām. Visu laikrindu kopējais garums ir 1000. Grupu apraksts ar to galveno parametru atšķirībām 6.:

- 1. Grupa: normālas darbības perioda garums ir 100 laika vienības, atkausēšanas perioda garums ir 20 laika vienības, atkausēšanas asimetrijas koeficients *n* ir 0.8, atkausēšanas perioda temperatūras pieaugums ir 15 grādi un normālas darbības perioda temperatūras vidējā vērtība ir 0 grādi. 6a.
- 2. Grupa: normālas darbības perioda garums ir 50 laika vienības, atkausēšanas perioda garums ir 7 laika vienības, atkausēšanas asimetrijas koeficients *n* ir 100, atkausēšanas perioda temperatūras pieaugums ir 20 grādi un normālas darbības perioda temperatūras vidējā vērtība ir -5 grādi. 6b.

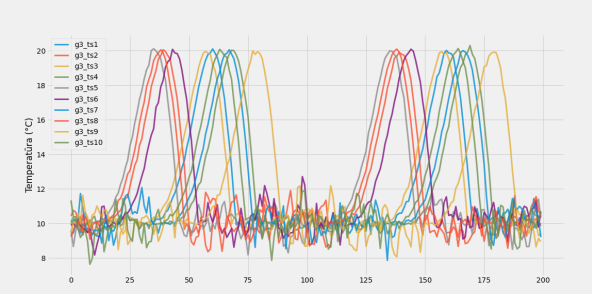
- 3. Grupa: normālas darbības perioda garums ir 60 laika vienības, atkausēšanas perioda garums ir 40 laika vienības, atkausēšanas asimetrijas koeficients n ir 1, atkausēšanas perioda temperatūras pieaugums ir 10 grādi un normālas darbības perioda temperatūras vidējā vērtība ir 10 grādi. 6c.



(a) **1. grupas laikrindu grafiks**



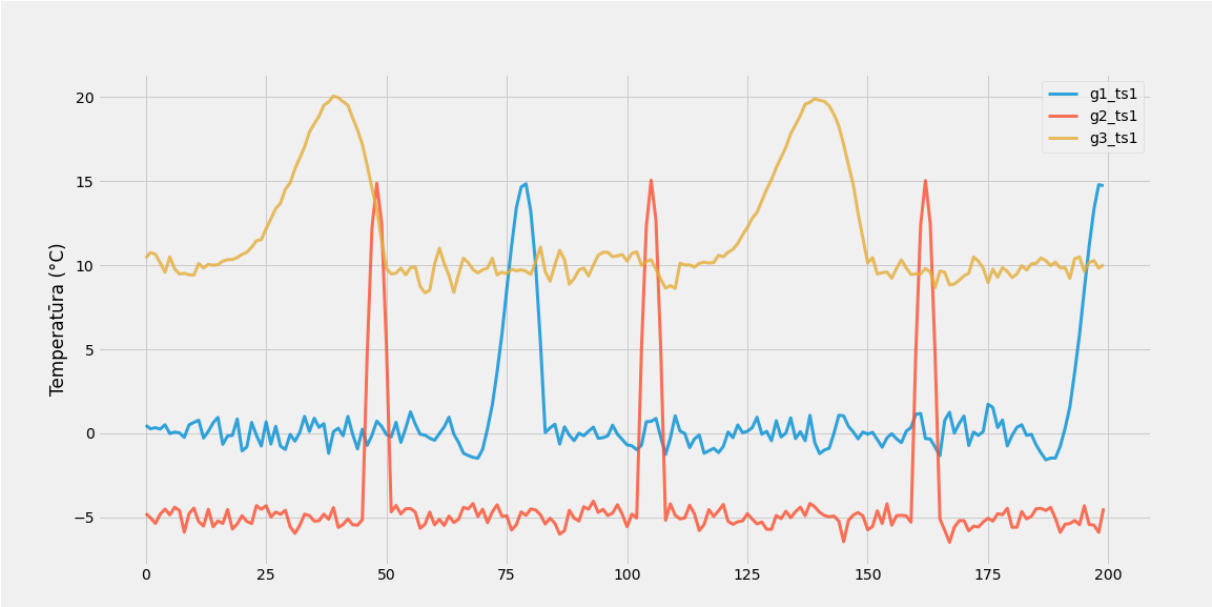
(b) **2. grupas laikrindu grafiks**



(c) **3. grupas laikrindu grafiks**

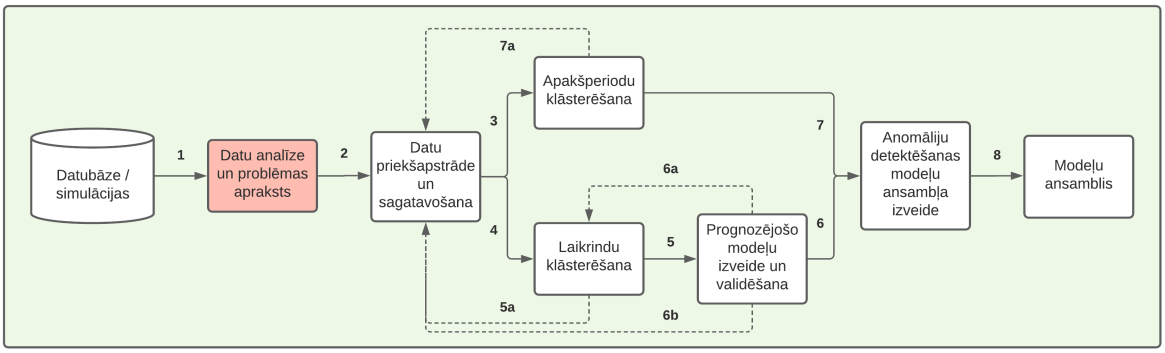
6. att. **Trīs simulēto grupu laikrindu grafiki**

Apskatot visu trīs grupu laikrindu grafikus 6. attēlotajos grafikos, varam secināt, ka grupas ietvaros laikrindas, lai gan ar nobīdītu sezonalitāti, ir līdzīgas. Ir vērts pieminēt, ka simulējot vienas grupas laikrindas tika mainīti arī citi parametri, precīzas parametru vērtības ir atrodamas pielikumā esošajā simulācijas darba grāmatā B.1.. Lai salīdzinātu visu trīs laikrindu grupu laikrindas savā starpā skatīt 7. attēlu, kur ir attēlotas visu trīs grupu laikrindas, pa vienai laikrindai no katras grupas. Kā varam redzēt no 7., laikrindas starp grupām ir atšķirīgas gan pēc normālās darbības vidējās temperatūras, gan pēc sezonas garuma, gan arī pēc normālas darbības un atkausēšanas periodu formas un garuma. Šī atšķirību ir būtiski atcerēties, jo tā būs nozīmīga 6. nodaļā apskatīto metožu izpētē.



7. att. **Visu trīs grupu laikrindu salīdzinājums**

4. LAIKRINDU VISPĀRĒJA ANALĪZE UN IZPĒTE



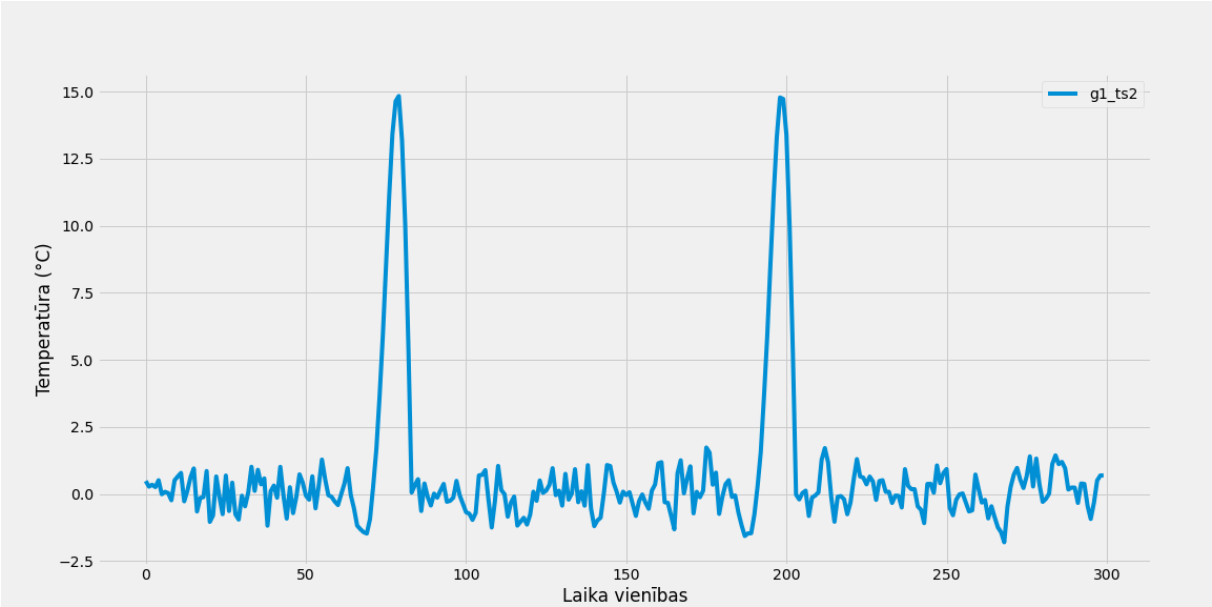
8. att. **Datu analīzes un modeļa izstrādes diagramma, laikrindu analīzes fāze**

Šajā nodaļā tiks apskatītas metodes, kas palīdz iepazīt un saprast apskatāmās laikrindas īpašības(8.), lai atvieglotu turpmāko modeļu izvēli un izveidi. Daļa no šajā nodaļā apskatītajām metodēm ir ļoti būtiski pielietot, lai spētu veiksmīgi izveidot tādus statistiskos modeļus, kā piemēram, ARIMA. Tāpēc turpmākajās nodaļās, pie nepieciešamības, atsauksimies uz šajā nodaļā apskatītajām metodēm. Lai gan galvenā pētāmā laikrinda ir stacionāra ar izteiktu sezonalitāti, nodaļā tiks pieminētas metodes, kas var palīdzēt analizēt nestacionāras laikrindas ar citādāku struktūru. Nodaļā veiktās analīzes darba grāmatu var aplūkot B.2. pielikumā.

Nodaļā izmantotais datu piemērs - Simulētie 1. grupas 2. laikrindas temperatūras mērījumu dati no 3. nodaļas. Kopējais laikrindas novērojumu skaits - 1000.

4.1. Vizualizācija

Viena no pirmajām lietām ar ko būtu jāsāk, uzreiz pēc datu ielādes, ir laikrindas vizualizācija. Laikrindas grafisks attēlojums var palīdzēt saskatīt tās tipiskās iezīmes, īpašības, izlēcējus, datu problēmas un kopējo struktūru.

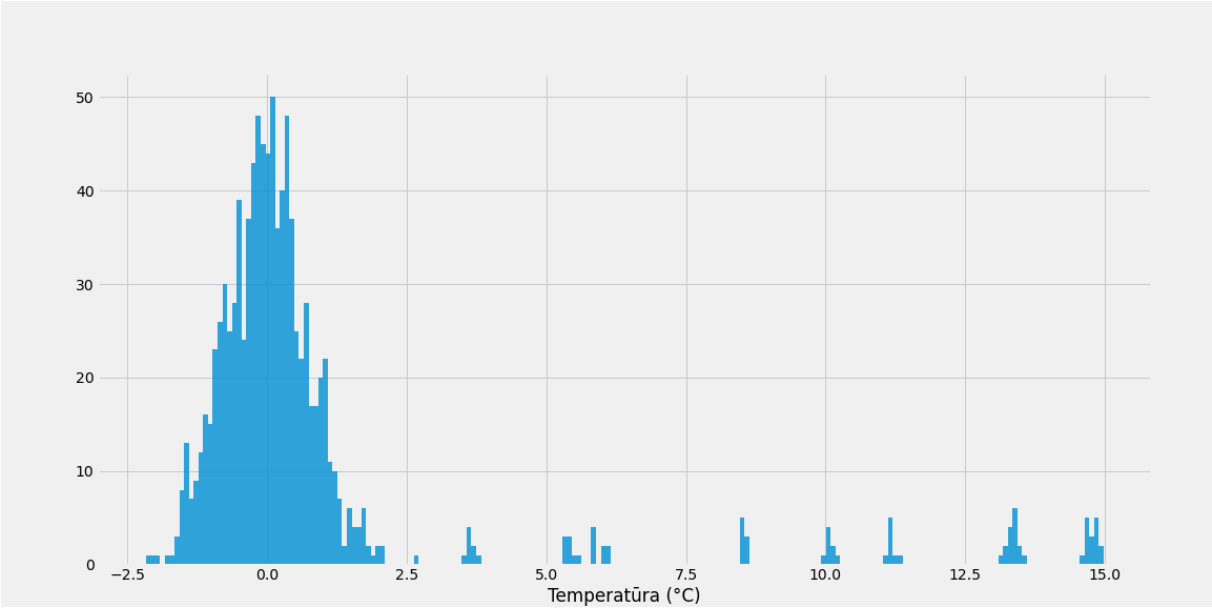


9. att. **1. grupas 2. laikrindas temperatūras pirmo 300 mērījumu attēlojums**

Kā redzams 9. attēlā, laikrinda izskatās stacionāra ar izteiktu sezonalitāti un attēlotajā laika intervālā nav novērojamas neparastas vērtības (izlēcēji). Ņemot vērā, ka ir attēlota laikrinda un veikti sākotnējie secinājumi uz kā pamata ir iespējams veikt pieņēmumus par apskatāmo laikrindu, bet lai apstiprinātu šos pieņēmumus, tiks tuvāk aplūkotas metodes, kas palīdzēt apstiprināt vai apgāzt pieņemot pieņēmumus no grafiku analīzes.

Papildus, gadījumos kad laikrindu dati ir zemākā detalizācijā un ir pamats domāt, ka kādā no zemākām detalizācijām var būt novērojama sezonāla uzvedība, tādā situācijā var palīdzēt apakšsēriju izdalīšana un salīdzināt laikrindu uzvedību starp, piemēram, nedēļas dienām, nedēļām, mēnešiem, ceturkšņiem vai gadiem. Šāda veida sezonāla uzvedība nav novērojama apskatāmajā piemērā, kā arī zemāka datu detalizācija arī nav pieejama, tāpēc dziļāka sezonalitātes analīze netiek veikta.

Lai aplūkotu, kā sadalās laikrindu novērojumu vērtības, tiek vērtības tiek attēlotas histogrammas grafikā (skatīt 10. attēlu). Aplūkojot 10. attēlu varam pamanīt, ka ap nulli vērtības ir sadalītas pēc normālā sadalījuma, kas raksturo normālas darbības periodu un vērtības, kas ir lielākas par 2.5 ir vērtības no atkausēšanas perioda.



10. att. **1. grupas 2. laikrindas temperatūras mērījumu histogramma**

4.2. Dekompozīcija

Jebkura laikrinda var tikt sadalīta sekojošās komponentēs:

- Trends - pieaugošā vai dilstošā laikrindas vērtība, kas iekļauj arī cikliskas svārtības.
- Sezonalitāte - atkārtojošs īslaicīgs cikls.
- Kļūda/baltais troksnis/atlikums - nejaušas gadījuma svārstības.

Cikliska uzvedība netiek atdalīta kā atsevišķa komponente, jo cikliskām izmaiņām nav fiksēts biežums un frekvence. Komponentes iedala divās grupas:

1. Sistemātiskas komponentes - komponentes, kas var tik aprakstītas un modelētas (trends un sezonalitāte).
2. Nesistemātiskas komponentes - komponentes, kas tiešā veidā nevar tikt modelētas (kļūda/atlikums).

Laikrindas komponentēm un to noteikšanai ir būtiska nozīme, lai saprastu laikrindas sarežģītību un palīdzētu izvēlēties modeli, kas spētu vislabāk modelēt šīs komponentes. [7]

4.1. Definīcija. Par laikrindas **dekompozīciju** sauc laikrindas sadalīšanu pa tās komponentēm. Pamatā ir divu veidu dekompozīcijas - **aditīvā dekompozīcija**:

$$y_t = T_t + S_t + R_t,$$

kur T ir trenda kompoenente, S is sezonālā komponente, R ir kļūdas/atlikuma komponente un y ir orģinālā laikrinda.

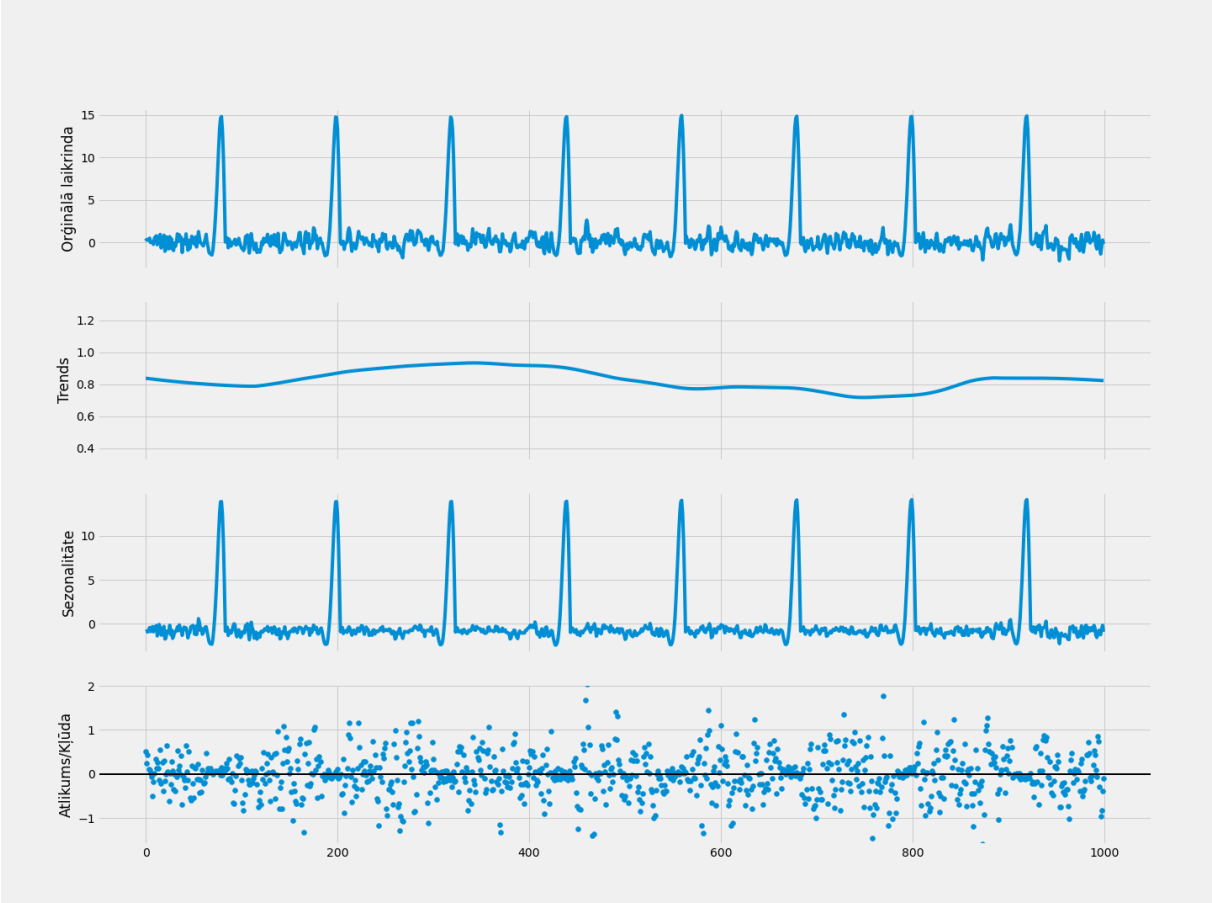
un **multiplikatīva dekompozīcija**:

$$y_t = T_t \cdot S_t \cdot R_t,$$

kur T ir trenda kompoenente, S is sezonālā komponente, R ir kļūdas/atlikuma komponente un y ir orģinālā laikrinda.

Klasiskā dekompozīcijas metode ir sezonālā trenda dekompozīcija izmantojot LOESS (STL), kuras pamatā tiek izmantota LOESS metode.[19]

Kā varam redzēt 11. attēlā, temperatūras mērījumu laikrindai tiek veikta laikrindu dekompozīcija izmantojot aditīvo dekompozīcijas metodi un visas trīs komponentes tiek attēlotas kopā ar orģinālo laikrindu, lai tās varētu salīdzinātu.



11. att. **Temperatūras mērījumu laikrindas aditīvā dekompozīcija**

Temperatūras mērījumu laikrindas dekompozīcijā 11. attēlā, varam novērot, ka laikrindai nav trenda, ir izteiktu sezonālu uzvedību un atlikuma sadalījums izskatās tuvu normālajam sadalījumam. Apskatot dekompozīcijas grafiku varam secināt, ka dekompozīcijas metode labi sadala laikrindu komponentēs, jo ir novērojama izteiktā sezonalitāte un atlikumu vērtības ir cieši izkārtotas ap konstantu vērtību. [19]

4.3. Stationaritāte un tās noteikšana

Stacionārām procesam vidējā vērtība, dispersija un atokorelācija ir nemainīga laikā. Vairums statistisko laikrindu prognozēšanas metožu ir veidotas priekš stacionāru procesu prognozēšanas, tāpēc šis nosacījums ir īpaši svarīgs tieši priekš šīm metodēm. Stacionārus procesus ir vieglāk modelēt un gandrīz jebkuru procesu ir iespējams padarīt stacionāru ar attiecīgām transformācijām. Ir trīs pieejas, kā var novērtēt vai process ir stacionārs:

1. Grafiski - attēlojot procesu grafiski un izvērtējot visus nosacījumus, kas ir nepieciešami, lai process būtu uzskatāms par stacionāru. Šī pieeja nav precīza un skaitliski pamatota, bet ir situācijas, kur pietiek ar šo metodi.

2. Izvēloties divas vai vairāk apakšsērijas un aprēķināt nepieciešamās statistikas un salīdzināt tās. Arī šī metode nav pilnīgi precīza, jo var gadīties, ka izvēlētās apakšsērijas nereprezentē visu procesa uzvedību.

3. Izmantojot statistiskos testus:

    - Paplašināto Dikija-Fullera (ADF) testu
    - Kwiatkowski-Phillips-Schmidt-Shin (KPSS) testu
    - Philips Perron (PP) testu

Praksē visbiežāk tiek lietots tieši ADF tests, tāpēc arī šajā nodaļā izmantosim tieši šo testu, lai pārbaudītu temperatūras mērījumu laikrindas stacionaritāti. ADF testa nulles hipotēze ir ka laikrinda satur vienības sakni un līdz ar to ir nestacionāra.

Izmantojot iepriekš minēto ADF testu, tiek aprēķināta apskatāmās temperatūras mērījumu laikrindas ADF statistika un p-vērtība.

2. tabula

ADF testa rezultāti

	ADF statistika	10%	5%	1%	p-vērtība
Temperatūras mērījumi	-11.52	-2.568	-2.864	-3.437	0

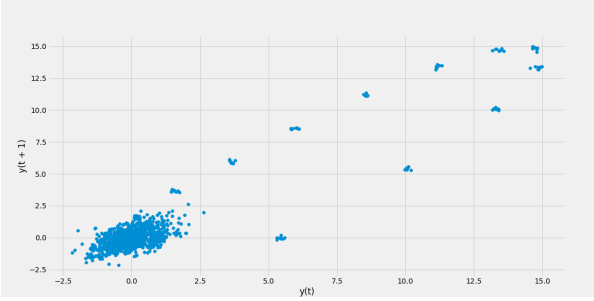
Balstoties uz ADF testa rezultātiem (2.) varam secināt, ka temperatūras mērījumu laikrinda ir stacionāra (p-vērtība ir stingri mazāka par 0.01, līdz ar to nulles hipotēzes, ka laikrinda nav stacionāra, tik noraidīta), kas apstiprina iepriekš izvirzītos pieņēmumus balstoties uz grafisko analīzi.

4.4. Laika nobīdes analīze

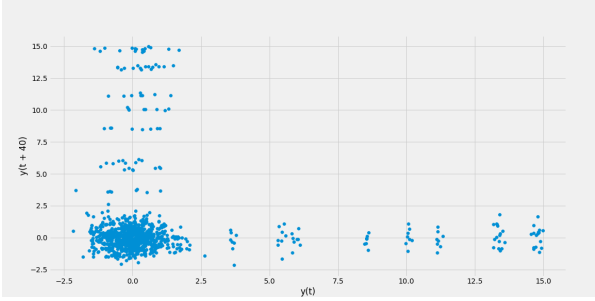
Nobīdes laikā, jeb lagu, grafiks ir grafisks salīdzinājums starp diviem laika momentiem $y(t)$ un $y(t+i)$, kur $i \in \mathbb{N}$. Salīdzinājums starp vērtībām dažādos laika momentos dod iespēju analizēt

sezonālu uzvedību un vērtību salīdzinājums divos dažādos laika momentos ir tiek izmantots, lai noteiktu autokorelāciju.

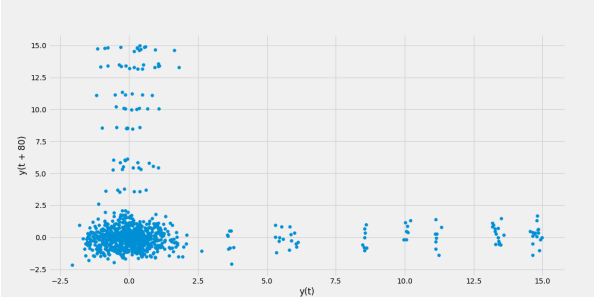
Aplūkojot 12. grafiku varam redzēt temperatūras mērījumu vērtību salīdzinājuma ar četriem dažādiem laika nobīdes periodiem - 1, 40, 80 un 120. Un varam uzreiz ieraudzīt, ka mērījumu vērtības starp -2 un 2 ir izkārtotas tādā kā mākonī. Šo mērījumu izkārtojums, ja neskaita grafiku pie laika nobīdes 1, izskatās kā baltais troksnis un nav redzama korelācija un tieši šīs vērtības ir tās, kas raksturo regulāro darbības periodu. Savukārt ar vērtībām, kas raksturo atkausēšanas periodu ir savādāk, to korelācija manāmi mainās jau sākot no laika nobīdes 1 (12a.). Pie laika nobīdes 40 (12b.) un 80 (12c.) atkausēšanas perioda laika mērījumos nav vērojama korelācija. Bet pie laika nobīdes 120 (12d.) ir vērojama korelācija starp atkausēšanas periodu mērījumiem, kas liecina par to, ka šai laikrindai varētu būt sezonāla uzvedība ar soli 120, ko šajā gadījumā varam apstiprināt, jo atgādināsim, ka šie ir simulēti dati un šī konkrētā laikrinda tika simulēta ar sezonālu uzvedību garumā 120 laika vienības. [7]



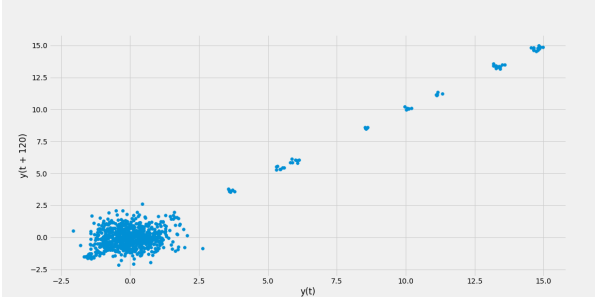
(a) **Temperatūras mērījumu grafiks ar laika nobīdi 1**



(b) **Temperatūras mērījumu grafiks ar laika nobīdi 40**



(c) **Temperatūras mērījumu grafiks ar laika nobīdi 80**



(d) **Temperatūras mērījumu grafiks ar laika nobīdi 120**

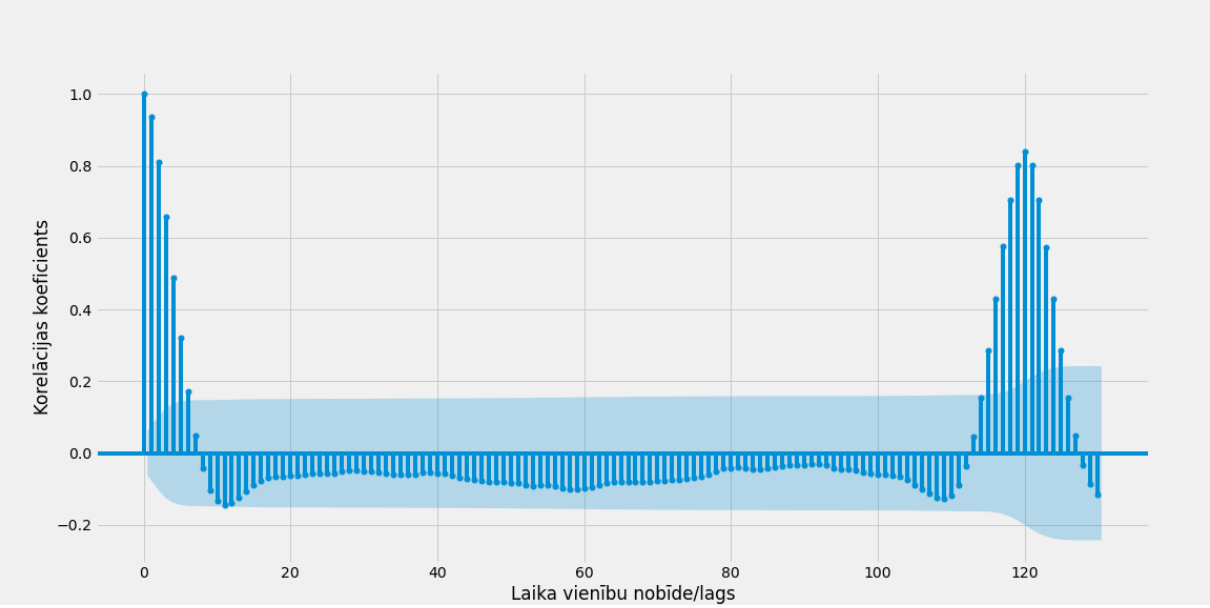
12. att. **Temperatūras mērījumu nobīdes laikā/lagu grafiki**

4.5. ACF un PACF grafiki un to skaidrojums

4.5.1. Autokorelācijas funkcija (ACF)

Autokorelācija ir korelācija starp pašas laikrindas novērojumiem divos dažādos laika momentos, kas tika aprakstīta 4.4. nodaļā. Motivācija autokorelācijas fukcijas (ACF) aprēķināšanai

un tās grafiku attēlošanai ir identificēt tos laika momentus, kuros novēroto vērtību korelācija ir statistiski nozīmīga, kas liecina par to, ka vērtība konkrētā laika momentā $y(t-i)$ var palīdzēt prognozēt pašreizējā laika momenta vērtību $y(t)$. ACF grafikā tiek attēlots korelācijas koeficienti pirmajiem n laika nobīdes salīdzinājumiem, kur $n \in \mathbb{N}$. Temperatūras mērījumu laikrindas piemēra ACF grafiks ir attēlots 13.. Attēlā gaiši zilā josla iezīmē robežu, lai noteiktu kurš laika nobīdes korelācijas koeficients ir statistiski nozīmīgs (vērtība ir ārpus šī intervāla) un kurš tāds nav (vērtība atrodas iekšpusē šai joslai). Kā redzam grafikā, pirmie 7 lagi ir ar statistiski nozīmīgu korelācija un tad nākamie lagi ar statistiski nozīmīgu korelāciju parādās pie laga 115 un ar spēcīgu korelāciju (virs 0.8) pie laga 120, kas nav pārsteigums, jo kā iepriekš secinājām, 120 ir laikrindas sezonas garums, ko apstiprina arī ACF grafiks. Ir vērts piezīmēt, ka parasti šāda veida korelācijas koeficientu lēcieni pēc fiksēta laika vienību skaita liecina par sezonalitāti. Lai gan laikrinda ir konstruēta tā, ka normālas darbības periodā ir ARMA(1,1) process, korelācija laika nobīdēs 2, 3, 4 un 5 ir statistiski nozīmīgas dēļ atkausēšanas perioda simulētajām mērījumu vērtībām. Ja ir nepieciešamība saprast katras laika nobīdes korelācijas struktūru var atgriezties 4.4. un uzzīmēt katras interesējošās laika nobīdes grafiku un analizēt to atsevišķi. ACF grafiks kalpo kā labs rīks MA (slīdošā vidējā) modeļa parametra q novērtēšanai. [17]

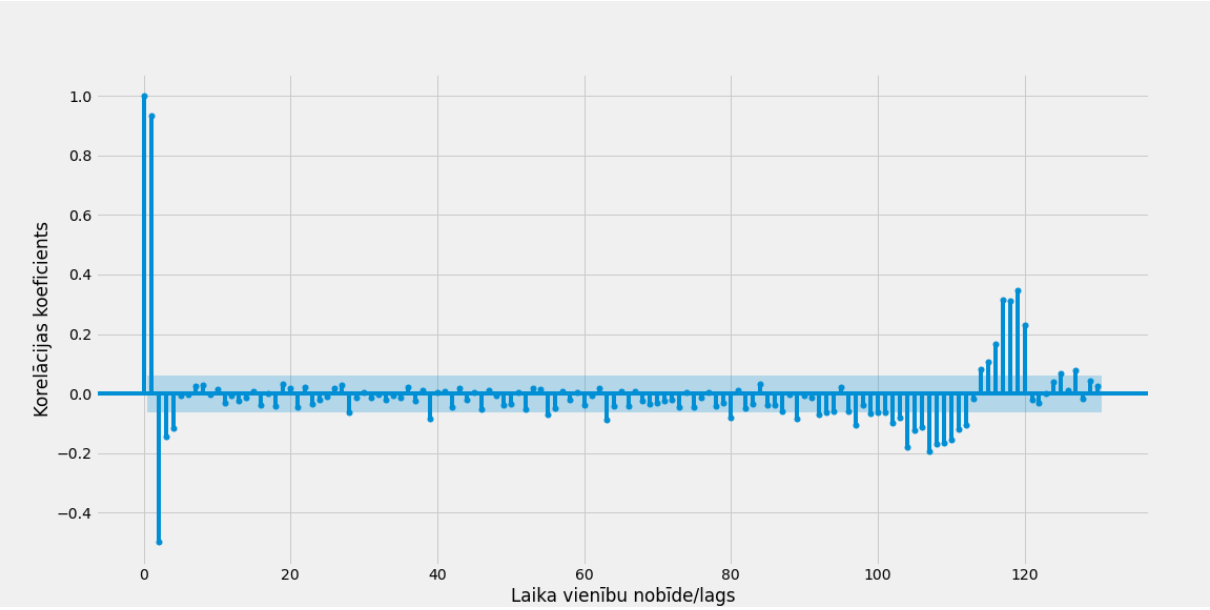


13. att. **Temperatūras mērījumu laikrindas ACF grafiks**

4.5.2. Parciālā autokorelācijas funcijas (PACF)

Parciālā autokorelācija funkcija, tāpat kā ACF, rēķina korelāciju starp tās pašas laikrindas novērojumiem divos dažādos laika momentos, vienīgi būtiskākā atšķirība ar ACF ir tāda, ka PACF aprēķinātajā korelācijas koeficientā netiek ņemta citu laika nobīdes $y(t+j)$, kur $j \in \mathbb{N}$ un $j < i$, ietekme uz korelācijas koeficientu starp vērtībām laika momentā $y(t)$ un $y(t+i)$. Tādējādi tiek iegūta tieša korelācija tikai starp attiecīgajiem diviem laika nobīdes novērojumiem.

Grafikā 14. var aplūkot temperatūras mērījumu laikrindas PACF grafiku, kur, līdzīgi kā ACF grafikā, ir attēlots konkrētā laika vienības nobīdes korelācijas koeficients un to vai tas ir statistiski nozīmīgs vai nav, nosaka tas vai punkts atrodas ārpus zilās joslas (ir statistiski nozīmīgs) vai iekšpus zilās joslas (nav statistiski nozīmīgs). Kā redzam grafikā pirmie 4 lagi ir statistiski nozīmīgi, no kuriem 2., 3. un 4. lags ir negatīvi. Pēctam statistiski nozīmīgi lagi parādās pie 102 laga. Ja ir nepieciešams var veikt tuvāku analīzi, lai saprastu vai attiecīgie korelācijas koeficienti ir starp attiecīgajiem laika momentiem ir nozīmīgi un tos ir nepieciešams iekļaut modelī vai arī ne, bet tā kā mums ir zināms, ka šī laikrinda daļēji ir ARMA(1,1) process un daļēji satur nobīdītu sinusoīdu, tad tas izskaidro stipro korelāciju pie laika vienību nobīdes 1 un negatīvos korelācijas koeficientus. PACF grafiks kalpo kā labs rīks AR (autoregresīvā) modeļa parametra p novērtēšanai. [17]



14. att. **Temperatūras mērījumu laikrindas PACF grafiks**

4.6. Prognozējamības sarežģītības novērtēšana

Šajā nodaļā tiek apskatīta lainkrindas struktūras sarežģītības novērtēšanas metode - modeļa entropijas (*sample entropy*, SampEn) novērtējums. SampEn novērtējums pieņem vērtības tuvu nullei, ja laikrinda ir vienkārša un bez baltā trokšņa, piemēram, sinus funkcija, savukārt baltā trokšņa gadījumā novērtējums pieņem vērtību lielāku par 2.

Tabulā 3. ir attēlots temperatūras mērījumu laikrindas sarežģītība snovērtējums. No rezultātiem varam spriest, ka laikrindai ir salīdzinoši vienkārša struktūra, bet mērķis šim novērtējumam ir izmantot to pie laikrindu gludināšanas, lai pārbaudītu cik lielu pienesumu dod attiecīgā gludināšanas metode.

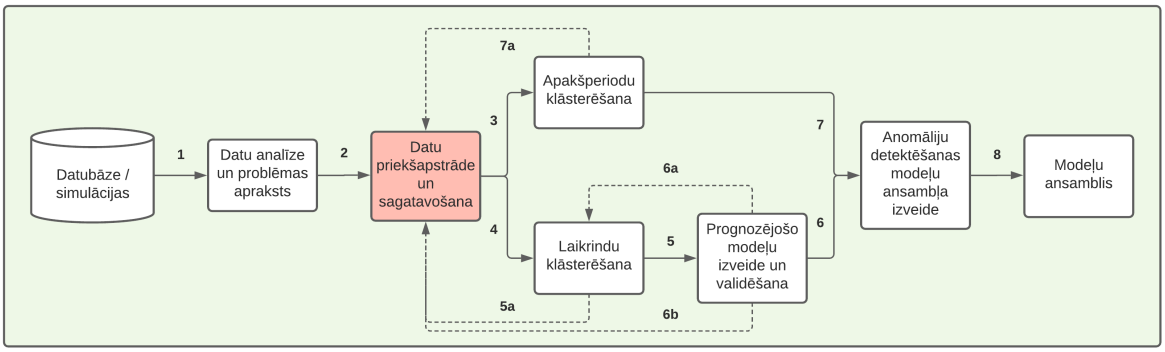
3. tabula

SampEn novērtējuma rezultāti

	SampEn
Temperatūras mērījumi	0.681

SampEn metodes novērtējumu var ņemt vērā veicot prognozējošo algoritmu izvēli, jo vienkāršāka ir laikrindas struktūra (SampEn novērtējums ir mazāks), jo potenciāli vienkāršākas metodes var izmantot, lai modelētu attiecīgo laikrindu.

5. DATU SAGATAVOŠANA UN PRIEKŠAPSTRĀDE



15. att. **Datu analīzes un modeļa izstrādes diagramma, laikrindu datu sagatavošanas fāze**

Datu sagatavošanas nodaļā tiks apskatītas datu sagatavošanas un priekšapstrādes metodes (15.), izmantojot arī 4. nodaļā iegūto informāciju, lai sagatavotu datus modeļu veidošanai nākamajās nodaļās. Šis solis ir nozīmīgs, jo ir modeļi vai metodes, priekš kurām datus ir nepieciešams apstrādāt un sagatavot konkrētā formā, kā piemēram, klāsterēšanā ir jāizmanto mērogoti dati un ARMA modeļus var izmantot tikai stacionāriem datiem. Katrā no apakšnodaļām tiks uzskaitītas populārākās metodes un tās metodes, kuras parādīja labākos rezultātus tieši uz apskatāmā piemēra, tiks apskatītas sīkāk. Datu sagatavošanas un priekšapstrādes darba grāmatu var aplūkot B.3. pielikumā.

5.1. Trūkstošo datu aizpildīšana/imputācija

Mēdz gadīties kad datos iztrūkst kādas vērtības, ja tās nav sistemātiskas kļūdas, tad šo datu trūkumu var labot sekojošos veidot:

1. Neņemt vērā attiecīgo periodu. Modeļiem, priekš kuriem treniņa kopa sastāv no vairākiem novērojumiem ar n garu vektoru ar neatkarīgajiem lielumiem, šis būtu pieņemams risinājums, bet priekš statistiskajiem modeļiem, kuriem nepieciešama nepārtraukta novērojumu virkne, šīs risinājums varētu nebūt piemērotākais, jo būtu nepieciešams atstāt pēdējos novērojumus līdz pirmajai iztrūkstošajai vērtībai.

2. Izmantot kādu no modeļiem - gadījuma meži, ARMA vai kNN, lai prognozētu trūkstošās vērtības ar esošajām vērtībām.

3. Veiktu imputāciju izmantojot sekojošas metodes ([17]):

   (a) Aizpildot ar pēdējo pieejamo vērtību.

   $$\hat{y}(t) = y(t-n),$$

   kur $n = \min\{x \in \mathbb{N} \mid y(t-x) \in \mathbb{R}\}$

(b) Aizpildot ar nākamo pieejamo vērtību.

$$\hat{y}(t) = y(t+m),$$

kur $m = \min\{x \in \mathbb{N} \mid y(t+x) \in \mathbb{R}\}$

(c) Lineārā interpolācija.

$$\hat{y}(t) = y(t-n) + (t-n)\frac{y(t+m) - y(t-n)}{(t+m) - (t-n)},$$

kur $n = \min\{x \in \mathbb{N} \mid y(t-x) \in \mathbb{R}\}$ un $m = \min\{x \in \mathbb{N} \mid y(t+x) \in \mathbb{R}\}$.

(d) Kubiskā interpolācija.

(e) K tuvāko kaimiņu vidējā vērtība.

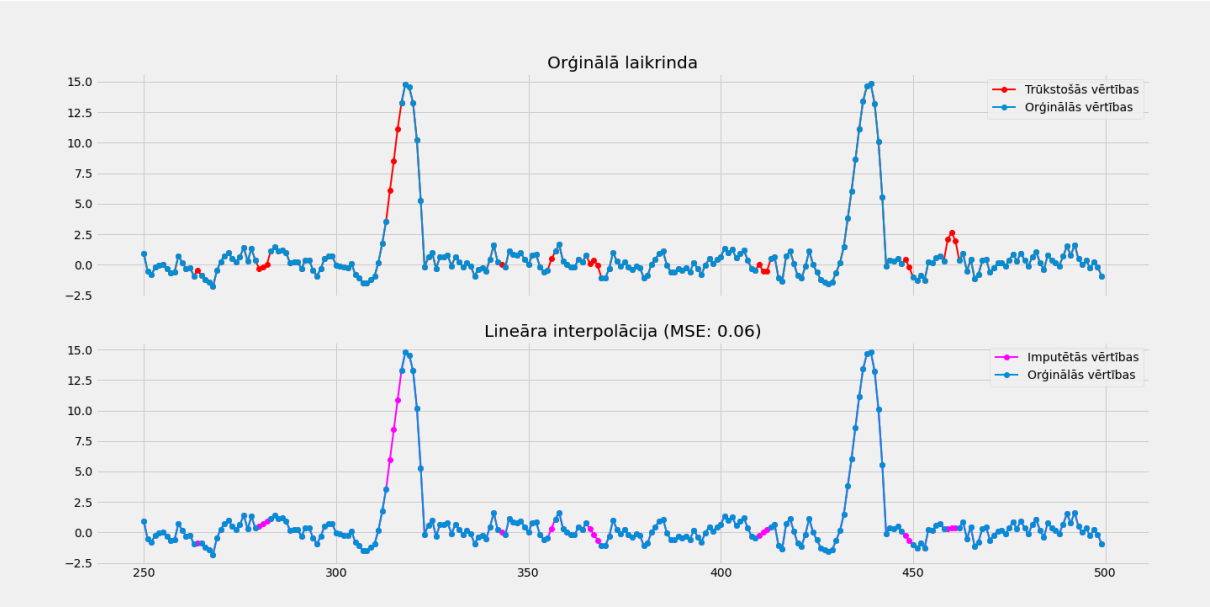
(f) Sezonālā vidējā vērtība.

4. tabula

Dažādu imputācijas metožu salīdzinājums izmantojot MSE

	MSE
Lineāra interpolācija	0.06
Kubiska interpolācija	0.09
KNN vidējā vērtība	0.11
Sezonālā vidējā vērtība	0.12
Aizpilda ar pēdējo pieejamo vērtību	0.41
Aizpilda ar nākamo pieejamo vērtību	0.42

Izvēloties metodi ir nepieciešams ņemt vērā datu struktūru, cik daudz ir iztrūkstošo vērtību un kāds ir šo iztrūkstošo vērtību periodu garums. Šī darba ietvaros tiek aplūkotas visas imputācijas metodes, to MSE novērtējums ir attēlots 4. tabulā, kā arī visu metožu salīdzinājums ir attēlots A.1. attēlā. Kā varam redzēt pēc rezultāta un grafikiem, vislabāk trūkstošos datus, apskatītajam piemēram, restaurē lineārās interpolācijas metode. Grafikā 16. var aplūkot orģinālās laikrindas datus (augšējais grafiks), kuros ar sarkanu iezīmēti tie datu punkti, kas tiks restaurēti izmantojot lineāro interpolācijas metodi un laikrindas datus (apakšējais grafiks) ar restaurētām trūkstošajām vērtībām (iekrāsotām purpursarkanā krāsā) izmantojot lineārās interpolācijas metodi.



16. att. **Lineārās imputācijas grafisks attēlojums**

Šajā nodaļā uzskaitītās metodes ir vienas no populārākajām, bet šīs nav vienīgās metodes. Veicot datu analīzi ir nepieciešams saprast un novērtēt kura metode vislabāk spēj atjaunot trūkstošos datus. Iespējams dažādās situācijās ir nepieciešams izmantot dažādas metodes, piemēram, gadījumos ja trūkstošo datu perioda garums ir līdz 3 laika vienībām, tad izmanto lineāro interpolāciju, bet, kad iztrūkstošo datu perioda garums ir lielāks par 3, tiek izmantot kubisko interpolācijas metodi.

5.2. Transformācijas

Laikrindu nepieciešams transformēt, lai padarītu to stacionāru, likvidētu trenda, atbrīvotos no sezonalitātes vai stabilizētu dispersiju. Gandrīz jebkuru laikrindu ir iespējams padarīt stacionāru izmantojot attiecīgas transformācijas [17]. Ir vairākas transformācijas metodes, bet populārākās ir šādas metodes:

- Diferencēšana - palīdz padarīt laikrindu stacionāru un atbrīvoties no trenda. Izvēloties diferencēšanas perioda garumu vienādu ar laikrindas sezonas perioda garumu var atbrīvoties no sezonalitātes, to arī sauc par sezonālo diferencēšanu. Laikrindu var diferencēt vairākas reizes, par otrās kārtas diferencēšanu sauc laikrindas diferencēšanu divas reizes. (definīciju skatīt 2. nodaļā.)

- Box-Cox transformācija, kas, atkarībā no izvēlētās λ vērtības, var būt gan kā logaritma transformācija, gan arī kā pakāpes transformāciju.

$$w_t = \begin{cases} \log(y_t) & \text{ja } \lambda = 0; \\ (y_t^{\lambda} - 1)/\lambda & \text{citādi.} \end{cases}$$

- Atņemt trenda komponenti, kas iegūta izmantojot laikrindas dekompozīciju (4.).
- Atņemt laikrindas vidējo vērtību.

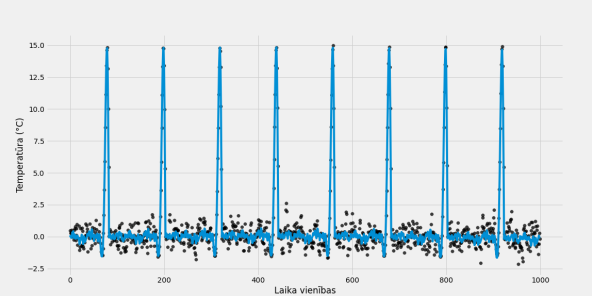
Nepieciešamības gadījumā var kombinēt vairākas transformācijas. Darbā analizētā laikrinda, kā tika noskaidrots 4. nodaļā, izmantojot ADF testu, ir stacionāra, tāpēc nav nepieciešamas transformācijas.

5.3. Laikrindu gludināšanas metodes

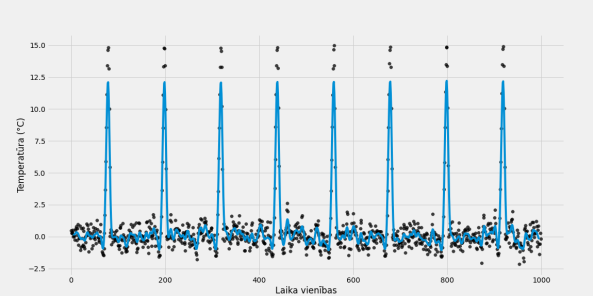
Arī laikrindas gludināšanas metodes ir daudz un dažādas, šajā nodaļā tiks apskatītas pāris populārākās. Laikrindas nepieciešams gludināt, lai samazina baltā trokšņa ietekmi, modelētu vai attēlot un saprast laikrindas struktūru. Populārākās gludināšanas metodes ir:

- Dekompozīcija gludināšana - izmanto dekompozīcijas metodi (4.) un tiek attēlotas sezonālā un trenda kompoenentes 17a..
- Slīdošās vidējās vērtības gludināšana 17b..
- Lokālās svērtās regresija gludināšana 17c..
- Eksponenciālā gludināšana 17d..

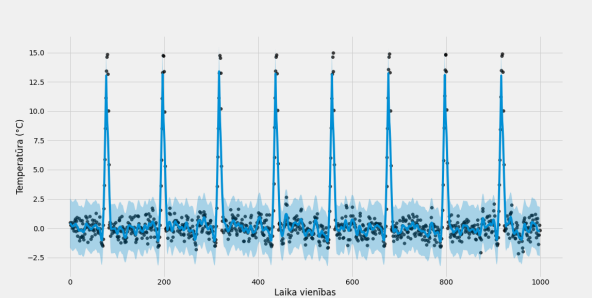
Darba ietvaros gludināšanai ir divas nozīmīgas lomas, datu sagatavošanu laikrindu klāsterēšanai un laikrindas gludināšanai priekš prognozējošo modeļu treniņu datu kopas izveides. Tāpēc vispirms tiek apskatītas visas četras metodes, lai varētu novērtēt, kura no tām ir piemērotākā temperatūras mērījumu laikrindas gludināšanai. Izmantojot visas četra iepriekš minētās metodes tiek nogludināta temperatūras mērījumu laikrinda un visu četru gludināšanas metožu grafiki ir attēloti 17. grafikā. Ar zilo krāsu tiek iezīmēta nogludinātā laikrinda, savukārt ar melnajiem punktiem tiek attēloti orģinālie novērojumi. Lokālās svērtās regresijas un eksponenciāli nogludinošās metodēm papildus tiek aprēķinātas un attēlotas ticamības joslas (caurredzami zilā krāsā).



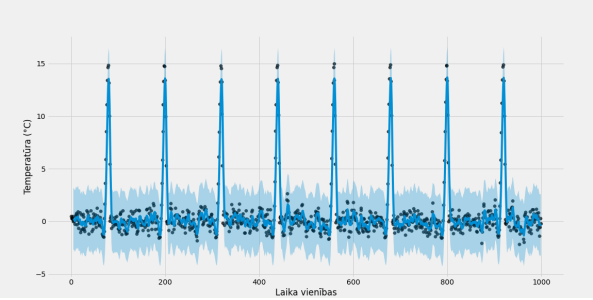
(a) **Dekompozīcijas gludināšanas grafiks**



(b) **Slīdošās vidējās vērtības gludināšanas grafiks**



(c) **Lokālās svērtās regresija gludināšanas grafiks**



(d) **Eksponenciālās gludināšanas grafiks**

17. att. **Laikrindu gludinošo metožu grafiks attēlojums**

Kā varam redzēt grafikos visas metodes laikrindu nogludina salīdzinoši labi, bet, lai novērtētu un salīdzinātu visas četras laikrindu gludināšanas metodes, tiek izmantota SampEn metode no 4. nodaļas. Tabulā 5. ir attēloti visu četru metožu SampEn novērtējumi. No iegūtajiem rezultātiem varam secināt, ka visas metodes ir vienkāršojušas laikrindas struktūru, tajā pašā laika saglabājot laikrindas tipisko uzvedību un to parāda SampEn novērtējuma samazinājums no oriģinālā temperatūras mērījumu laikrindas novērtējuma - 0.681 3.. Vislabākie rezultāti gan pēc grafiskā novērtējuma, gan arī pēc SampEn laikrindas sarežģītības novērtējuma ir dekompozīcijas gludināšanas metodei. Grafiski analizējot visas gludinātās laikrindas (17.) varam novērtot, ka dekompozīcijas gludināšanas metode acīmredzami vislabāk raksturo atkausēšanas periodu pīķus, kā arī regulārā perioda trokšņainā uzvedība ir nogludināta, tāpēc arī zemāks SampEn novērtējums. Citas struktūras laikrindām rezultāti var būt citādāki, bet ņemot vērā, ka visas metodes salīdzinoši labi veica savu darbu, metodes izvēle nav kritiska tālāko modeļu veidošanā.

5. tabula

Laikrindu gludināšanas metožu salīdzinājums izmantojot SampEn novērtējumu

	SampEn novērtējums
Dekompozīcija gludināšana	0.09
Slīdošās vidējās vērtības gludināšana	0.13
Lokālās svērtās regresija gludināšana	0.14
Eksponenciālā gludināšana	0.21

5.4. Mērogošana

Mērogošana ir laikrindu novērojumu transformācija izvēlētā skalā. Mērogošanas mērķis ir saglabājot laikrindas struktūru un izmantojot izvēlētu skalu novērtēt un transformēt orģinālos novērojumus, kas pēc tam ļauj salīdzināt laikrindas ar, piemēram, dažādām vidējām vērtībām un līdzīgu uzvedību/struktūru. Mērogošana ir īpaši būtiska un nepieciešama datu sagatavošanas procesā priekš klāsterēšanas, jo klāsterējot laikrindas mums neinteresē absolūtās vērtības, mūs interesē salīdzināt laikrindas uzvedība/struktūra. Lai gan mērogošana parasti tiek izmantota priekš klāsterēšanas, tā var tikt izmantot arī dziļās apmācības datu sagatavošanai, kas gan nav obligāta prasība, bet var palīdzēt atvieglot skaitliskos aprēķinus un līdz ar to samazināt modeļa trenēšanas/apmācības laiku. Šajā nodaļā aplūkotās mērogošanas metodes būs populārākās un efektīvākās metodes, kas tiek izmantotas praksē, kā arī dažas šo metožu modifikācijas.

Izvērtējot dažādas mērogošanas metodes priekš konkrētas problēmas ir nepieciešams veikt analīzi un noskaidrot, kā konkrētā metode transformē divas vai vairāk atšķirīgas laikrindas no visu laikrindu kopas, kas tiks izmantota klāsterēšanas proceša un pēctam izvērtēt kura no metodēm labāk transformē laikrindu, tajā pašā laikā saglabājot būtiskākās laikrindu īpašību un/vai strukturālās atšķirības. Kā lielākajai daļu metožu, dažāda veida laikrundu datiem var būt piemērotāka dažādākas mērogošanas metodes, tāpēc būtiski ir izvērtēt un izvēlēties piemērotāko metodi, kas palīdz atrisināt attiecīgo problēmu.

Priekš mērogošanas metodēm, kas tiks aplūkotas šajā darbā, nepieciešam zināt sekojošu vērtību aprēķinus[3]:

- Lielākā vērtība. $X_{max} = max\{x : x \in X\}$
- Mazākā vērtība. $X_{min} = min\{x : x \in X\}$
- Vidējā vērtība. $X_{\mu} = \frac{\sum x_i}{n}$, kur $x \in X$ un n ir kopējais X novērojumu skaits.
- Standartnovirze. $X_{\sigma} = \sqrt{\frac{1}{N-1}\sum_{i=1}^{N}(x_i - \overline{x})^2}$
- Vidējā absolūtā kļūda.

$$X_{MAD} = \frac{\sum |x_i - \bar{x}|}{n},$$

kur $x \in X$, x_i ir i-tais novērojums un n ir kopējais novērojumu skaits kopā X.

Temperatūras mērījumu datus mērogosim izmantojot sekojošas metodes:

1. Mazākās un lielākās vērtības mērogošana. (*Min max scaling*)

$$X_{sc} = \frac{X - X_{min}}{X_{max} - X_{min}},$$

kur X_{max} ir laikrindas lielākā vērtība un X_{min} ir mazākā vērtība.

2. Standartizēšanu.

$$X_{sc} = \frac{X - X_{\mu}}{X_{\sigma}},$$

kur μ ir laikrindas vidējā vērtība un σ ir laikrindas dispersija.

3. Robustā mērogošana.

$$X_{sc} = \frac{X - X_{median}}{X_{Q3} - X_{Q1}},$$

kur X_{median} ir laikrindas mediāna, jeb otrā kvantile, X_{Q3} un X_{Q1} ir laikrindas pirmās un trešās kvantiles vērtības.

4. Kvantiļu transformācija. Šī metode transformē lainkrindas vērtības, lai tās būtu sadalītas pēc normālā sadalījuma.

5. Mērogošana izmantojot mediānu un MAD.

$$X_{sc} = \frac{X - X_{median}}{X_{MAD}},$$

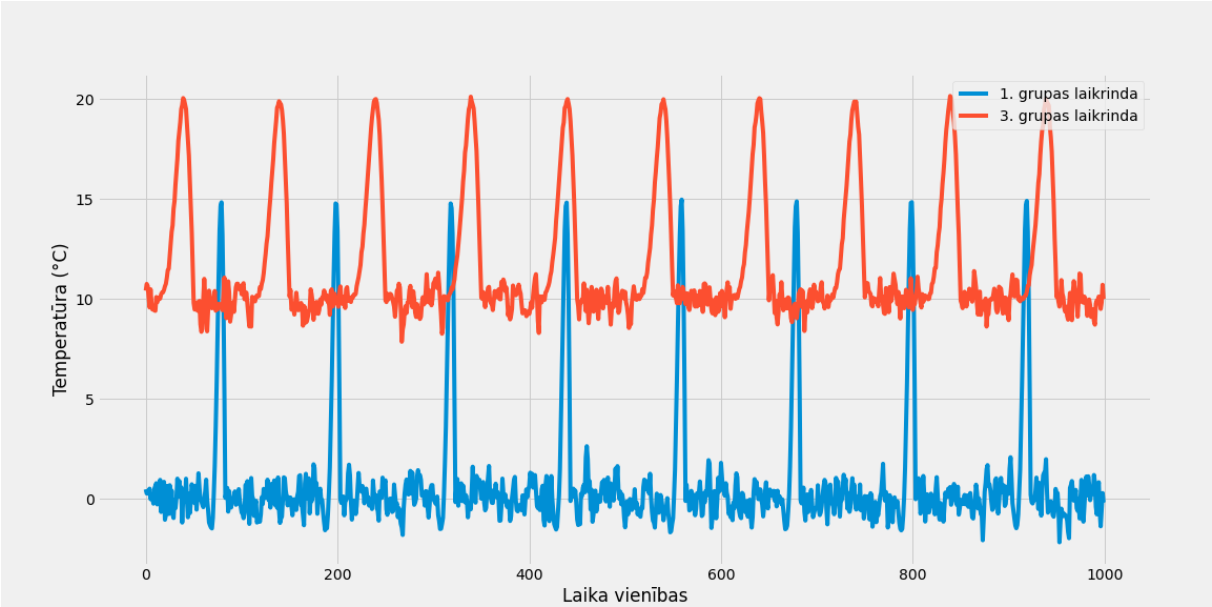
kur X_{median} ir laikrindas novērojumu mediāna un X_{MAX} ir laikrindas vidējā absolūtā kļūda.

6. Mērogošana izmantojot mediānu un standartnovirzi.

$$X_{sc} = \frac{X - X_{median}}{X_{\sigma}},$$

kur X_{median} ir laikrindas novērojumu mediāna un X_{σ} ir laikrindas novērojumu standartnovirze.

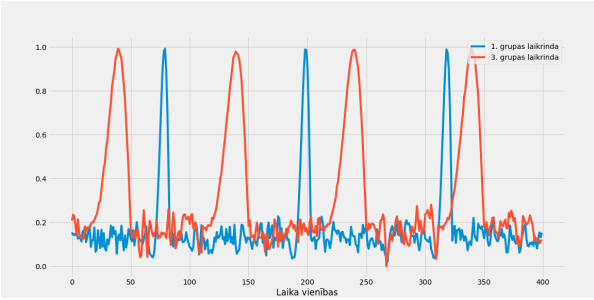
Pirms salīdzinām mērogošanas metodes ir nepieciešams izvēlēties vairāk kā vienu laikrindu, tāpēc, kā piemērs, tiek izvēlēts salīdzināt pirmās un trešās grupas temperatūras mērījumu laikrindas. Kā redzams grafikā 18. laikrindām atšķirās pamata temparatūra un tām ir atšķirīgs atsaldēšanas perioda garums un temperatūras amplitūda.



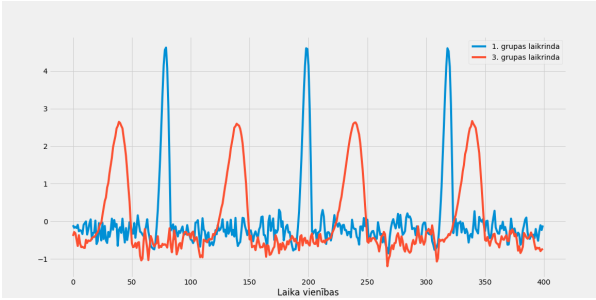
18. att. **1. un 3. grupas laikrindu grafiks**

Veicot mērogošanu ar visām sešām iepriekš minētajām metodēm tiek iegūtas mērogotas laikrindas, kuras var apskatīt 19. attēlā. Aplūkojot katru grafiku tuvāk varam secināt

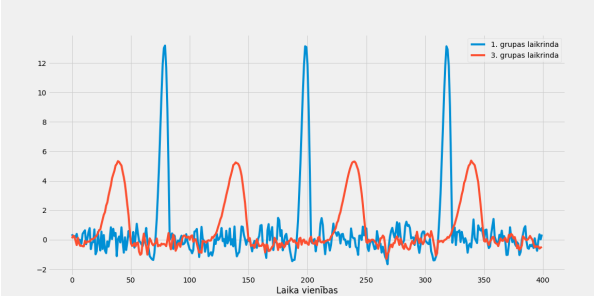
- Mazākās un lielākās vērtības mērogošana pazaudē atkausēšanās periodu amplitūdas atšķirības, bet saglabā abu laikrindu formu. 19a.
- Mērogojot izmantojot standartizāciju tiek saglabātas abu laikrindu īpašības, tajā skaitā atkausēšanas periodu temperatūras izmaiņu amplitūda. Vienīgā lieta, kas ir manāma ir regulārā perioda vidējās vērtības atšķirība, ko rada atkausēšanās perioda amplitūdas atšķirība. 19b.
- Robustā mērogošana pārāk izmaina 3. grupas laikrindas atkausēšanas perioda formu un atšķirība starp abu mērījumu atkausēšanas periodiem ir nesamērīgi liela. 19c.
- Kvantiļu mērogošana pavisam izmaina abu laikrindu strukrūru un regulārā perioda novērojumi vairs nav tik labi nošķirami no atkausēšanas perioda novērojumiem. 19d.
- Mērogošanā izmantojot meidānu un MAD situācija ir līdzīgi kā robustajā mērogošanā. 19e.
- Mērogojot izmantojot mediānu un standartnovirzi tiek iegūti līdzīgi rezultāti kā izmantojot standartizēšanu, vienīgā būtiskā atšķirība ir ka netiek izmantota vidējā vērtība, bet gan mediāna, līdz ar to tiek atrisināta regulārā perioda vidējās vērtības nobīde, kas bija manāma standartizēšanas salīdzinājuma grafikā. 19f.



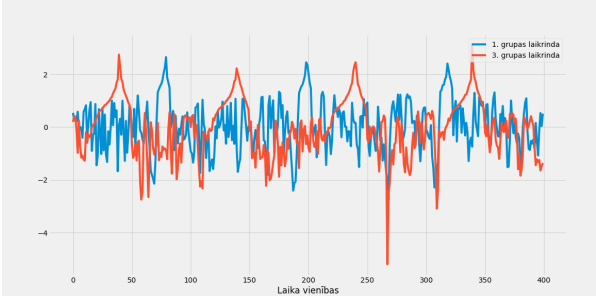
(a) **Mazākās un lielākās vērtības mērogošana**



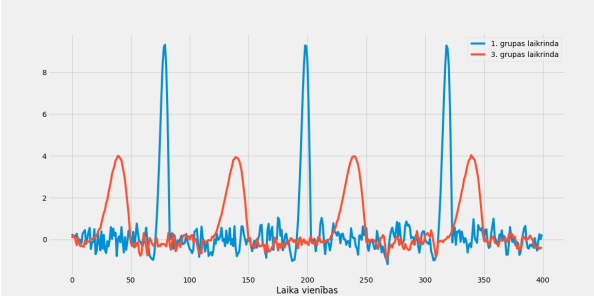
(b) **Mērogošana izmantojot standartizēšanu**



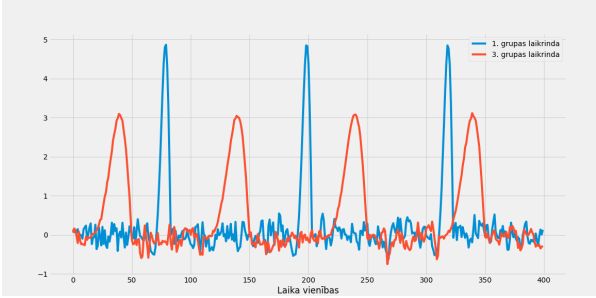
(c) **Robustā mērogošana**



(d) **Kvantiļu mērogošana**



(e) **Mērogošana izmantojot mediānu un MAD**

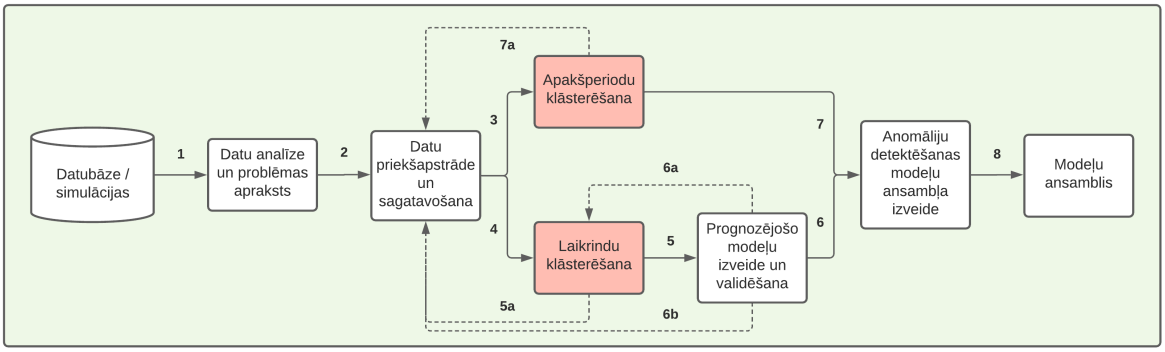


(f) **Mērogošana izmantojot mediānu un standartnovirzi**

19. att. **Laikrindu mērogošanas metožu salīdzinājuma grafiki**

Pēc iepriekš veiktās grafiku analīzes varam secināt, ka vislabākā izvēle temperatūras mērījumu mērogošanai ir standartizēšanas metode vai mērogošana izmantojot mediānu un standartnovirzi. Tā kā **mērogošana ar mediānu un standartnovirzi** dod nedaudz vēlamāku transformāciju, tad tieši ši metode tiks izmantota, lai mērogotu laikrindas 6. un 7. nodaļās.

6. KLĀSTERĒŠANA

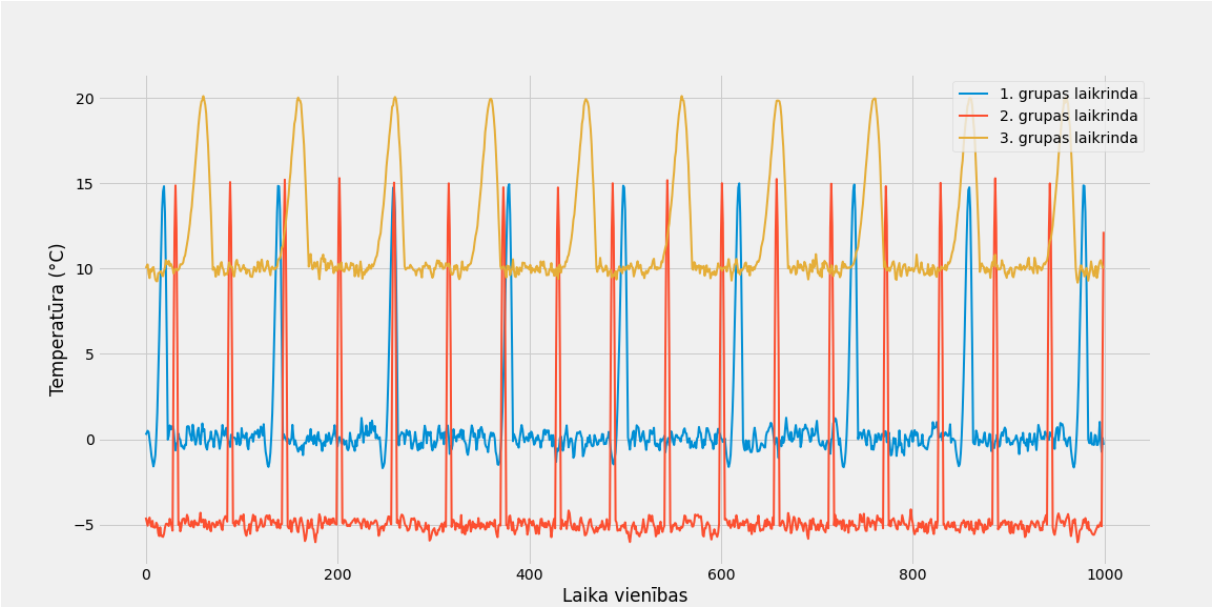


20. att. **Datu analīzes un modeļa izstrādes diagramma, laikrindu klasterēšanas fāzes**

Šajā nodaļā tiks apskatītas dažādas klasterēšanas metodes un atšķirīgas pieejas, kā izmantot klasterēšanu, lai analizētu un grupētu laikrindas, kā arī detektētu anomālijas. Kā redzams 20. attēlā, klasterēšanas tiek veikt pēc vispārējas datu analīzes un datu sagatavošanas un priekšapstrādes posmiem un nodaļas ietvaros tiks apskatīti divas klasterēšanas pieejas - pilna laikrindas klasterēšana un laikrindu apakšperiodu klasterēšana. Veicot klasterēšanas procesu var nākties secināt, ka dati nav pietiekami veiksmīgi sagatavoti, piemēram, nav izvēlēta piemērotākā mērogošanas metode, tāpēc izstrādes diagrammā ir attēlotas bultas *7a* un *5a*, kas norāda, ka nepieciešamības gadījumā var nākties atgriezties iepriekšējā izstrādes posmā. Klasterēšanas darba grāmatu var aplūkot B.4. pielikumā.

Nodaļa tiks izmantota temperatūras mērījumu laikrindas, kuras tika simulētas 3. nodaļā. Kā tika secināts 4. nodaļā, dati ir stacionāri ar sezonālu uzvedību. No 5. nodaļas rezultātiem un secinājumiem šajā nodaļā tiks izmantotas piemērotākās laikrindu gludināšanas (*dekompozīcijas gludināšanas metode*) un mērogošanas (*mērogošana izmantojot MAD un standartnovirzi*) metodes. Temperatūras mērījumu laikrindas nav nepieciešams transformēt vai tajās aizvietot trūkstošās vērtības, jo laikrindas ir stacionāras ar izteiktu sezonālu uzvedību un dēļ tā, ka dati tika simulēti varam būt pārliecināti, ka šajos datos nav iztrūkstošu vērtību. Ir vērts pieminēt, ka šī darba ietvaros netiek apskatītas metodes optimālā klasteru skaita noteikšanai.

Šajā nodaļā tiks apskatīti 3 dažādu grupu temperatūras mērījumu laikrindas, kas tika simulētas 3. nodaļā. Attēlā 21. varam aplūkot trīs temperatūras mērījumu laikrindu salīdzinājumu, kas reprezentē visas trīs simulētās temperatūras mērījumu grupas. Grafiku ar vienas grupas laikrindu salīdzinājumu skatīt 52. pielikumā. No salīdzinājumiem starp visām trim grupām un laikrindām grupas ietvaros varam secināt, ka laikrindām ir tipiska uzvedība - atkausēšanas perioda garums un temperatūras izmaiņu svārstība, regulāras darbības vidējā temperatūra un garums. Lai gan vienas grupas ietvaros visām laikrindām vidējā regulārā perioda temperatūra ir vienāda, mērogojot laikrindas mēs atbrīvojamies no absolūtajām vērtībām saglabājot laikrindas struktūru, tādējādi salīdzinot laikrindas tiek izslēgta atšķirīgu vidējo vērtību ietekme.



21. att. **Visu trīs simulēto temperatūras mērījumu grupu laikrindu salīdzinājums**

6.1. Laikrindu klasterēšana

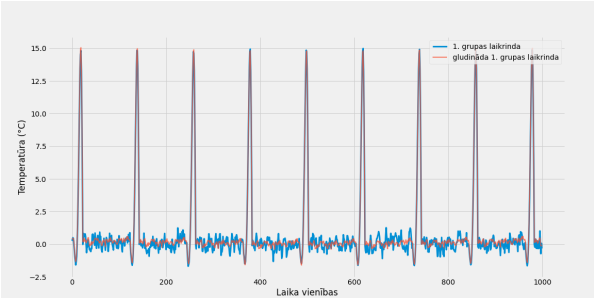
Laikrindu klasterēšanai ir līdzīga motivācija kā jeb kāda cita veida datu klasterēšanai - atrast un sagrupēt līdzīgas laikrindas n grupās. Lai atrastu un sagrupētu līdzīgas laikrindas mums ir jāizvēlas attāluma mērs, kas tiek izmantots, lai skaitliski raksturotu laikrindu līdzību un metode, kas tiks izmantota klasteru veidošanai. Attāluma mēra un klasterēšanas metodes izvēlei ir būtiska nozīme un to lielā mērā ietekmē kādi ir laikrindu dati un kāds ir klasterēšanas mērķis. Izpētes procesā tika apskatīti un izvērtēti vairāki attāluma mēri un klasterēšanas metodes [1] [8] [20] [16].
Tika salīdzināti un izmantoti, kopā ar dažādām klasterēšanas metodēm, sekojoši attālumu mēri:

- Eiklīda attālums.
- Dinamiskās deformācijas laikā metodes attālums (izmantojot eiklīda attālumu) (DTW).
- Gludināta dinamiskās deformācijas laikā metode (Soft-DTW).
- Globālā kodola izlīdzināšanas metodes attālums (GAK).
- Uz formu balstīts attālums.

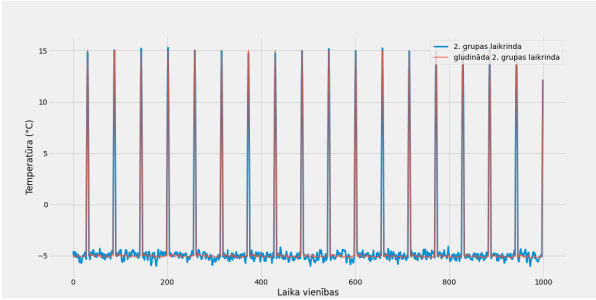
Salīdzinātās un izmantotās klasterēšanas metodes:

- K-vidējo klasterēšanas metode
- K-medoīdu klasterēšanas metode
- Hierarhiskās klasterēšanas metode

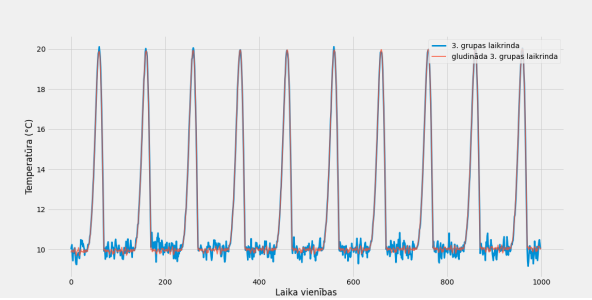
Pirms klasterēšanas laikrindas ir nepieciešams sagatavot izmantojot metodes no 4. un 5. nodaļām. Kā iepriekš minēts, temperatūras mērījumu laikrindu priekšapstrādē priekš klasterēšanas tiks izmantots gludināšana un mērogošana, vispirms nogludinot laikrindu un pēctam veicot mērogošanu izmantojot nogludinātās laikrindas. Gludināšanai tiek izmantot dekompozīcijas gludināšanas metode un katras grupas laikrindas salīdzinājumu ar gludinātu šīs laikrindas versiju var aplūkot 22. attēlā. Aplūkojot attēlus var redzēt, ka laikrindas ir nogludinātas samērā labi, saglabājot laikrindu sezonālos atkausēšanas perioda pīķus un regulārās darbības periodā samazinot temperatūras svārstības.



(a) **Gludinātas un negludinātas 1. grupas laikrindas salīdzinājums**



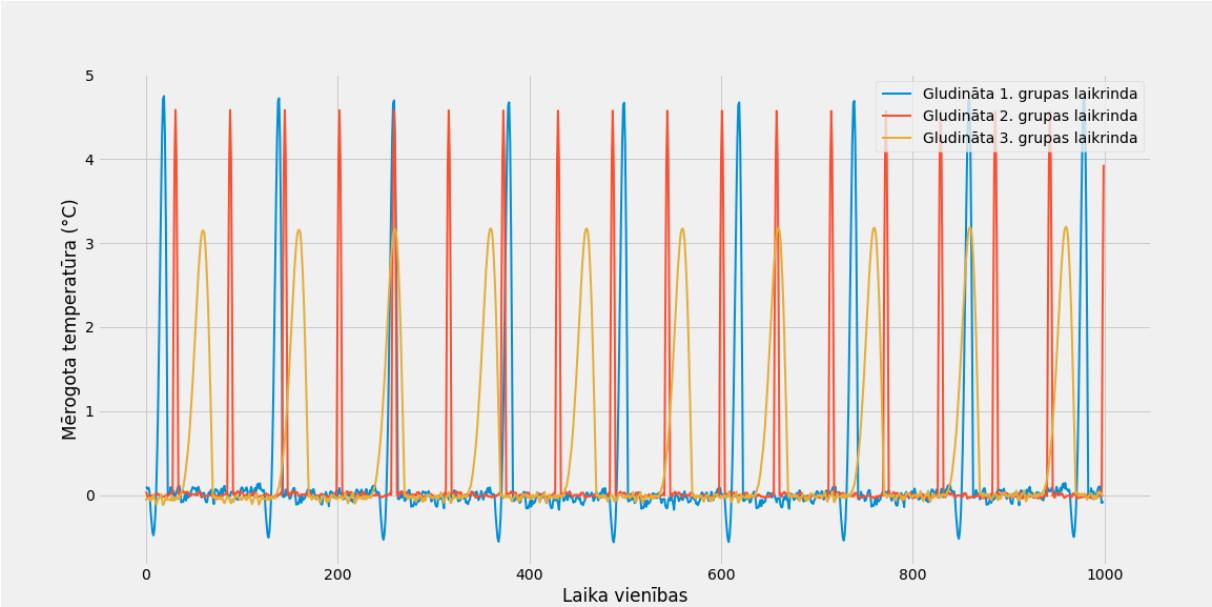
(b) **Gludinātas un negludinātas 2. grupas laikrindas salīdzinājums**



(c) **Gludinātas un negludinātas 3. grupas laikrindas salīdzinājums**

22. att. **Trīs temperatūras mērījumu grupu gludinātu un negludinātu laikrindu salīdzinājums**

Tālāk tiek veikta laikrindu mērogošana izmantojot mērogošanu izmantojot MAD un standartnovirzi un attēlā 23. varam redzēt trīs gludinātas un mērogotas laikrindas no visām trijām simulētajām grupām. Aplūkojot attēlu varam secināt, ka, salīdzinājumā ar tām pašām laikrindām 21. grafikā, laikrindās ir novērojama mazāka baltā trokšņa klātbūtne, visas tās ir vienā skalā un pēc priekšapstrādes tās ir saglabājušas savu struktūru. Iepriekš izdarītie secinājumi apstiprina, ka laikrindas ir veiksmīgi sagatavotas klasterēšanai.



23. att. **Trīs gludinātu un mērogotu temperatūras mērījumu laikrindu salīdzinājums no visām trijām grupām**

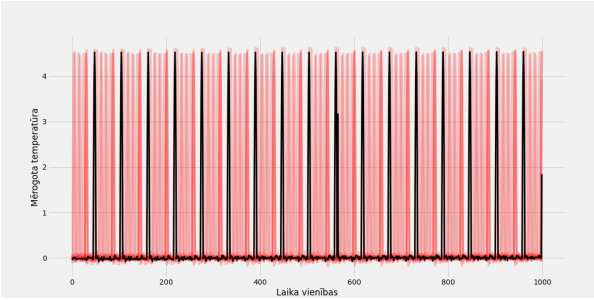
6.2. klasterēšana laikrindu grupēšanas nolūkam

Vispirms apskatīsim pilnu laikrindas klasterēšanu. Motivācija šāda veida klasterēšanai ir atrast līdzigas uzvedības laikrindas un sagrupēt tās. Sagrupējot laikrindas ļauj izstrādāt mērogojamu risinājumu, kas ļauj izstrādāt un pārbaudīt prognozēšanas un anomāliju noteikšanas metodes un modeļus attiecīgajai laikrindu grupai un līdz ar to atvieglojot modeļu analīzes un izstrādes procesu. Ar mērogojamību tiek domāts, ka nav svarīgi cik daudz modeļu ir attiecīgajā klasterī, bet zinot, ka visas attiecīgajam klasterim piederošās laikrindas ir līdzīgas savā uzvedībā, varam izmantot izstrādāto un pārbaudīto modeli vai metodi uz visām attiecīgā klastera laikrindām. Papildus sagrupētās laikrindas var izmantot, lai veiktu prognozi grupas ietvaros, piemēram, ja uzņēmumam ir produktu pārdošanas rādītāji kādā laika periodā, kas tiek sagrupēti izmantojot kādu no klasterēšanas metodēm, var tikt veikta prognozējošā modeļa izstrāde attiecīgās sensoru grupas sensoriem - izmantojot atsevišķu sensoru prognozējošo modeļu ansambli vai veidojot vairāku mainīgo prognozējošo modeli, kas ietver vairāku sensoru mērījumus konkrētā laika punktā.

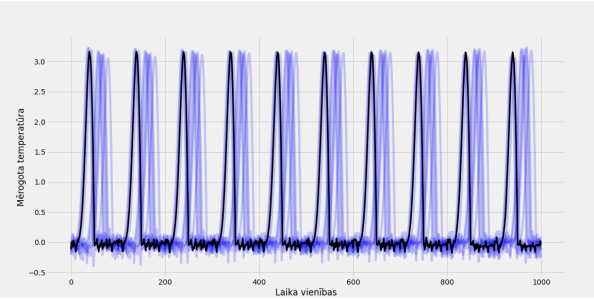
Tā kā apskatam visas laikrindas no visām trijām grupām, tad datu kopa satur 10 laikrindas no katras grupas, kas sastāda kopā ir 30 laikrindas. Lai varētu salīdzināt dažādas metodes savā starpā, piemērā izmantotās laikrindas satur vienādu novērojumu skaitu, bet ir vērts piezīmēt, ka ir metodes kam šis nosacījums nav nepieciešams, piemēram, DTW klasterēšanas metode. Praktiski tika apskatītas vairākas metodes (sīkāk skatīt B.4. pielikumu). Precizitāte aprakstot salīdzinātos rezultātus ir aprēķināta procentuāli izsakot pareizi saklasterēto laikrindu skaitu visām trim grupām.

Vislabākos rezultātus parādīja DTW (24.) un uz formu balstītas klasterēšanas attāluma (54.)

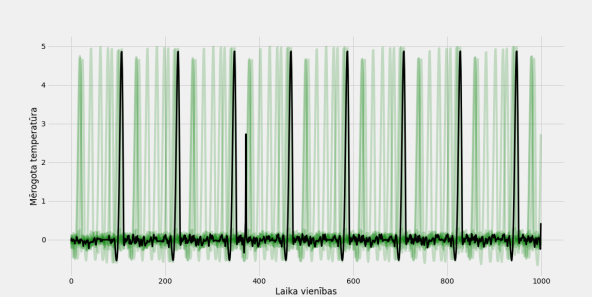
mēri kopā ar k-vidējo klasterēšanas metodi. Ir vērts uzvērt ka abu šo metožu lielākā priekšrocība ir tāda, ka tās neietekmē līdzīgu laikrindu nobīde laikā. Attiecīgi DTW un uz formu balstīta klasterēšanas metode 3 grupu temperatūras mērījumu laikrindas saklasterēja ar ideālu preciziāti (100%), savukārt populārās eiklīda attāluma k-vidējo klasterēšanas metodes precizitāte bija tikai 60% (53.). K-vidējo (ar DTW attāluma mēru) metodes rezultātus var aplūkot 24. attēlā, kur ar zilu, sarkanu un zaļu krāsu ir attēloti temperatūras mērījumu laikrindas dati un ar melnu krāsu - klastera centroīda laikrinda.



(a) **1. klasteris**



(b) **2. klasteris**

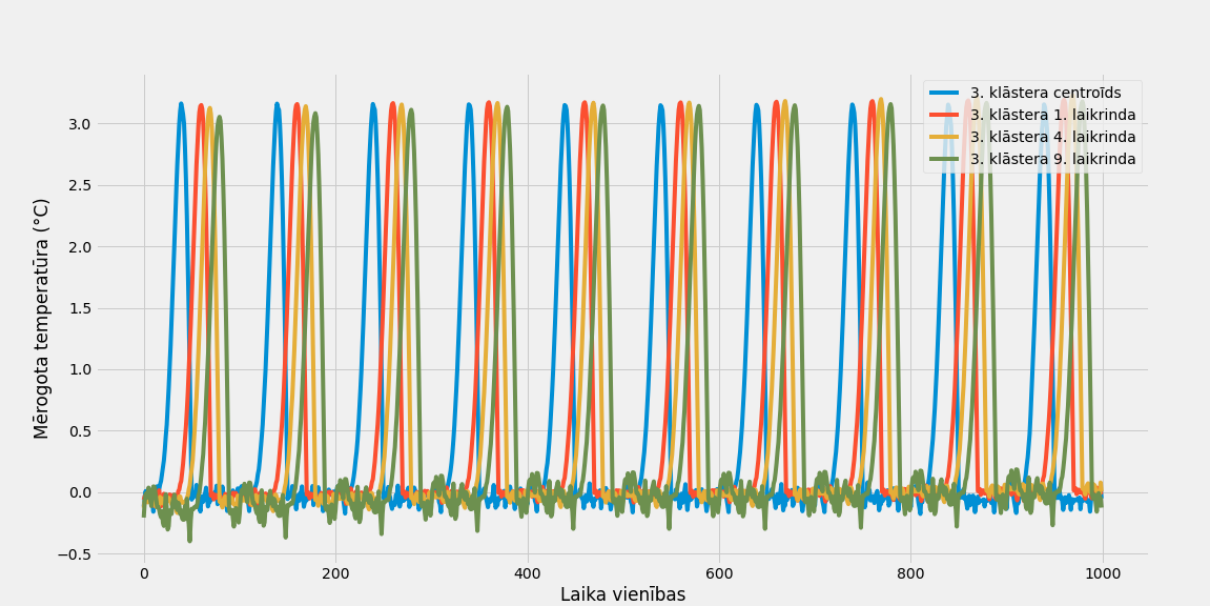


(c) **3. klasteris**

24. att. **K-vidējo klasterēšanas metodes (izmantojot DTW attāluma mēru) izveidoto klasteru un to centroīdu attēlojums**

Aplūkojot grafiku varam secināt, ka šī metode pietiekami labi raksturo katras grupas laikrindu tipisko struktūru un lai par to pārliecinātos tuvāk varam aplūkot 25. attēlu, kur ir attēlas trīs 3. grupas laikrindas kopā ar to piederošā klastera (klasteru numerācija var nesakrist ar grupu numerāciju) centroīdu. Rezultāti varētu šķist pašsaprotami un triviāli, bet salīdzinot ar eiklīda distances iegūtajiem rezultātiem (attēlu skatīt A.3. pielikumā), kur varam redzēt, ka dažāda veida laikrindas ir saliktas vienā klasterī un klasteru centroīdi nepavisam nav līdzīgi nevienai no laikrindu grupu struktūrām, var secināt, ka metode spēj atšķir visu trīs grupu laikrindas. Veicot šo analīzi ir skaidrs, ka Eiklīda attāluma mērs nav piemēros apskatāmajai problēmai. Neskatoties uz to, ka izmantojot Eiklīda mēru mūsu temperatūras mērījumu laikrindas netika saklasterētas precīzi, balstoties uz [8] rakstu, kur tika salīdzinātas vairākas metodes, tajā skaitā k-vidējās vērtības (ar DTW attāluma mēru) un k-vidējās vērtības (ar Eiklīda attāluma mēru), izmantojot 128 samerķētas laikrindas, abas šīs iepriekš minētās metodes bija vienas no precīzākajām un kopumā neviena no šīm metodēm neparādīja manāmi labākus rezultātus. Tāpēc ir vērts apskatīt vairākas

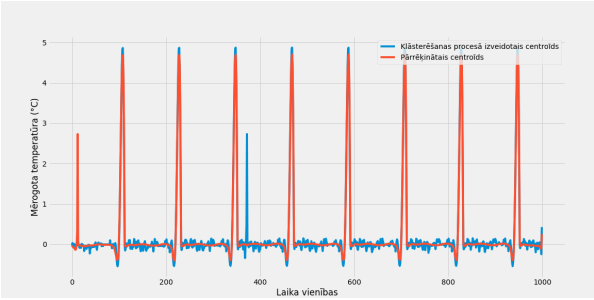
metodes un atrast piemērotāko attiecīgajai problēmai un mērķim, kas tiek risināts. Savukārt, aplūkojot A.3. attēlotos uz formu balstītus klasterēšanas rezultātus varam redzēt, ka laikrindas ir, līdzīgi kā ar DTW metodi, saklasterētas pilnīgi precīzi un klasteru centroīdi raksturo katram klasterim piederošo laikrindu tipisko struktūru. Vērīgs lasītās noteikti pamanīs, ka centroīdi ir nobīdīti un tas ir tādēļ, ka uz formu balstītas klasterēšanas metode sevī ietver mērogošanu izmantojot standart novirzi un vidējo aritmētisko vērtību. Papildus tika apskatīta Soft-DTW metode, kas ir gludināta versija DTW metodei, bet šī metode priekš klasterēšanas aizņēma krietni vairāk laika un iegūtie rezultāti nebija labāki kā vienkārši DTW metodei, tādēļ šī metode nemaz netiek izmantota salīdzinājumā. Visātrākais skaitliskā aprēķina laiks bija metodēm, kas izmanto Eiklīda attālumu, tas bija mazāks par 1 sekundi. Uz formu balstītas klasterēšanas metodes aprēķina laiks aizņēma aptuveni 42 sekundes (veicot 3 atkārtotus aprēķinus ar dažādiem sākuma stāvokļiem) un DTW metode (viecot trīs atkārtotus aprēķinus) aizņēma nedaudz vairāk par minūti. Šiem laikiem gan ir informatīva nozīme, lai saprastu kura metode ir ātrāka par kuru metodi, bet aprēķina laiki noteikti var variēt atkarībā no izvēlētajiem parametriem un citiem nosacījumiem. Ir vērts pieminēt, ka uz formu balstīta metode, lai gan aprēķini tika veikti ātrāk, ne vienmēr (pat ar 3 atkārtotiem aprēķiniem) spēja nokonverģēt un precīzi saklasterēt dotās laikrindas. Ņemot vērā visus iepriekš minētos secinājumus, kā galvenā klasterēšanas metode tiek izvēlēta k-vidējo klasterēšanas metode ar DTW kā attāluma mēru.



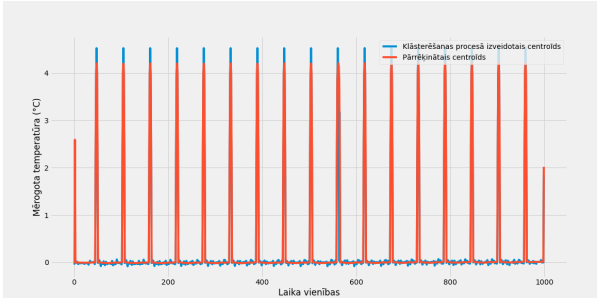
25. att. **Trīs 3. grupas laikrindu salīdzinājums ar tās klastera centroīdu**

Varam novērot, ka visu trīs klasteru centroīdi diezgan precīzi raksturo katras grupas laikrindu struktūru 24., bet ir atsevišķas vietas, kur ir novērojami neparasti novērojumi. Lai gan tas neietekmē esošo laikrindu klasterēšanu, tas var ietekmēt jaunu laikrindu klasifikāciju. Lai uzlabotu klastera centroīda izveidi, kad laikrindas ir saklasterētas, katrai no laikrindām tiek pārrēķināts tās centroīds izmantojot Soft-DTW metodi, kas kā atceramies iepriekš netika izmantots pašā klasterēšanas procesā, jo aprēķina laiks aizņēma daudz vairāk laika nekā citas metodes.

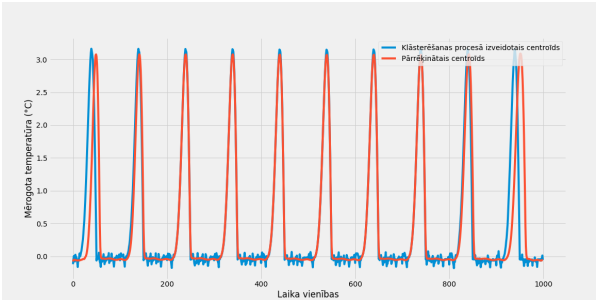
Soft-DTW tiek izmantots laikrindas klastera baricentra aprēķināšanai un parāda daudz labākus klastera centroīda novērtējumu kā DTW klasterēšanas metode. Vēlāk šīe centroīdi var tikt izmantoti, lai pievienotu jaunus sensoru datus pie klastera ar kura centroīdu attālums pēc DTW attāluma mēra ir vismazākais. Katra klastera pārrēķināto centroīdu salīdzinājumu ar klasterēšanas procesā izveidotajiem centroīdiem var apskatīt 26. attēlā. Kā redzam centroīdu salīdzinājuma grafikos, nav manāmas būtiskas izmaiņas, bet tomēr ir redzams, ka pārrēķinātie centroīdi ir daudz gludāki un tajos ir izlabotas dažās samērā nelielas nepilnības, kā piemēram - 3. klasterim pirmais un pēdējais atkausēšanas periods bijis nedaudz nobīdīts un 1. klastera centroīdam ir novērojams neizprotams lēciens pie aptuveni 380. laika vienības. Ir vērts piebilst, ka jaunizveidotajiem jeb uzlabotajiem centroīdiem - 1. un 2. klasterim pēdējais un pirmais atkausēšanās periods ir zemāks dēļ tā, ka klasterī esošajām laikrindām ir nobīde un tas nozīmē, ka šajā klasterī ietilpst laikrindas ar dažādu atkausēšanas periodu skaitu un tas veido šo centroīdu galapunktu neprecīzo attēlojumu. Bet ja neņemam vērā galapunktus, tad katra pudura centroīdi diezgan precīzi raksturo laikrindu struktūru, kas tika maskēta ar trokšņa pievienošanu un periodu nobīdi.



(a) **1. klastera centroīdi**



(b) **2. klastera centroīdi**



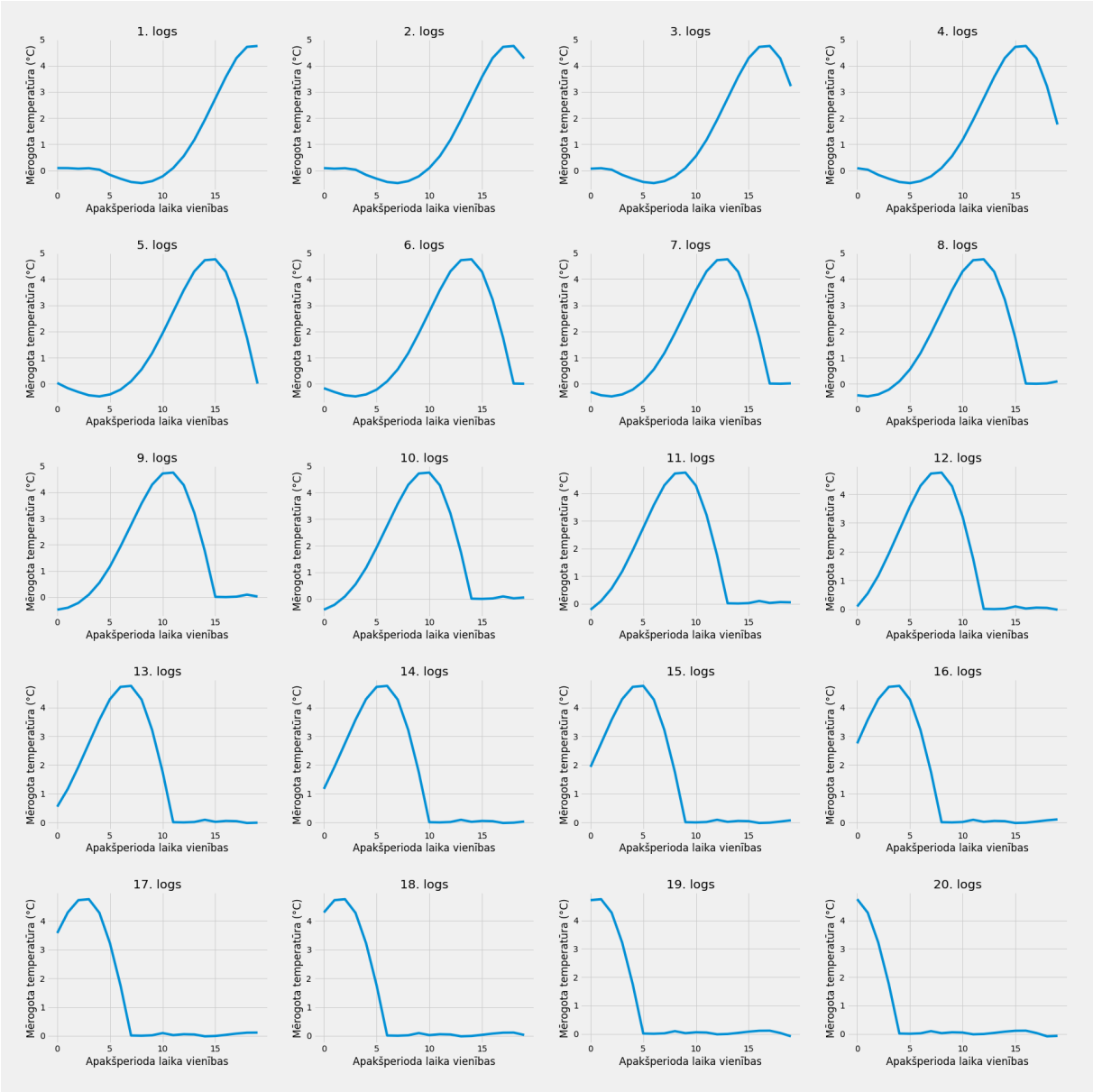
(c) **3. klastera centroīdi**

26. att. **Katra klastera pārrēķināto centroīdu salīdzinājums ar to centroīdiem, kas iegūti klasterēšanas procesā**

Iegūtie rezultāti ir nozīmīgi tieši modeļu izstrādes un testēšanas procesā, kas detaļās netiks aprakstīts šajā darbā, jo darba mērķis ir aprakstīt modeļu un metožu izpēti, lai parādītu kā konceptuāli atrisināt attiecīgo problēmu. Bet tas nekādā mērā nemazina šīs apakšnodaļā iegūto rezultātu un izpētes nozīmīgumu, jo iegūtie rezultāti ir būtiski tālāk minēto metožu mērogojama risinājuma tālākai izstrādei.

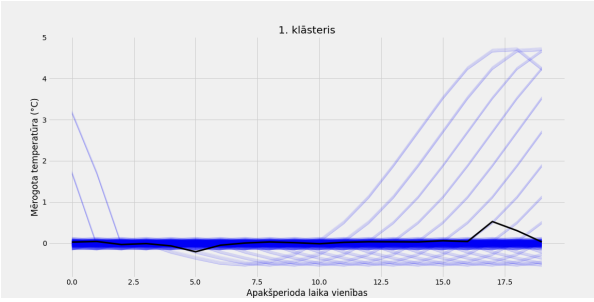
6.3. Apakšperiodu klasterēšana anomāliju noteikšanai

Otra darbā apskatītā klasterēšanas pieeja ir klasterēt konkrētas laikrindas apakšperiodus. Ar to ir domāts, sadalīt laikrindu apakšperiodos un saklasterējot šos apakšperiodus, lai identificētu tipiskos laikrindas periodus un to uzvedību, kas pēc tam var tikt izmantots, lai gan klasificēt periodus, gan arī lai noteiktu anomālus periodus. Pati klasterēšana neatšķiras no pilnas laikrindu klasterēšanas procesa, bet būtiskākā atšķirība ir tāda, ka šajā pieejā mūs interesē aplūkot tikai vienu laikrindu un visus tās apakšperiodus. Laikrindu sadalīšanai apakšperiodus nepieciešams nodefinēt apakšperioda jeb apskatāmā laika loga garumu un laika vienību skaitu starp i un $i + 1$ laika loga sākuma laika vienībām. Papildus var apsvērt domu, pēc nepieciešamības, katru apakšperiodu papildus gludināt un/vai mērogot, bet priekš mūsu apskatāmā piemēra tas nav nepieciešams. Priekš temperatūras mērījumu laikrindu datiem tiek izvēlēts laikrindu dalīt laika logos 20 vienību garumā un atstarpe starp laika logiem ir 1 laika vienība. Tas tiek darīts, lai parādītu, ka pat ja atstarpe starp laika logiem tiek izvēlēta 1 vienības garumā, tad neskatoties uz to, ka eksistēs tādi apakšperiodu kas pieder dažādiem klasteriem un tie atšķirsies tikai ar vienu novērojumu, klasterēšanas rezultātā tiek iegūti klasteri, kas parāda laikrindas atšķirīgos periodus. Simulētajiem temperatūras mērījumiem, kā atminamies no 3. ir divi periodi - atkausēšanas un regulārais periods, tāpēc veicot apakšperiodu klasterēšanu tiek izvēlēts klasterēt šos apakšperiodus 2 klasteros. No visām temperatūras mērījumu simulētajām laikrindām tiek izvēlēta 1. grupas 1. laikrinda un tā tiek sadalīta apakšperiodos (loga garums - 20, atstarpe starp laika logiem - 1). Attēlā 27. varam aplūkot pirmos 20 laikrindas apakšperiodus.

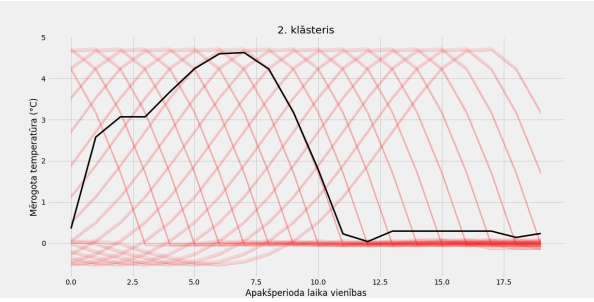


27. att. **1. grupas 1. laikrindas pirmie 20 laika logi**

Pēc apakšperiodu izdalīšanas nākamais solis ir saklasterēt šos apakšperiodus ar jebkuru no 6.1. apakšnodaļā minētajām metodēm. Balstoties uz 6.2. apakšnodaļu, izvēle vairāk sliecas uz DTW vai uz formu balstītu klasterēšanu, bet tieši apakšperiodu klasterēšanā varētu būt situācijas, kur piemērotāka varētu būt klasterēšanas metode, kas izmanto Eiklīda attālumu. Pieturoties pie 6.2. apakšnodaļā izvēlētās metodes, apakšperiodi tiek klasterēti izmantojot k-vidējo klasterēšanas metodi ar DTW kā attāluma mēru. Apskatot 28. attēlu varam redzēt abus iegūtos klasterus, kur ar zilu vai sarkanu krāsu tiek attēloti klasterim piederošie apakšperiodi, savukārt ar melnu krāsu tiek attēloti abu klasteru centroīdi.



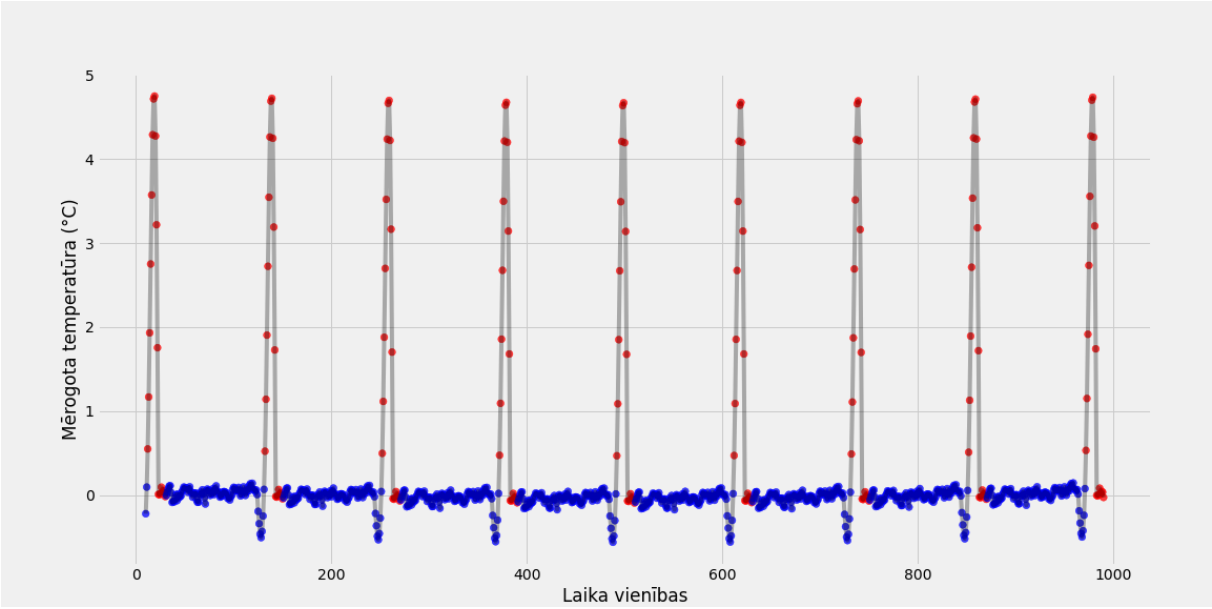
(a) **1. klasteri ar tā centroīdu**



(b) **2. klasteris ar tā centroīdu**

28. att. **1. grupas laikrindas apakšperiodu klasteri**

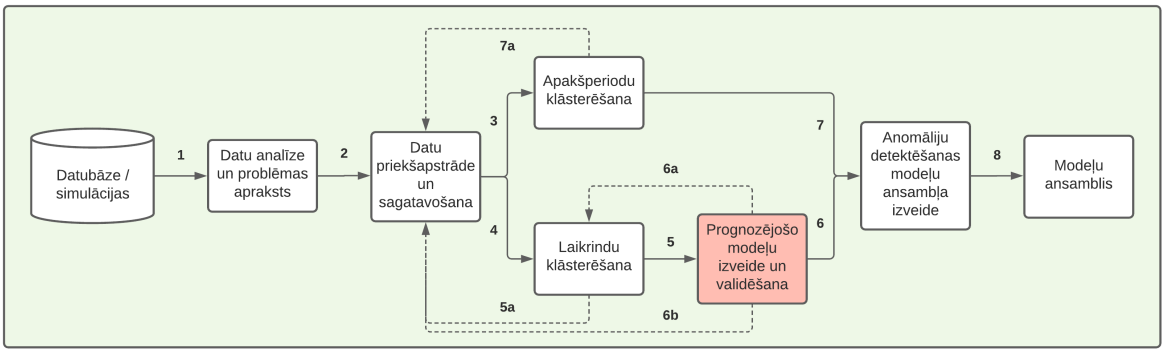
Apskatot abu klasteru piederošo apakšperiodu attēlojumus (28.) varam novērot, ka abi klasteri satur dažādākas formas apakšperiodus un izskatās, ka 1. klasteris satur regulāro temperatūras mērījumu apakšperiodus, bet 2. klasteris atkausēšanas periodu mērījumu apakšperiodus. Lai labāk ilustrētu katra punkta piederību konkrētam klasterim, tiek zīmēts grafiks (29.), kas satur klasterēto laikrindu un tās novērojumus, kas ir iekrāsoti to piederošā klastera krāsā. Aplūkojot 29. attēlu apstiprinās iepriekš izdarītais pieņēmums, ka viens klasteris raksturo regulāra temperatūras mērījuma periodu novērojumus, bet otrs atkausēšanas perioda mērījumus.



29. att. **1. grupas 1. laikrindas pirmie 20 laika logi**

Esam veiksmīgi atraduši parametrus un metodi ar ko varam klasterēt temperatūras mērījumu apakšperiodus un tālāk šajā apakšnodaļā aplūkotie rezultāti tiks izmantoti 8., kur iegūtie apakšperiodu klasteri un to centroīdi tiks izmantoti, lai identificētu anomālus apakšperiodus.

7. PROGNOZĒŠANA



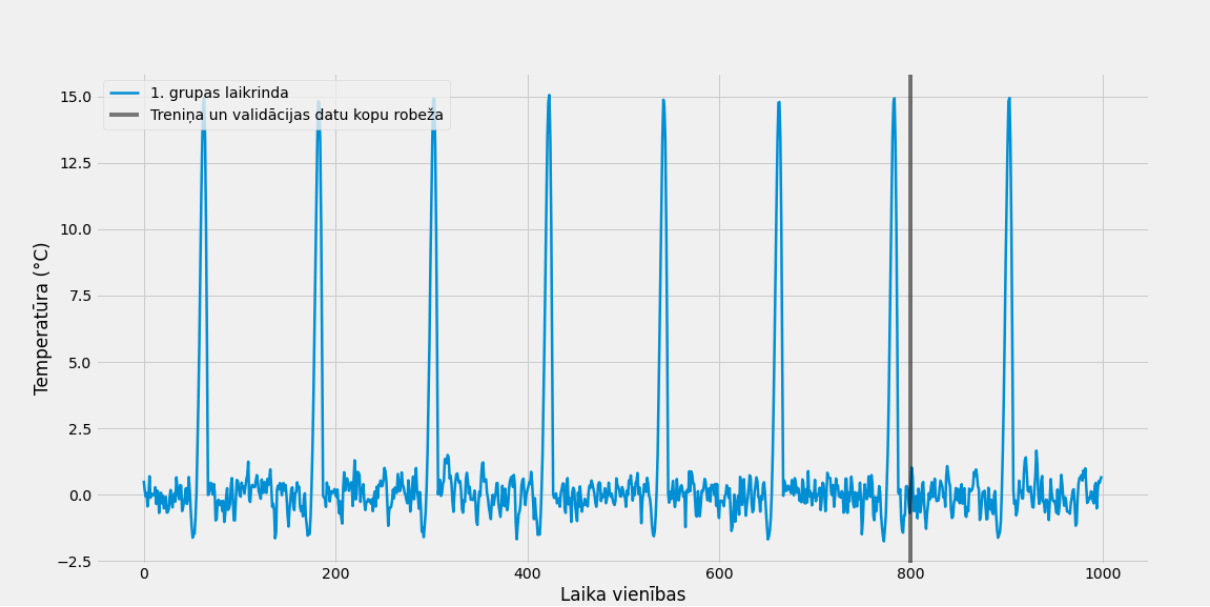
30. att. **Datu analīzes un modeļa izstrādes diagramma, laikrindu prognozēšanas fāze**

Šajā nodaļā tiks apskatīti dažādi laikrindu prognozējošie modeļi, kas ir piemēroti temperatūras mērījumu laikrindas datu prognozēšanai. Vispirms tiks definēti modeļu prognozes novērtēšanas metrikas un krosvalidācijas metode priekš laikrindu modeļu trenēšanas. Pēctam tiks apskatīti naivie modeļi, kas ir vienkārši modeļi ar mazu parametru daudzumu un balstīti uz konkrētiem nosacījumiem, kam sekos statistiskie, mašīnmācīšanās un dziļās apmācības modeļi. Prognozējošo algoritmu izstrādes darba grāmatu var aplūkot B.5. pielikumā.

Visu modeļu trenēšanai tiks izmantota treniņa datu kopa ar pirmajiem 1. grupas temperatūras mērījumu laikrindas novērojumiem. 1. grupas laikrindu, kas tiks izmantota šajā nodaļā apskatīto modeļu salīdzināšanai var aplūkot 31. attēlā, ar zilu krāsu tiek attēlota pati laikrinda un ar melnu krāsu tiek iezīmēta robeža, kas atdala treniņa un validācijas kopas. Validācijas kopa satur 200 novērojumus. Visi šajā nodaļā izmantotie tiks novērtēti izmantojot validācijas kopu un ir vērts piebilst, lai gan būs modeļi, kuri tiks trenēti, lai veiktu viena soļa prognozi - prognoze tiks veikta iteratīvi izmantojot iepriekšējā laika solī prognozēto vērtību, tādā veidā izvairoties no validācijas kopas datu/informācijas ”noplūdes” (angliski - *data leakage*). Aplūkojot 31. varam secināt, ka dēļ atkārtojošā un stacionārās uzvedības, modeļiem būtu precīzi jāprognozē izvēlētā laikrinda. Modeļu/metožu izvērtēšanā tiks ņemta vērtā ne tikai modeļa precizitāte, bet arī modeļa trenēšanas laiks, kas ir būtiska, jo mērķis ir izpētīt un atrast vispārināmu, precīzu un aprēķina ziņā ātru risinājumu, kas ļautu izveidot mērogojamu modeļu izveidi. Katra tipa modeļiem atšķirsies treniņu datu kopas izveides process, kā arī, dēļ modeļu īpašībām, atšķirsies parametru novērtēšanas process izmantojot korvalidāciju. Nodaļā tiek izmantoti trīs dažādi datu kopu jēdzieni treniņa, testa un validācijas kopa:

- Treniņa kopa - datu kopa, kas tiek izmantota, lai trenētu modeli.
- Testa kopa - datu kopa, kas tiek izmantota modeļa novērtēšanā krosvalidācijas procesā. Ja modeļa trenēšanas procesā netiek veikta krosvalidācija, tad test kopa netiek izdalīta.

- Validācijas kopa - datu kopa, kas tiek izmantota, lai novērtētu gala modeli. Šīs kopas dati netiek izmantoti neviena modeļa trenēšanas procesā, nedz arī prognožu veikšanas procesā.



31. att. **1. grupas laikrinda ar treniņa un validācijas kopas robežu**

7.1. Prognozes novērtēšanas metrikas

Lai skaitliski novērtētu prognozes precizitāti ir nepieciešams izvēlēties un definēt novērtēšanas metrikas, kas tiks izmantotas, lai novērtētu katru no aplūkotajām metodēm un palīdzētu pieņemt lēmumu par to, kura metode ir piemērotākā mūsu problēmas risināšanai. [9]
Lai novērtētu laikrindas prognozes precizitāti tiek apsvērtas sekojošas metodes:

- *R kvadrāts* - R^2, kas var tikt interpretēts, kā neizskaidrotās kļūdas procents, vērtību apgabals $(-\inf, 1]$:

$$SS_{res} = \sum_{i=1}^{n} (\hat{y}_i - y_i)^2$$

$$SS_{tot} = \sum_{i=1}^{n} (\overline{y} - y_i)^2$$

$$R^2 = 1 - \frac{SS_{res}}{SS_{tot}}$$

- *Vidējā absolūtā kļūda* (**MAE**), iterpretējama metrika, jo saglabā novērojumu skalu, vērtību apgabals $[0, +\inf)$:

$$MAE = \frac{\sum_{i=1}^{n} |y_i - \hat{y}_i|}{n}$$

- *Absolūtās kļūdas mediāna* (**MedAE**), robusta un interpretējama metrika, jo saglabā novērojumu skalu un izmanto mediānu, vērtību apgabals $[0, +\inf)$:

$$MedAE = median(|y_1 - \hat{y}_1|, ..., |y_n - \hat{y}_n|)$$

- *Vidējā kvadrātiskā kļūda* (**MSE**), viena no populārākajām metrikām, šī metrika soda lielākas kļūdas, vērtību apgabals $[0, +\text{inf})$:

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

- *Vidējā kvadrātiski logaritmiskā kļūda* (**MSLE**), līdzīga metrika kā MSE, tikai tiek ņemts logaritms no laikrindas, tā kā šo metodi visbiežāk izmanto gadījumos, kur laikrindas trendam ir eksponenciāla rakstura pieagums, vērtību apgabals $[0, +\text{inf})$:

$$MSLE = \frac{1}{n}\sum_{i=1}^{n}(log(1 + y_i) - log(1 + \hat{y}_i))^2$$

- *Vidējās absolūtās kļūdas procents* (**MAPE**), raksturo tieši to pašu ko MAE, tikai izteikts kā procents, vērtību apgabals $[0, +\text{inf})$:

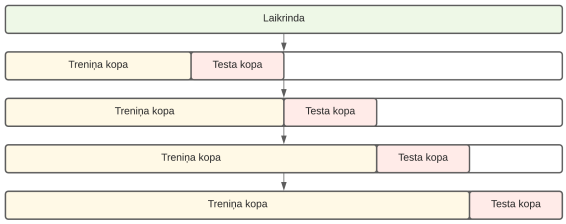
$$MAPE = \frac{100}{n}\sum_{i=1}^{n}\frac{|y_i - \hat{y}_i|}{y_i}$$

Novērtējot temperatūras mērījumu laikrindu prognozes tiks aprēķinātas visas iepriekš minētās metrikas (izņemot MSLE, jo mūsu piemērā laikrinda pieņem negatīvas vērtības), bet lielāka uzmanība tiks pievērsta sekojošām metrikām - R^2, MAPE un MSE.

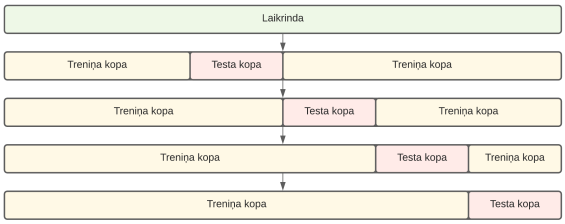
7.2. Laikrindu prognozējošo modeļu trenēšana izmantojot krosvalidāciju

Laikrindu modeļu trenēšanai, tāpat kā cita veidu datu trenēšanā, ir iespējams veikt korsvalidāciju. Atkarībā no izvēlētā modeļa, atšķirsies arī korsvalidācijas treniņa un testa kopas izveide. Attēlā 32. varam aplūkot divus krovalidācijas procesus, kuru izvēli ietekmē izvēlētais laikrindu prognozējošais modelis. Priekš modeļiem, kam ir nepieciešama nepārtraukta novērojumu virkne, krosvalidācija tiek veikta izmantojot krosvalidācijas procesu attēlotu 32a. attēlā. Kā varam novērot krosvalidācijas procesa attēlojumā, kā pirmā krosvalidācijas datu kopa tiek ņemta laikrindas pirmie 40% novērojumu kā treniņa, sekojošie 15% novērojumu kā testa kopa un atlikušie 45% netiek vispār ņemti vērā treniņa procesā. Nākamajā krosvalidācijas datu kopā treniņa datu kopa tiek palielināta par 15% un tiek ņemti pirmie 55% novērojumu kā treniņa kopa, sekojošie 15% novērojumu kā testa kopa un atlikušie 30% netiek izmantoti. Analoģiski, palielinot par 15% treniņa kopu un izmantojot sekojošos 15% kā testa kopu tiek izveidota trešā datu kopa un pēdējā datu kopa, kas satur gan treniņa, gan testa datu kopas, tiek izveidota ņemot pirmos 85% novērojumu kā treniņa kopu un atlikušos 15% novērojumu kā testa kopu. Kopā, pie šāda sadalījuma izvēles, sanāk 4 datu kopas, kuras var izmantot, lai veiktu korsvalidāciju. Savukārt 32b. attēlā ir attēlota korsvalidāciju datu kopu izveide, kas ir līdzīga cita veida datu krosvalidācijas procesam. Šāda veida krosvalidāciju procesu var veikt modeļiem, kam nav nepieciešama nepārtraukta novērojumu rinda. Šo krosvalidācijas pieeju var izmantot tādiem algoritmiem, kā

piemēram, LightBOOST un LSTM, jo laikrindas dati tiek transformēti novērojumus ar m neatkarīgiem mainīgajiem un n atkarīgiem mainīgajiem, kur attiecīgā novērojuma m neatkarīgie mainīgie ir secīgi laikrindas apakšperioda novērojumi un atkarīgie mainīgie, skaitā n, ir neatkarīgo mainīgo laikrindas apakšperioda sekojošais apakšperiods.



(a) **Nepārtrauktas testa un treniņa novērojumu virkņu krosvalidācija**



(b) **Pārtrauktas testa un treniņa novērojumu virkņu krosvalidācija**

32. att. **Laikrindu krosvalidāciju veidi**

7.3. Naivie modeļi

Naivie modeļi ir pavisam vienkārši modeļi, kas dod iespēju novērtēt sarežģītāku modeļu pievienoto vērtību. Ja sarežģītāki modeļi nespēj dot labāku precizitāti kā naivie modeļi, tad šie modeļi tiek uzskatīti par nepiemērotiem. Šīm metodēm nav nepieciešama trenēšana un ir nepieciešami tikai pēdējie novērojumu (skaits atšķiras atkarībā no izvēlētās metodes). [4]

1. Pēdējās vērtības prognoze - ļoti elementāra metode, kas veic prognozē nākamo vērtību izmantojot iepriekšējā laika soļa vērtību, prognozi var apskatīt 33. attēlā.

$$\hat{y}_t = y_{t-1}$$

2. Vidējās vērtības prognoze - šī metode pieņem, ka visas prognozētās vērtības ir vienādas ar laikrindas vidējo vērtību. Šīs metodes, balstoties uz R^2 definīciju, R^2 novērtējums būs tuvu nullei, tāpēc šīs metodes mērķis ir apskatīt un salīdzināt citu prognožu novērtēšanas metriku vērtības.

$$\hat{y}_t = \overline{y}$$

3. Pēdējo k vērtību vidējās vērtības prognoze - kalpo kā nogludinoša funkcija, pieņemot vērtību, kas ir vienāda ar pēdējo k novērojumu vidējo vērtību. Šīs metodes prognozi var aplūkot 33. attēlā.

$$\hat{y}_t = \frac{1}{k}\sum_{n=1}^{k} y_{t-n}$$

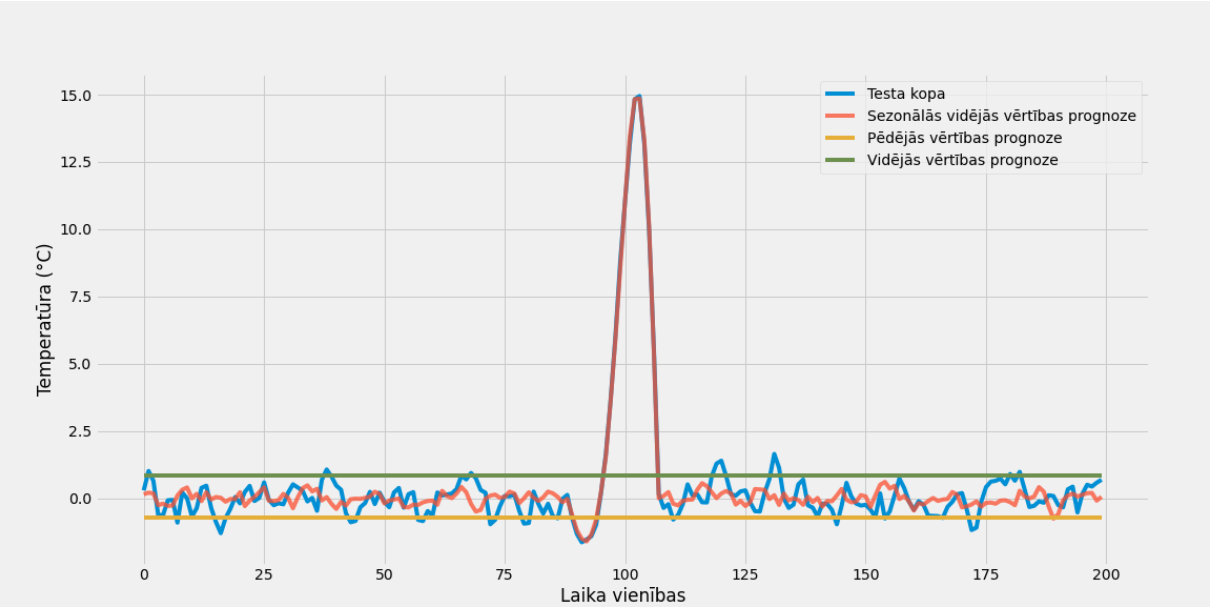
4. Sezonālās vērtības prognoze - pieņem novērojuma vērtību pirms m laika vienībām. Tā kā dati ir sezonāli, tad šī metode varētu būt piemērotāka kā pēdējās vērtības prognoze.

$$\hat{y}_t = y_{t-m}$$

5. Vidējās sezonālās vērtības prognoze - līdzīgi kā sezonālās vērtības prognoze, tiek ņemtas vērtā vērtības ņemot vērā sezonas perioda garumu un aprēķinot vidējo vērtību no šī pēdējām k vērtībām. Sī metode ietver sezonalitāti un gludināšanu. Šīs metodes prognozi var aplūkot 33. attēlā.

$$\hat{y}_t = \frac{1}{k}\sum_{n=1}^{k} y_{t-nm}$$

Tiek izvēlēts modeļu salīdzinājumā izmantot 3 no 5 naivajiem modeļiem. Grafiski izvērtējot izvēlētos naivos modeļus varam novērot, ka vidējās vērtības un pēdējās vērtības prognozes nav precīzas un iespējams vidējās vērtības prognoze būtu vērts aizvietot ar laikrindas mediānas vērtības prognozi. Savukārt sezonālās vidējās vērtības prognoze izskatās, ka diezgan labi spēj prognozēt šo laikrindu un pārējām metodēm varētu būt sarežģīti pārspēt šīs salīdzinoši vienkāršās metodes rezultātus. Papildus, šīs metodes prognožu veikšanas/aprēķina laiks ir viens no īsākajiem. Metožu rezultātus un salīdzinājumus var aplūkot 7. tabulā.



33. att. **Naivo modeļu prognožu salīdzinājums**

7.4. Statistiskie modeļi

Darbā par statistiskajiem modeļiem tiek saukti un uzskatīti modeļi, kuru pamatā ir statistiskās analīzes metodes, pamatā tie ir modeļi, kas plaši tiek izmantoti ekonometrijā. Visi šajā apakšnodaļā apskatītajiem modeļiem nepieciešams sagatavot treniņa datus tā, lai tie ir nepārtraukti un gala modeļa trenēšanā tiek izmantota visa treniņa kopa izmērā n, lai var veikt nākamo $m \in \mathbb{N}$ novērojumu prognozes $t_{n+i}, \forall i \in [1, m]$. [2]

7.4.1. Trīskāršās eksponenciālās gludināšanas modelis, jeb Holt-Winters modelis

Šis ir vienkāršā eksponenciālās gludināšanas (SES) un divkāršās eksponenciālās gludināšanas modeļu papildinājums, kas ņem vērā ne tikai trendu, bet arī sezonalitāti. Trīskāršās eksponenciālās gludināšanas (HW) aditīvais modelis sastāv no sekojošām komponentēm:

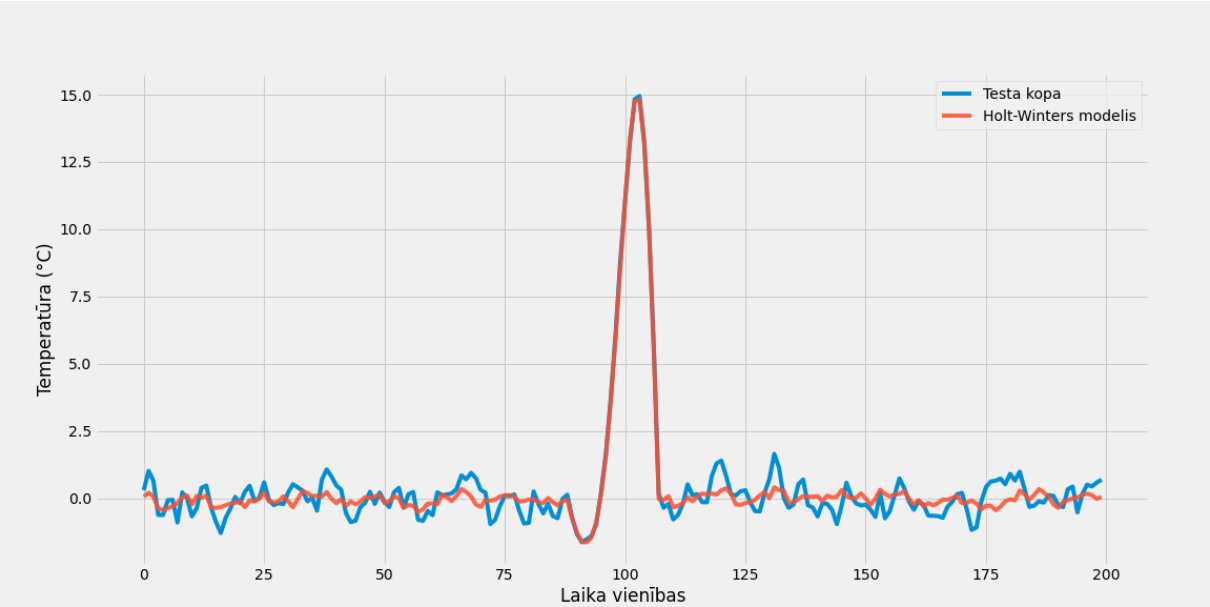
$$
\begin{aligned}
\text{Modeļa vienādojums} \quad & \hat{y}_{t+h|t} = \ell_t + hb_t + s_{t+h-m(k+1)} \\
\text{Līmeņa kompoenente} \quad & \ell_t = \alpha(y_t - s_{t-m}) + (1-\alpha)(\ell_{t-1} + b_{t-1}) \\
\text{Trenda kompoenente} \quad & b_t = \beta^*(\ell_t - \ell_{t-1}) + (1-\beta^*)b_{t-1} \\
\text{Sezonālā komponente} \quad & s_t = \gamma(y_t - \ell_{t-1} - b_{t-1}) + (1-\gamma)s_{t-m},
\end{aligned}
$$

kur h ir prognozējamās vērtības laika solis, m ir sezonas garums un k ir veselā daļa no $(h-1)/m$. l_t ir vidējais svērtais novērtējums starp sezonāli pielāgoto novērojumu $(y_t - s_{t-m})$ un nesezonālo $(\ell_{t-1} + b_{t-1})$ prognozi laika momentā t un $0 \leq \alpha \leq 1$. b_t ir līmeņa starpības - $\ell_t - \ell_{t-1}$ un pagājušā laika momenta trenda - b_{t-1} vidējā svērtā vērtība, kur $0 \leq \beta^* \leq 1$. s_t ir pašreizējās sezonālas indeksa - $(y_t - \ell_{t-1} - b_{t-1})$ un pagājušās sezonas (m laika vienības atpakaļ) indeksa vidējais svērtais novērtējums, kur $0 \leq \gamma \leq 1 - \alpha$. ([7])

Lai novērtētu un iegūtu ticamības intervālus ir nepieciešams izmantot Brutlag metodi, kur ticamības intervāls tiek novērtēts sekojoši:

$$
\begin{aligned}
\hat{y}_{max_x} &= \ell_{x-1} + b_{x-1} + s_{x-T} + m \cdot d_{t-T} \\
\hat{y}_{min_x} &= \ell_{x-1} + b_{x-1} + s_{x-T} - m \cdot d_{t-T} \\
d_t &= \gamma|y_t - \hat{y}_t| + (1-\gamma)d_{t-T}.
\end{aligned}
$$

Tātad, HW modelim ir viens fiksēts lielums, kas ir jāfiksē - sezonas garums un trīs parametri - α, β^* un γ, kurus ir nepieciešams novērtēt. Lai novērtētu visus trīs parametrus tiek izmantota iepriekš minētā krosvalidācijas metode. Aplūkojot 34. attēlu varam redzēt HW modeļa prognozi temperatūras mērījumu validācijas perioda (garumā 200 laika vienības) prognozi un vizuāli novērtējot varam secināt, ka modeļa veiktā prognoze ir precīza. Modeļu rezultātus un salīdzinājumus var aplūkot 7. tabulā.

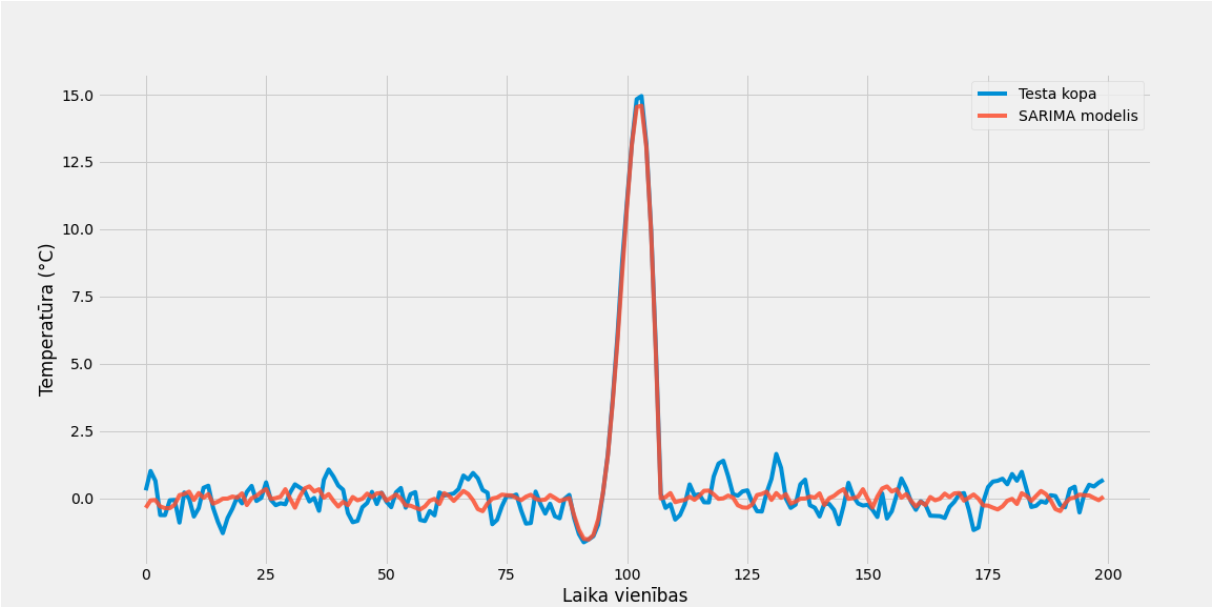


34. att. **Holt-Winters modeļa validācijas perioda prognoze**

7.4.2. SARIMA

SARIMA modelis ir iepriekš minētais ARIMA modelis ar pievienotu sezonālo komponenti. SARIMA satur nesezonālos paramterus (p, d, q), sezonalos parametrus $(P, D, Q)s$ un modeļa prametrus pieraksta sekojoši - $(p, d, q) \times (P, D, Q)_s$. SARIMA modeļa definīcija ir atrodama 2. nodaļā.

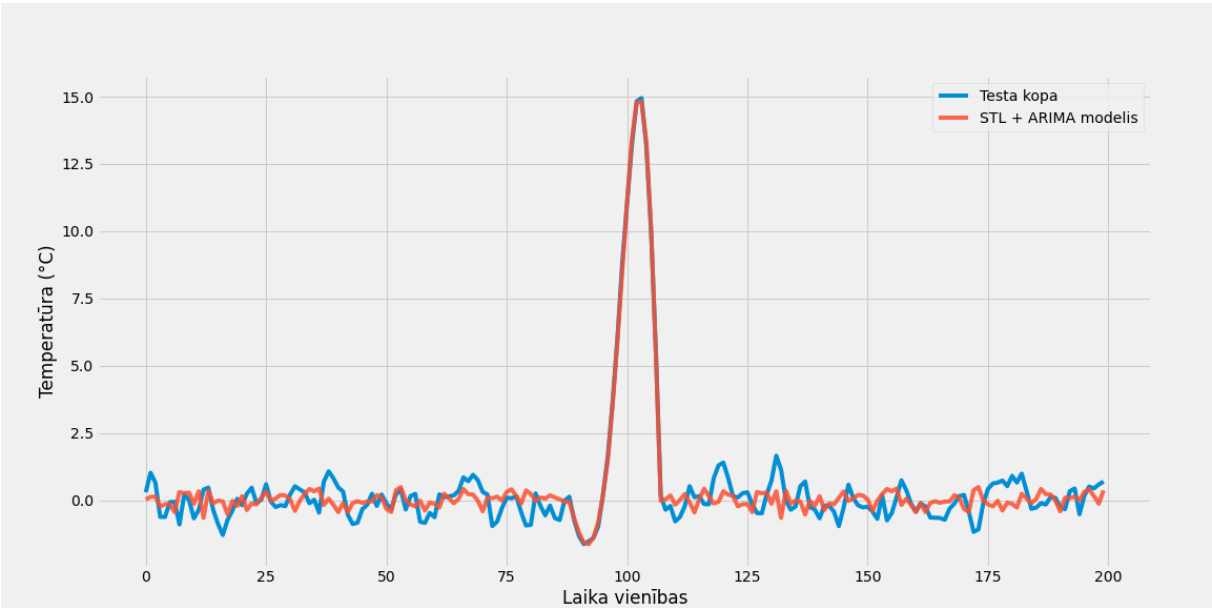
SARIMA modeļa prognozi var aplūkot 35. attēlā. Aplūkojot SARIMA modeļa prognozes salīdzinājumu ar laikrindas validācijas periodu ir redzams, ka modelis labi prognozē laikrindas uzvedību, bet ir vērts pieminēt, ka SARIMA modeļa trenēšanas laiks laikrindai ar 120 vienību garu sezonālu bija krietni lielāks par visiem citiem apskatītājiem modeļiem. Modeļu rezultātus un salīdzinājumus var aplūkot 7. tabulā.



35. att. **SARIMA modeļa validācijas perioda prognoze**

7.4.3. STL un ARIMA hibrīd modelis

STL un ARIMA hibrīd modeļa princips ir izmantot STL 4. nodaļā apskatīto metodi, kas spēj sadala laikrindu trijās komponentēs - trenda, sezonālajā un kļūdas/atlikuma, lai prognozētu trenda un sezonalitātes daļu un iepriekš apskatīto ARIMA modeli, lai prognozētu atlikušo atlikuma/kļūdas daļu. Hibrīd modeļa prognozi var aplūkot 36. attēlā. Attēlā varam redzēt, ka arī šis modelis labi prognozē laikrindas uzvedību un tā salīdzinājumu ar pārējiem modeļiem var aplūkot 7. tabulā.



36. att. **STL un ARIMA modeļu validācijas perioda prognoze**

7.4.4. Theta modelis

Theta modelis arī viens no populārāk izmantotajiem statiskajiem modeļiem un šis modelis tiek definēts sekojoši:

$$\hat{X}_{T+h|T} = \frac{\theta - 1}{\theta}\hat{b}_0 \left[h - 1 + \frac{1}{\hat{\alpha}} - \frac{(1-\hat{\alpha})^T}{\hat{\alpha}} \right] + \tilde{X}_{T+h|T},$$

kur b_0 tiek novērtēts izmantojot regresiju ar mazāko kvadrātu metodi (OLS):

$$X_t = a_0 + b_0(t-1) + \epsilon_t$$

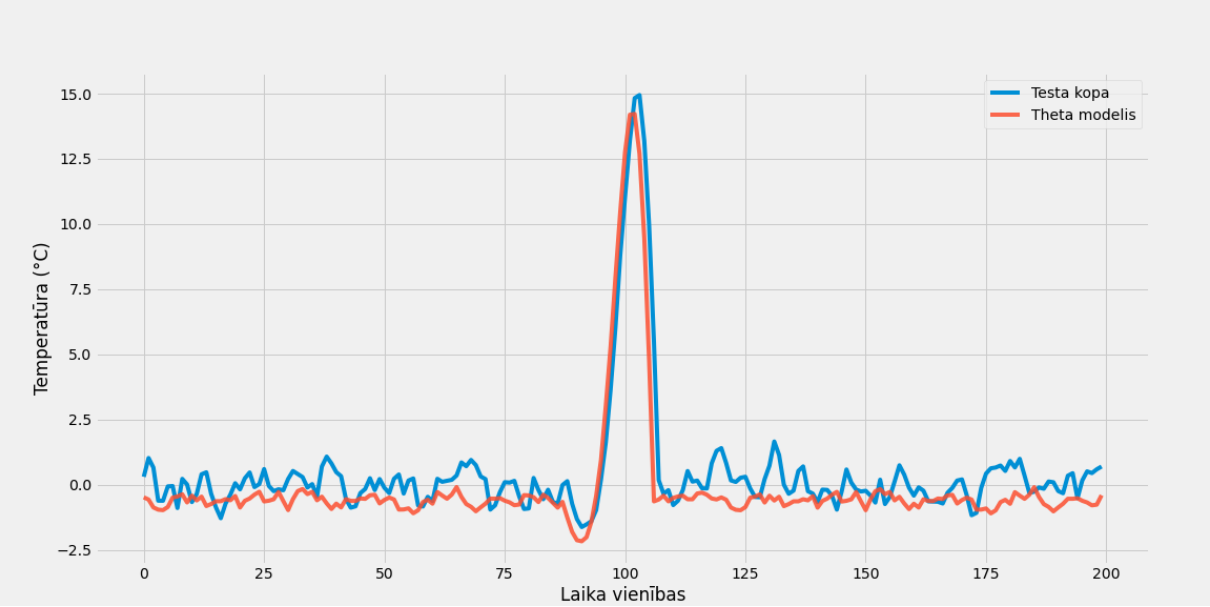
un α tiek novērtēts izmantojot vienkāršo eksponenciālo gludināšanas metodi:

$$\tilde{X}_t = (1-\alpha)X_t + \alpha\tilde{X}_{t-1}.$$

Savukārt θ nosaka to cik daudz tiks samazināta trenda ietekme, parasti θ pieņem vērtību 2, bet ja θ tiecas uz bezgalību, tad theta modelis vienāds ar IMA modeli (daļa no iepriekš apskatītā no ARIMA modeļa):

$$X_t = X_{t-1} + b_0 + (\alpha - 1)\epsilon_{t-1} + \epsilon_t.$$

Attēlā 37. var aplūkot theta modeļa prognozi un no attēla varam secināt, ka modelis nav tik precīzs kā iepriekš apskatītie modeļi un varam grafiski novērot, ka modelis pieļauj sistemātisku kļūdu, jo novērtētās vērtības ir zemākas kā sagaidāmās vērtības. Kā arī ir novērojama neliela laika nobīde, līdz ar to šis modelis ir viens no sliktākajiem modeļiem, rezultātus un salīdzinājumus var aplūkot 7. tabulā.



37. att. **Theta modeļa validācijas perioda prognoze**

7.5. Mašīnmācīšanās modeļi

Par mašīnmācīšanās modeļiem (ML) tiek uzskatīti modernie modeļi, kas sevī neietver statistiskos un dziļās apmācības modeļus. Datu kopas sagatavošana priekš mašīnmācīšanās algoritmiem atšķiras no iepriekš apskatīto statistisko modeļu datu kopas sagatavošanas. Šajā nodaļā apskatīsim tādus modeļus kā XGBoost, LightGBM, CatBoost un SVM.

7.5.1. Treniņa kopa izveide

Priekš ML modeļiem dati tiek sagatavoti savādāk kā tas tika darīts ar statiskajiem modeļiem. Ja statistiskajiem modeļiem ir nepieciešams nepārtraukts treniņa periods, kas tiek izmantot, lai trenētu modeli, tad ML modeļiem priekš katra laika momenta y_t prognozes ir nepieciešams izveidot novērojumu ar izvēlētiem mainīgajiem. Tabulā 6. var apskatīt transponētu sagatavotu datu tabulu ar pirmajiem 5 novērojumiem. Kā varam redzēt 6. tabula satur 13 neatkarīgos mainīgos un 1 atkarīgo mainīgo (y). Atkarīgo mainīgo izvēle un izveide ir modeļa izstrādātāja uzdevums un tie iedalās sekojošās kategorijās ([22]):

- Datuma raksturojošie mainīgie - mainīgie, kas raksturo izvēlēto novērojuma dienu vai datumu saistītu lielumu, piemēram, nedēļas dienu.
- Laika raksturojoši mainīgie - mainīgie, kas raksturo izvēlēto novērojuma laiku vai kādu citu ar laiku saistītu lielumu, piemēram, novērojuma dienas stundu.
- Lagu/Nobīdes laikā raksturojošie mainīgie - mainīgais, kas pieņem vērtību n laika vienību pagātnē, piemēram, vērtība iepriekšējā laika solī (lag 1).
- Slīdošā loga mainīgie - mainīgie, kas raksturo laikrindu no y_{t-n}, kur t ir loga garums, līdz y_t novērojumam, piemēram, pēdējo 10 novērojumu vidējā vērtība.
- Paplašinošā loga mainīgie - mainīgie, kas raksturo laikrindu sākot no laikrindas sākuma y_1 līdz y_t novērojumam, piemērām, vēsturisko pīķu skaits.
- Biznesa definētie mainīgie - šie ir mainīgie, kas tiek definēti izmantojot industrijas/biznesa informāciju, piemēram, akciju periodu raksturojošie mainīgie.

6. tabula

Mašīnmācīšanās modeļu datu kopa

Mainīgais	1. nov.	2. nov.	3. nov.	4. nov.	5. nov.
temp lag 0	0.10	0.01	-0.16	0.57	0.07
temp lag 1	0.45	0.10	0.01	-0.16	0.57
temp lag 2	0.73	0.45	0.10	0.01	-0.16
temp lag 3	0.55	0.73	0.45	0.10	0.01
temp lag 4	0.29	0.55	0.73	0.45	0.10
temp lag 60	11.30	13.29	14.73	14.97	13.34
temp lag 118	-0.07	0.06	-0.44	-0.05	0.69
temp lag 119	0.16	-0.07	0.06	-0.44	-0.05
temp lag 120	0.52	0.16	-0.07	0.06	-0.44
temp rolling 10 mean	0.30	0.33	0.29	0.31	0.29
temp rolling 10 min	-0.28	0.01	-0.16	-0.16	-0.16
temp rolling 10 max	0.73	0.73	0.73	0.73	0.73
temp rolling 10 median	0.31	0.31	0.31	0.31	0.28
y	0.01	-0.16	0.57	0.07	0.08

Nozīmīgi mainīgo izveidē ir izprast apskatāmo problēmu un laikrindu uzvedību (4.). Mainīgo izveide ir viena no būtiskākajām daļām ML modeļu izveidē priekš laikrindu prognozēšanas, tāpēc pēc modeļu izveides ir nepieciešamas izvērtēt izveidotos mainīgos un novērtēt modeļa jaudu, lai saprastu vai ir nepieciešama papildus mainīgo izveide. Būtiskākie mainīgie, kas būtu jāiekļauj ir laikā nobīdītās vērtības, kurām ir novērojama korelācija ACF un it īpaši PACF grafikos. Atcerēsimies, ka iepriekšējā izstrādes posmā 6. tika izveidota metode, kas sagrupē laikrindas klāsteros, tas var palīdzēt izvērtēt mainīgo izveidi visā laikrindām konkrēta klāstera ietvaros apskatots tikai pāris laikrindas no attiecīgā klāstera. Darbā apskatāmājam piemēram priekš temperatūras mērījumu prognozes tiek izveidoti sekojoši mainīgie (gatavu transponētu datu kopu pirmajiem 5 novērojumiem var apskatīt 6. tabulā):

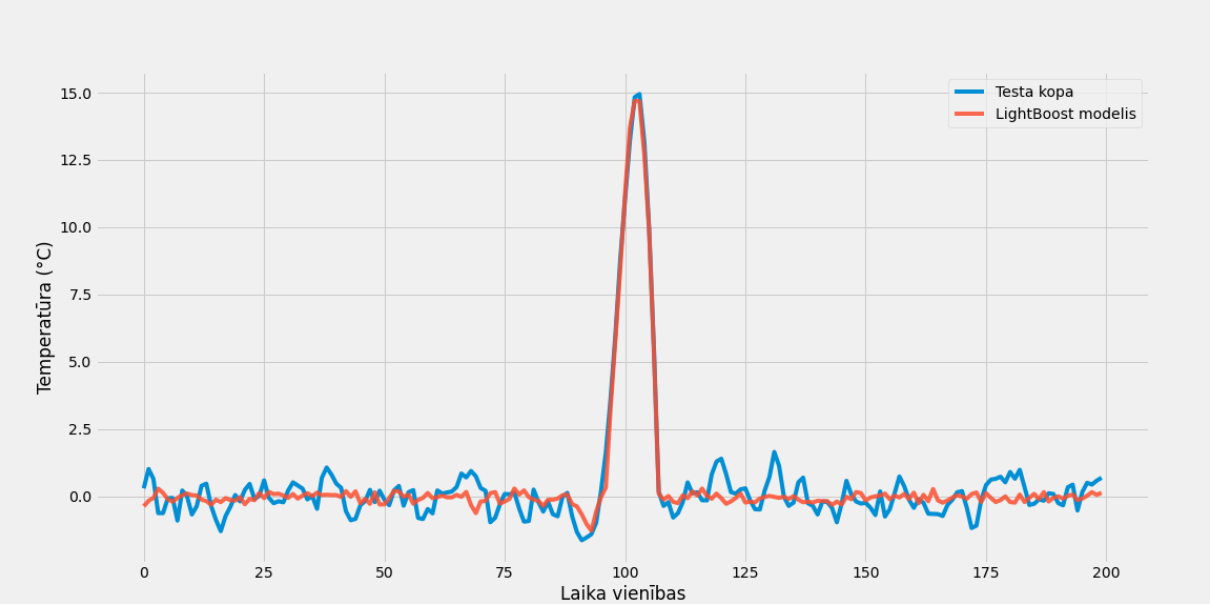
- temp lag 0, temp lag 1, temp lag 2, temp lag 3, temp lag 4, temp lag 60, temp lag 118, temp lag 119 un temp lag 120 - ir laikā nobīdīto vērtību mainīgie un skaitlis mainīgā nosaukumā norāda kāds ir novērojuma lags. Ir vērt piezīmēt, ka lag 0 ir attiecīgā novērojuma vērtība, bet skatoties uz prognozējamo vērtību, kas tiek apzīmēta ar y, tas ir uzskatāms kā lag 1 mainīgais.
- temp rolling 10 mean - pēdējo 10 novērojumu vidējās vērtības mainīgais
- temp rolling 10 min - pēdējo 10 novērojumu mazākās vērtības mainīgais
- temp rolling 10 max - pēdējo 10 novērojumu maksimālās vērtības mainīgais

- temp rolling 10 median - pēdējo 10 novērojumu mediānas vērtības mainīgais
- y - ir atkarīgais mainīgais, kas ir nākamais temperatūras mērījums, ko vēlamies prognozēt.

Atgādināsim, ka pieejamās treniņu datu kopas izmērs ir 800 novērojumi. Veidojot ML modeļa datu kopu mums ir nepieciešamas, lai būtu pieejamas/eksistētu visas mainīgo vērtības katram no treniņa datu kopā iekļautajiem novērojumiem. Tā kā tiek izmantoti mainīgie, kas izmanto vēsturisku informāciju, līdz pat 120 laika vienībām, tas nozīmē, ka novērojumi var tikt veidoti sākot no 121 temperatūras mērījuma. Atceroties, ka tiek skatīta arī nākamā novērojuma vērtība, kas tiek glabāta mainīgajā y, tas nozīmē, ka modelim pieejamā datu kopa ir izmērā 678 novērojumi ($800 - 121 - 1 = 678$).

7.5.2. LightGBM modelis

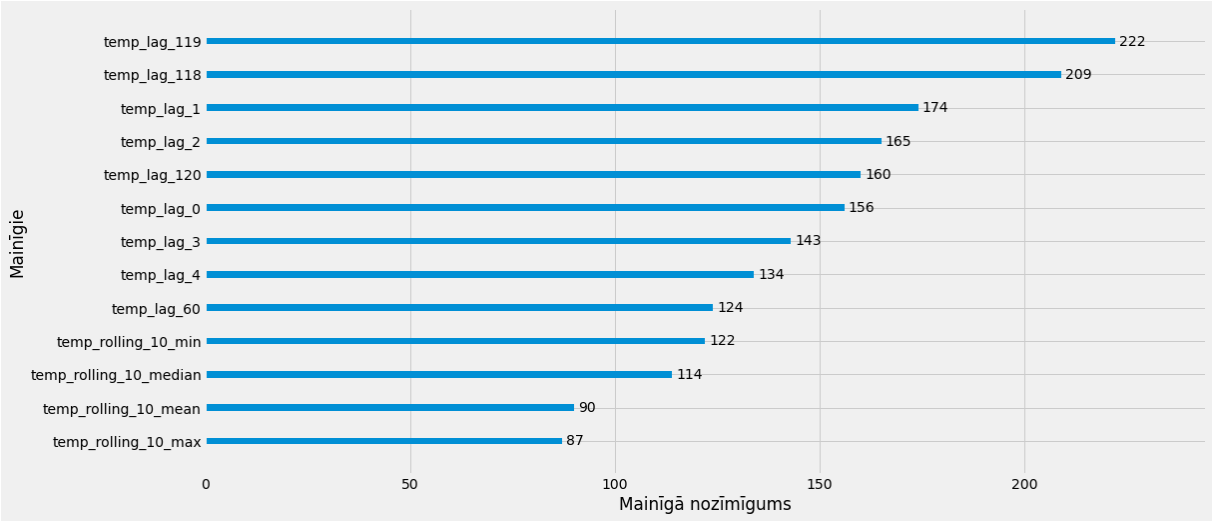
LightGBM ir gradientu pastiprināšanas algritma (GBT) paveids. [5] Prognozējot LightGBM tika izmantoti izmantota iepriekš definētā datu kopa ar 13 neatkarīgajiem mainīgajiem un vienu atkarīgo mainīgo 6., modeļa trenēšanas procesā tika izmantota krosvalidācija minimizējot vidējo kvadrātisko kļūdu. Attēlā 38. varam aplūkot LightGBM modeļa prognozi un tā, pēc grafiska novērtējuma, izskatās pietiekami laba.



38. att. **LightGBM modeļa validācijas perioda prognoze**

Papildus39. attēlā varam aplūkot visu mainīgo nozīmīgumu. Kā varam redzēt grafikā, visnozīmīgākais mainīgais ir vērtība 119 laika vienības atpakaļ, kas ir 120 laika vienības pirms prognozējamās vērtības, kas ir vienāda ar laikrindas perioda garumu. Interesanti, ka lag 0 mainīgā, kas ir iepriekšējā vērtība pirms prognozējamās vērtības ir tikai 6 nozīmīgākais mainīgais. Un varam novērot, ka slīdošo vērtību mainīgie ir visnenozīmīgākie mainīgie, kas var liecināt

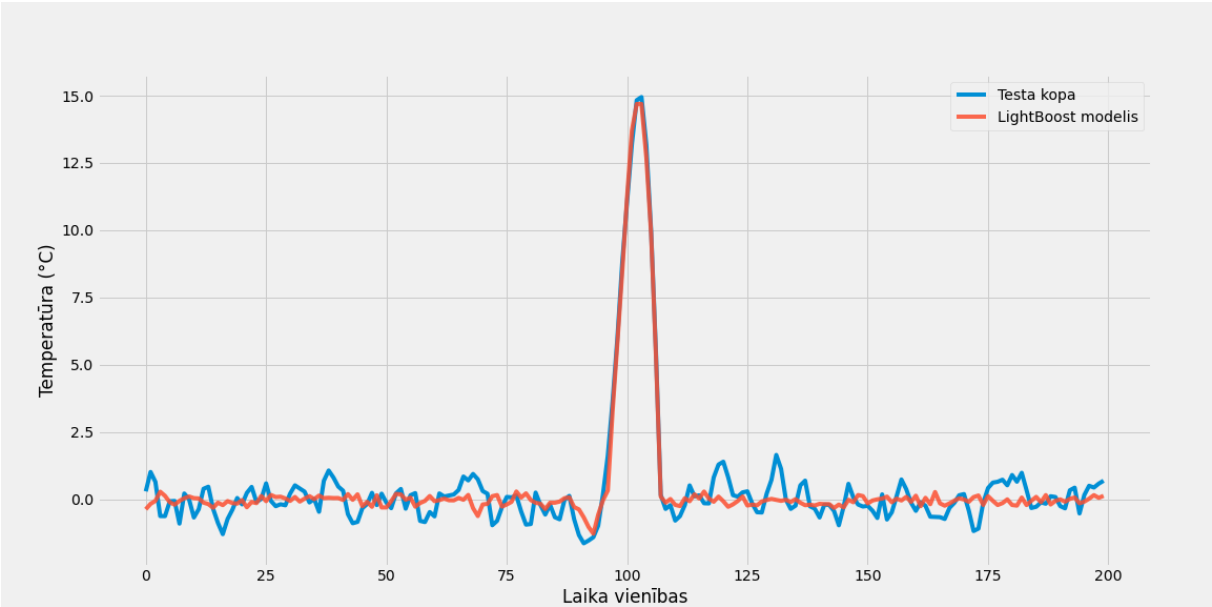
par to, ka šos mainīgos varētu izņemt no ML modaļu datu kopas vai aizvietot ar kādiem citiem mainīgajiem.



39. att. **LightGBM modeļa mainīgo nozīmīgums**

7.5.3. XGBoost modelis

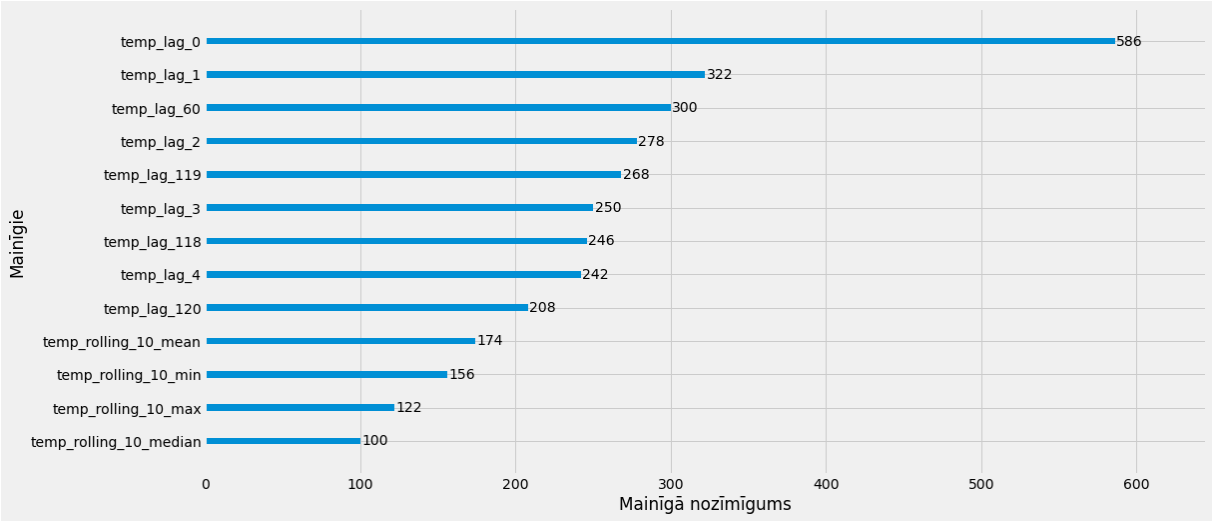
Arī XGBoost ir populārs GBT algoritma paveids. [23] Tāpat kā LightGBM, arī XGBoost tiek izmantota tā pati datu kopa. Attēlā 40. varam aplūkot modeļa prognozi salīdzinājumā ar validācijas kopu un varam redzēt, ka prognoze ir diezgan precīza.



40. att. **XGBoost modeļa validācijas perioda prognoze**

Arī XGBoost algoritmam ir iespējams novērtēt mainīgo nozīmīgumu. Grafikā 41. varam aplūkot XGBoost uztrenētā modeļa mainīgo nozīmīgumu un interesanti ir tas, ka XGBoost visnozīmīgākais ir tieši pēdējais novērojums (lag 0) un novērojums ar laika nobīdi 119, kas ir vien-

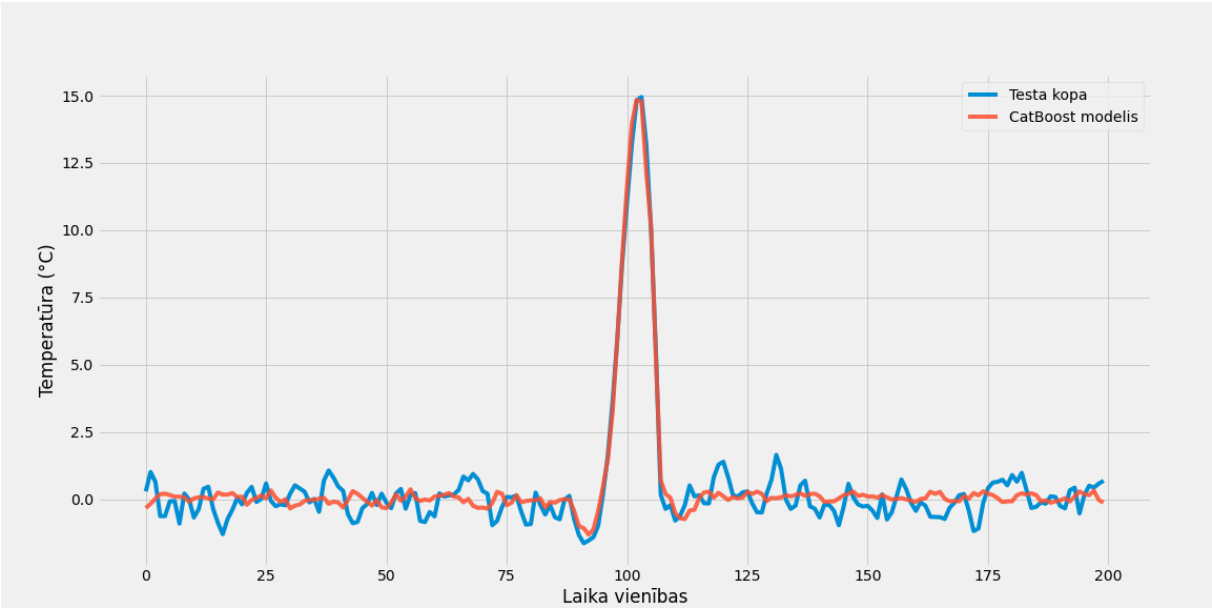
āds ar sezonas garumu ir tikai 5 nozīmīgākais mainīgais. Ja aplūkojam sliktākos prediktorus, tad arī šim algoritmam, tāpat kā LightGBM, varam novērot, ka visnenozīmīgākie mainīgie ir tieši slīdošo vidējo vērtību mainīgie, kas apstiprina iepriekš izvirzīto pieņēmumu, ka šos mainīgos varētu apsvērt izņemt no ML modaļa datu kopas.



41. att. **XGBoost modeļa mainīgo nozīmīgums**

7.5.4. CatBoost modelis

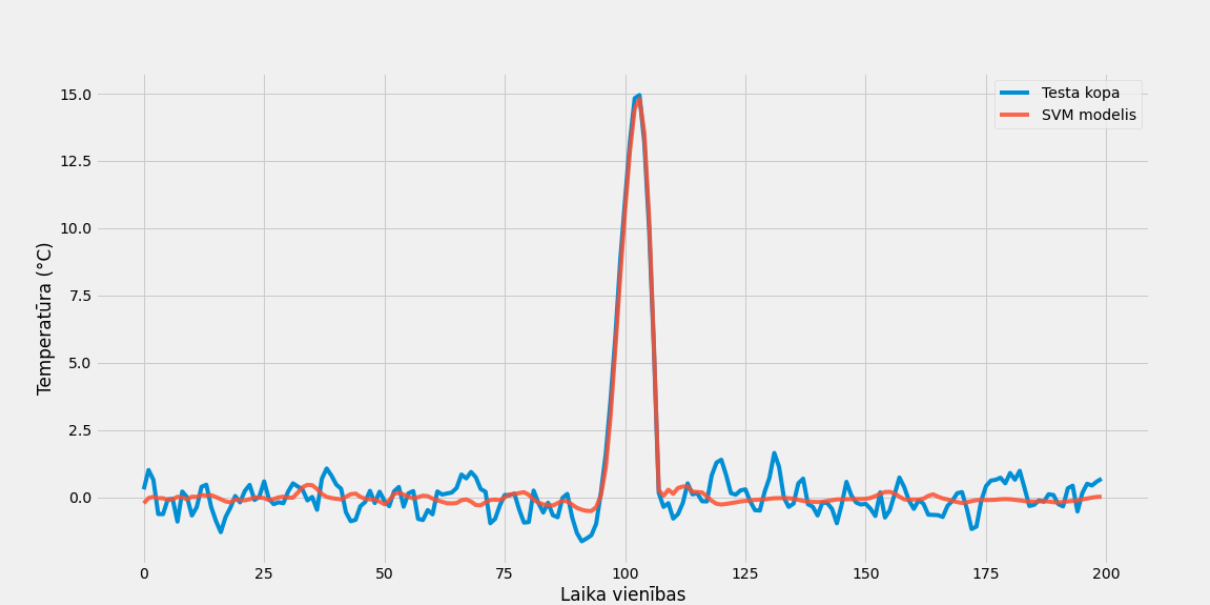
Tāpat kā iepriekšējās divas metodes arī CatBoost ir GBT algoritma paveids. [6] Lai gan orģināli šis algoritms ir domāts, lai izmantotu kategoriskiem mainīgajiem, tas diezgan labi veic prognozes izmantojot skaitliskus mainīgos. Attēlā 42. varam aplūkot modeļa prognozi un rezultāti ir līdzīgi kā iepriekšējiem algoritmiem.



42. att. **CatBoost modeļa validācijas perioda prognoze**

7.5.5. SVM modelis

Kā pēdējo modeli šajā apakšnodaļā apskatīsim atbalsta vektoru mašīnas (SVM) modeli. Modeļa prognozes rezultātus var aplūkot 43. attēlā. No attēla varam secināt, ka prognoze nav bijusi tik veiksmīga kā iepriekšējiem modeļiem, jo modelis nav prognozējis temperatūras kritumu atkausēšanas perioda sākumā.



43. att. **SVM modeļa validācijas perioda prognoze**

7.6. Dziļās apmācības modeļi

Šajā nodaļā tiks apskatītas dziļās apmācības metodes, kuras var tiks izmantotas laikrindu prognozēšanai. [18]

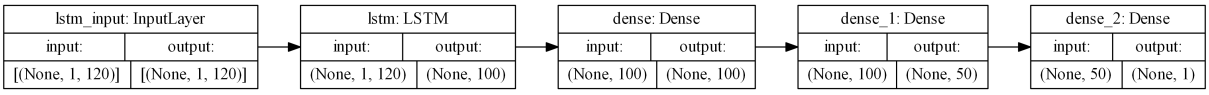
7.6.1. Treniņa kopas izveide

Priekš neironu tīklu modeļiem dati tiek sagatavoti līdzīgi kā tie tika sagatavoti priekš ML modeļiem - laikrindas treniņa un validācijas kopa tiek sadalīta novērojumos, kas satur atkarīgos un neatkarīgos mainīgos. Tā kā ir plānot izmantot neironu tīklus ar LSTM šūnām, kas ņem vērā mainīgo secību, līdz ar to t laika momenta novērojuma mainīgie sastāv no pēdējo 120 novērojumu mērījumiem un vienu nesakarīgo mainīgo, kas ir nākamā vērtība $\hat{y}_{t+1}$, ko vēlamies prognozēt laika momentā t.

7.6.2. LSTM modelis

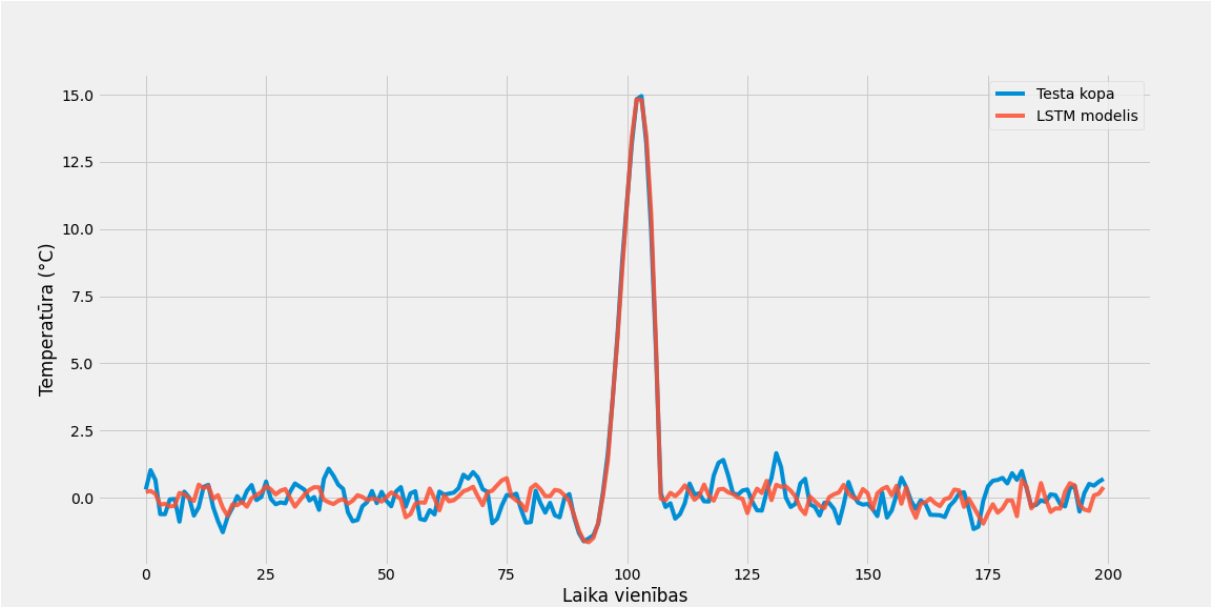
Temperatūras mērījumu datu modelēšanai tika atrasta piemērotākā dziļo neironu tīklu arhitektūra, kas sastāv no viena LSTM neironu slāņa un diviem regulāru neironu slāņiem, 44. var aplūkot modeļa arhitektūru. Modeļa arhitektūrā ir redzams ieejošā slāņa izmērs, kas ir vienāds ar (1,

120). LSTM neironu slānis sastāv no 100 LSTM neironiem un tam seko divi regulāru neironu slāņi izmērā 100 un 50, pēdējais slānis satur prognozēto vērtību $\hat{y}$ un tā kā regresijas prognoze tiek veikta vienam novērojumam, tad pēdējais/izejas slānis satur vienu neironu un ir vērts pieminēt, ka izejošā slāņa neironi nesatur aktivācijas funkcijas. Neironu tīklu modeļi var prognozēt vairākus laika soļus uz priekšu, bet tā kā visbūtiskāk ir saprast nākamā soļa vērtību tika izvēlēts, tāpat kā ar ML modeļiem, veikt prognozi vienam laika solim, bet pie nepieciešamības veikt atkārtotas iterācijas, lai iegūtu vēlamo skaitu soļu prognozes izmantojot iepriekšējā soļa novērtēto vērtību, t.i. lai prognozētu pirmā laika soļa vērtību $\hat{y}_{t+1}$ tiek izmantoti novērojumi $x_{t-119}, x_{t-118}, ...x_{t-1}, x_t$, savukārt prognozējot vērtību otrajā laika solī $\hat{y}_{t+2}$ tiek izmantoti sekojoši neatkarīgie mainīgie - $x_{t-118}, x_{t-117}, ...x_t, \hat{y}_{t+1}$, kur $\hat{y}_{t+1}$ ir iepriekšējā iterācijā novērtētā vērtība un attiecīgi analoģiski tiek turpināts, lai iegūtu nepieciešamo prognozēto laika soļu skaitu.



44. att. **Dziļās apmācības modeļa arhitektūra**

Attēlā 45. varam aplūkot LSTM modeļa prognozi un varam novērot, ka arī šis modelis diezgan labi prognozē temperatūras mērījumu.



45. att. **Dziļās apmācības modeļa arhitektūra**

7.6.3. Modeļu rezultātu salīdzinājums un secinājumi

Šajā nodaļā tiks aplūkoti un salīdzināti visu iepriekš minēto prognozējošo metožu un modeļu rezultāti. Rezultātu kopsavilkumu var aplūkot 7. tabulā. Tabula ir sakārtota secībā vispirms pēc R^2 novērtējuma, pēc tam pēc MAPE un kā trešā prioritāte ir MSE novērtējums. Divu naivo

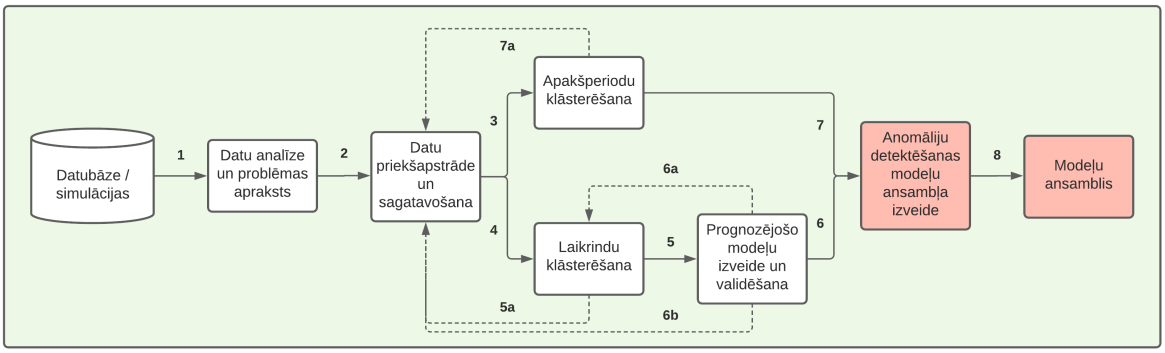
modeļu - vidējās vērtības prognozes un pēdējās vērtības prognozes, rezultāti ir vissliktākie, par ko varējām pārliecināties aplūkojot naivo modeļu prognožu grafiku 33.. Bet sezonālās vidējās vērtības metodes, kas ir kā naivais modelis, rezultāti ir pietiekami labi un var kalpot, lai novērtētu pārējo modeļu jaudu. Varam novērot, ka lielākajai daļai modeļu R^2 novērtējums svārstās no 0.94 līdz 0.96, tāpēc būtiskāk būtu pievērst uzmanību MAPE un MSE metriku novērtējumiem. Labie rādītāji ir izskaidrojami ar to, ka tika veikta izpēte un tika atstāti tikai tie modeļi, kas ir piemēroti tieši šāda tipa problemātikai. Kopumā novērtējot pēc visiem rādītājiem pirmo vietu ieņem Holt-Winters modelis un tam seko LightGBM modelis, kas ir labāks pēc MAPE, bet nedaudz iepaliek pēc MSE novērtējuma. Arī SARIMA modelis ir devis pietiekami labus rezultātus, bet tie ir daudz tuvāki naivā modeļa jaudai un papildus tā trenēšanas laiks aizņēma manāmi vairāk laika kā citiem apskatītajiem modeļiem/metodēm. ETS + ARIMA modeļa jauda ir līdzīga kā naivajai modelim. SVM, XGBoost, CatBoost un LSTM modeļi nedaudz iepaliek, bet šo modeļu jaudu varētu uzlabot papildus mainīgo izveide, kā arī šie modeļi, kopā ar LightGBM, varētu dod labākus rezultātus laikrindu prognozēšanai ar mainīgu sezonālu uzvedību. Ir vērts pieminēt, ka ML un DL tipa modeļus ir salīdzinoši vienkārši papildināt ar papildus mainīgajiem vai pārveidot par vairāku mainīgo problēmu, kur var tikt izmantoti gan dažāda cita tipa mainīgie, gan arī mērījumi no citiem sensoriem. Balsoties uz iegūtajiem rezultātiem prognozējošo algoritmu vidū ir modeļi ar lielāko jaudu - **Holt-Winters** un **LightGBM**. Attiecīgi arī šie modeļi tiks izmantoti anomāliju noteikšanā, papildus tiks iekļauts LSTM modelis, jo tāds bija sākotnējais darba mērķis un ņemot vērā, ka šis modelis nav bijis krietni sliktāks, tas var dot pozitīvu pienesumu anomāliju noteikšanā. Labāko modeļu trenēšanas un prognozes veikšanas laiks ir pietiekami īss, lai tos varētu izmantot reāllaika anomāliju noteikšanai. Papildus palīdz 6. nodaļā iegūtie rezultāti, kas ļauj grupēt laikrindas tādējādi atvieglot modeļu trenēšanu izmantojot iepriekš uztrenētus modeļus vai to parametrus kā sākuma stāvokļus.

7. tabula

Prognozējošo modeļu novērtējumi

Modelis	Modeļa tips	Izpildes laiks, sekundes	R^2	MAE	MedAE	MSE	MAPE
Holt-Winters	Stat	2.12	0.96	0.37	0.31	0.22	158.75
LightGBM	ML	3.13	0.95	0.45	0.36	0.32	113.75
SARIMA	Stat	33.76	0.95	0.44	0.38	0.31	169.50
Sezonālās vidējās vērtības prognoze	Naivais	0.01	0.95	0.44	0.36	0.31	195.67
ETS + ARIMA	Stat	1.25	0.95	0.45	0.34	0.33	213.73
LSTM	DL	77.65	0.95	0.46	0.40	0.31	580.15
SVM	ML	0.03	0.94	0.47	0.40	0.34	122.29
CatBoost	ML	12.82	0.94	0.47	0.43	0.34	155.01
XGBoost	ML	2.96	0.94	0.46	0.39	0.34	224.62
Theta	Stat	0.04	0.81	0.78	0.64	1.16	468.61
Vidējās vērtības prognoze	Naivais	0.00	-0.02	1.34	0.97	6.10	603.96
Pēdējās vērtības prognoze	Naivais	0.01	-0.23	1.24	0.72	7.32	493.00

8. ANOMĀLIJU NOTEIKŠANA



46. att. **Datu analīzes un modeļa izstrādes diagramma, laikrindu anomālijas noteikšanas modeļu ansambļa izveide**

Šajā nodaļā tiks apskatītas pēdējais modeļa izstrādes posms - anomāliju noteikšanas modeļu ansambļa izveide, kas ir attēlots 46. attēlā. Ar modeļu ansambli ir domāts dažādu modeļu kopums, kas ar balsošanas palīdzību spēj noteikt vai konkrētais novērojums ir anomāls. Anomāliju noteikšanai tiks izmantoti 3 dažāda tipa labākie prognozējošie modeļi - Holt-Winters, LSTM un LightGBM, kas tika izvērtēti 7. nodaļā. Lai noteiktu vai periods ir anomāls tiks izmantota laikrindu apakšperiodu klasterēšana, kas tika apskatīta 6. nodaļā. Anomāliju noteikšanas modeļu, modeļu ansambļa izveides un citu ar anomāliju noteikšanu saistīto darbību darba grāmatu var aplūkot B.6. pielikumā.

Anomāliju noteikšanas izmantojot izvēlētās metodes/modeļus tiek veikta izmantojot ticamības intervālu, kas Holt-Winter modeļa gadījumā tiek aprēķināts izmantojot 7. definēto *Brutlag* metodi un atlikušo divu modeļu- LightGBM un LSTM, gadījumā tiks izmantots prognozēto validācijas kopas vērtību un īsto validācijas kopas vērtību atlikumu sadalījums, rēķinoties, ka šiem atlikumiem būtu jābūt sadalītiem pēc normālā sadalījuma. Savukārt klasterēšanas metodei tiks izmantoti attālumi starp visiem klasterētajiem apakšperiodiem un klasteru centriem.

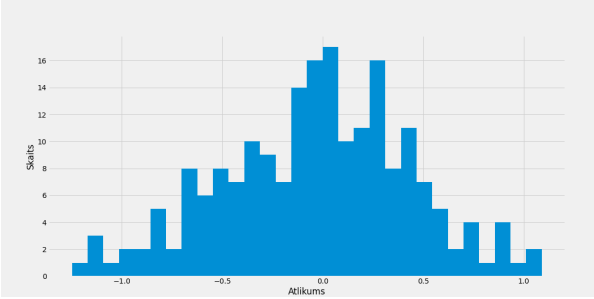
8.1. Atlikumu sadalījuma pārbaude

Lai gan ir sagaidāms, ka atlikumu sadalījums būs normāli sadalīts, ir nepieciešams veikt statistiku testu, lai pārliecinātos vai tik tiešām atlikumi ir sadalīti pēc normālā sadalījuma. Tam tiks izmantots Šapiro-Vilka tests.

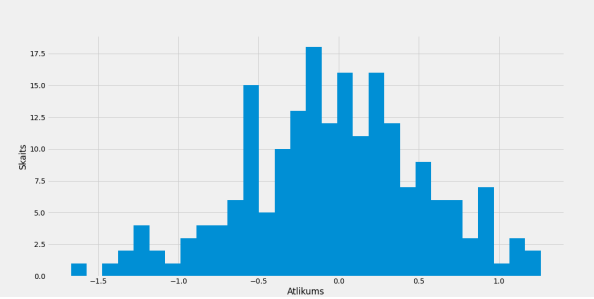
Šapiro-Vilka tests aprēķina W statistiku, kas pārbauda vai izlase $x_1, x_2, ..., x_n$ ir nāk no normālā sadalījuma, t.i. izlase ir normāli sadalīta. Maza W statistikas vērtība liecina par to ka izlase nenāk no normālā sadalījuma. W statistika tiek aprēķināta sekojoši:

$$W = \frac{\left(\sum_{i=1}^{n} a_i x_{(i)}\right)^2}{\sum_{i=1}^{n}(x_i - \bar{x})^2},$$

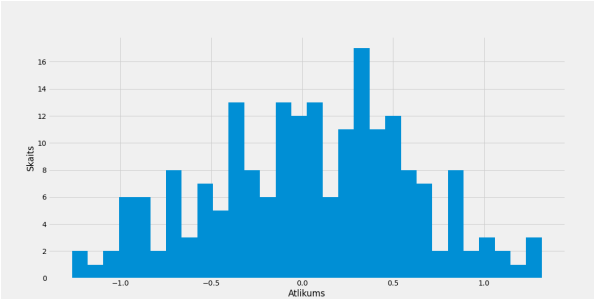
kur $x_{(i)}$ ir i-tā ranga x vērtība, jeb sakārtotas izlases (augošā secībā) i-tā vērtība un a_i ir konstante ģenerēta no izlases izmērā n, kas nāk no normālā sadalījuma, vidējā vērtības, dispersijas un kovariances. Un testa nulles hipotēze ir, ka izlase nāk no sadalījuma, kas ir normāli sadalīts. Ir vērts piezīmēt, ka izlasēm, kuru apjoms ir lielāks par 5000 novērojumiem, p-vērtību aprēķins var nebūt precīzs, bet W statistika tiek aprēķināta precīzi. Tāpēc lielām izlasēm var izmantot būtstrapa metodi, lai novērtētu p-vērtības.



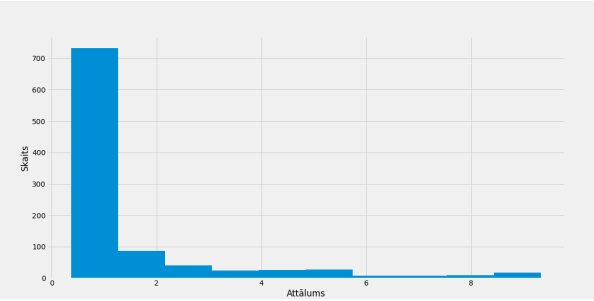
(a) **Holt-Winters modeļa atlikumu sadalījums**



(b) **LightGBM modeļa atlikumu sadalījums**



(c) **LSTM modeļa atlikumu sadalījums**



(d) **Apakšperiodu attālumi līdz klasteru centriem**

47. att. **Anomāliju noteikšanas metožu/modeļu mēri**

Apskatot visu trīs prognozējošo modeļu atlikumu sadalījumus 47a., 47b. un 47c., varam novērot, ka atlikumi varētu būt normāli sadalīti, tāpēc tiek veikts Šapiro-Vilka tests, aprēķināta vidējā vērtība, standartnovirze, pīķainības koeficients un nobīdes koeficients. Sadalījumu analīzes rezultātus var aplūkot 8. tabulā, no tabulas varam secināt, ka visiem modeļiem atlikumu vidējā vērtība ir tuvu nullei un balstoties uz Šapiro-Vilka testu nulles hipotēze netiek noraidīta pie nozīmības līmeņa $\alpha = 0.1$, kas nozīmē, ka varam pieņemt, ka visu modeļu prognožu atlikumi ir sadalīti pēc normālā sadalījuma.

Apūkojot 47d. varam redzēt sadalījumu klasterēto apakšperiodu attālumiem (izmantojot DTW attālumu) līdz klasteru centriem. Kā redzmas vērtības ir pozitīvas un šajā gadījumā ir jāizvēlas vērtība, kas kalpos kā robeža pie kuras tiks uzskatīts, ka apakšperiods nepieder nevienam no klasteriem un ir uzskatāms par apakšperiodu, kas satur anomālijas. Maksimālā attāluma vērtība starp klasterētajiem apakšperiodiem ir 9.3, tāpēc par apakšperiodiem, kas satur anomālijas tiek izvēlēts uzskatīt apakšperiodus, kuru attālums līdz tuvākajam klastera centram ir lielāks par 10.

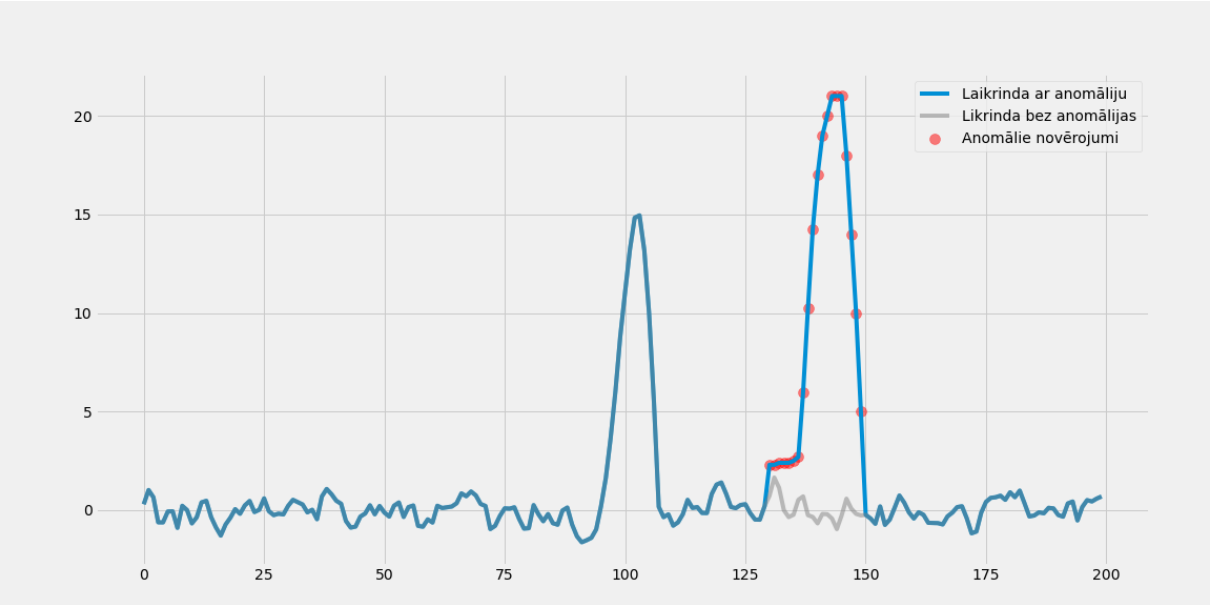
8. tabula

Prognozējošo modeļu atlikumu analīze

Modelis	W statistika	p-vērtības	Nobīde	Pīķainība	Vidējā vērtība	Standartnovirze
LSTM	0.9906	0.22	-0.11	-0.52	0.05	0.56
LightGBM	0.9940	0.60	-0.19	-0.08	-0.03	0.56
Holt-Winters	0.9935	0.53	-0.16	-0.25	-0.03	0.47

8.2. Anomāliju ievietošana simulētajos datos

Lai pārbaudītu modeļu anomāliju noteikšanas spēju tiek ievietota anomālija 1. grupas laikrindā, kas tika izmantota visu modeļu trenēšanā. Attēlā 48. ar zilu krāsu var aplūkot jaunizveidoto laikrindu, kas satur anomāliju, ar sarkaniem punktiem tiek atzīmēti pievienotie anomālie novērojumi un ar pelēku krāsu tiek attēlots periods, kas tika aizvietots ar anomāliem novērojumiem.



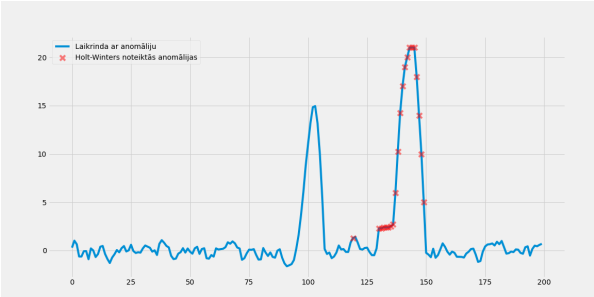
48. att. **Laikrinda ar pievienotām anomālijām**

8.3. Anomāliju noteikšana

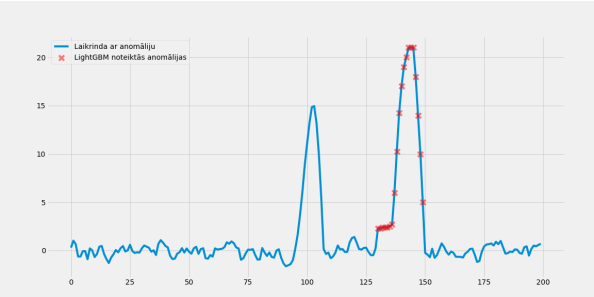
Izmantojot iepriekš uztrenētos modeļus un saklasterētos apakšperiodus tiek pārbaudīti visi novērojumi. Gadījumā kad modelis ir novērtējis novērojumu kā anomāliju, tā grafikā (49.) tiek iezīmēta ar sarkanu krustu. Kā varam novērot attēlos 49a., 49b. un 49c. prognozējošie modeļi ir identificējuši visas anomālijas, vienīgi Holt-Winters modelis ir identificējis vienu papildus novērojumu (aptuveni 120. laika momentā), kas atradies ārpus ticamības intervāla. Klasterēšanas pieeja arī ir veiksmīgi atradusi anomālijas, lai gan šī metode nav identificējusi visas anomālijas to var pielāgot mainot apakšperiodu garumu, kā arī tā var identificēt cita tipa anomālijas, kā

piemēram, ilglaicīgu un salīdzinoši mazu temperatūras izmaiņu. Papildus ir vērts piebilst ka par anomāliju tiek uzskatīti novērojumu, kas apmierina vienu no sekojošiem nosacījumiem:

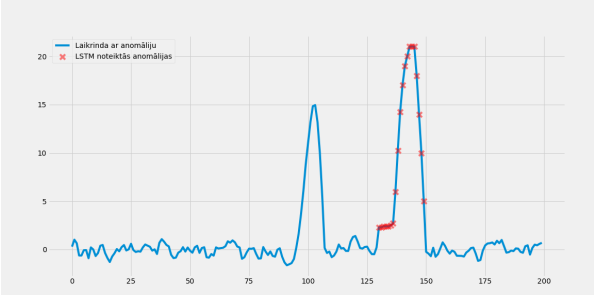
- Ir noteikti n secīgi anomāli novērojumi, kur n ir definēts lielums, kas novērš retas nejaušas svārstības, kas neliecina par problēmām ar iekārtu kurā atrodas sensors.
- Ir noteikti n anomāli novērojumi periodā $[a,b]$, kur n, a un b ir speciālista definēti lielumi atkarībā no sensora un iekārtas kurā atrodas attiecīgais sensors.



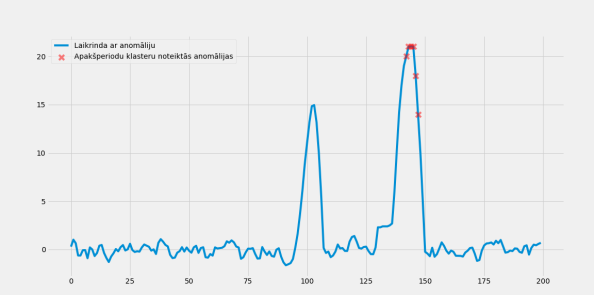
(a) **Holt-Winters modeļa noteiktās anomālijas**



(b) **LightGBM modeļa noteiktās anomālijas**



(c) **LSTM modeļa noteiktās anomālijas**

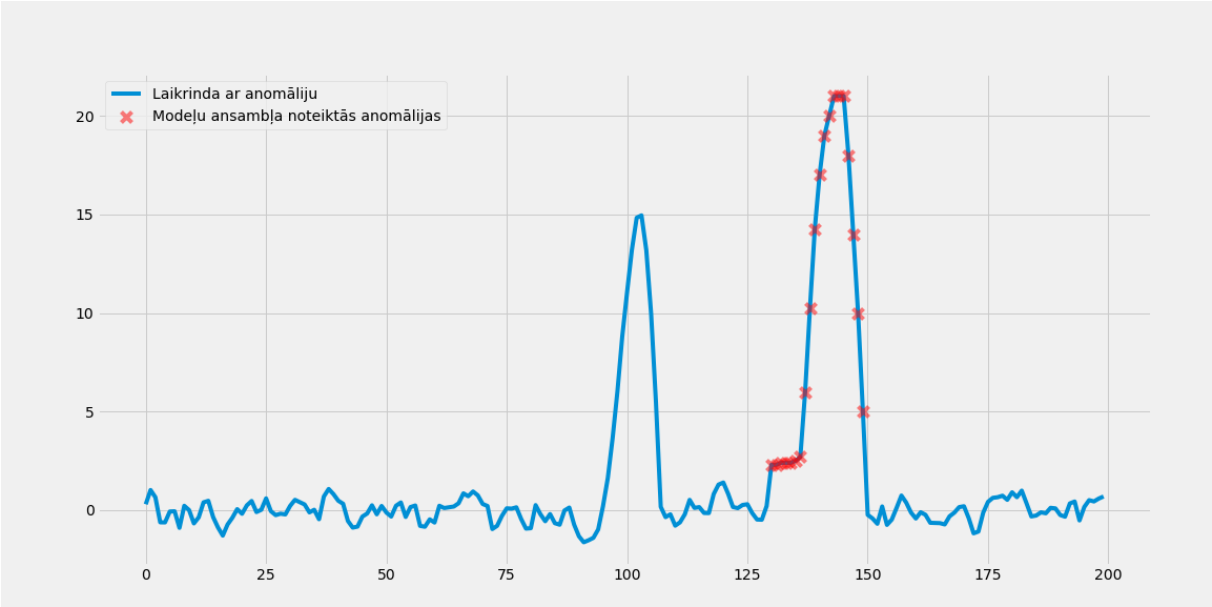


(d) **Klasterēšanas metodes noteiktās anomālijas**

49. att. **Metožu/modeļu noteiktās anomālijas**

8.4. Anomāliju noteikšanas modeļu ansambļa izveide

Modeļu ansamblis anomāliju noteikšanai tiek veidots no visiem izvēlētajiem modeļiem, kur katrs modelis ar savu prognozi piedalās kā balsotājs par to vai attiecīgais novērojums ir vai nav anomālija. Modaļu balsojumam var atšķirties konkrēta modeļa balss nozīmīgums, bet darbā apskatītajā piemērā visu modeļu balsis ir vienlīdzīgas. Attēlā 50. var aplūkot modeļa ansambļa noteiktās anomālijas un kā varam redzēt, tad Holt-Winters identificētā anomālija, kas nebija anomālija netiek uzskatīta kā anomālija, jo neviens no pārējiem modeļiem šo novērojumu nav identificējis kā anomāliju. Modeļu ansamblis dod robustākus rezultātus, jo tiek ņemta vērā vairākuma balsis, kā arī, ir iespējas ieviest papildus risinājumus, ka tiek atsevišķi pētīti novērojumi, kurus kaut viens modelis ir identificējis kā anomālijas.



50. att. **Modeļu ansambļa noteiktās anomālijas**

9. REZULTĀTI

Darba tika veiksmīgi izpētīti un izveidoti modeļi priekš reāllaika temperatūras mērījumu sensoru anomāliju noteikšanas. Izvērtējot modeļus tika ņemts vērā modeļu trenēšanas un prognožu veikšanas laiks un tika izvēlēti optimālākie modeļi. Papildus uzstādītajam uzdevumam, izmantot pamatā LSTM tipa modeļus, tika atrastas alternatīvas metodes/modeļi, kas tiek iekļautas modeļu ansamblī veidojot gala modeli robustāku. Tika izstrādāts un aprakstīti visi prototipa izstrādes soļi - datu ieguve, datu analīze, datu priekšapstrāde un sagatavošana, klasterizācija, prognozējošo modeļu izveide, anomāliju noteikšana un modeļu ansambļa izveide. Papildus tika izveidota simulāciju process, kas var palīdzēt turpmākai izpētei. Un izstrādātā modeļa prototips ir spējīgs veikt prognozes reālā laika.

Veiskmīgi tika izveidoti modeļi/metodes, kas risina darba uzdevumā definēto problēmu un to jauda bija lielāka par naivajiem modeļiem. Galvenie modeļi un metodes, kas tika izmantoti gala modelī ir sekojoši:

1. Pilna laikrindu klasterēšana - izmantojot DTW attālumu un k-vidējo klasterēšanas metodi. Šī metode palīdz grupēt laikrindas, lai pārējo modeļu realizācija būtu mērogojama.

2. Laikrindu apakšperiodu klasterizācija - izmantojot DTW attālulumu un k-vidējo klasterēšanas metodi. Šī metode palīdz noteikt tipiskos laikrindu apakšperiodus, kas kalpo kā anomālu periodu noteikšanas rīks.

3. LightGBM, LSTM un Holt-Winters prognozējošo modeļi. Šie modeļi veic anomāliju noteikšanu un sastāda gala modeļu ansambli.

4. Modeļu ansamblis - iepriekš minēto modeļu kopums, kas veic anomāliju noteikšanu izmantojot balsošanas principu.

10. UZLABOJUMI UN TĀLĀKA IZPĒTE

Šajā nodaļā tiks uzskaitītas lietas, kas izpētes un izstrādes procesā tika pamanītas un piezīmētas kā potenciāli uzlabojumi vai turpmākas izpētes materiāls. Lai cik labs būtu modelis, vienmēr to var uzlabot un papildināt, kā arī attīstoties jaunām metodēm ir būtiski aplūkot ar vien jaunas metodes un pārbaudīt ar vien jaunas idejas.

Lai potenciāli uzlabotu darbā iegūtos rezultātus var veikt sekojošas darbības:

- Apskatīt papildus metodes no jaunākajām laikrindu sacensībām (aplūkotas 1. nodaļā), ar domu atrast papildus modeļus, kuru jauda ir lielāka par naivajiem modeļiem vai uzlabot esošos modeļus.
- Apskatīt citas arhitektūras dziļās apmācības modeļus.
- Aplūkot papildus klasterēšanas metodes.
- Novērtēt cita tipa anomālija noteikšanas metodes - autoenkoderu modeļus un laikrindu matricu profilus.
- Visu modeļu trenēšanā izmantot krosvalidāciju.
- Veikt smalkāku izvēlēto modeļu hiperparametru noteikšanu.
- Notestēt modeļus uz laikrindām ar neregulāru sezonalitāti.

Izpētes gaitā, papildus darbā apskatītajām metodēm, tika atrastas sekojošas metodes/idejas, kas varētu kalpot par pamatu turpmākai izpētei, lai uzlabotu anomāliju noteikšanas gala modeli:

- Mašīnmācīšanās modeļiem izpētīt iespēju izmantot Šepli vērtības, lai noteikti mainīgo nozīmīgumu.
- Izpētīt ARNN (autoregresīvo neironu tīklu) modeli. Šis ir neironu tīklu un ARMA modeļu hibrīdmodelis, kas atšķiras no ARMA ar to, ka šim modelim nav nepieciešams stacionaritātes nosacījums.
- Atrast mērogojamu risinājumu, kas spētu automātiski ar labu precizitāti noteikt precīzu laikrindas sezonālā perioda garumu.
- Izspētīt metodes kā automātiski apstrādāt treniņa datus, lai no tiem izslēgtu anomālijas (pašlaik tas tiek darīts manuāli analīzes veikšanas procesā).
- Izpētīt slēpto markova modeļu (HMM) izmantošana laikrindu analīzē.
- Apskatīt laikrindu dekompozīcija izmantojot Furjē transformācijas.
- Izpētīt tādu modeļus kā - GMM, AAE, VAE un DeepAnT.

- Izpētīt iespēju lietot maiņas punktu analīzi anomāliju noteikšanā. Maiņas punktu analīze varētu būt nozīmīga ilgtermiņa izmaiņu noteikšanā.

11. NOBEIGUMS

Darbā aprakstītais projekta pamata darba uzdevums - izstrādāt anomāliju noteikšanas prototipu temperatūras mērījumu sensoriem, tika veiksmīgi izpildīts un realizēts. Tādējādi tika konceptuāli parādot kā realizēt anomāliju noteikšanu sensoru datiem. Lai gan galvenais mērķis bija apskatīt dziļās apmācības metodes, lai salīdzinātu un iespējams iegūtu piemērotāku modeli, darba uzdevums tika paplašināts iekļaujot klasterēšanas, statistikas un mašīnmācīšanās modeļus/metodes. Papildus darbs apraksta strukturētu modeļa izstrādes procesu, kas ļauj ērti identificēt potenciālos uzlabojumu un tos efektīvi ieviest nākamo modeļu versiju izstrādes procesā.

Papildus izstrādātajam modelim nozīmīgs ieguvums ir dažādo modeļu aplūkošana un salīdzināšana, kas dod ieskatu turpmāko modeļu potenciālajā attīstībā. Un galvenā darba atziņa ir, ka galvenais ir sākt ar vienkāršu vai jebkādu pazīstamu metodi vai modeli un pēc tam strādāt pie tā, lai izvērtētu kuras metodes vai modeļi var uzlabot sākotnēji iegūtos rezultātus, ņemot vērā gan precizitātes, gan arī modeļa interpretējamību un sarežģītību. Dažkārt arī pavisam vienkārši modeļi var dot vēlamo rezultātu. Un vienmēr ir vērts paturēt prātā statistiķa Džordža Koksa teicienu - **"Visi modeļi ir nepareizi, bet daži ir noderīgi."** ("All models are wrong, but some are useful.").

IZMANTOTĀ LITERATŪRA UN AVOTI

[1] Saeed Aghabozorgi, Ali Seyed Shirkhorshidi, and Teh Ying Wah. Time-series clustering – a decade review. *Information Systems*, 53:16–38, 2015.

[2] Peter J Brockwell and Richard A Davis. *Time Series: Theory and Methods 2nd Edition*. Springer-Verlag, Berlin, Heidelberg, 1991.

[3] J. Brownlee. *Introduction to Time Series Forecasting With Python: How to Prepare Data and Develop Models to Predict the Future*. Machine Learning Mastery, 2017.

[4] Jason Brownlee. *Introduction to Time Series Forecasting With Python*. MACHINE LEARNING MASTERY, 2020.

[5] Microsoft Corporation. Lightgbm's documentation. https://lightgbm.readthedocs.io/en/latest/, 2021. [Accessed on 2021-06-18].

[6] CatBoost developers. Catboost documentation. https://catboost.ai/docs, 2021. [Accessed on 2021-06-18].

[7] G. Hyndman, R.J. & Athanasopoulos. *Forecasting: principles and practice, 3rd edition*. OTexts: Melbourne, Australia. OTexts.com/fpp3, 2021. [Accessed on 2021-06-02].

[8] Ali Javed, Byung Suk Lee, and Donna M. Rizzo. A benchmark study on time series clustering. *Machine Learning with Applications*, 1:100001, 2020.

[9] Yury Kashnitsky. Time series analysis in python. https://www.kaggle.com/kashnitsky/topic-9-part-1-time-series-analysis-in-python, 2021. [Accessed on 2021-06-15].

[10] Spyros Makridakis, A. Andersen, R. Carbone, Robert Fildes, Michele Hibon, R. Lewandowski, Howard Newton, Emanuel Parzen, and R. Winkler. The accuracy of extrapolation (time series) methods: Results of a forecasting competition. *Journal of Forecasting*, 1:111 – 153, 04 1982.

[11] Spyros Makridakis, Chris Chatfield, Michèle Hibon, Michael Lawrence, Terence Mills, Keith Ord, and LeRoy F. Simmons. The m2-competition: A real-time judgmentally based forecasting study. *International Journal of Forecasting*, 9(1):5–22, 1993.

[12] Spyros Makridakis and Michele Hibon. The m3-competition: Results, conclusions and implications. *International Journal of Forecasting*, 16:451–476, 10 2000.

[13] Spyros Makridakis, Evangelos Spiliotis, and Vassilios Assimakopoulos. The m4 competition: 100,000 time series and 61 forecasting methods. *International Journal of Forecasting*, 36(1):54–74, 2020. M4 Competition.

[14] Spyros Makridakis, Evangelos Spiliotis, and Vassilis Assimakopoulos. The m5 accuracy competition: Results, findings and conclusions. 10 2020.

[15] Makridakis Competitions. Makridakis competitions — Wikipedia, the free encyclopedia, 2010. [Online; last edited on 18 December 2020].

[16] François Petitjean, Alain Ketterlin, and Pierre Gançarski. A global averaging method for dynamic time warping, with applications to clustering. *Pattern Recognition*, 44(3):678–693, 2011.

[17] Selva Prabhakaran. Time series analysis in python – a comprehensive guide with examples. https://www.machinelearningplus.com/time-series/time-series-analysis-python/, 2018. [Accessed on 2021-06-05].

[18] B. Ripley. *Pattern Recognition and Neural Networks*. Cambridge University Press, 1996.

[19] ritvikmath. Time series analysis. `https://github.com/ritvikmath/Time-Series-Analysis`, 2020. [Accessed on 2021-06-02].

[20] Patrick Schäfer. *Scalable time series similarity search for data analytics*. PhD thesis, 10 2015.

[21] Skipper Seabold and Josef Perktold. statsmodels: Econometric and statistical modeling with python. In *9th Python in Science Conference*, 2010.

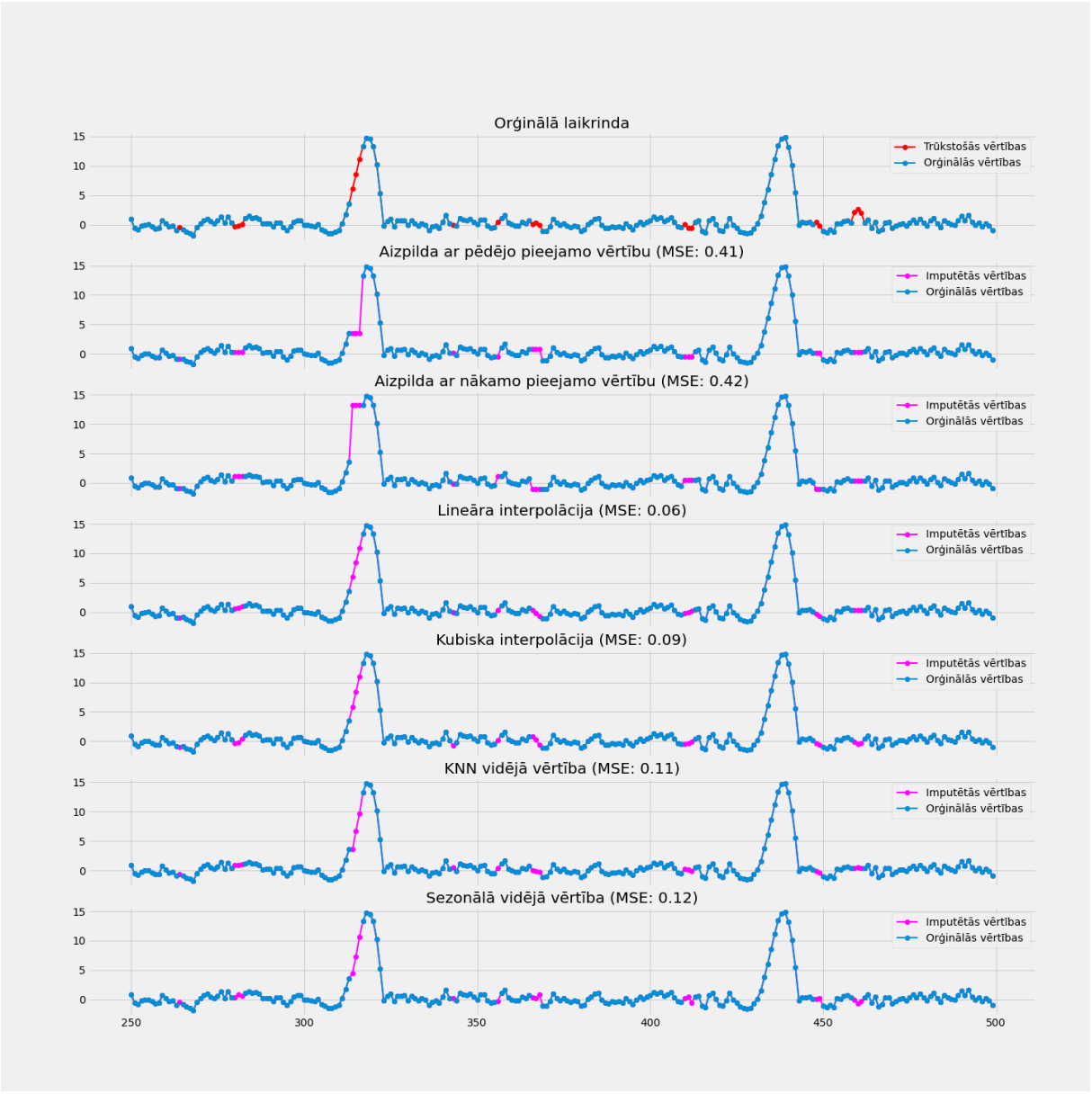
[22] Aishwarya Singh. 6 powerful feature engineering techniques for time series data (using python). https://www.analyticsvidhya.com/blog/2019/12/6-powerful-feature-engineering-techniques-time-series/, 2019. [Accessed on 2021-06-18].

[23] xgboost developers. Xgboost documentation. https://xgboost.readthedocs.io/en/latest/index.html, 2020. [Accessed on 2021-06-18].

PIELIKUMI

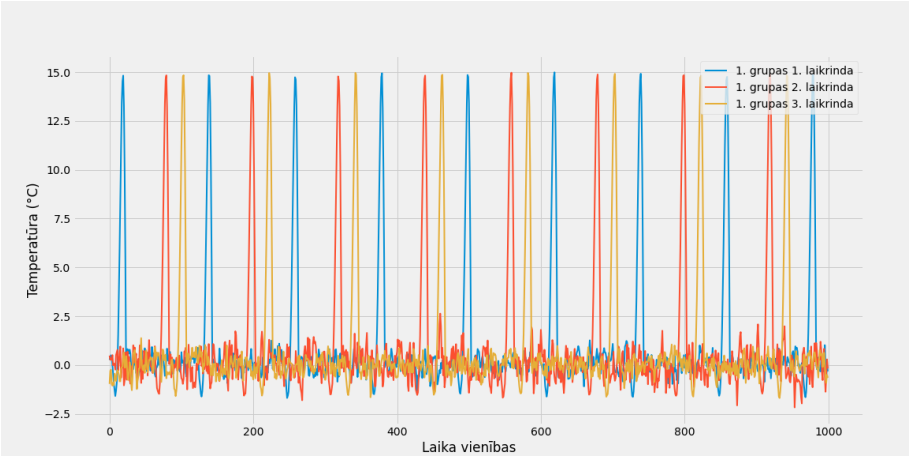
A. GRAFIKI

A.1. Imputācijas metožu salīdzinājuma grafiks

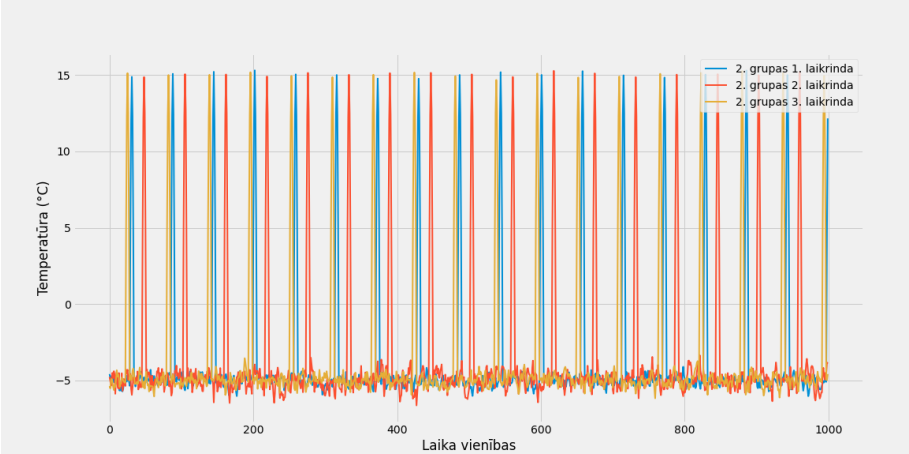


51. att. **Dažādu imputācijas metožu salīdzinājuma grafiks**

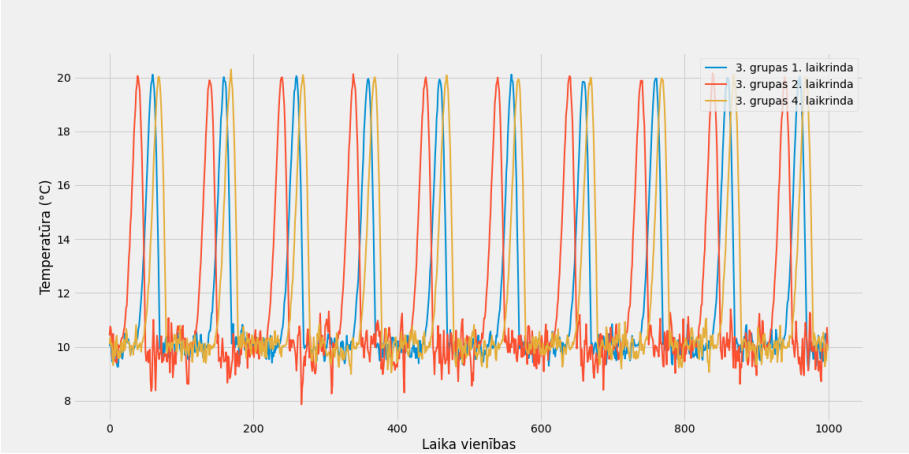
A.2. Laikrindu salīdzinājums grupu ietvaros



(a) **1. grupas 3 laikrindu salīdzinājums**



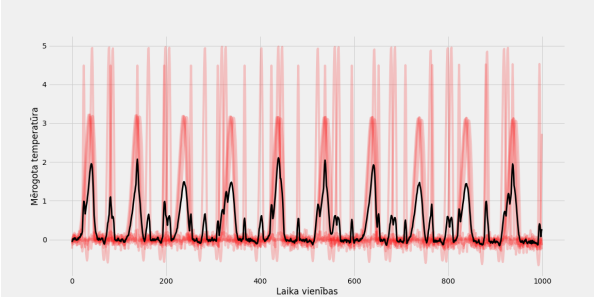
(b) **2. grupas 3 laikrindu salīdzinājums**



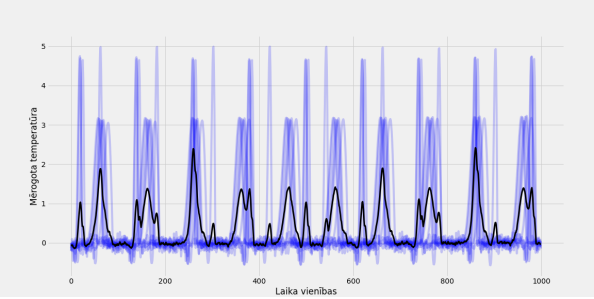
(c) **3. grupas 3 laikrindu salīdzinājums**

52. att. **Temperatūras mērījumu laikrindu salīdzinājums katrai no trīs grupām**

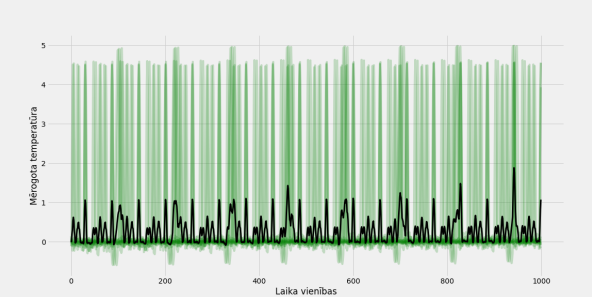
A.3. Dažādu klāsterēšanas metožu rezultāti



(a) **1. klāsteris**

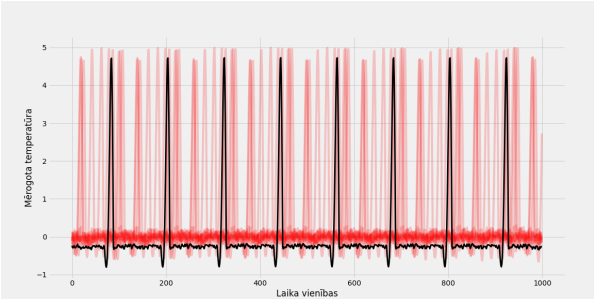


(b) **2. klāsteris**

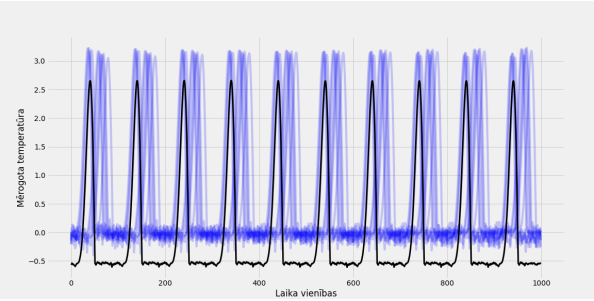


(c) **3. klāsteris**

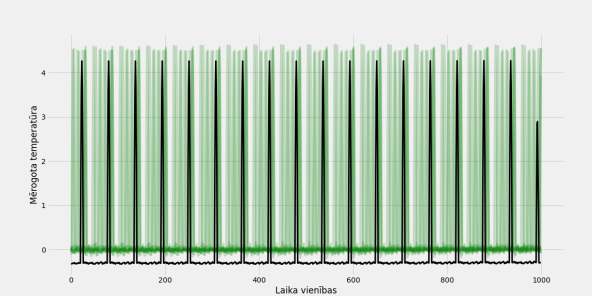
53. att. **K-vidējo klāsterēšanas metodes (izmantojot Eiklīda attāluma mēru) izveidoto klāsteru un to centroīdu attēlojums**



(a) **1. klāsteris**



(b) **2. klāsteris**



(c) **3. klāsteris**

54. att. **Uz formu balstītas klāsterēšanas metodes izveidoto klāsteru un to centroīdu attēlojums**

B. DARBA GRĀMATAS

B.1. Simulāciju darba grāmata

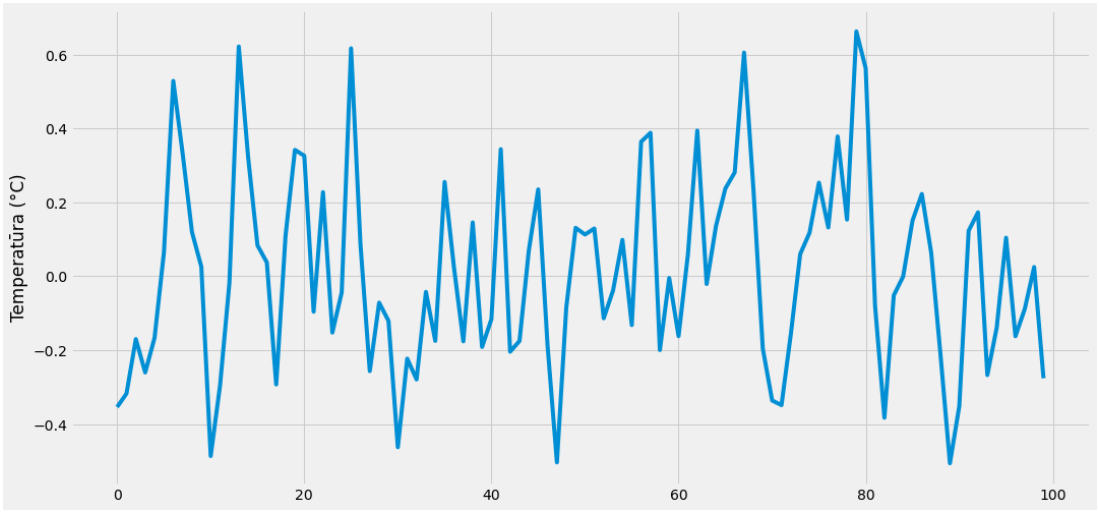
1 Temperatūras mērījumu laikrindu simulācijas

[1]:
```
# Pakotņu ielāde un iestatījumi
import numpy as np
import pandas as pd
import random
import math
import matplotlib.pyplot as plt

%matplotlib inline
import statsmodels.api as sm
from statsmodels.tsa.arima_process import arma_generate_sample

plt.style.use('fivethirtyeight')
plt.rcParams["figure.figsize"] = (16,8)

np.random.seed(42)
```

1.0.1 Normālas darbības perioda simulācija

[2]:
```
# Piemērs no https://www.elenacuoco.com/2016/07/31/simulating-time-series/ &
 ↪https://www.statsmodels.org/stable/generated/statsmodels.tsa.arima_process.
 ↪ArmaProcess.html
# Normālas darbības periods tiek simulēts izmantojot ARMA(1,1) procesu!
ar_coef = np.r_[1, -.1] # Koeficienti AR polinomā (iekļaujut t0, jeb lag 0)!
ma_coef = np.r_[1, .5] # Koeficienti MA polinomā (iekļaujut t0, jeb lag 0)!

# Definējam ARMA procesu!
arma_process = sm.tsa.ArmaProcess(ar_coef, ma_coef)
print(f'Vai process ir stacionārs:', arma_process.isstationary)

n_arma = 100
y_arma = arma_process.generate_sample(n_arma, scale=.25)
y_arma = arma_generate_sample(ar_coef, ma_coef, n_arma, scale=.25)

# Grafiks + saglabā grafiku!
fig, ax = plt.subplots()
ax.plot(range(n_arma), y_arma);
ax.set(ylabel=u'Temperatūra (\u00B0C)')

plt.savefig('images/simulacijas/ARMA_process.png')
plt.show();
```

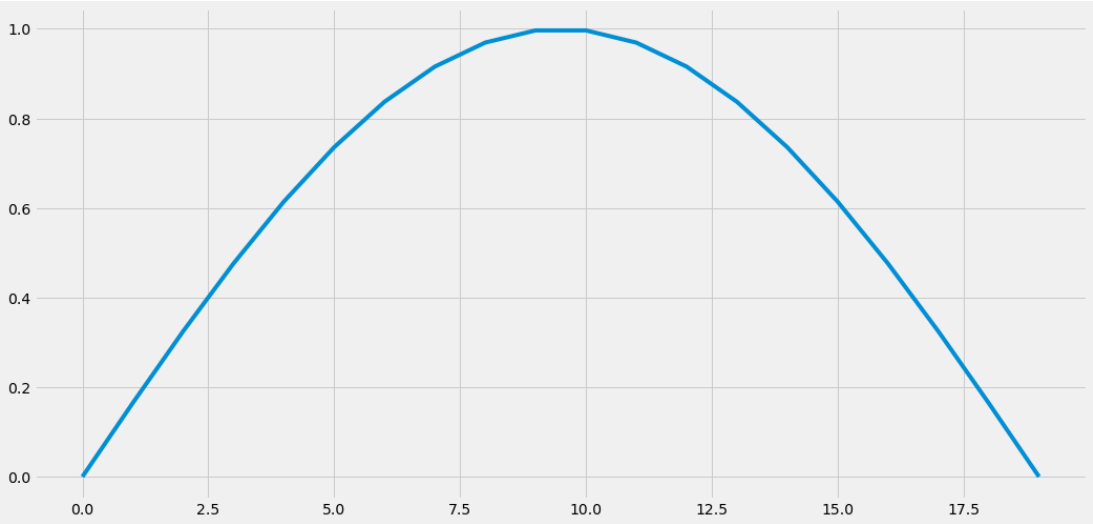
```
Vai process ir stacionārs: True
```



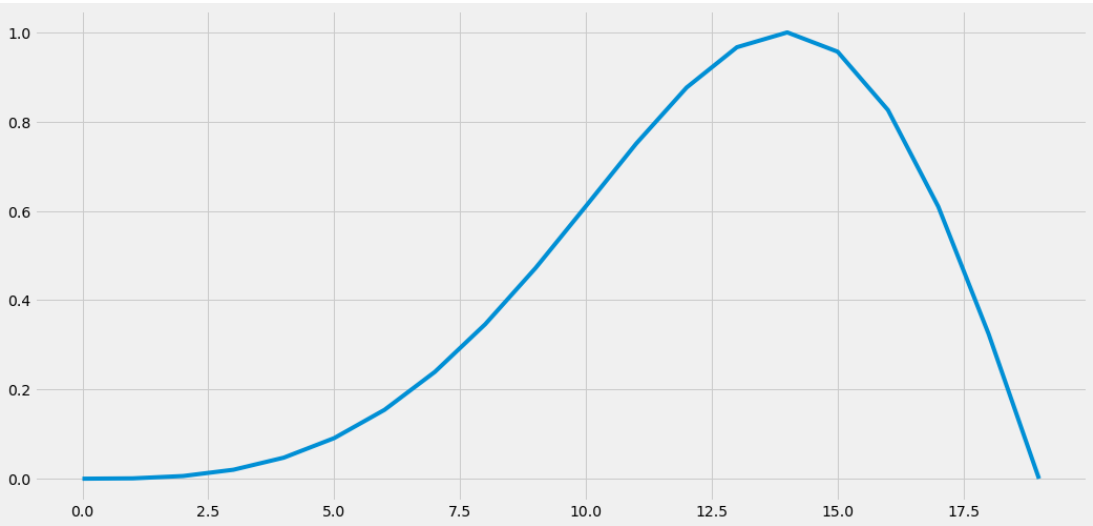
1.0.2 Atkausēšanas perioda simulācija

[3]:
```
# Atkausēšanas periods garums
n_defrost = 20
n_skew = 1

# Tiek ģenerētas vērtības no 0 līdz pi, vērtību daudzums tiek konrolēts ar␣
 ↪mainīgo n_defrost, kas
# attiecīgi nosaka to cik garš būs atkausēšanas periods! Un pēctam tiek␣
 ↪izmantota sinusoīda funkcija,
# lai saģerētu vērtības.
# Papildus, lai kontrolētu asimetriju tiek izmantota funkcija y = sin(x + y/n)
x = np.linspace(0, np.pi, n_defrost)
sin_y = np.sin(x)
y_defrost = np.sin(x - sin_y/n_skew)
```

[4]:
```
# Sinusoīda grafiks + saglabā grafiku!
fig, ax = plt.subplots()
ax.plot(range(n_defrost), sin_y)
ax.set(ylabel=u'')

plt.savefig('images/simulacijas/sinusoid.png')
plt.show();
```

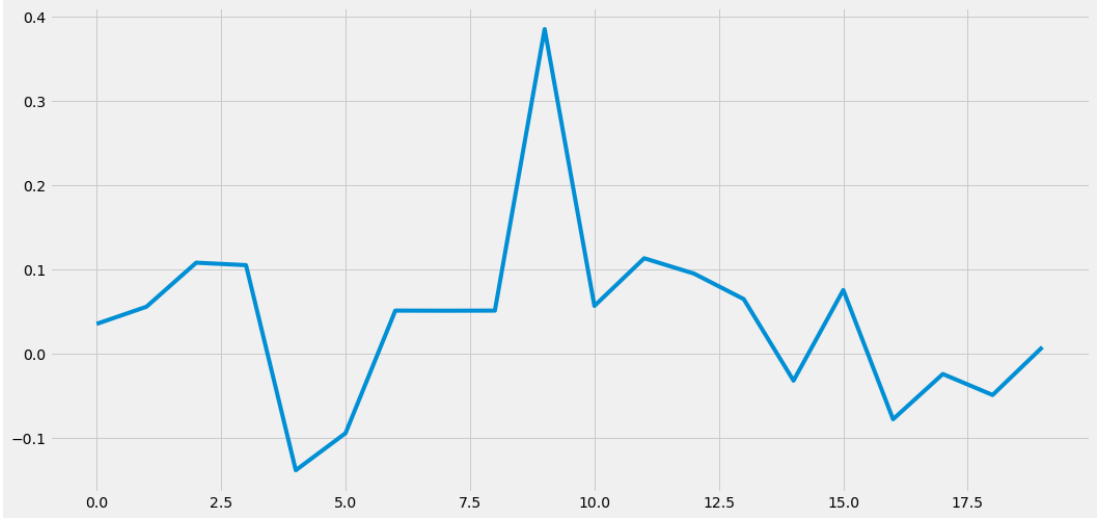


[5]:
```
# Sašķiebts/asimetrisks (atkarībā no iepriekš nodefinētā n) sinusoīda grafiks
fig, ax = plt.subplots()
ax.plot(range(n_defrost), y_defrost)
# ax.set(ylabel=u'Temperatūras (\u00B0C) izmaiņa robežās no 0 līdz 1')

plt.savefig('images/simulacijas/skewd.png')
plt.show();
```



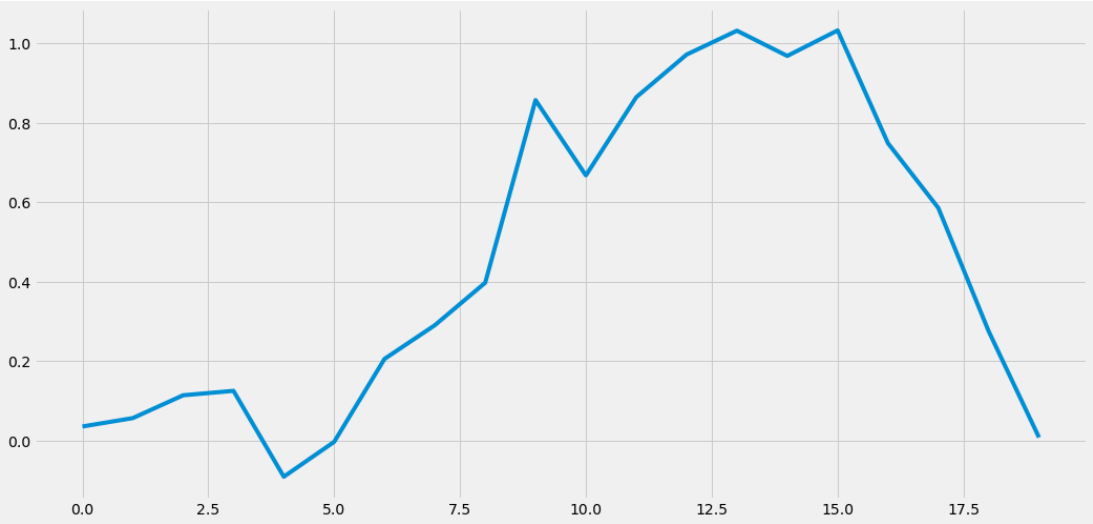
1.0.3 Baltā trokšņa simulācija!

[6]:
```
n_noise = n_defrost
sigma = 0.1
y_noise = sigma * np.random.normal(size=n_noise)

fig, ax = plt.subplots()
ax.plot(range(n_noise), y_noise);

plt.savefig('images/simulacijas/white_noise.png')
plt.show();
```



1.0.4 Baltrā trokšņa pievienošana atkausēšanas perioda simulācijai (sinusoīda funkcijai)!

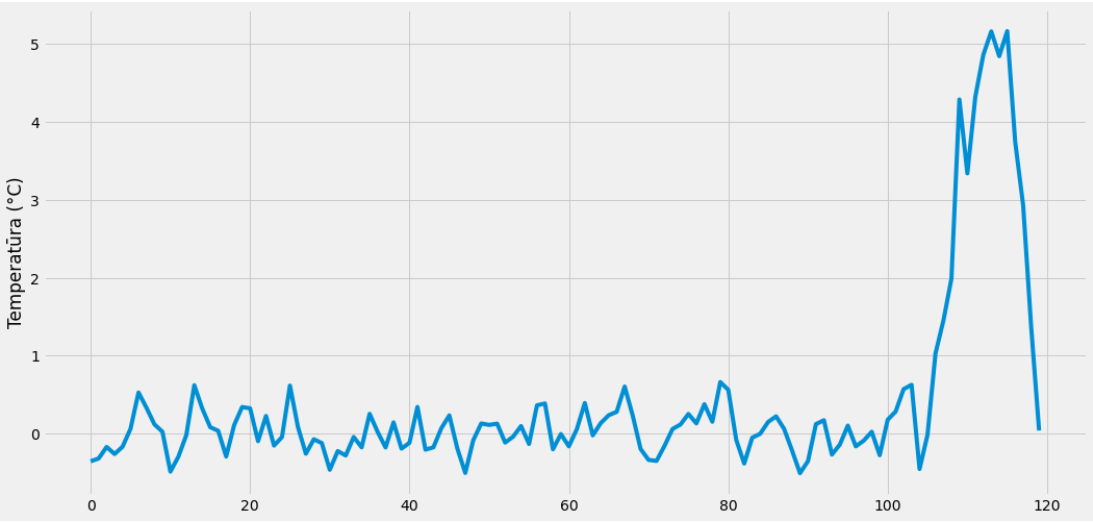
[7]:
```
y_defrost_w_noise = y_defrost + y_noise

fig, ax = plt.subplots()
ax.plot(range(n_defrost), y_defrost_w_noise)

plt.savefig('images/simulacijas/sinusoid_w_white_noise.png')
plt.show();
```



1.0.5 Laikrindu periodu apvienojums

[8]:
```
defrost_scaler = 5
y = np.concatenate((y_arma, defrost_scaler*y_defrost_w_noise))
```

[9]:
```
fig, ax = plt.subplots()
ax.plot(range(len(y)), y)
ax.set(ylabel=u'Temperatūra (\u00B0C)');
```



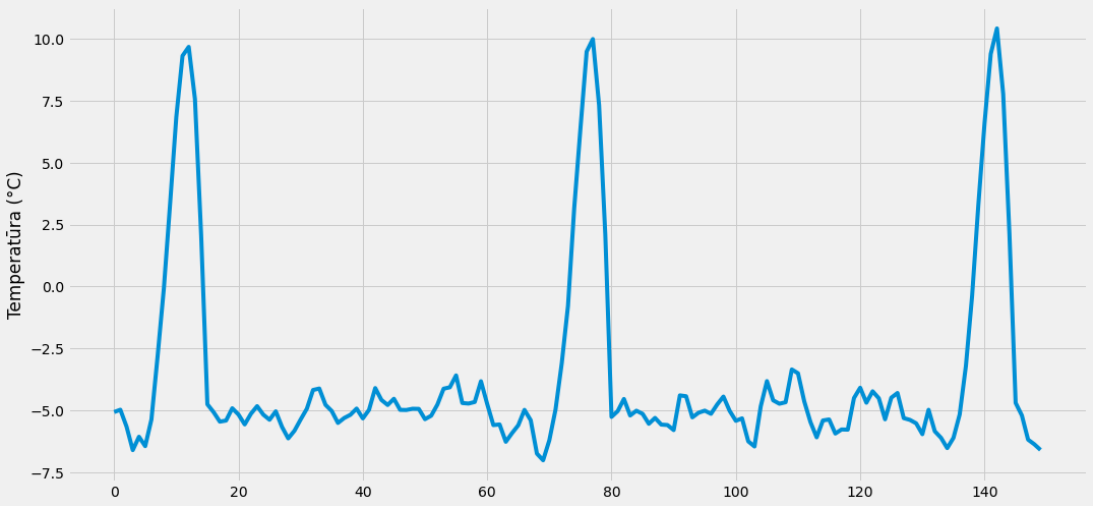
1.0.6 Simulācijas funkcija!

[10]:
```
def ts_sim(n_regular = 100, regular_scale = .25, regular_noise_sigma = 0,
           n_defrost = 20, n_defrost_skew = 2, defrost_scaler = 5,␣
↪defrost_noise_sigma = 0.1,
           n_ts = 1000, ts_mean = 0
          ):
    # Aprēķina sezonu skaitu!
    n_seasons = math.ceil((n_ts+n_arma) / (n_regular + n_defrost))
    y_ts = np.array([])

    for i in range(n_seasons):
        # Simulā normālo periodu ar ARMA procesu!
        ar_coef = np.r_[1, -random.randint(1, 5)/10] # Koeficienti AR polinomā␣
↪(iekļaujut t0, jeb lag 0)!
        ma_coef = np.r_[1, random.randint(1, 5)/10] # Koeficienti MA polinomā␣
↪(iekļaujut t0, jeb lag 0)!
        y_regular = arma_generate_sample(ar_coef, ma_coef, n_regular,␣
↪scale=regular_scale)
        y_regular_noise = regular_noise_sigma * np.random.normal(size=n_regular)

        # Simulē atkausēšanas periodu ar sinusoīdu un pievieno balto troksni!
        x = np.linspace(0, np.pi, n_defrost)
        sin_y = np.sin(x)
        y_defrost = defrost_scaler * np.sin(x - sin_y/n_defrost_skew)
        y_defrost_noise = defrost_noise_sigma * np.random.normal(size=n_defrost)

        # Apvieno regulāro periodu ar atkausēšanas periodu (kopā ar balto␣
↪troksni)
        y_ts = np.concatenate([y_ts, y_regular + y_regular_noise, y_defrost +␣
↪y_defrost_noise])

    # Izvēlas skaitli no 0 līdz n_arma, lai noteiktu laikrindas sākuma punktu!
    # Tas tiek darīts, lai nobīdītu sezonālos pīķus līdzīgām laikrindām.
    rand_start = random.randint(0, n_regular)

    return y_ts[rand_start:n_ts+rand_start] + ts_mean
```

[11]:
```
sim_rez = ts_sim(ts_mean = -5, defrost_scaler = 15, n_defrost = 15,␣
↪n_defrost_skew = 0.8, n_regular = 50, n_ts = 150, defrost_noise_sigma = .25,␣
↪regular_scale = .5)
# sim_rez = ts_sim(n_regular = 100, regular_scale = .35, regular_noise_sigma =␣
↪0, n_defrost = 20, n_defrost_skew = 0.8, defrost_scaler = 15,␣
↪defrost_noise_sigma = 0.1, n_ts = 1000, ts_mean = 0)
```

[12]:
```
fig, ax = plt.subplots()
ax.plot(range(len(sim_rez)), sim_rez)
```

```
ax.set(ylabel=u'Temperatūra (\u00B0C)');

# plt.savefig('images/simulacijas/sim_ts.png')
# plt.show();
```



1.0.7 Laikrindu simulēšana - 3 laikrindu grupas/tipi pa 10 laikrindām katrā grupā!

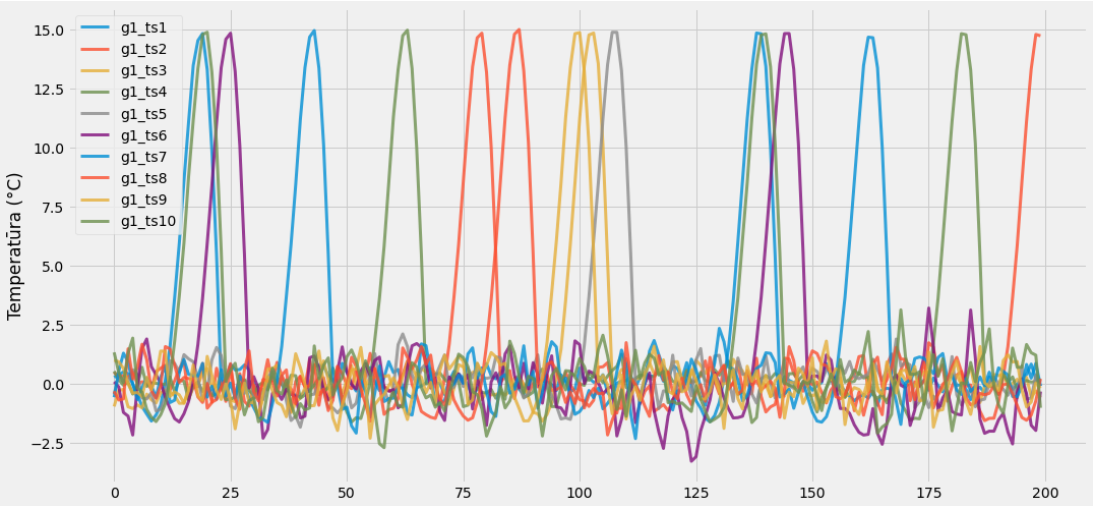
[13]:
```
# 1. grupa!
# Reagular = 100, Defrost = 20, skew = .8
g1_ts1  = ts_sim(n_regular = 100, regular_scale = .35, regular_noise_sigma = 0,␣
 ↪n_defrost = 20, n_defrost_skew = 0.8, defrost_scaler = 15,␣
 ↪defrost_noise_sigma = 0.1, n_ts = 1000, ts_mean = 0)
g1_ts2  = ts_sim(n_regular = 100, regular_scale = .55, regular_noise_sigma = 0,␣
 ↪n_defrost = 20, n_defrost_skew = 0.8, defrost_scaler = 15,␣
 ↪defrost_noise_sigma = 0.1, n_ts = 1000, ts_mean = 0)
g1_ts3  = ts_sim(n_regular = 100, regular_scale = .40, regular_noise_sigma = 0,␣
 ↪n_defrost = 20, n_defrost_skew = 0.8, defrost_scaler = 15,␣
 ↪defrost_noise_sigma = 0.1, n_ts = 1000, ts_mean = 0)
g1_ts4  = ts_sim(n_regular = 100, regular_scale = .40, regular_noise_sigma = 0,␣
 ↪n_defrost = 20, n_defrost_skew = 0.8, defrost_scaler = 15,␣
 ↪defrost_noise_sigma = 0.1, n_ts = 1000, ts_mean = 0)
g1_ts5  = ts_sim(n_regular = 100, regular_scale = .60, regular_noise_sigma = 0,␣
 ↪n_defrost = 20, n_defrost_skew = 0.8, defrost_scaler = 15,␣
 ↪defrost_noise_sigma = 0.1, n_ts = 1000, ts_mean = 0)
g1_ts6  = ts_sim(n_regular = 100, regular_scale = .90, regular_noise_sigma = 0,␣
 ↪n_defrost = 20, n_defrost_skew = 0.8, defrost_scaler = 15,␣
 ↪defrost_noise_sigma = 0.1, n_ts = 1000, ts_mean = 0)
```

```
g1_ts7  = ts_sim(n_regular = 100, regular_scale = .80, regular_noise_sigma = 0,␣
 ↪n_defrost = 20, n_defrost_skew = 0.8, defrost_scaler = 15,␣
 ↪defrost_noise_sigma = 0.1, n_ts = 1000, ts_mean = 0)
g1_ts8  = ts_sim(n_regular = 100, regular_scale = .70, regular_noise_sigma = 0,␣
 ↪n_defrost = 20, n_defrost_skew = 0.8, defrost_scaler = 15,␣
 ↪defrost_noise_sigma = 0.1, n_ts = 1000, ts_mean = 0)
g1_ts9  = ts_sim(n_regular = 100, regular_scale = .75, regular_noise_sigma = 0,␣
 ↪n_defrost = 20, n_defrost_skew = 0.8, defrost_scaler = 15,␣
 ↪defrost_noise_sigma = 0.1, n_ts = 1000, ts_mean = 0)
g1_ts10 = ts_sim(n_regular = 100, regular_scale = .85, regular_noise_sigma = 0,␣
 ↪n_defrost = 20, n_defrost_skew = 0.8, defrost_scaler = 15,␣
 ↪defrost_noise_sigma = 0.1, n_ts = 1000, ts_mean = 0)
```

[14]:
```
ts_g1 = pd.DataFrame({'g1_ts1': g1_ts1,
                      'g1_ts2': g1_ts2,
                      'g1_ts3': g1_ts3,
                      'g1_ts4': g1_ts4,
                      'g1_ts5': g1_ts5,
                      'g1_ts6': g1_ts6,
                      'g1_ts7': g1_ts7,
                      'g1_ts8': g1_ts8,
                      'g1_ts9': g1_ts9,
                      'g1_ts10': g1_ts10,
                     })

# Saglabā datu tabulu!
#ts_g1.to_csv('data/sim_ts_g1.csv')
```

[15]:
```
ts_g1 = pd.read_csv('data/sim_ts_g1.csv', index_col=0)
```

[16]:
```
ts_g1.iloc[0:200].plot(alpha=.8, linewidth=3)
plt.ylabel(u'Temperatūra (\u00B0C)')
plt.savefig('images/simulacijas/ts_g1.png')
```



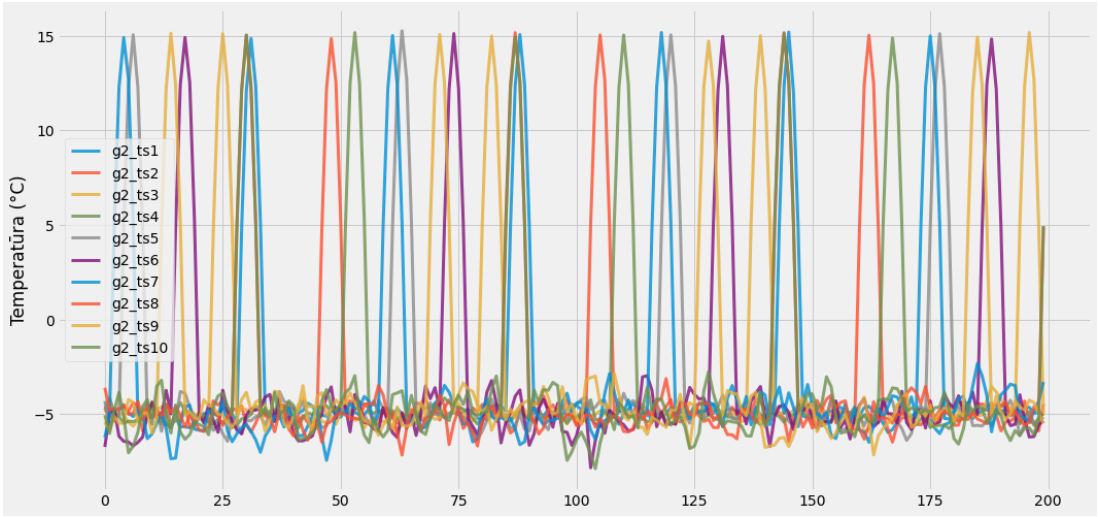
[17]:
```
# 2. grupa!
# Reagular = 50, Defrost = 7, skew = 100
g2_ts1  = ts_sim(n_regular = 50, regular_scale = .25, regular_noise_sigma = 0,␣
 ↪n_defrost = 7, n_defrost_skew = 100, defrost_scaler = 20,␣
 ↪defrost_noise_sigma = 0.15, n_ts = 1000, ts_mean = -5)
g2_ts2  = ts_sim(n_regular = 50, regular_scale = .45, regular_noise_sigma = 0,␣
 ↪n_defrost = 7, n_defrost_skew = 100, defrost_scaler = 20,␣
 ↪defrost_noise_sigma = 0.15, n_ts = 1000, ts_mean = -5)
g2_ts3  = ts_sim(n_regular = 50, regular_scale = .30, regular_noise_sigma = 0,␣
 ↪n_defrost = 7, n_defrost_skew = 100, defrost_scaler = 20,␣
 ↪defrost_noise_sigma = 0.15, n_ts = 1000, ts_mean = -5)
g2_ts4  = ts_sim(n_regular = 50, regular_scale = .30, regular_noise_sigma = 0,␣
 ↪n_defrost = 7, n_defrost_skew = 100, defrost_scaler = 20,␣
 ↪defrost_noise_sigma = 0.15, n_ts = 1000, ts_mean = -5)
g2_ts5  = ts_sim(n_regular = 50, regular_scale = .50, regular_noise_sigma = 0,␣
 ↪n_defrost = 7, n_defrost_skew = 100, defrost_scaler = 20,␣
 ↪defrost_noise_sigma = 0.15, n_ts = 1000, ts_mean = -5)
g2_ts6  = ts_sim(n_regular = 50, regular_scale = .80, regular_noise_sigma = 0,␣
 ↪n_defrost = 7, n_defrost_skew = 100, defrost_scaler = 20,␣
 ↪defrost_noise_sigma = 0.15, n_ts = 1000, ts_mean = -5)
g2_ts7  = ts_sim(n_regular = 50, regular_scale = .70, regular_noise_sigma = 0,␣
 ↪n_defrost = 7, n_defrost_skew = 100, defrost_scaler = 20,␣
 ↪defrost_noise_sigma = 0.15, n_ts = 1000, ts_mean = -5)
g2_ts8  = ts_sim(n_regular = 50, regular_scale = .60, regular_noise_sigma = 0,␣
 ↪n_defrost = 7, n_defrost_skew = 100, defrost_scaler = 20,␣
 ↪defrost_noise_sigma = 0.15, n_ts = 1000, ts_mean = -5)
g2_ts9  = ts_sim(n_regular = 50, regular_scale = .65, regular_noise_sigma = 0,␣
 ↪n_defrost = 7, n_defrost_skew = 100, defrost_scaler = 20,␣
 ↪defrost_noise_sigma = 0.15, n_ts = 1000, ts_mean = -5)
```

```
g2_ts10 = ts_sim(n_regular = 50, regular_scale = .75, regular_noise_sigma = 0,␣
↪n_defrost = 7, n_defrost_skew = 100, defrost_scaler = 20,␣
↪defrost_noise_sigma = 0.15, n_ts = 1000, ts_mean = -5)
```

[18]:
```
ts_g2 = pd.DataFrame({'g2_ts1': g2_ts1,
                      'g2_ts2': g2_ts2,
                      'g2_ts3': g2_ts3,
                      'g2_ts4': g2_ts4,
                      'g2_ts5': g2_ts5,
                      'g2_ts6': g2_ts6,
                      'g2_ts7': g2_ts7,
                      'g2_ts8': g2_ts8,
                      'g2_ts9': g2_ts9,
                      'g2_ts10': g2_ts10,
                     })

# Saglabā datu tabulu!
#ts_g2.to_csv('data/sim_ts_g2.csv')
```

[19]:
```
ts_g2 = pd.read_csv('data/sim_ts_g2.csv', index_col=0)
```

[20]:
```
ts_g2.iloc[0:200].plot(alpha=.8, linewidth=3)
plt.ylabel(u'Temperatūra (\u00B0C)')
plt.savefig('images/simulacijas/ts_g2.png')
```



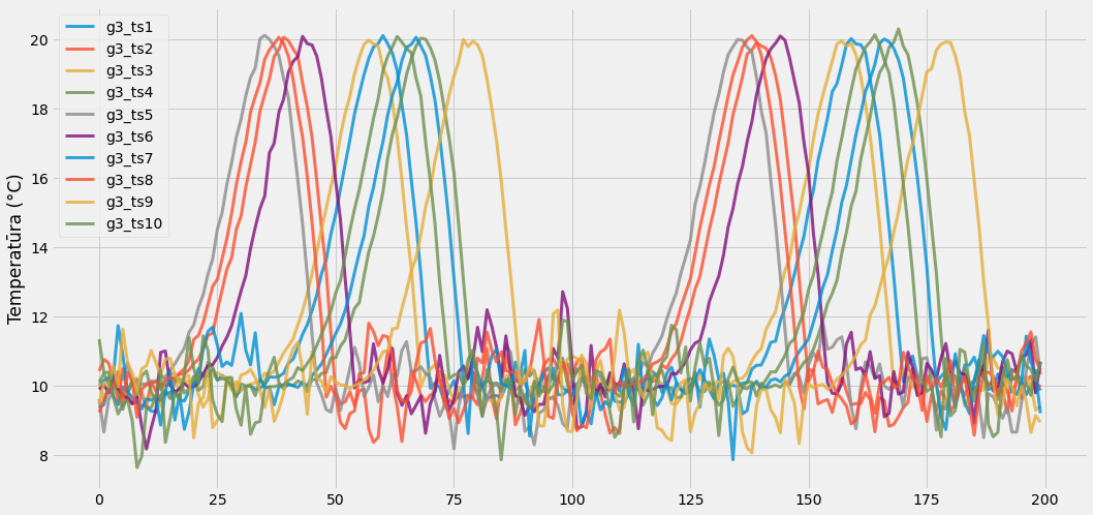
[21]:
```
# 3. grupa!
# Reagular = 50, Defrost = 10, skew = 1
```

```
g3_ts1  = ts_sim(n_regular = 60, regular_scale = .25, regular_noise_sigma = 0,␣
 ↪n_defrost = 40, n_defrost_skew = 1, defrost_scaler = 10, defrost_noise_sigma␣
 ↪= 0.10, n_ts = 1000, ts_mean = 10)
g3_ts2  = ts_sim(n_regular = 60, regular_scale = .45, regular_noise_sigma = 0,␣
 ↪n_defrost = 40, n_defrost_skew = 1, defrost_scaler = 10, defrost_noise_sigma␣
 ↪= 0.10, n_ts = 1000, ts_mean = 10)
g3_ts3  = ts_sim(n_regular = 60, regular_scale = .30, regular_noise_sigma = 0,␣
 ↪n_defrost = 40, n_defrost_skew = 1, defrost_scaler = 10, defrost_noise_sigma␣
 ↪= 0.10, n_ts = 1000, ts_mean = 10)
g3_ts4  = ts_sim(n_regular = 60, regular_scale = .30, regular_noise_sigma = 0,␣
 ↪n_defrost = 40, n_defrost_skew = 1, defrost_scaler = 10, defrost_noise_sigma␣
 ↪= 0.10, n_ts = 1000, ts_mean = 10)
g3_ts5  = ts_sim(n_regular = 60, regular_scale = .50, regular_noise_sigma = 0,␣
 ↪n_defrost = 40, n_defrost_skew = 1, defrost_scaler = 10, defrost_noise_sigma␣
 ↪= 0.10, n_ts = 1000, ts_mean = 10)
g3_ts6  = ts_sim(n_regular = 60, regular_scale = .80, regular_noise_sigma = 0,␣
 ↪n_defrost = 40, n_defrost_skew = 1, defrost_scaler = 10, defrost_noise_sigma␣
 ↪= 0.10, n_ts = 1000, ts_mean = 10)
g3_ts7  = ts_sim(n_regular = 60, regular_scale = .70, regular_noise_sigma = 0,␣
 ↪n_defrost = 40, n_defrost_skew = 1, defrost_scaler = 10, defrost_noise_sigma␣
 ↪= 0.10, n_ts = 1000, ts_mean = 10)
g3_ts8  = ts_sim(n_regular = 60, regular_scale = .60, regular_noise_sigma = 0,␣
 ↪n_defrost = 40, n_defrost_skew = 1, defrost_scaler = 10, defrost_noise_sigma␣
 ↪= 0.10, n_ts = 1000, ts_mean = 10)
g3_ts9  = ts_sim(n_regular = 60, regular_scale = .65, regular_noise_sigma = 0,␣
 ↪n_defrost = 40, n_defrost_skew = 1, defrost_scaler = 10, defrost_noise_sigma␣
 ↪= 0.10, n_ts = 1000, ts_mean = 10)
g3_ts10 = ts_sim(n_regular = 60, regular_scale = .75, regular_noise_sigma = 0,␣
 ↪n_defrost = 40, n_defrost_skew = 1, defrost_scaler = 10, defrost_noise_sigma␣
 ↪= 0.10, n_ts = 1000, ts_mean = 10)
```

[22]:
```
ts_g3 = pd.DataFrame({'g3_ts1': g3_ts1,
                      'g3_ts2': g3_ts2,
                      'g3_ts3': g3_ts3,
                      'g3_ts4': g3_ts4,
                      'g3_ts5': g3_ts5,
                      'g3_ts6': g3_ts6,
                      'g3_ts7': g3_ts7,
                      'g3_ts8': g3_ts8,
                      'g3_ts9': g3_ts9,
                      'g3_ts10': g3_ts10,
                     })

# Saglabā datu tabulu!
#ts_g3.to_csv('data/sim_ts_g3.csv')
```

[23]:
```
ts_g3 = pd.read_csv('data/sim_ts_g3.csv', index_col=0)
```

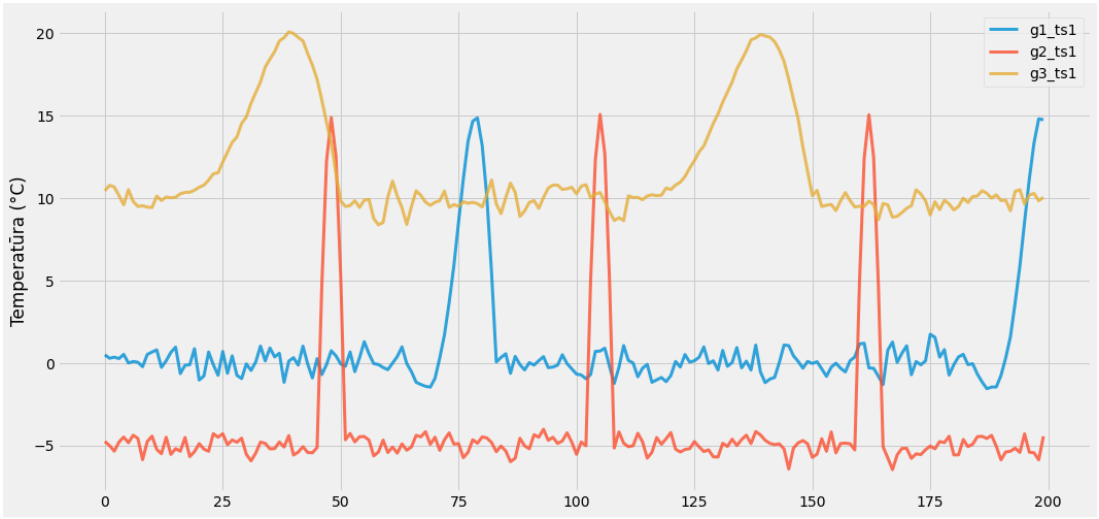
[24]:
```
ts_g3.iloc[0:200].plot(alpha=.8, linewidth=3)
plt.ylabel(u'Temperatūra (\u00B0C)')
plt.savefig('images/simulacijas/ts_g3.png')
```



[25]:
```
# Visu 3 grupu laikrindu salīdzinājums!
```

[26]:
```
ts_g1 = pd.read_csv('data/sim_ts_g1.csv', index_col=0)
ts_g2 = pd.read_csv('data/sim_ts_g2.csv', index_col=0)
ts_g3 = pd.read_csv('data/sim_ts_g3.csv', index_col=0)
```

[27]:
```
pd.DataFrame({'g1_ts1': ts_g1.iloc[0:200,1],
              'g2_ts1': ts_g2.iloc[0:200,1],
              'g3_ts1': ts_g3.iloc[0:200,1],
             }).plot(alpha=.8, linewidth=3);
plt.ylabel(u'Temperatūra (\u00B0C)')
plt.savefig('images/simulacijas/ts_comp.png');
```


g1_ts1
g2_ts1
g3_ts1
Temperatúra (°C)
20
15
10
5
0
−5
0
25
50
75
100
125
150
175
200

B.2. Laikrindu analīzes darba grāmata

1 Laikrindu vispārēja analīze un izpēte

[1]:
```
# Pakotņu ielāde un iestatījumi
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from statsmodels.tsa.stattools import adfuller # Stacionaritātes pārbaudes␣
↪funkcija
from statsmodels.tsa.seasonal import STL # Priekš sezonālās-trenda␣
↪dekompozīcijas
from statsmodels.tsa.seasonal import seasonal_decompose # Klasiskā sezonālā␣
↪dekompozīcija
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf # ACF & PACT␣
↪grafiku zīmēšanai
#import statsmodels.formula.api as smf
#import statsmodels.tsa.api as smt
#import statsmodels.api as sm
#import scipy.stats as scs

%matplotlib inline
plt.style.use('fivethirtyeight') # Grafiku stils!
plt.rcParams["figure.figsize"] = (16,8) # Grafiku izmērs
```

1.0.1 Datu ielāde

[2]:
```
# Simulētie temperatūras mērījumu dati
ts_g1 = pd.read_csv('data/sim_ts_g1.csv', index_col=0)
ts_g2 = pd.read_csv('data/sim_ts_g2.csv', index_col=0)
ts_g3 = pd.read_csv('data/sim_ts_g3.csv', index_col=0)

df_g1 = ts_g1.iloc[:, [1]].copy() # Sezonas garums - 120
df_g2 = ts_g2.iloc[:, [1]].copy() # Sezonas garums - 57
df_g3 = ts_g3.iloc[:, [1]].copy() # Sezonas garums - 100

# df = pd.DataFrame({'g1_ts1': ts_g1.iloc[0:200,1],
#                    'g2_ts1': ts_g2.iloc[0:200,1],
#                    'g3_ts1': ts_g3.iloc[0:200,1],
#                   })

# Airpassengers data
df_ap = pd.read_csv('data/AirPassengers.csv', parse_dates=True)
df_ap['Month'] = pd.to_datetime(df_ap.Month)
df_ap.set_index('Month', inplace=True)
```

[55]:
```
min(df_ap.index)
```

[55]:
```
Timestamp('1949-01-01 00:00:00')
```

[56]:
```
max(df_ap.index)
```

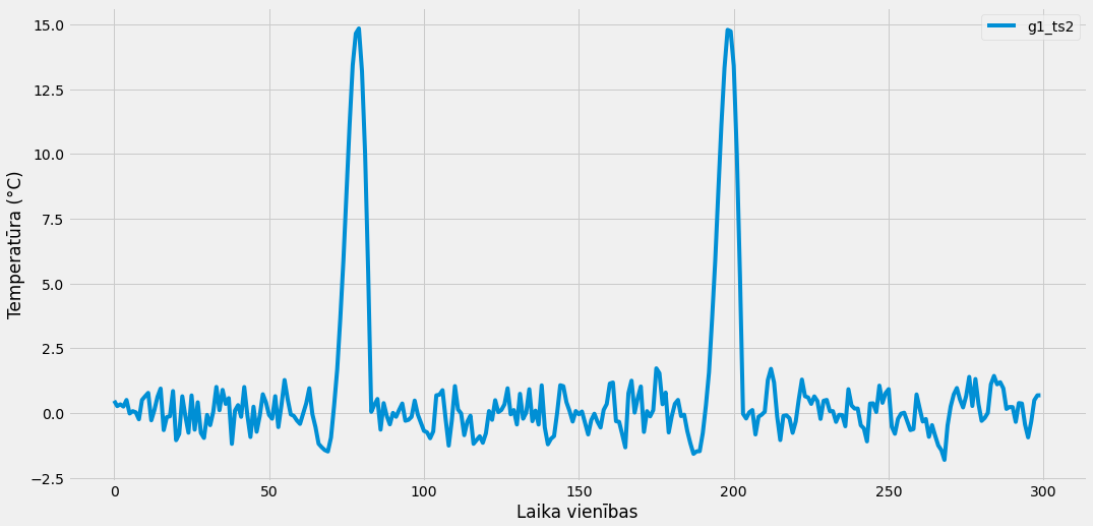
[56]:
```
Timestamp('1960-12-01 00:00:00')
```

[3]:
```
print(df_g1.shape)
print(df_ap.shape)
```

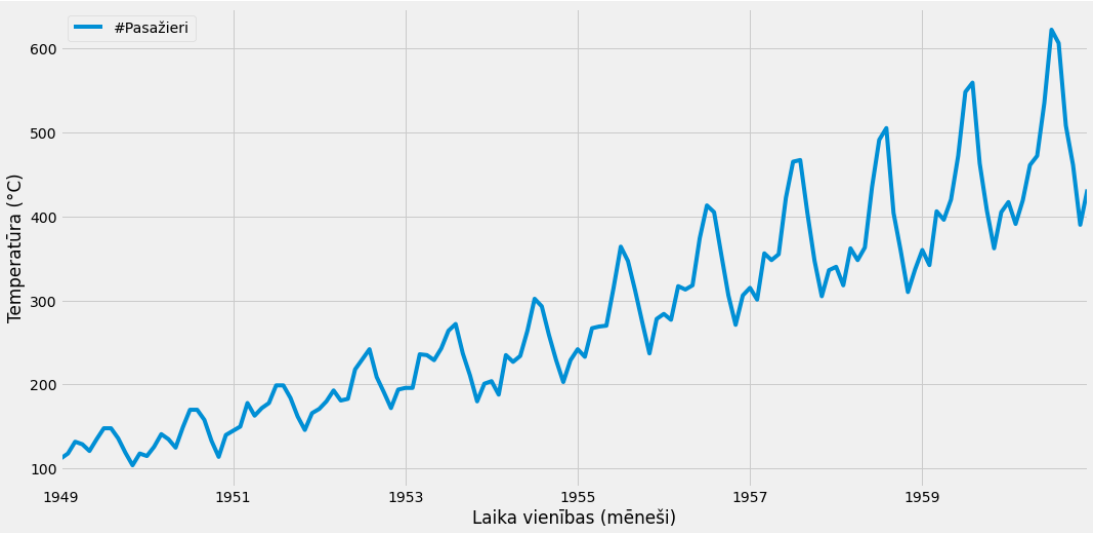
```
(1000, 1)
(144, 1)
```

1.0.2 Datu vizualizācija

[5]:
```
df_g1.iloc[:300, :].plot(ylabel=u'Temperatūra (\u00B0C)',
                         xlabel='Laika vienības'
                        );
plt.savefig('images/analize/g1_ts2_plot.png');
```

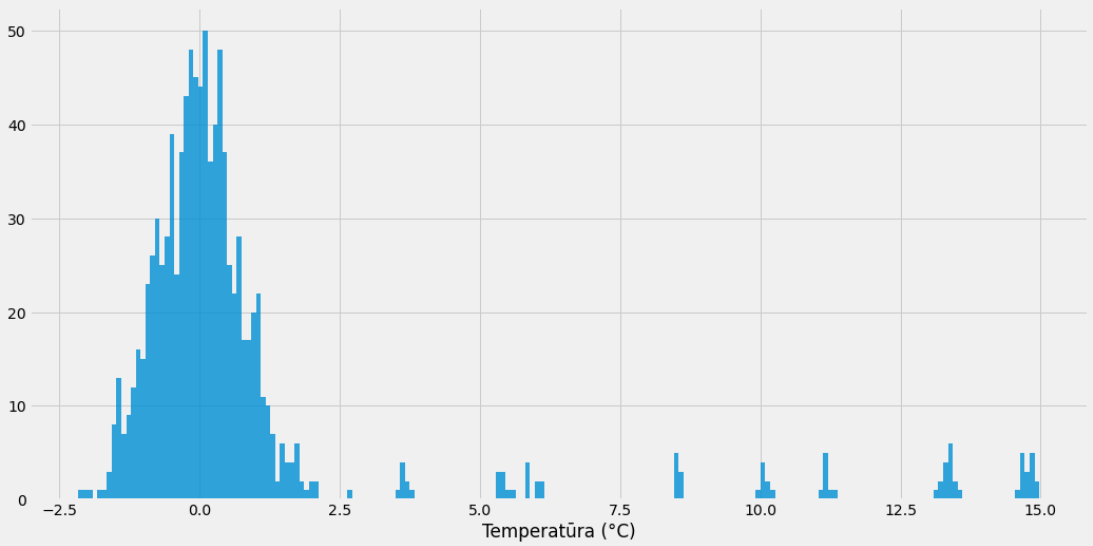


[32]:
```
df_ap.plot(ylabel=u'Temperatūra (\u00B0C)', xlabel='Laika vienības (mēneši)');
plt.legend(['#Pasažieri'])
plt.savefig('images/analize/ap_plot.png');
```



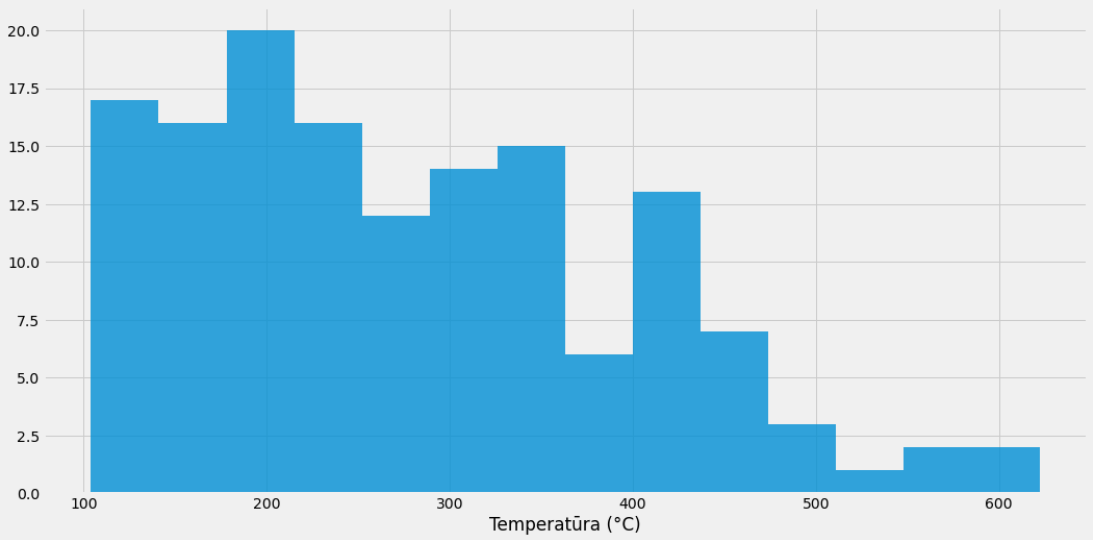
1.0.3 Laikrindas vērtību histogramma

[7]:
```
plt.hist(ts_g1.iloc[:, [1]], bins=200, alpha=.80)
plt.xlabel(u'Temperatūra (\u00B0C)')
plt.savefig('images/analize/g1_ts2_hist.png');
plt.show();
```



[8]:
```
df_ap.hist(bins=14, alpha=.80)
plt.title('')
plt.savefig('images/analize/ap_hist.png');
```

```
plt.xlabel(u'Temperatūra (\u00B0C)');
```



1.0.4 Sezonāla-trenda dekompozīcija izmantojot LOESS metodi (STL)

Time Series Components

A useful abstraction for selecting forecasting methods is to break a time series down into systematic and unsystematic components.

- Systematic: Components of the time series that have consistency or recurrence and can be described and modeled.
- Non-Systematic: Components of the time series that cannot be directly modeled.

A given time series is thought to consist of three systematic components including level, trend, seasonality, and one non-systematic component called noise.

These components are defined as follows:

- Level: The average value in the series.
- Trend: The increasing or decreasing value in the series.
- Seasonality: The repeating short-term cycle in the series.
- Noise: The random variation in the series.

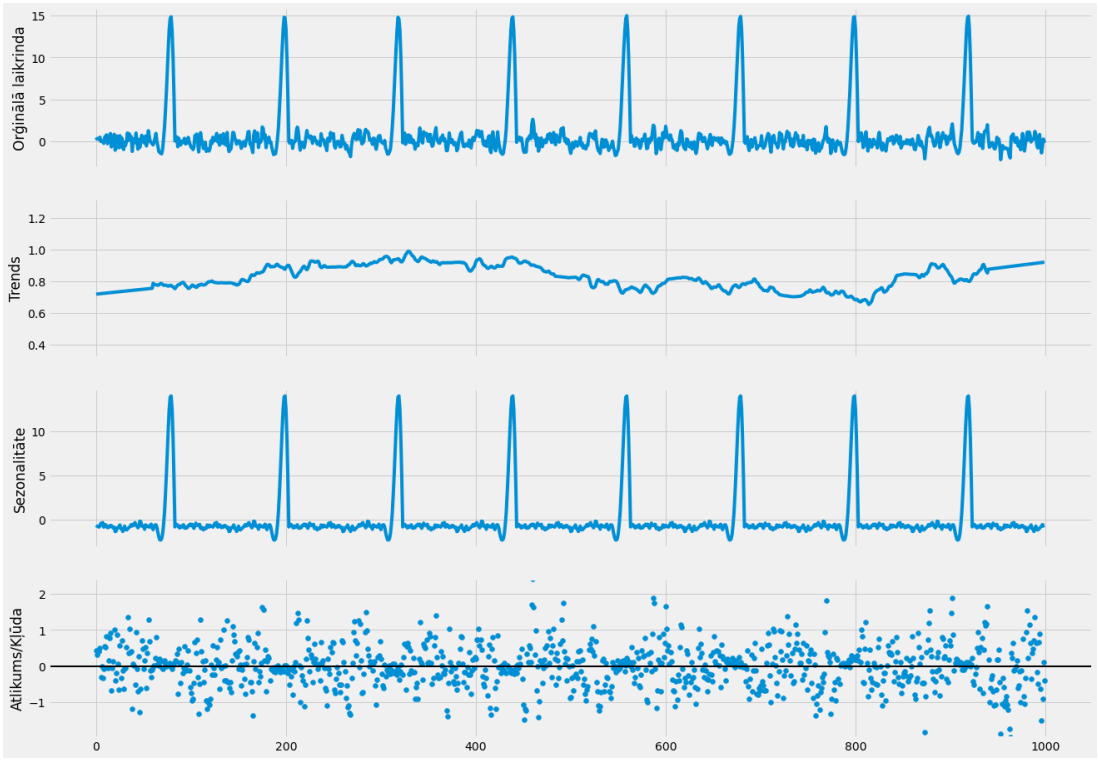
It provides a structured way of thinking about a time series forecasting problem, both generally in terms of modeling complexity and specifically in terms of how to best capture each of these components in a given model.

Additive components $ y(t) = Level + Trend + Seasonality + Noise $

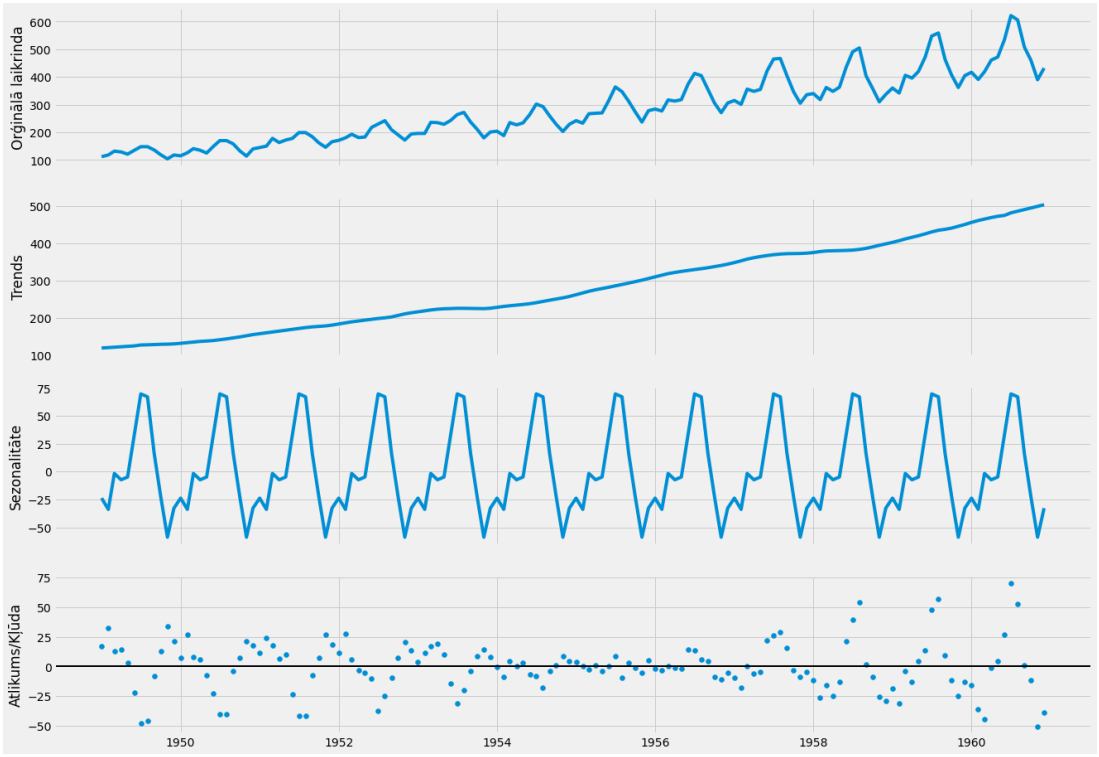
Multiplicative componentes $ y(t) = Level * Trend * Seasonality * Noise $

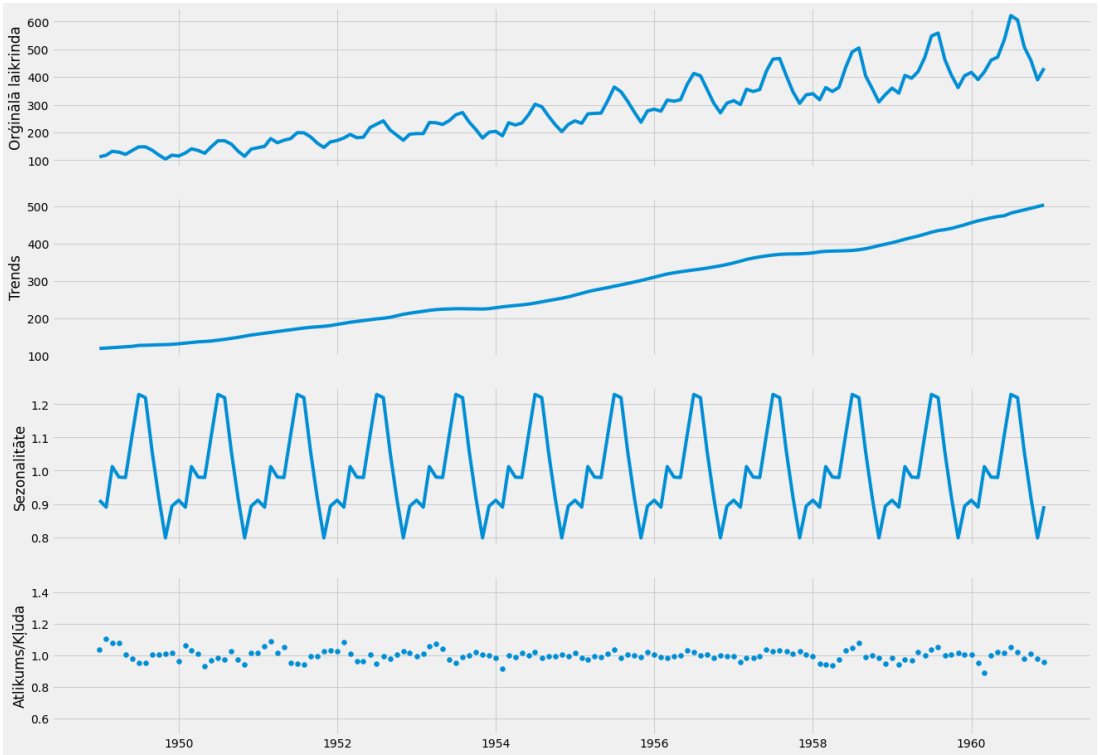
[59]:
```python
def decomp_plot(result, save = False, file_name = 'none'):
    fig, axs = plt.subplots(4, sharex=True, figsize=(20, 15))
    axs[0].plot(result.observed)
    axs[0].set(ylabel='Oŗģinālā laikrinda')
    axs[1].plot(result.trend)
    if max(result.trend) - min(result.trend) <= 1:
        lower = (max(result.trend) + min(result.trend)) / 2 - 0.5
        upper = (max(result.trend) + min(result.trend)) / 2 + 0.5
        axs[1].set_ylim([lower, upper])
    axs[1].set(ylabel='Trends')
    axs[2].plot(result.seasonal)
    axs[2].set(ylabel='Sezonalitāte')
    axs[3].scatter(result.observed.index, result.resid)
    axs[3].axhline(linewidth=2, color='black')
    axs[3].set(ylabel='Atlikums/Kļūda')
    if max(result.resid) - min(result.resid) <= 1:
        lower = (max(result.resid) + min(result.resid)) / 2 - 0.5
        upper = (max(result.resid) + min(result.resid)) / 2 + 0.5
        axs[3].set_ylim([lower, upper])
    if save==True:
        plt.savefig(f'images/analize/{file_name}.png')
    plt.show()

    return
```

[25]:
```python
# Multiplikatīva dekompozīcija
# mul_res = seasonal_decompose(df_g2, model='multiplicative',␣
 ↪extrapolate_trend='freq')

# Adtīva dekompozīcija
add_res = seasonal_decompose(df_g1, model='additive', extrapolate_trend='freq',␣
 ↪period=120)
# !!! extrapolate_trend nodrošina trūkstošu vērtību imputāciju !!!

decomp_plot(add_res, save=True, file_name='g1_ts2_decomp')
```



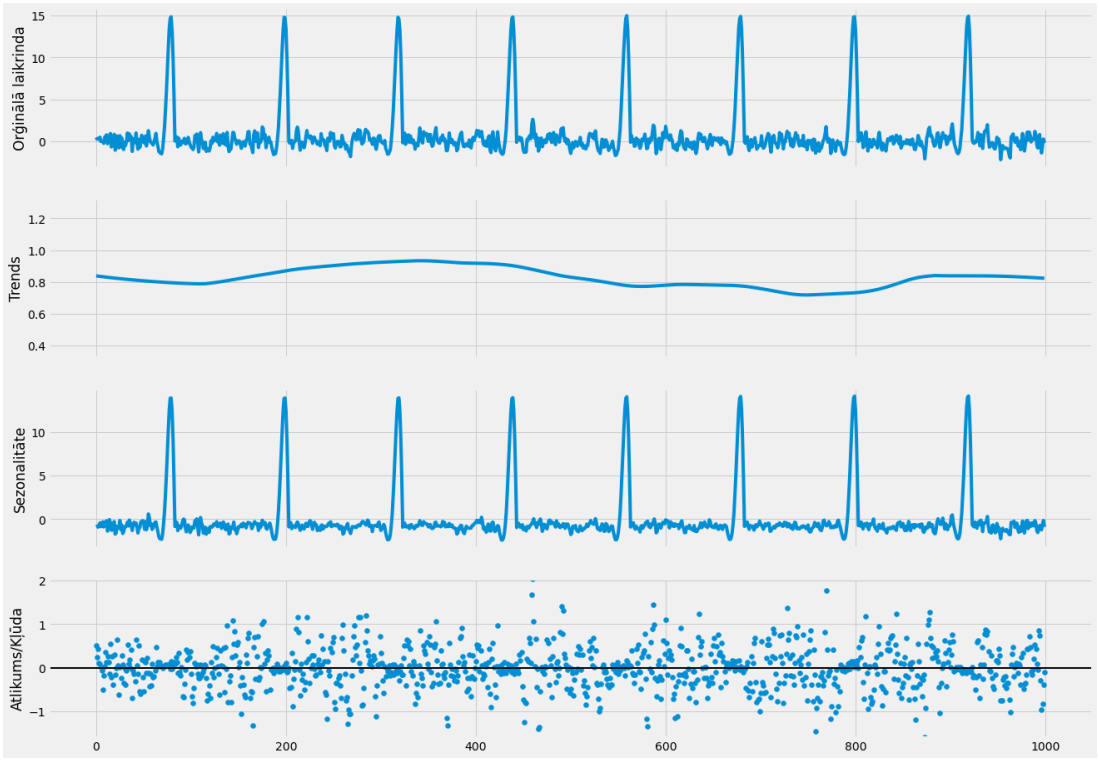
[61]:
```
# Multiplikatīva dekompozīcija
mul_res = seasonal_decompose(df_ap, model='multiplicative',␣
 ↪extrapolate_trend='freq')

# Adtīva dekompozīcija
add_res = seasonal_decompose(df_ap, model='additive', extrapolate_trend='freq')
# !!! extrapolate_trend nodrošina trūkstošu vērtību imputāciju !!!

decomp_plot(add_res, save=True, file_name='ap_decomp_add')
decomp_plot(mul_res, save=True, file_name='ap_decomp_mul')
```


Orģinālā laikrinda
Trends
Sezonalitāte
Atlikums/Kļūda
1950
1952
1954
1956
1958
1960

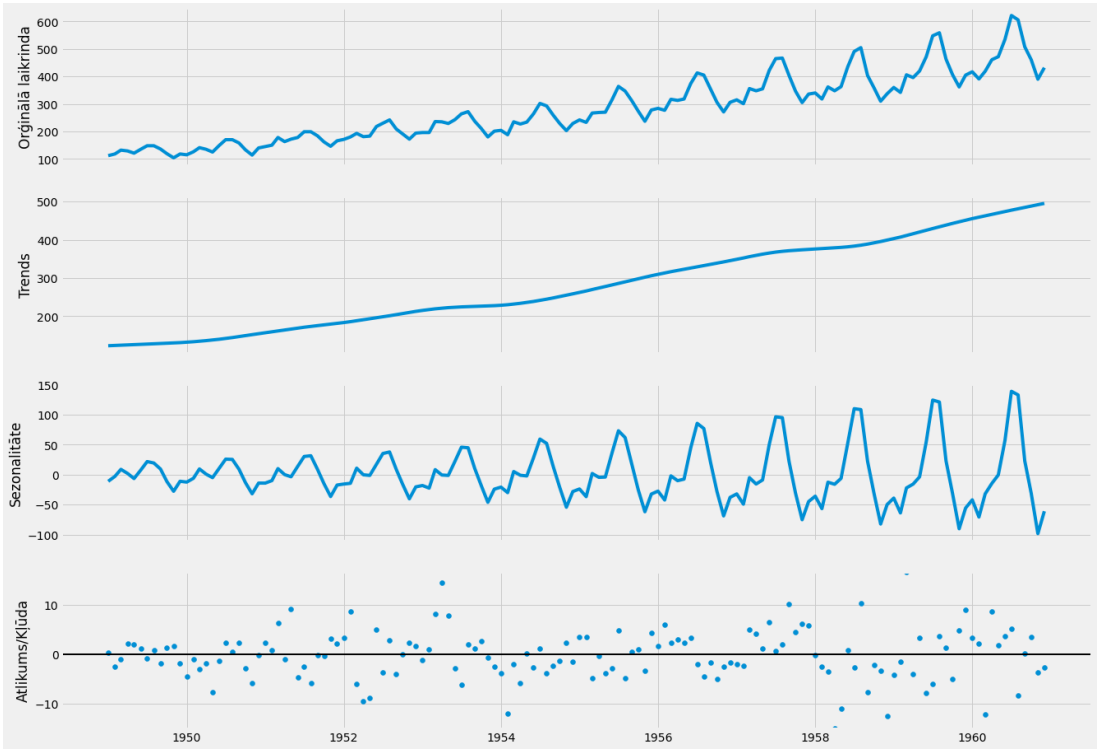

Orģinālā laikrinda
Trends
Sezonalitāte
Atlikums/Kļūda
1950
1952
1954
1956
1958
1960

1.0.5 STL

[28]:
```
# STL dekompozīcija!
stl = STL(df_g1, period=120)
stl_res = stl.fit()

decomp_plot(stl_res, save=True, file_name='g1_ts2_decomp_stl')
```



[29]:
```
stl = STL(df_ap)
stl_res = stl.fit()

decomp_plot(stl_res, save=True, file_name='ap_decomp_stl')
```



1.0.6 Stacionaritātes pārbaude

[62]:
```
# Laik pārbaudītu stacionaritāti tiek izmantots uzlabotais Dikija-Fullera tests!
 ↪!!
def adf_test(series):
    result = adfuller(series)
    print('ADF statistika: %f' % result[0]) # Uzlabotā Dikija-Fullera testa␣
 ↪statistika
    print('p-vērtība: %f' % result[1]) # ADF testa p-vērtība
    print('Kritiskās vērtības:')
    for key, value in result[4].items():
        print('\t%s: %.3f' % (key, value))
```

[265]:
```
adf_test(df_g1.values)
```

```
ADF statistika: -11.519534
p-vērtība: 0.000000
Kritiskās vērtības:
        1%: -3.437
        5%: -2.864
        10%: -2.568
```

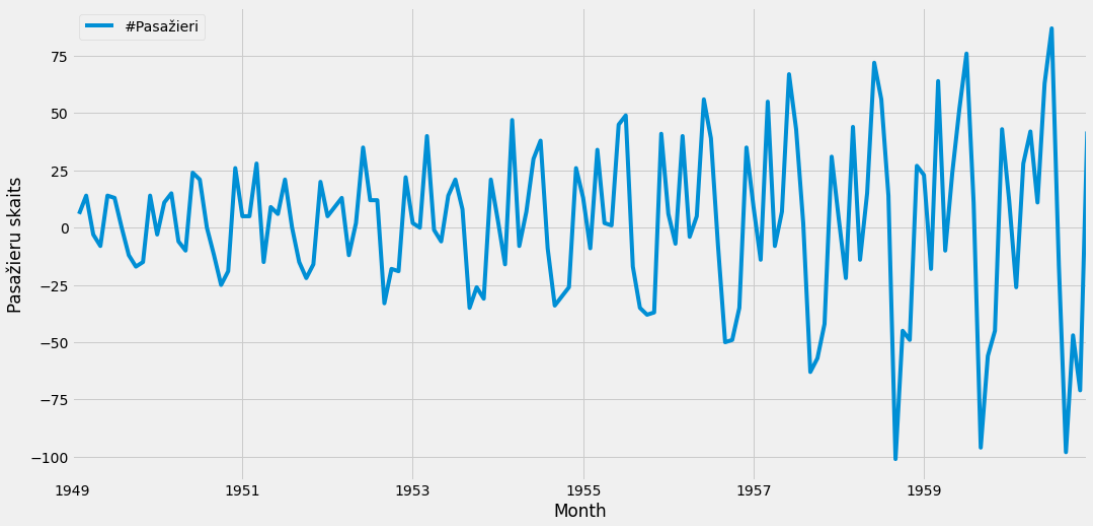
[266]:
```
adf_test(df_ap)
```

```
ADF statistika: 0.815369
p-vērtība: 0.991880
Kritiskās vērtības:
        1%: -3.482
        5%: -2.884
        10%: -2.579
```

[268]:
```
adf_test(df_ap.diff().dropna())
```

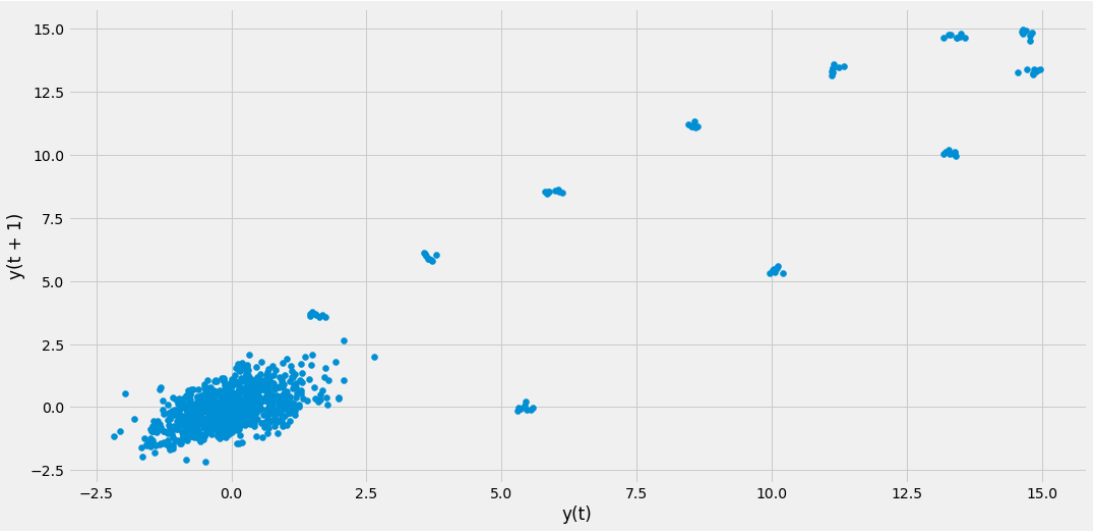
```
ADF statistika: -2.829267
p-vērtība: 0.054213
Kritiskās vērtības:
        1%: -3.482
        5%: -2.884
        10%: -2.579
```

[33]:
```
df_ap.diff().plot(ylabel='Pasažieru skaits')
plt.legend(['#Pasažieri'])
plt.savefig('images/analize/ap_1x_diff.png');
```

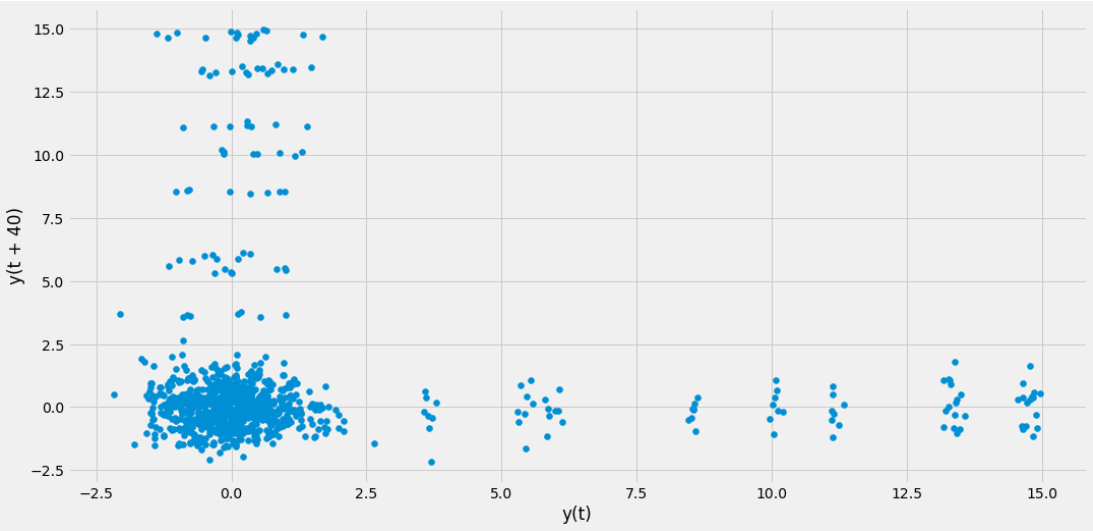


1.0.7 Lagu grafiki!

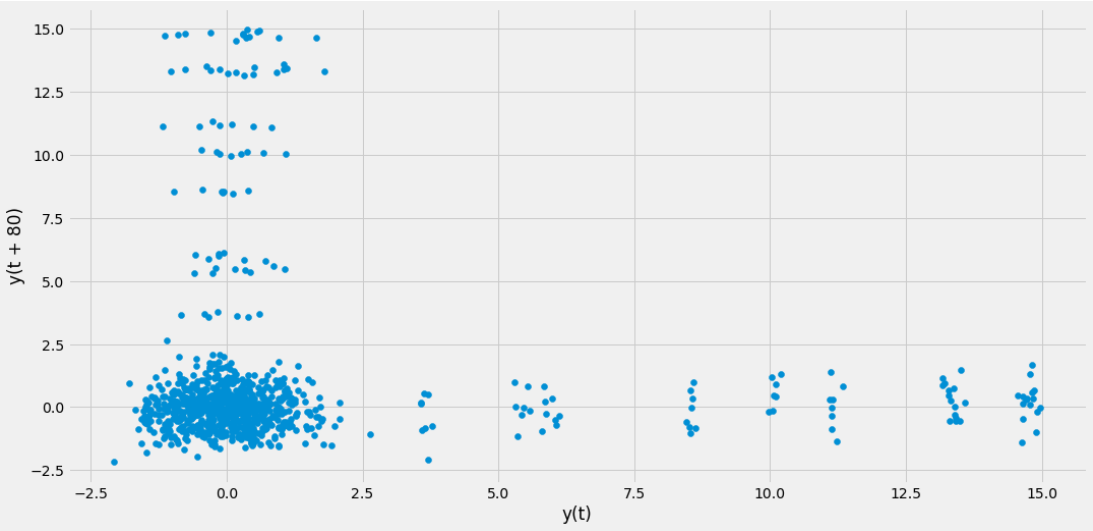
[34]:
```
# LAG = 1
pd.plotting.lag_plot(df_g1, lag=1);
plt.savefig('images/analize/g1_ts2_lag1.png');
```



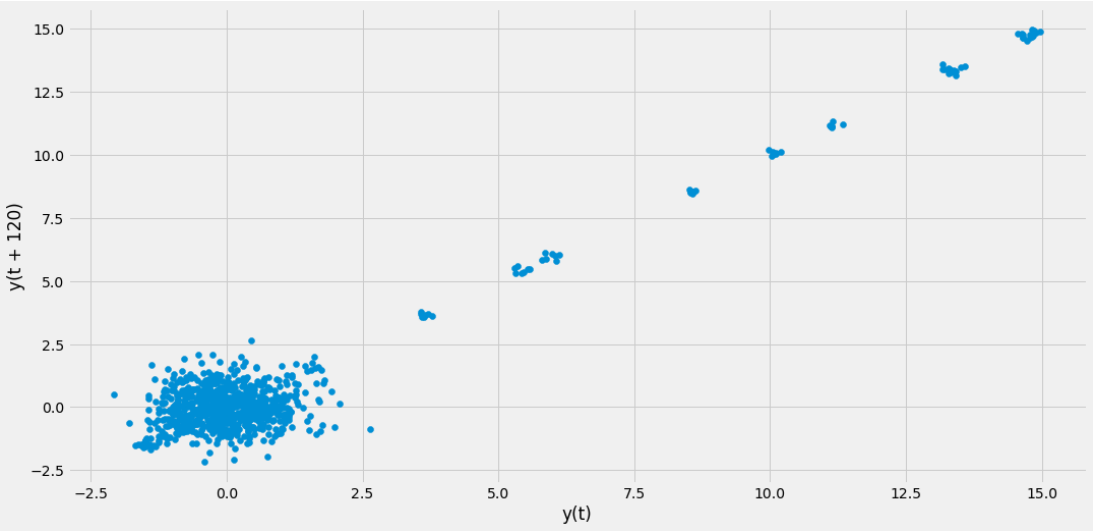
[35]:
```
# LAG = 40
pd.plotting.lag_plot(df_g1, lag=40);
plt.savefig('images/analize/g1_ts2_lag40.png');
```



[36]:
```
# LAG = 80
pd.plotting.lag_plot(df_g1, lag=80);
plt.savefig('images/analize/g1_ts2_lag80.png');
```



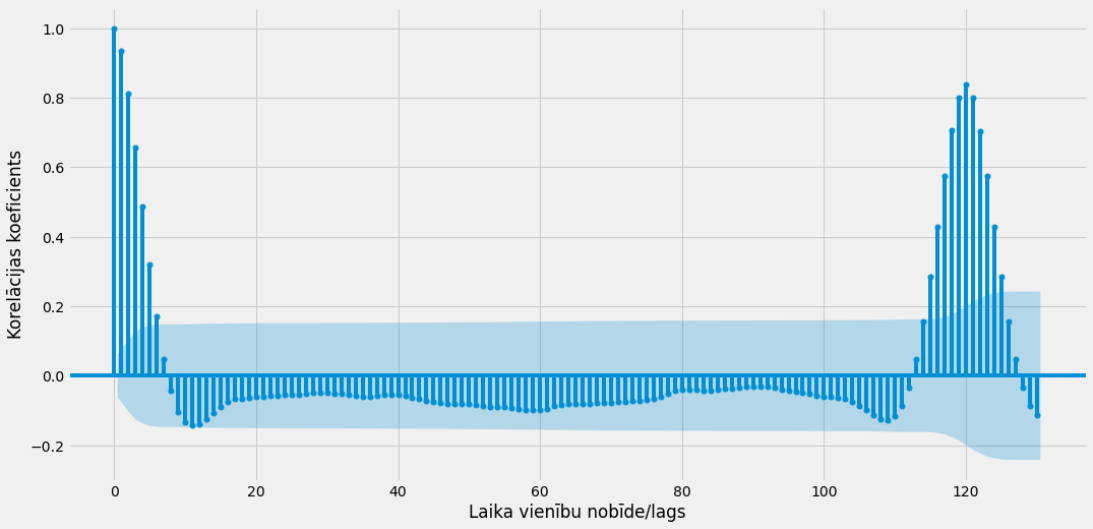
[37]:
```
# LAG = 120
pd.plotting.lag_plot(df_g1, lag=120);
plt.savefig('images/analize/g1_ts2_lag120.png');
```



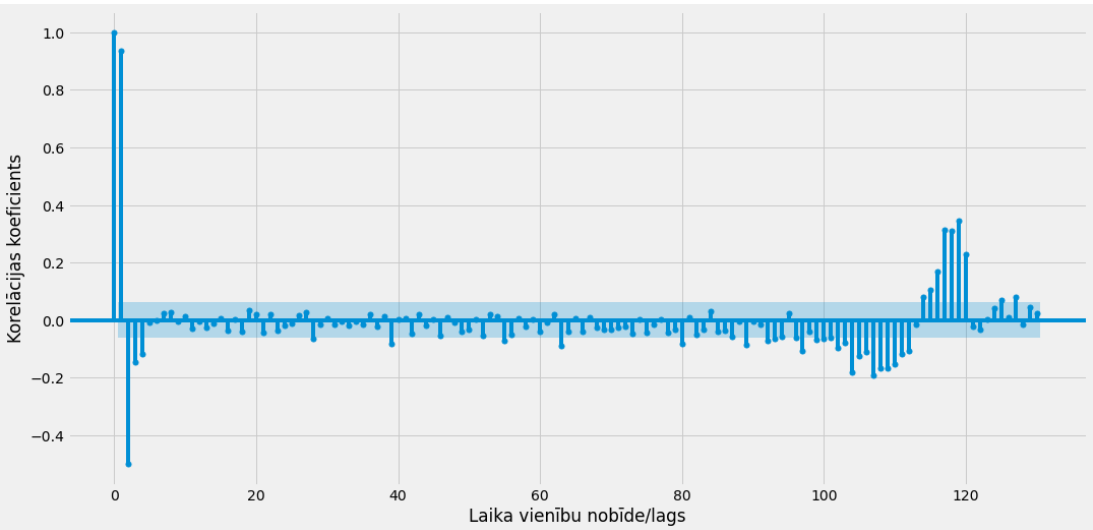
1.0.8 ACF & PACF grafiki!

[38]:
```
# ACF
acf_plot = plot_acf(df_g1, lags=130)
plt.title('')
plt.ylabel('Korelācijas koeficients')
```

```
plt.xlabel('Laika vienību nobīde/lags');
plt.savefig('images/analize/g1_ts2_ACF.png');
```



[39]:
```
# PACF
pacf_plot = plot_pacf(ts_g1.iloc[:, [1]], lags=130, method='ols')
plt.title('')
plt.ylabel('Korelācijas koeficients')
plt.xlabel('Laika vienību nobīde/lags');
plt.savefig('images/analize/g1_ts2_PACF.png');
```



1.0.9 Sample entropy

[47]:
```
def SampEn(U, m, r):
    # Compute Sample entropy
    def _maxdist(x_i, x_j):
        return max([abs(ua - va) for ua, va in zip(x_i, x_j)])

    def _phi(m):
        x = [[U[j] for j in range(i, i + m - 1 + 1)] for i in range(N - m + 1)]
        C = [len([1 for j in range(len(x)) if i != j and _maxdist(x[i], x[j])␣
 ↪<= r]) for i in range(len(x))]
        return sum(C)

    N = len(U)
    return -np.log(_phi(m+1) / _phi(m))
```

[53]:
```
# https://en.wikipedia.org/wiki/Approximate_entropy labāka alternatīva =>␣
 ↪Sample entropy
print(SampEn(df_ap['#Passengers'].values, m=2, r=0.2*np.
 ↪std(df_ap['#Passengers'].values)))
print(SampEn(df_g1.values, m=2, r=0.2*np.std(df_g1['g1_ts2'].values)))
```

```
0.6251424514458879
0.6813177678252744
```

B.3. Datu priekšapstrādes darba grāmata

1 Datu sagatavošana un priekšapstrāde

[63]:
```
# Pakotņu ielāde un iestatījumi
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.metrics import mean_squared_error
from tsmoothie.smoother import * # gludināšanas funkcijas
from sklearn.preprocessing import MinMaxScaler, Normalizer, StandardScaler,␣
↪RobustScaler, QuantileTransformer # mērogošanas funkcijas
from statsmodels.tsa.seasonal import seasonal_decompose # Klasiskā sezonālā␣
↪dekompozīcija
import statistics

%matplotlib inline
plt.style.use('fivethirtyeight') # Grafiku stils!
plt.rcParams["figure.figsize"] = (16,8) # Grafiku izmērs
```

1.0.1 0. Datu ielāde

[6]:
```
# Simulētie temperatūras mērījumu dati
ts_g1 = pd.read_csv('data/sim_ts_g1.csv', index_col=0)
df_g1 = ts_g1.iloc[:, [1]].copy() # Sezonas garums - 120

ts_g3 = pd.read_csv('data/sim_ts_g3.csv', index_col=0)
df_g3 = ts_g3.iloc[:, [1]].copy() # Sezonas garums - 100

# Datu ielāde ar trūkstošiem ierakstiem
df_g1_missing = pd.read_excel('data/missing_data_g1_ts2.xlsx', index_col=0)

# Airpassengers data
df_ap = pd.read_csv('data/AirPassengers.csv', parse_dates=True)
df_ap['Month'] = pd.to_datetime(df_ap.Month)
df_ap.set_index('Month', inplace=True)
```

1.0.2 1. Trūkstošo datu imputācija

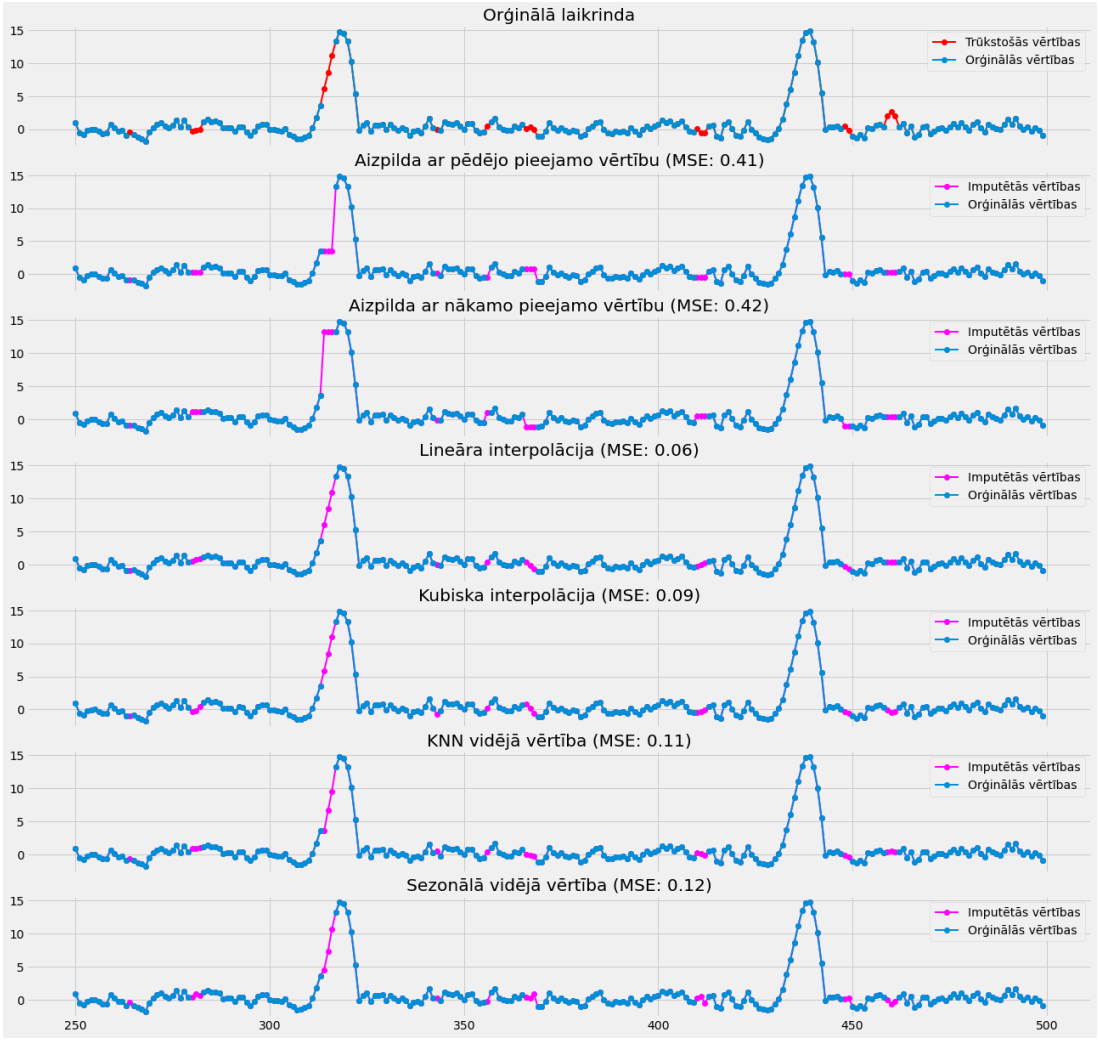
[52]:
```
# Tā kā mūsu simulētie dati satur visus novērojumus, ir nepieciešams
```

[53]:
```
fig, axes = plt.subplots(7, 1, sharex=True, figsize=(20, 20))

### Orģinālā laikrinda!
df_g1.iloc[250:500, :].plot(title='Orģinālā laikrinda', ax=axes[0],␣
↪label='Actual', color='red', style="o-", linewidth=2)
df_g1_missing.iloc[250:500, :].plot(title='Orģinālā laikrinda', ax=axes[0],␣
↪label='Actual', style="o-", linewidth=2)
axes[0].legend(["Trūkstošās vērtības", "Orģinālās vērtības"])
```

```
### Aizpilda ar pēdējo pieejamo vērtību!
df_ffill = df_g1_missing.ffill()
error = np.round(mean_squared_error(df_g1['g1_ts2'].iloc[250:500],␣
 ↪df_ffill['g1_ts2'].iloc[250:500]), 2)
df_ffill['g1_ts2'].iloc[250:500].plot(title='Aizpilda ar pēdējo pieejamo␣
 ↪vērtību (MSE: ' + str(error) +")", ax=axes[1], label='Forward Fill',␣
 ↪color='magenta', style="o-", linewidth=2)
df_g1_missing.iloc[250:500, :].plot(ax=axes[1], label='Forward Fill',␣
 ↪style="o-", linewidth=2)
axes[1].legend(["Imputētās vērtības", "Orģinālās vērtības"])

### Aizpilda ar nākamo pieejamo vērtību!
df_bfill = df_g1_missing.bfill()
error = np.round(mean_squared_error(df_g1['g1_ts2'].iloc[250:500],␣
 ↪df_bfill['g1_ts2'].iloc[250:500]), 2)
df_bfill['g1_ts2'].iloc[250:500].plot(title='Aizpilda ar nākamo pieejamo␣
 ↪vērtību (MSE: ' + str(error) +")", ax=axes[2], label='Back Fill',␣
 ↪color='magenta', style="o-", linewidth=2)
df_g1_missing.iloc[250:500, :].plot(ax=axes[2], label='Back Fill', style="o-",␣
 ↪linewidth=2)
axes[2].legend(["Imputētās vērtības", "Orģinālās vērtības"])

### Lineāra interpolācija!
df_linear = df_g1_missing.interpolate()
error = np.round(mean_squared_error(df_g1['g1_ts2'].iloc[250:500],␣
 ↪df_linear['g1_ts2'].iloc[250:500]), 2)
df_linear['g1_ts2'].iloc[250:500].plot(title='Lineāra interpolācija (MSE: ' +␣
 ↪str(error) +")", ax=axes[3], label='Linear Fill', color='magenta',␣
 ↪style="o-", linewidth=2)
df_g1_missing.iloc[250:500, :].plot(ax=axes[3], label='Linear Fill',␣
 ↪style="o-", linewidth=2)
axes[3].legend(["Imputētās vērtības", "Orģinālās vērtības"])


### Kubiska interpolācija!
df_cubic = df_g1_missing.interpolate(method='cubic')
error = np.round(mean_squared_error(df_g1['g1_ts2'].iloc[250:500],␣
 ↪df_cubic['g1_ts2'].iloc[250:500]), 2)
df_cubic['g1_ts2'].iloc[250:500].plot(title='Kubiska interpolācija (MSE: ' +␣
 ↪str(error) +")", ax=axes[4], label='Cubic Fill', color='magenta',␣
 ↪style="o-", linewidth=2)
df_g1_missing.iloc[250:500, :].plot(ax=axes[4], label='Cubic Fill', style="o-",␣
 ↪linewidth=2)
axes[4].legend(["Imputētās vērtības", "Orģinālās vērtības"])
```

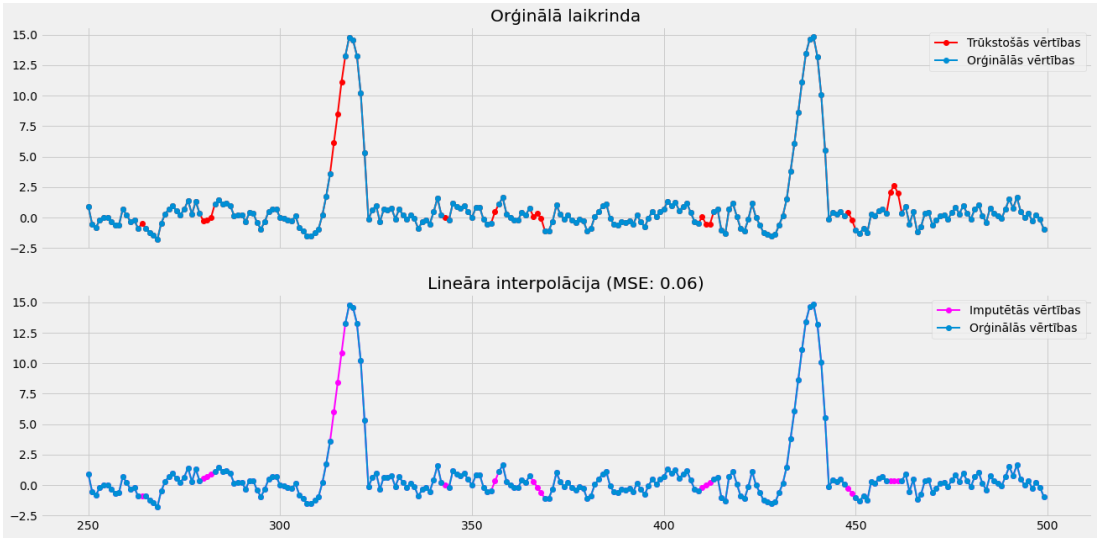
```
# https://www.machinelearningplus.com/time-series/time-series-analysis-python/
### Vidējā vērtība no pēdējiem n novērojumiem
def knn_mean(ts, n):
    out = np.copy(ts)
    for i, val in enumerate(ts):
        if np.isnan(val):
            n_by_2 = np.ceil(n/2)
            lower = np.max([0, int(i-n_by_2)])
            upper = np.min([len(ts)+1, int(i+n_by_2)])
            ts_near = np.concatenate([ts[lower:i], ts[i:upper]])
            out[i] = np.nanmean(ts_near)
    return out

df_knn = df_g1_missing.copy()
df_knn['g1_ts2'] = knn_mean(df_g1_missing.values, 8)
error = np.round(mean_squared_error(df_g1['g1_ts2'].iloc[250:500],␣
↪df_knn['g1_ts2'].iloc[250:500]), 2)
df_knn['g1_ts2'].iloc[250:500].plot(title='KNN vidējā vērtība (MSE: ' +␣
↪str(error) +")", ax=axes[5], label='KNN', color='magenta', style="o-",␣
↪linewidth=2)
df_g1_missing.iloc[250:500, :].plot(ax=axes[5], label='KNN', style="o-",␣
↪linewidth=2)
axes[5].legend(["Imputētās vērtības", "Orģinālās vērtības"])

### Sezonālā vidējā vērtība
def seasonal_mean(ts, n, lr=.75):
    """
    ts: laikrinda kā 1D masīvs
    n: laikrindas sezonas garums
    """
    out = np.copy(ts)
    for i, val in enumerate(ts):
        if np.isnan(val):
            ts_seas = ts[i-1::-n]  # previous seasons only
            if np.isnan(np.nanmean(ts_seas)):
                ts_seas = np.concatenate([ts[i-1::-n], ts[i::n]])  # previous␣
↪and forward
            out[i] = np.nanmean(ts_seas) * lr
    return out
df_sm = df_g1_missing.copy()
df_sm['g1_ts2'] = seasonal_mean(df_g1_missing.values, n=120, lr=1.25)
error = np.round(mean_squared_error(df_g1['g1_ts2'].iloc[250:500],␣
↪df_sm['g1_ts2'].iloc[250:500]), 2)
df_sm['g1_ts2'].iloc[250:500].plot(title='Sezonālā vidējā vērtība (MSE: ' +␣
↪str(error) +")", ax=axes[6], label='SM', color='magenta', style="o-",␣
↪linewidth=2)
```

```
df_g1_missing.iloc[250:500, :].plot(ax=axes[6], label='SM', style="o-",␣
 ↪linewidth=2)
axes[6].legend(["Imputētās vērtības", "Orģinālās vērtības"])

plt.savefig('images/prieksapstrade/imputation_comp.png');
```



[54]:
```
fig, axes = plt.subplots(2, 1, sharex=True, figsize=(20, 10))

### Orģinālā laikrinda!
df_g1.iloc[250:500, :].plot(title='Orģinālā laikrinda', ax=axes[0],␣
 ↪label='Actual', color='red', style="o-", linewidth=2)
df_g1_missing.iloc[250:500, :].plot(title='Orģinālā laikrinda', ax=axes[0],␣
 ↪label='Actual', style="o-", linewidth=2)
axes[0].legend(["Trūkstošās vērtības", "Orģinālās vērtības"])
```

```
### Lineāra interpolācija!
df_linear = df_g1_missing.interpolate()
error = np.round(mean_squared_error(df_g1['g1_ts2'].iloc[250:500],␣
 ↪df_linear['g1_ts2'].iloc[250:500]), 2)
df_linear['g1_ts2'].iloc[250:500].plot(title='Lineāra interpolācija (MSE: ' +␣
 ↪str(error) +")", ax=axes[1], label='Linear Fill', color='magenta',␣
 ↪style="o-", linewidth=2)
df_g1_missing.iloc[250:500, :].plot(ax=axes[1], label='Linear Fill',␣
 ↪style="o-", linewidth=2)
axes[1].legend(["Imputētās vērtības", "Orģinālās vērtības"])

plt.savefig('images/prieksapstrade/imputation_small_comp.png');
```



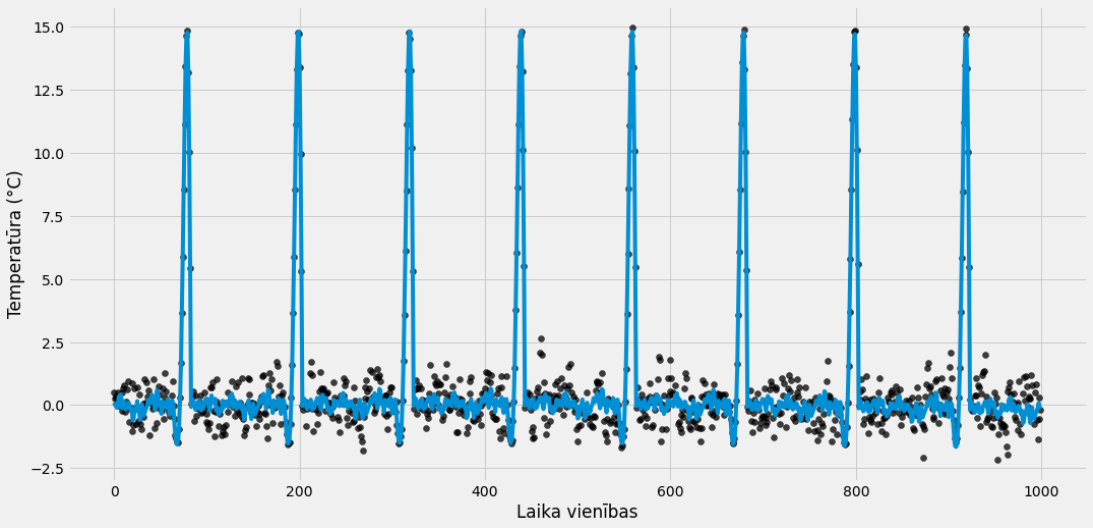
1.0.3 2. Transformācijas

[47]:
```
# ...
```

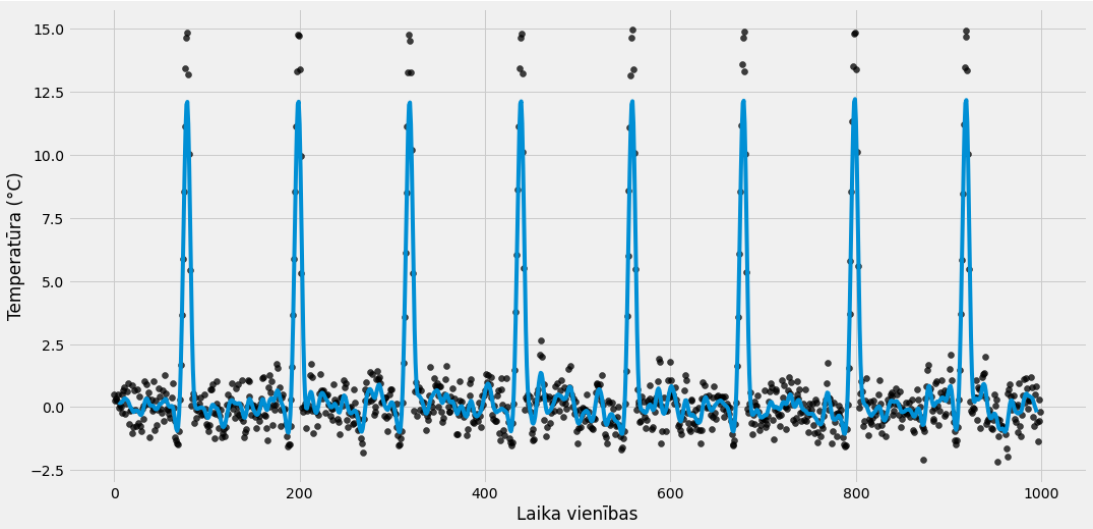
1.0.4 3. Laikrindas nogludināšana

[88]:
```
# Dekompozīcijas gludināšana
smoother = DecomposeSmoother(smooth_type='lowess', periods=120,␣
 ↪smooth_fraction=0.9)
smoother.smooth(df_g1.T)
df_decomp_smooth = pd.DataFrame(smoother.smooth_data.T, columns=df_g1.columns,␣
 ↪index=df_g1.index)
```

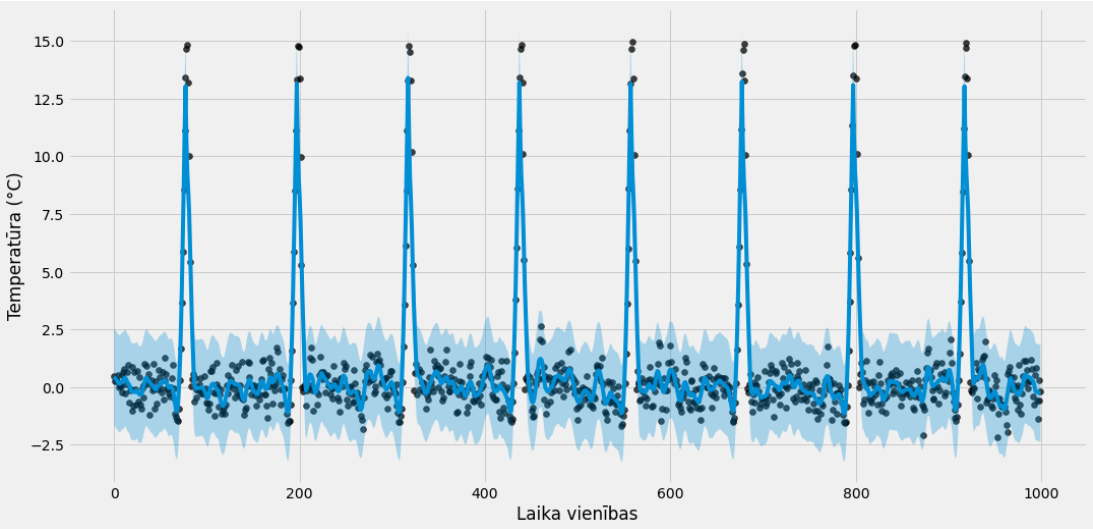
[210]:
```
plt.plot(df_decomp_smooth, alpha=1)
plt.scatter(df_g1.index, df_g1, alpha=.75, color='black')
plt.xlabel('Laika vienības')
plt.ylabel(u'Temperatūra (\u00B0C)')
plt.savefig('images/prieksapstrade/decomposition_smoothing.png');
```



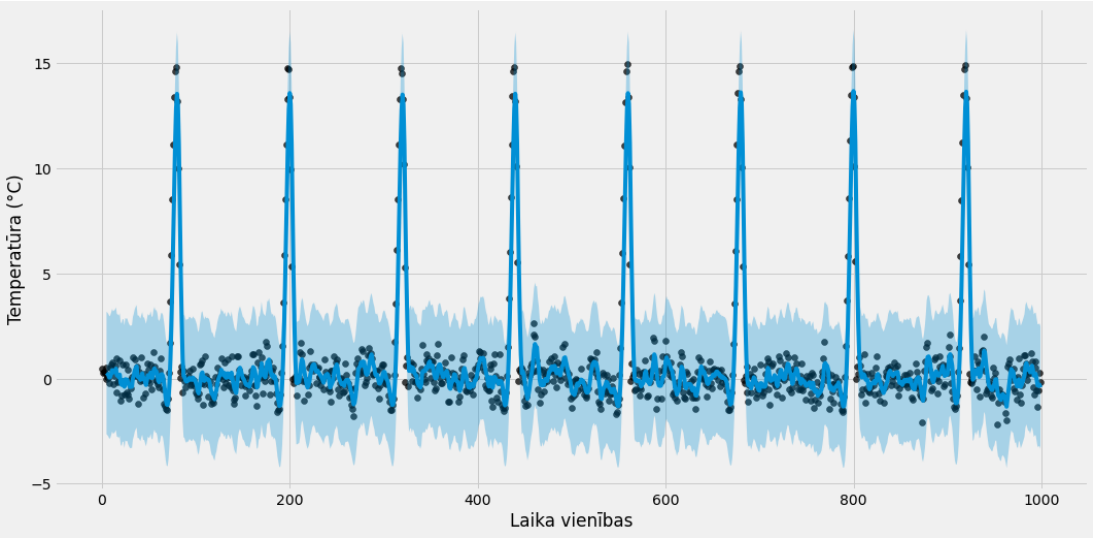
[214]:
```
# Slīdošās vidējās vērtības gludināšana
df_ma_smooth = df_g1.rolling(10, center=True, win_type='triang')\
                        .mean()\
                        .dropna(axis=0)\
                        .copy()

plt.plot(df_ma_smooth, alpha=1)
plt.scatter(df_g1.index, df_g1, alpha=.75, color='black')
plt.xlabel('Laika vienības')
plt.ylabel(u'Temperatūra (\u00B0C)')
plt.savefig('images/prieksapstrade/moving_average_smoothing.png');
```



[212]:
```
# LOWESS gludināšana
smoother = LowessSmoother(smooth_fraction=0.01, iterations=3)
smoother.smooth(df_g1.T)
df_lowess_smooth = pd.DataFrame(smoother.smooth_data.T, columns=df_g1.columns,␣
 ↪index=df_g1.index)

# Ticamības intervāli
low, up = smoother.get_intervals('sigma_interval')

# Grafiks
plt.plot(df_lowess_smooth)
plt.scatter(df_g1.index, df_g1, alpha=0.75, color='black')
plt.fill_between(df_lowess_smooth.index, low.reshape((-1)), up.reshape((-1)),␣
 ↪alpha=0.3)
plt.xlabel('Laika vienības')
plt.ylabel(u'Temperatūra (\u00B0C)')
plt.savefig('images/prieksapstrade/lowess_smoothing.png');
```



[213]:
```
# Eksponenciālā gludināšana
win_len = 5
smoother = ExponentialSmoother(window_len=win_len, alpha=.1)
smoother.smooth(df_g1.T)
df_es_smooth = pd.DataFrame(smoother.smooth_data.T, columns=df_g1.columns,␣
 ↪index=df_g1.index[win_len:])

# Ticamības intervāli
low, up = smoother.get_intervals('sigma_interval')

# Grafiks
plt.plot(df_es_smooth)
plt.scatter(df_g1.index, df_g1, alpha=0.75, color='black')
plt.fill_between(df_es_smooth.index, low.reshape((-1)), up.reshape((-1)),␣
 ↪alpha=0.3)
plt.xlabel('Laika vienības')
plt.ylabel(u'Temperatūra (\u00B0C)')
plt.savefig('images/prieksapstrade/exponential_smoothing.png');
```



[205]:
```
def SampEn(U, m, r):
    # Compute Sample entropy
    def _maxdist(x_i, x_j):
        return max([abs(ua - va) for ua, va in zip(x_i, x_j)])

    def _phi(m):
        x = [[U[j] for j in range(i, i + m - 1 + 1)] for i in range(N - m + 1)]
        C = [len([1 for j in range(len(x)) if i != j and _maxdist(x[i], x[j])␣
↪<= r]) for i in range(len(x))]
        return sum(C)

    N = len(U)
    return -np.log(_phi(m+1) / _phi(m))
```

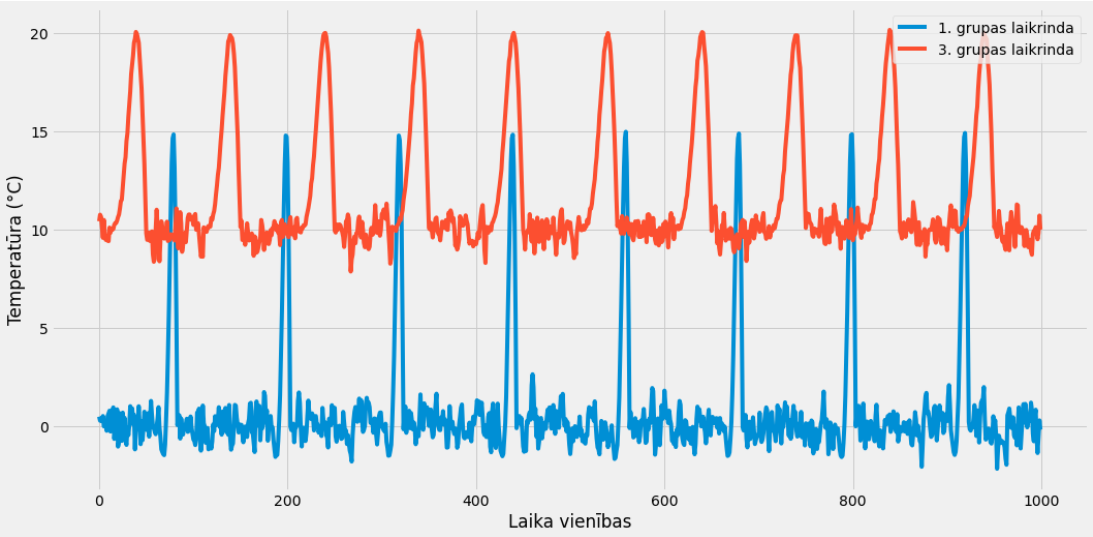
[209]:
```
# Visu četru gludināšanas metožu novērtējums izmantojot SampEn metodi!
print(f'Dekompozīcijas gludināšana: {round(SampEn(df_decomp_smooth.values, m=2,␣
↪r=0.2*np.std(df_decomp_smooth.values)),2)}')
print(f'Slīdošās vidējās vērtības gludināšana: {round(SampEn(df_ma_smooth.
↪values, m=2, r=0.2*np.std(df_ma_smooth.values)),2)}')
print(f'LOWESS gludināšana: {round(SampEn(df_lowess_smooth.values, m=2, r=0.
↪2*np.std(df_lowess_smooth.values)),2)}')
print(f'Eksponenciālā gludināšana: {round(SampEn(df_es_smooth.values, m=2, r=0.
↪2*np.std(df_es_smooth.values)),2)}')
```

```
Dekompozīcijas gludināšana: 0.09
Slīdošās vidējās vērtības gludināšana: 0.13
LOWESS gludināšana: 0.14
Eksponenciālā gludināšana: 0.21
```

1.0.5 4. Mērogošana

[8]:
```
# Apvienojam abas laikrindas vienā dataframe objektā
df = pd.concat([df_g1, df_g3], axis=1, join='inner')
```

[27]:
```
df.plot()
plt.legend(['1. grupas laikrinda', '3. grupas laikrinda'])
plt.xlabel('Laika vienības')
plt.ylabel(u'Temperatūra (\u00B0C)')
plt.savefig('images/prieksapstrade/ts_to_scale.png')
```

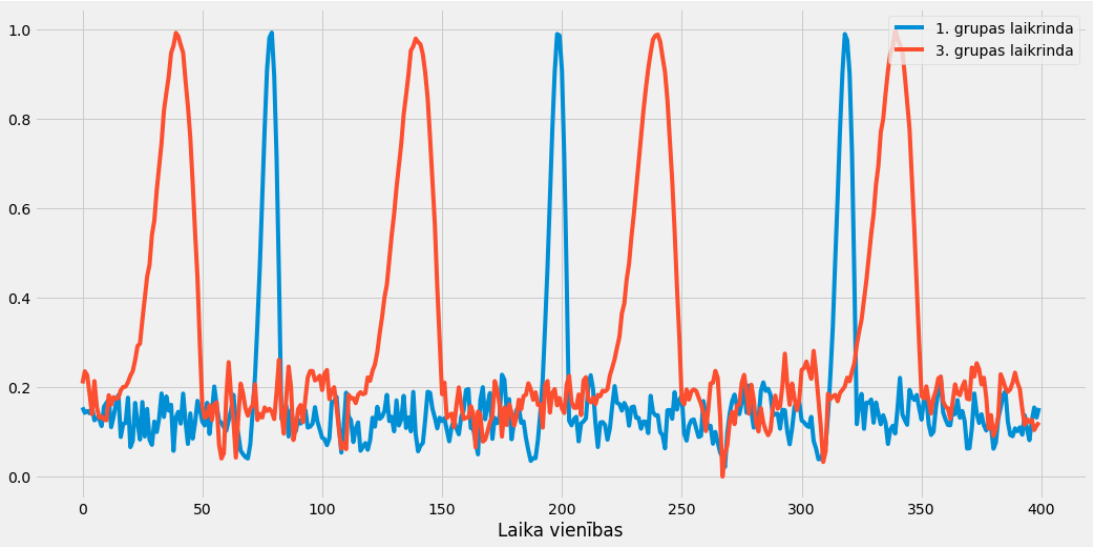


MinMaxScaler

https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.MinMaxScaler.html#sklearn.preprocessi
http://rasbt.github.io/mlxtend/user_guide/preprocessing/minmax_scaling/

[10]:
```
scaler = MinMaxScaler()
df_tmp = scaler.fit_transform(df)
df_MinMax = pd.DataFrame(df_tmp, columns=df.columns, index=df.index)
```

[31]:
```
df_MinMax[:400].plot()
plt.legend(['1. grupas laikrinda', '3. grupas laikrinda'], loc='upper right')
plt.xlabel('Laika vienības')
plt.savefig('images/prieksapstrade/minmax_scaled.png')
```



Normalization

https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.Normalizer.html#sklearn.preprocessing.

[12]:
```
scaler = Normalizer(norm='l2')
df_tmp  = scaler.fit_transform(df.T).T
df_Norm_L2 = pd.DataFrame(df_tmp, columns=df.columns, index=df.index)
```

[13]:
```
scaler = Normalizer(norm='l1')
df_tmp  = scaler.fit_transform(df.T).T
df_Norm_L1 = pd.DataFrame(df_tmp, columns=df.columns, index=df.index)
```

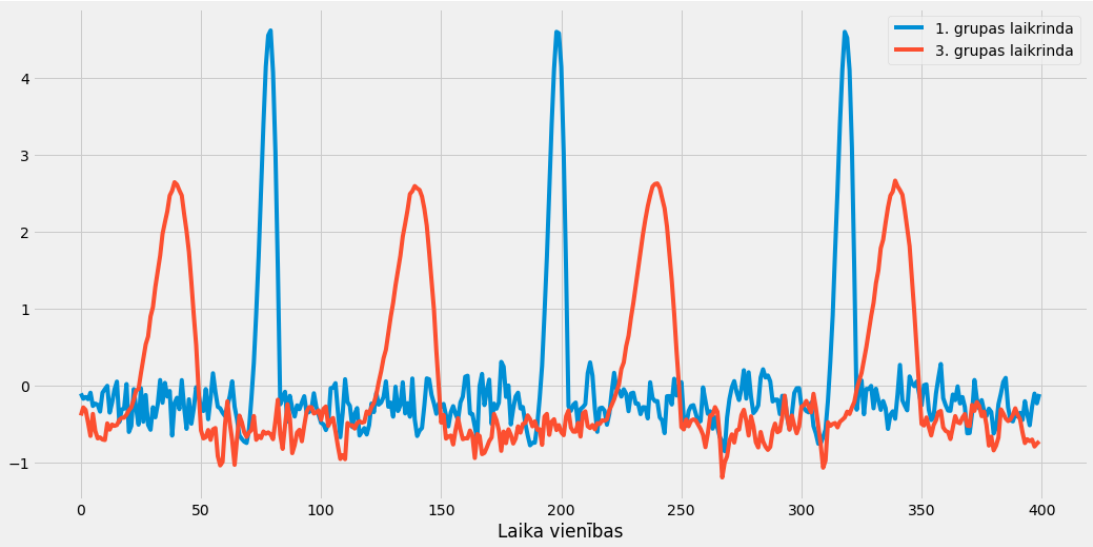
[14]:
```
scaler = Normalizer(norm='max')
df_tmp  = scaler.fit_transform(df.T).T
df_Norm_Max = pd.DataFrame(df_tmp, columns=df.columns, index=df.index)
```

Standart scaler

https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html#sklearn.preprocess

[15]:
```
scaler = StandardScaler()
df_tmp = scaler.fit_transform(df)
df_Standard = pd.DataFrame(df_tmp, columns=df.columns, index=df.index)
```

[32]:
```
df_Standard[:400].plot()
plt.legend(['1. grupas laikrinda', '3. grupas laikrinda'], loc='upper right')
plt.xlabel('Laika vienības')
plt.savefig('images/prieksapstrade/standard_scaled.png')
```

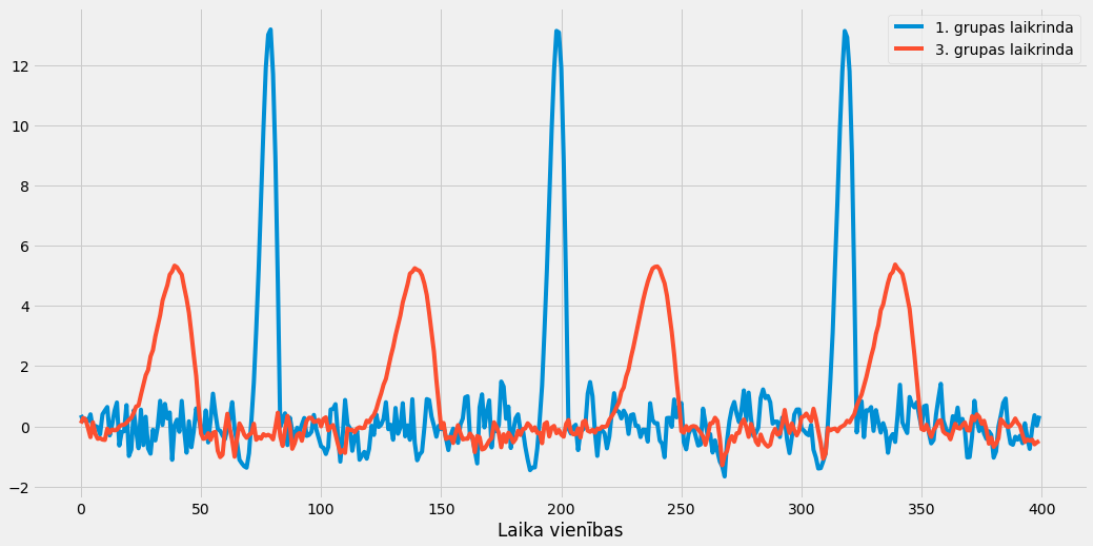


Robust scaler

https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.RobustScaler.html#sklearn.preprocessin

[17]:
```
scaler = RobustScaler()
df_tmp = scaler.fit_transform(df)
df_Robust = pd.DataFrame(df_tmp, columns=df.columns, index=df.index)
```

[33]:
```
df_Robust[:400].plot()
plt.legend(['1. grupas laikrinda', '3. grupas laikrinda'], loc='upper right')
plt.xlabel('Laika vienības')
plt.savefig('images/prieksapstrade/robust_scaled.png')
```

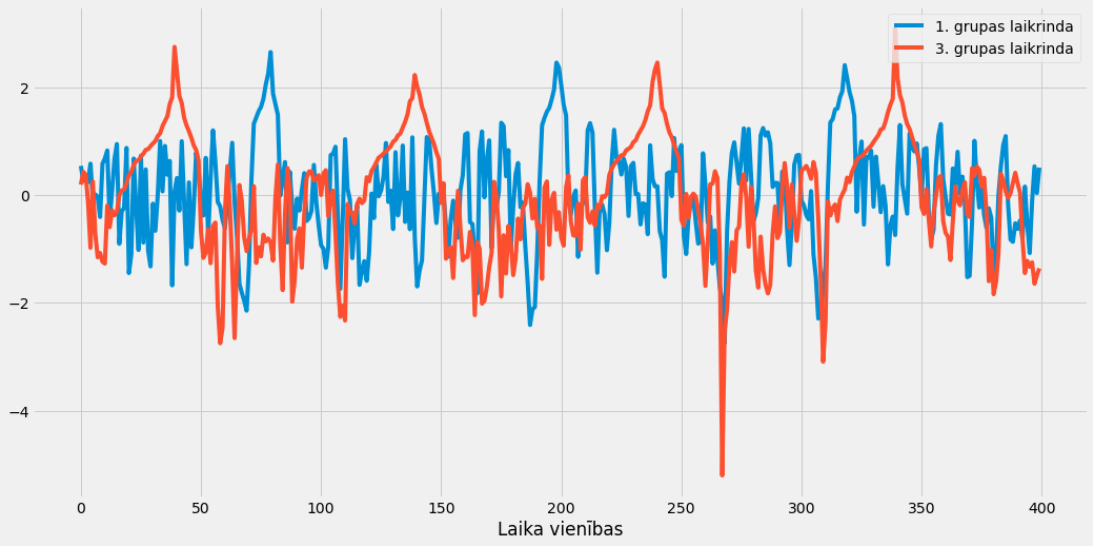


Quantile transformer

https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.QuantileTransformer.html#sklearn.prep

[19]:
```
scaler = QuantileTransformer(output_distribution='normal')
df_tmp = scaler.fit_transform(df)
df_Quantile = pd.DataFrame(df_tmp, columns=df.columns, index=df.index)
```

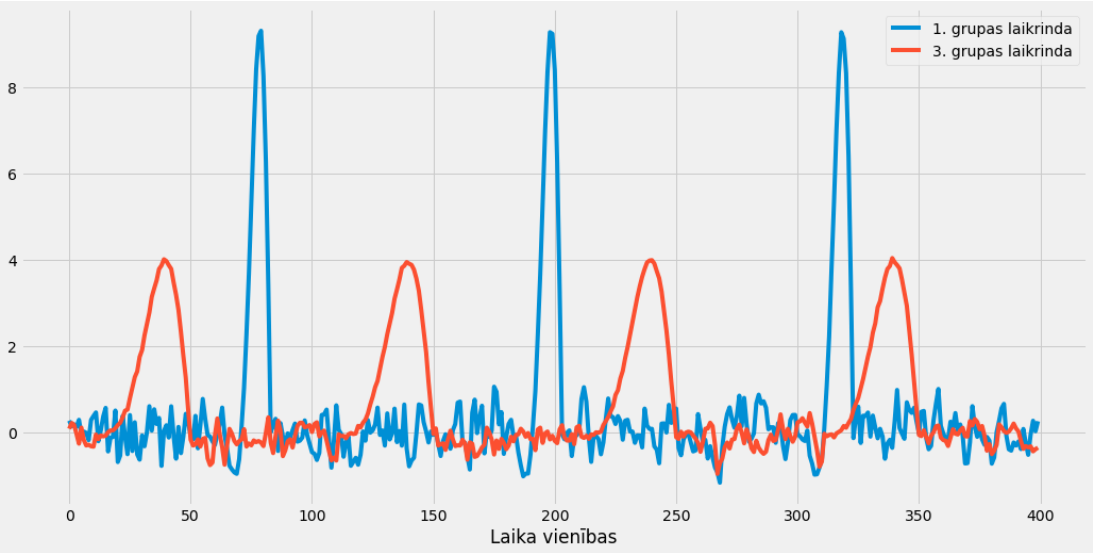
[34]:
```
df_Quantile[:400].plot()
plt.legend(['1. grupas laikrinda', '3. grupas laikrinda'], loc='upper right')
plt.xlabel('Laika vienības')
plt.savefig('images/prieksapstrade/quantile_scaled.png')
```



Median & MAD

[36]:
```
df_MM = (df - df.median(axis=0).values)/df.mad(axis=0).values
```

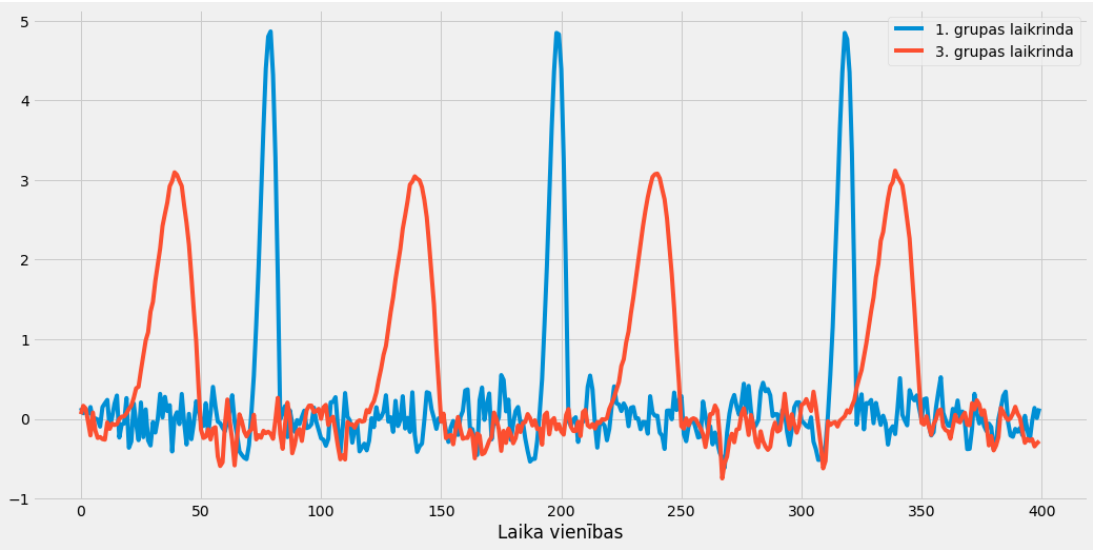
[37]:
```
df_MM[:400].plot()
plt.legend(['1. grupas laikrinda', '3. grupas laikrinda'], loc='upper right')
plt.xlabel('Laika vienības')
plt.savefig('images/prieksapstrade/median_mad_scaled.png')
```



Median & Std

[38]:
```
df_MS = (df - df.median(axis=0).values)/df.std(axis=0).values
```

[39]:
```
df_MS[:400].plot()
plt.legend(['1. grupas laikrinda', '3. grupas laikrinda'], loc='upper right')
plt.xlabel('Laika vienības')
plt.savefig('images/prieksapstrade/median_std_scaled.png')
```

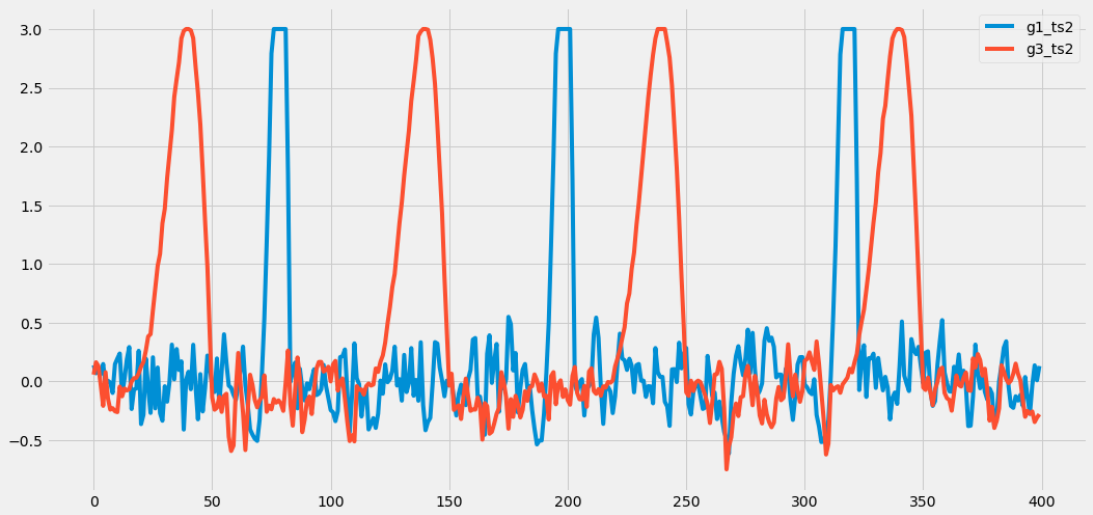


Median & Std with trunc

[40]:
```
df_std = df.std(axis=0)
df_MS_trunc = (df - df.median(axis=0).values)/df_std.values
for nm, i in zip(df_std.index, df_std.values):
    df_MS_trunc.loc[df_MS_trunc.loc[:,nm]>3, nm] = 3
    df_MS_trunc.loc[df_MS_trunc.loc[:,nm]<-3, nm] = -3
```

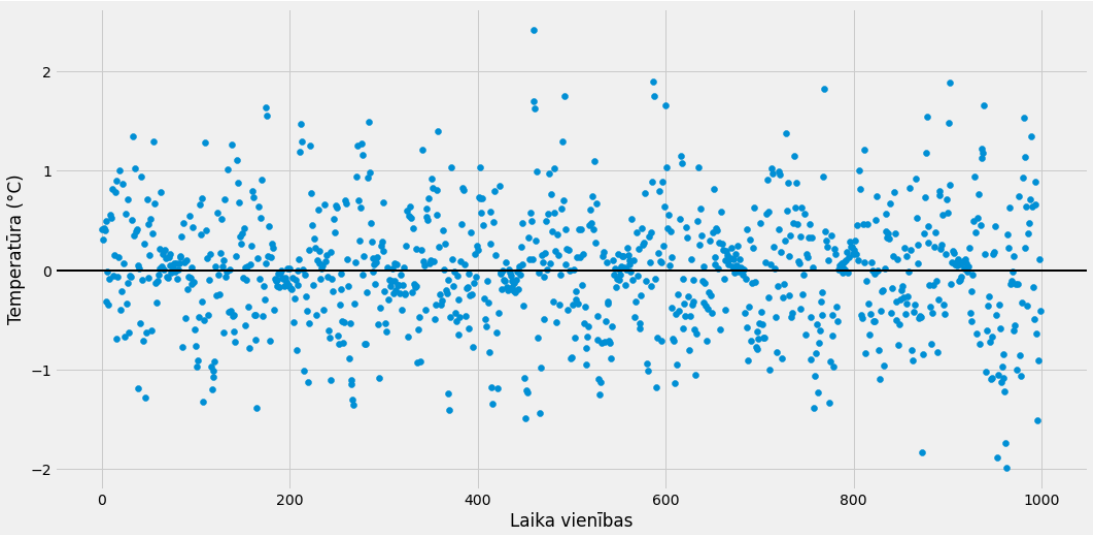
[41]:
```
df_MS_trunc[:400].plot()
```

[41]:
```
<AxesSubplot:>
```



1.0.6 Anomāliju likvidēšana treniņu datu kopā

[52]:
```
# Adtīva dekompozīcija
add_res = seasonal_decompose(df_g1, model='additive', extrapolate_trend='freq',␣
 ↪period=120)
# !!! extrapolate_trend nodrošina trūkstošu vērtību imputāciju !!!

plt.scatter(add_res.observed.index, add_res.resid)
plt.axhline(linewidth=2, color='black')
plt.xlabel('Laika vienības')
plt.ylabel(u'Temperatūra (\u00B0C)');
```

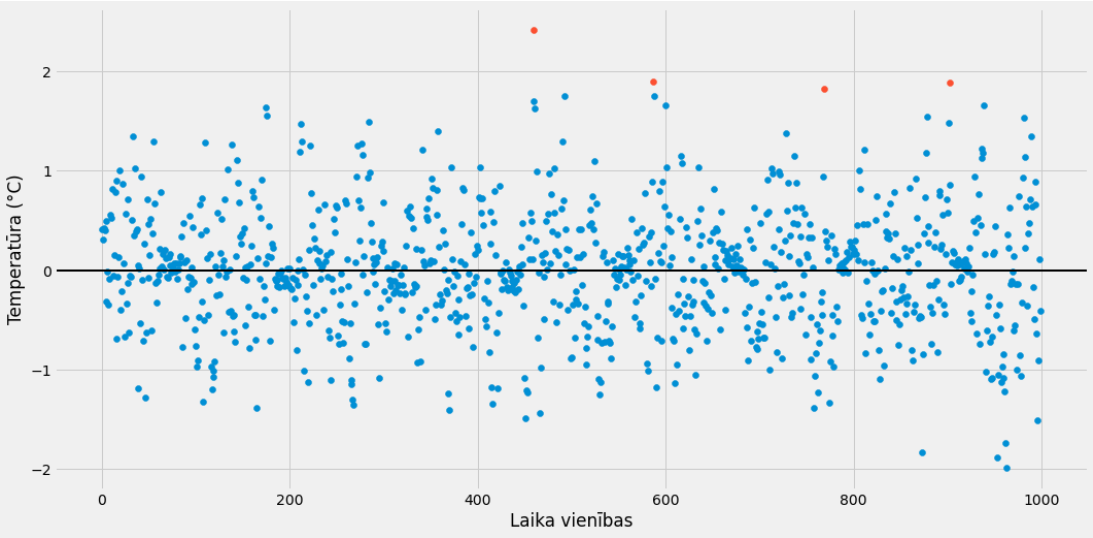


[116]:
```
print(f'Vidējā vērtība: {statistics.mean(add_res.resid)}')
print(f'Standartnovrize: {statistics.stdev(add_res.resid)*3}')
```

```
Vidējā vērtība: 0.0025378924398701
Standartnovrize: 1.7929603103443597
```

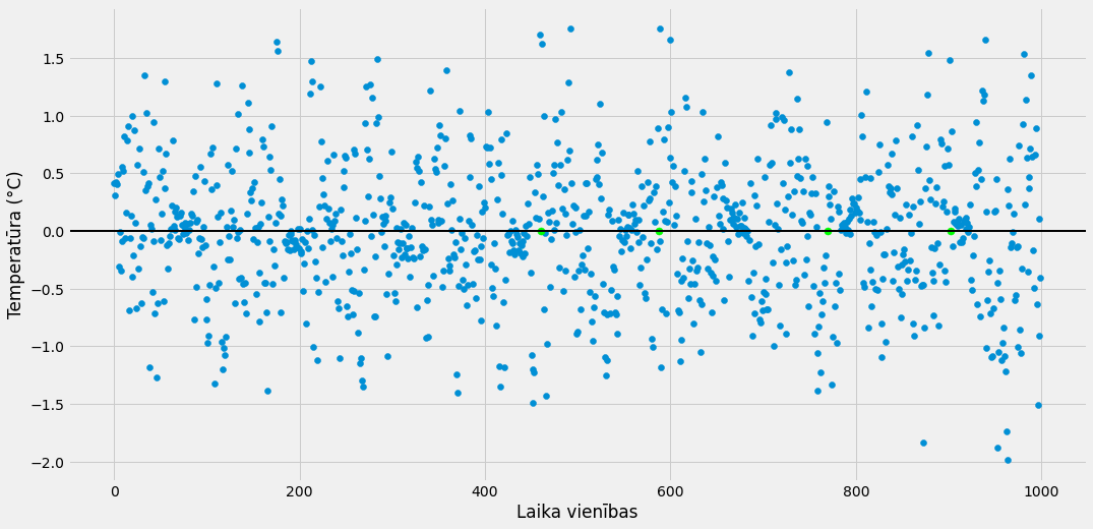
[82]:
```
resid_good = resid[resid <= statistics.stdev(add_res.resid)*3]
resid_bad = resid[resid > statistics.stdev(add_res.resid)*3]
```

[124]:
```
plt.scatter(resid_good.index, resid_good)
plt.scatter(resid_bad.index, resid_bad)
plt.axhline(linewidth=2, color='black')
plt.xlabel('Laika vienības')
plt.ylabel(u'Temperatūra (\u00B0C)')
plt.savefig('images/prieksapstrade/resid_with_bad.png')
```



[114]:
```
tmp = []
for i in range(len(resid_bad)):
    #tmp.append(statistics.stdev(add_res.resid)*3)
    tmp.append(statistics.mean(add_res.resid))
```

[125]:
```
plt.scatter(resid_good.index, resid_good)
plt.scatter(resid_bad.index, tmp, color='lime', s = 50)
plt.axhline(linewidth=2, color='black')
plt.xlabel('Laika vienības')
plt.ylabel(u'Temperatūra (\u00B0C)')
plt.savefig('images/prieksapstrade/resid_without_bad.png')
```



B.4. Klāsterēšanas koda darba grāmata

1 Klāsterēšana

[87]:
```
# Pakotņu ielāde un iestatījumi
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from tsmoothie.smoother import DecomposeSmoother, LowessSmoother
from tsmoothie import WindowWrapper
from tslearn.clustering import TimeSeriesKMeans, KShape, KernelKMeans
from tslearn.barycenters import dtw_barycenter_averaging, softdtw_barycenter
from tslearn.metrics import dtw
from dtaidistance.clustering import KMeans, Hierarchical, BaseTree, KMedoids,␣
 ↪Hooks, HierarchicalTree, LinkageTree
#from dtaidistance import dtw
import joblib # modeļu saglabāšanai!
import time
import math
from itertools import permutations
import pickle


%matplotlib inline
plt.style.use('fivethirtyeight') # Grafiku stils!
plt.rcParams["figure.figsize"] = (16,8) # Grafiku izmērs
```

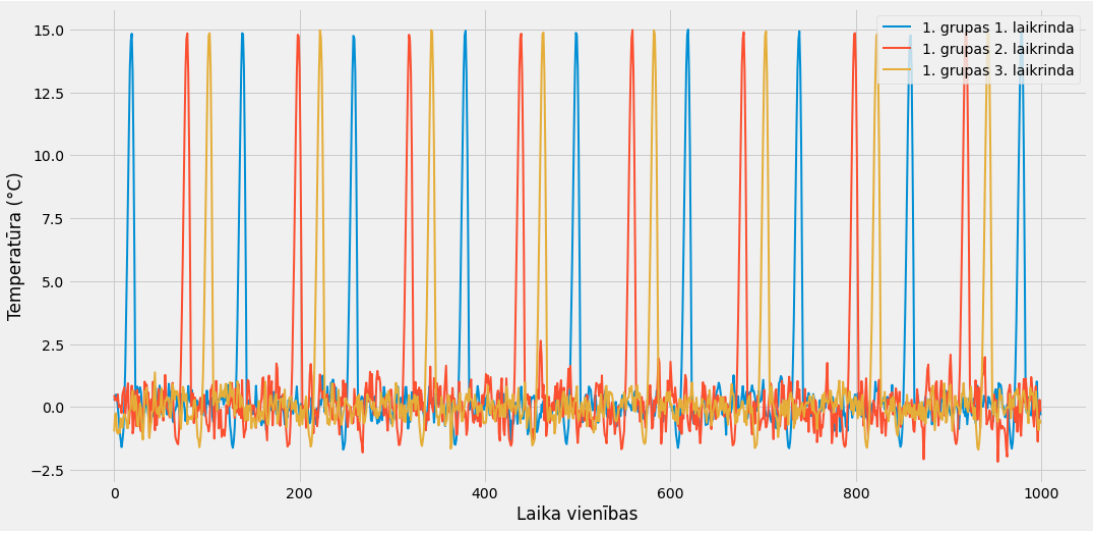
2 Datu ielāde

[2]:
```
# Simulētie temperatūras mērījumu dati
ts_g1 = pd.read_csv('data/sim_ts_g1.csv', index_col=0)
ts_g2 = pd.read_csv('data/sim_ts_g2.csv', index_col=0)
ts_g3 = pd.read_csv('data/sim_ts_g3.csv', index_col=0)

df = pd.concat([ts_g1, ts_g2, ts_g3], axis=1)
```

3 Datu vizualizācija

[3]:
```
df.loc[:, ['g1_ts1', 'g2_ts1', 'g3_ts1']].plot(linewidth=2)
plt.legend(['1. grupas laikrinda',
            '2. grupas laikrinda',
            '3. grupas laikrinda'
           ], loc='upper right')
plt.xlabel('Laika vienības')
plt.ylabel(u'Temperatūra (\u00B0C)');
plt.savefig('images/klasteresana/all_3_ts_plot.png')
```

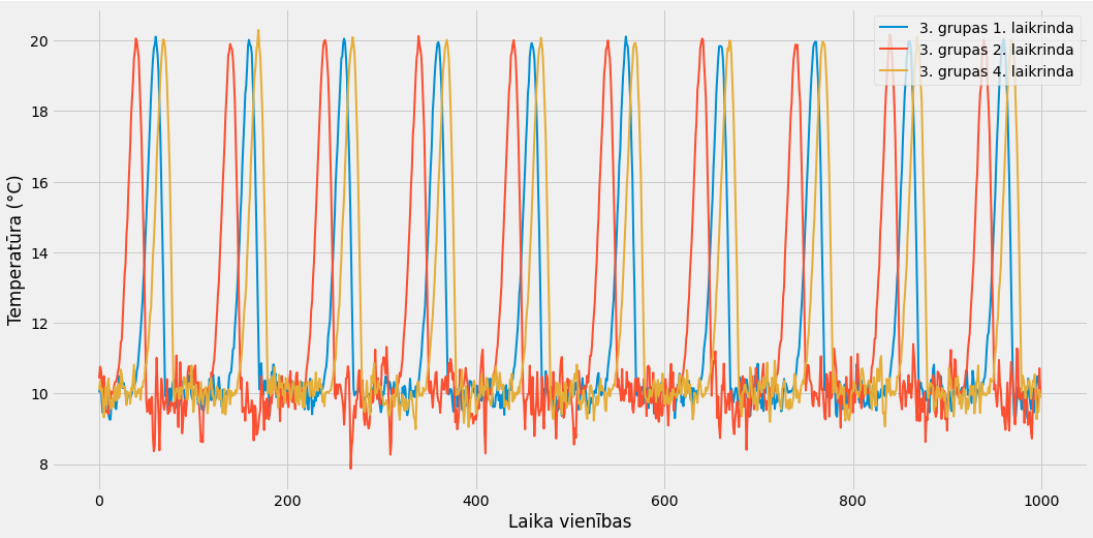
[4]:
```
# 1. grupas laikrindu mērogošanas attēlojums
df.loc[:, ['g1_ts1', 'g1_ts2', 'g1_ts3']].plot(linewidth=2)
plt.legend(['1. grupas 1. laikrinda',
            '1. grupas 2. laikrinda',
            '1. grupas 3. laikrinda'
           ], loc='upper right')
plt.xlabel('Laika vienības')
plt.ylabel(u'Temperatūra (\u00B0C)');
plt.savefig('images/klasteresana/group_1_ts_comp.png')
```



2

[5]:
```
# 2. grupas laikrindu mērogošanas attēlojums
df.loc[:, ['g2_ts1', 'g2_ts2', 'g2_ts3']].plot(linewidth=2)
plt.legend(['2. grupas 1. laikrinda',
            '2. grupas 2. laikrinda',
            '2. grupas 3. laikrinda'
           ], loc='upper right')
plt.xlabel('Laika vienības')
plt.ylabel(u'Temperatūra (\u00B0C)');
plt.savefig('images/klasteresana/group_2_ts_comp.png')
```

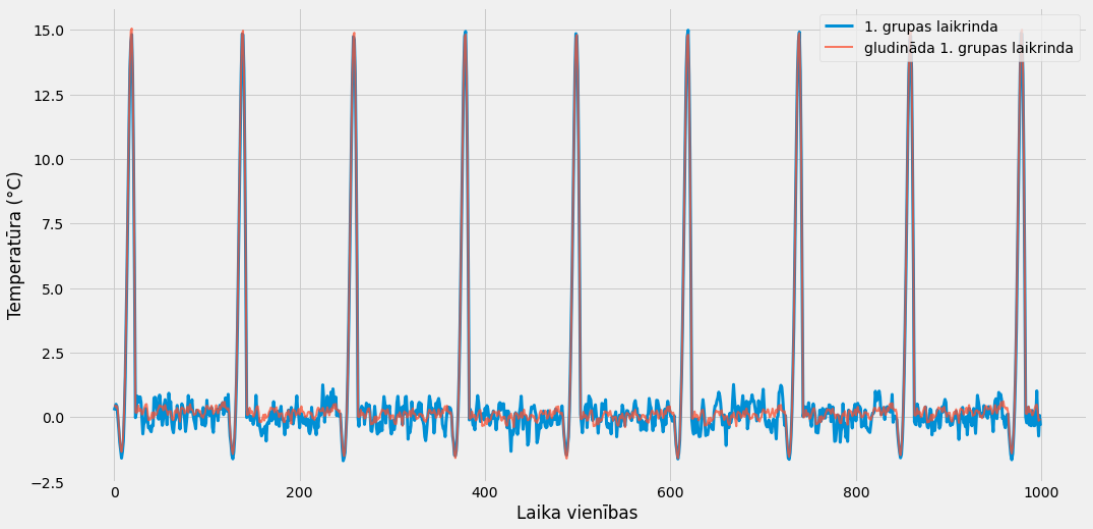
[6]:
```
# 3. grupas laikrindu mērogošanas attēlojums
df.loc[:, ['g3_ts1', 'g3_ts2', 'g3_ts4']].plot(linewidth=2)
plt.legend(['3. grupas 1. laikrinda',
            '3. grupas 2. laikrinda',
            '3. grupas 4. laikrinda'
           ], loc='upper right')
plt.xlabel('Laika vienības')
plt.ylabel(u'Temperatūra (\u00B0C)');
plt.savefig('images/klasteresana/group_3_ts_comp.png')
```



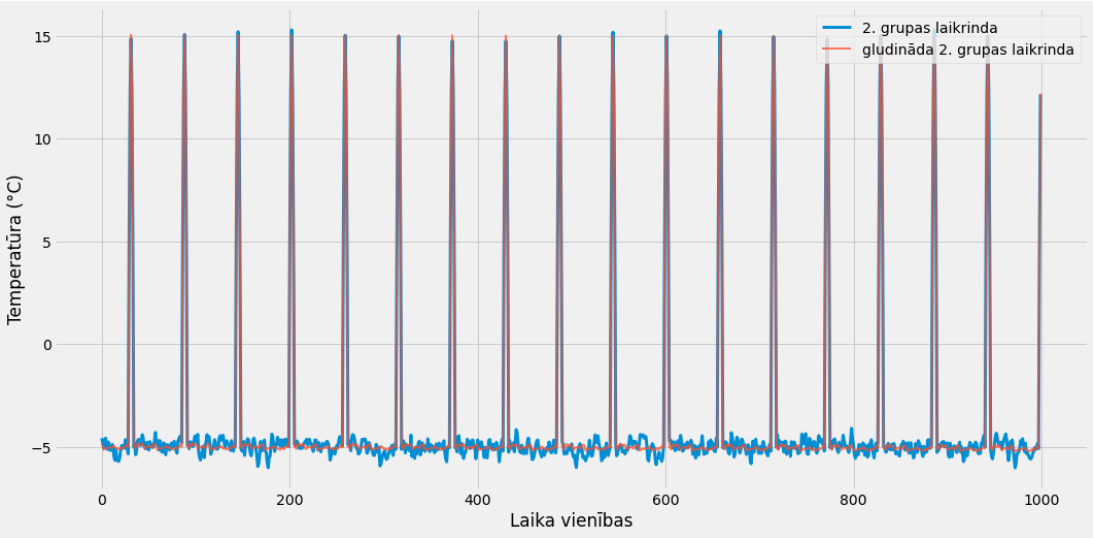
4 Datu sagatavošana un priekšapstrāde

[8]:
```
'''
Laikrindu gludināšana izmantojot dekompozīcijas metodi!
'''
# 1. grupas laikrindu gludināšana!
smoother = DecomposeSmoother(smooth_type='lowess', periods=120,␣
 ↪smooth_fraction=0.9)
smoother.smooth(ts_g1.T)
df_g1_smooth = pd.DataFrame(smoother.smooth_data.T, columns=ts_g1.columns,␣
 ↪index=ts_g1.index)

# 2. grupas laikrindu gludināšana!
smoother = DecomposeSmoother(smooth_type='lowess', periods=57,␣
 ↪smooth_fraction=0.9)
smoother.smooth(ts_g2.T)
df_g2_smooth = pd.DataFrame(smoother.smooth_data.T, columns=ts_g2.columns,␣
 ↪index=ts_g2.index)

# 3. grupas laikrindu gludināšana!
smoother = DecomposeSmoother(smooth_type='lowess', periods=100,␣
 ↪smooth_fraction=0.9)
smoother.smooth(ts_g3.T)
df_g3_smooth = pd.DataFrame(smoother.smooth_data.T, columns=ts_g3.columns,␣
 ↪index=ts_g3.index)
```

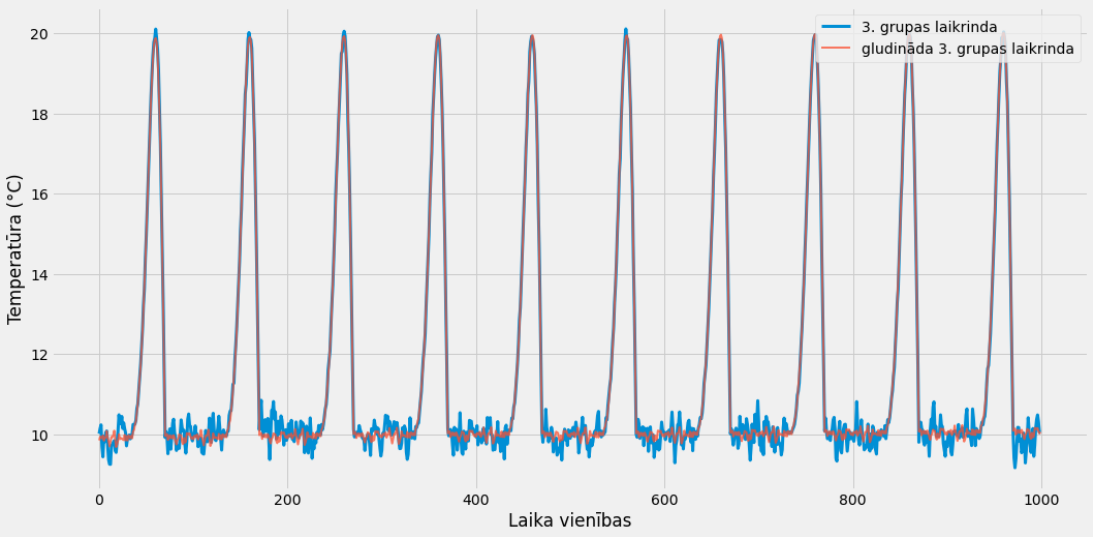
[9]:
```
# Gludināta 1. grupas laikrinda
plt.plot(ts_g1.iloc[:, 0], linewidth=3)
plt.plot(df_g1_smooth.iloc[:, 0], linewidth=2, alpha = .75)
plt.legend(['1. grupas laikrinda', 'gludināda 1. grupas laikrinda'], loc='upper␣
↪right')
plt.xlabel('Laika vienības')
plt.ylabel(u'Temperatūra (\u00B0C)');
plt.savefig('images/klasteresana/smooth_group_1_ts.png')
```



[10]:
```
# Gludināta 2. grupas laikrinda
plt.plot(ts_g2.iloc[:, 0], linewidth=3)
plt.plot(df_g2_smooth.iloc[:, 0], linewidth=2, alpha = .75)
plt.legend(['2. grupas laikrinda', 'gludināda 2. grupas laikrinda'], loc='upper␣
↪right')
plt.xlabel('Laika vienības')
plt.ylabel(u'Temperatūra (\u00B0C)');
plt.savefig('images/klasteresana/smooth_group_2_ts.png')
```



[11]:
```
# Gludināta 3. grupas laikrinda
plt.plot(ts_g3.iloc[:, 0], linewidth=3)
plt.plot(df_g3_smooth.iloc[:, 0], linewidth=2, alpha = .75)
plt.legend(['3. grupas laikrinda', 'gludināda 3. grupas laikrinda'], loc='upper␣
↪right')
plt.xlabel('Laika vienības')
plt.ylabel(u'Temperatūra (\u00B0C)');
plt.savefig('images/klasteresana/smooth_group_3_ts.png')
```



4.0.1 !!! Apvienojam gludinātu laikrindu datu tabulas !!!

[36]:
```
ts_g1
```

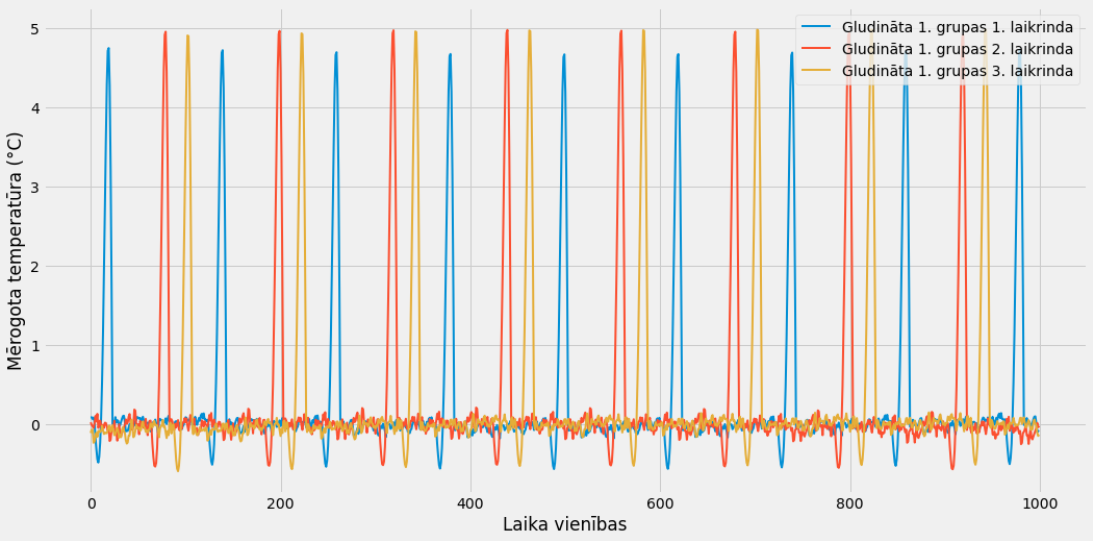
[36]:
```
       g1_ts1    g1_ts2    g1_ts3    g1_ts4    g1_ts5    g1_ts6    g1_ts7  \
0    0.263231  0.481515 -0.968285  0.523960 -0.443904  0.075724 -0.577119
1    0.375374  0.273325 -0.294613  0.163956  0.371454 -0.143816  0.532266
2    0.498885  0.340814 -0.240644 -0.069624  0.116820 -1.221406  1.287912
3    0.420796  0.255797 -0.982761  0.059856  0.095570 -1.375434  0.924860
4   -0.015273  0.510213 -1.057493 -0.443404  0.589535 -2.177723 -0.019701
..        …         …         …         …         …         …         …
995  1.019140 -0.584813  0.120834  0.165131 -0.504727 -2.256603 -0.011807
996 -0.044104 -1.369516 -0.685693  0.508555 -0.006078 -1.435184  1.571555
997 -0.728295 -0.564691 -0.915917  0.444065 -0.030191  0.725217  3.525535
998  0.070462  0.289490 -0.499239  0.588268 -0.304850  0.010786  5.845969
999 -0.324267 -0.174443 -0.653087  0.695761 -1.087453  0.675479  8.680356

       g1_ts8    g1_ts9   g1_ts10
0   -0.323540  0.991745  1.322013
1   -0.696787  0.924249  0.457556
2   -0.647288  0.640992  0.042196
3    1.473629 -0.014073  1.281025
4    0.191907 -0.402254  1.927115
..        …         …         …
995  0.063187  0.169363  0.125199
996 -0.096683 -0.800816 -1.625362
997  0.324860  0.679876 -0.408811
998  0.797122  1.461493 -0.476978
999  1.108980  0.471445 -0.080034

[1000 rows x 10 columns]
```

[12]:
```
df_smooth = pd.concat([df_g1_smooth, df_g2_smooth, df_g3_smooth], axis=1)
```

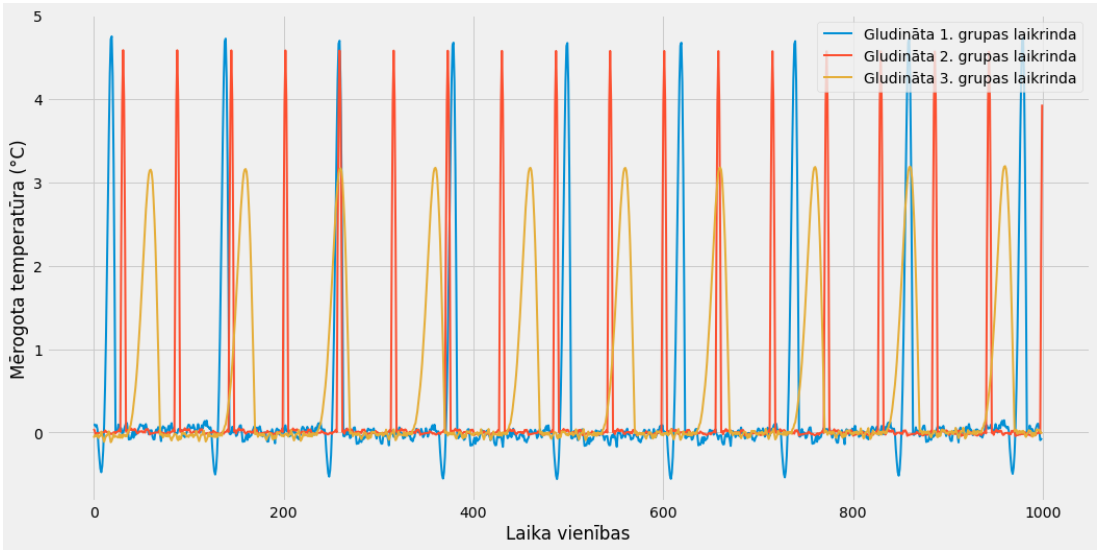
[13]:
```
'''
Laikrindu mērogošanai tiek izmantota mērogošana izmantojot MAD un␣
 ↪standartnovirzi!
'''
# Tā kā visām grupām tiek izmantota vien mērogošanas metode, tad nav␣
 ↪nepieciešams mērogot atsevišķi katras grupas tabulas
# Piezīme: izmantot gludinātos datus!!!
df_SS = (df_smooth - df_smooth.median(axis=0).values)/df_smooth.std(axis=0).
 ↪values
```

[14]:
```
# 1. grupas gludinātu laikrindu salīdzinājums!
df_SS.loc[:, ['g1_ts1', 'g1_ts2', 'g1_ts3']].plot(linewidth=2)
plt.legend(['Gludināta 1. grupas 1. laikrinda',
```

```
            'Gludināta 1. grupas 2. laikrinda',
            'Gludināta 1. grupas 3. laikrinda'
           ], loc='upper right')
plt.xlabel('Laika vienības')
plt.ylabel(u'Mērogota temperatūra (\u00B0C)');
plt.savefig('images/klasteresana/ss_group_1_ts_comp.png')
```



[15]:
```
# Visu grupu salīdzininājums!
df_SS.loc[:, ['g1_ts1', 'g2_ts1', 'g3_ts1']].plot(linewidth=2)
plt.legend(['Gludināta 1. grupas laikrinda',
            'Gludināta 2. grupas laikrinda',
            'Gludināta 3. grupas laikrinda'
           ], loc='upper right')
plt.xlabel('Laika vienības')
plt.ylabel(u'Mērogota temperatūra (\u00B0C)');
plt.savefig('images/klasteresana/ss_all_group_ts_comp.png')
```



5 Laikrindu klāsterēšana

[15]:
```
# Sagatavojam datus nepieciešamajā formātā!
df_train = df_SS.T.values.reshape(df_SS.shape[1], df_SS.shape[0], 1)

# Nodefinējam klāsteru skaitu!
clusters = 3

# Nodefinējam vektoru ar klāsteru klasēm!
c = [0] * 10 + [1] * 10 + [2] * 10
```

[31]:
```
# Klāsteru grafiku zīmēšanas funkcijas
def clust_plot(model, data, ylab='Mērogota temperatūra', xlab='Laika vienības'):
    colors = {0:'red',1:'blue',2:'green',3:'orange',
          4:'magenta',5:'blue',6:'green',7:'orange',
          8:'red',9:'blue'}

    plt.figure(figsize=(24,12))

    for c in range(model.n_clusters):
        plt.subplot(2, 2, c+1) #math.ceil(model.n_clusters/2)
        plt.plot(np.squeeze(data[model.labels_ == c],-1).T, c=colors[c],␣
↪alpha=0.2)
        plt.plot(np.squeeze(model.cluster_centers_, -1)[c], c='black',␣
↪linewidth=3)
        plt.title(f"{c+1}. klāsteris")
        plt.xlabel(xlab)
```

```
        plt.ylabel(ylab)

    plt.tight_layout(pad=3.0)
        #plt.show()
    return None
```

[123]:
```
# Klāsteru grafiku zīmēšanas funkcijas (atsevišķos grafikos) un sagalabā grafiku
def clust_sep_plot(model, data, method_name, ylab='Mērogota temperatūra',␣
↪xlab='Laika vienības',
                   save_path = 'images/klasteresana/',
                   save_name = 'time_series_clusters'
                ):
    colors = {0:'red',1:'blue',2:'green',3:'orange',
          4:'magenta',5:'blue',6:'green',7:'orange',
          8:'red',9:'blue'}

    #plt.figure(figsize=(24,12))

    for c in range(model.n_clusters):
        #plt.subplot(2, 2, c+1) #math.ceil(model.n_clusters/2)
        plt.plot(np.squeeze(data[model.labels_ == c],-1).T, c=colors[c],␣
↪alpha=0.2)
        plt.plot(np.squeeze(model.cluster_centers_, -1)[c], c='black',␣
↪linewidth=3)
        #plt.title(f"{c+1}. klāsteris")
        plt.xlabel(xlab)
        plt.ylabel(ylab)
        plt.savefig(f'{save_path}{save_name}_{c}_{method_name}.png')
        plt.show()

    #plt.tight_layout(pad=3.0)
    #plt.show()
    return None
```

[18]:
```
df_train.shape # 30 laikrindas, katra ar 1000 novērojumiem
```

[18]:
```
(30, 1000, 1)
```

[19]:
```
### Definējam modeļus ###
# k-means ar DTW
km = TimeSeriesKMeans(n_clusters=clusters,
                      metric="dtw",
                      n_jobs = 5,
                      max_iter=100,
                      n_init=3,
                      #random_state=42
                     )
```

```
# baricentru k-means ar DTW
# dba_km
dba_km = TimeSeriesKMeans(n_clusters=clusters,
                          n_init=3,
                          metric="dtw",
                          n_jobs = 5,
                          max_iter_barycenter=20,
                          #random_state=42
                         )

# k-means ar soft dtw
# sdtw_km
sdtw_km = TimeSeriesKMeans(n_clusters=clusters,
                           metric="softdtw",
                           metric_params={"gamma": .01},
                           n_jobs = 5,
                           #verbose=True,
                           #random_state=42
                          )

# k-means ar eiklīda distanci
# euc_km
euc_km = TimeSeriesKMeans(n_clusters=clusters,
                          metric="euclidean",
                          #verbose=True,
                          #random_state=42
                         )
```

1. DTW attālums

[20]:
```
# Trenējam k-means metodi ar dtw distances mēru
start = time.time()
km.fit_predict(df_train);
print(f'Modelis trenējās {round((time.time()-start)/60,2)} minūtes!')
```

```
Modelis trenējās 1.09 minūtes!
```

[21]:
```
unique, counts = np.unique(km.labels_, return_counts=True)
for i, j in zip(unique, counts):
    print(f'{i} - {j} laikrindas')
```

```
0 - 10 laikrindas
1 - 10 laikrindas
2 - 10 laikrindas
```

[22]:
```
# Modeļa saglabāšana!
# joblib.dump(km, '../models/KM_Clustering_model.pkl')
# km = joblib.load('../models/KM_Clustering_model.pkl')
```

[26]:
```
km.labels_
```

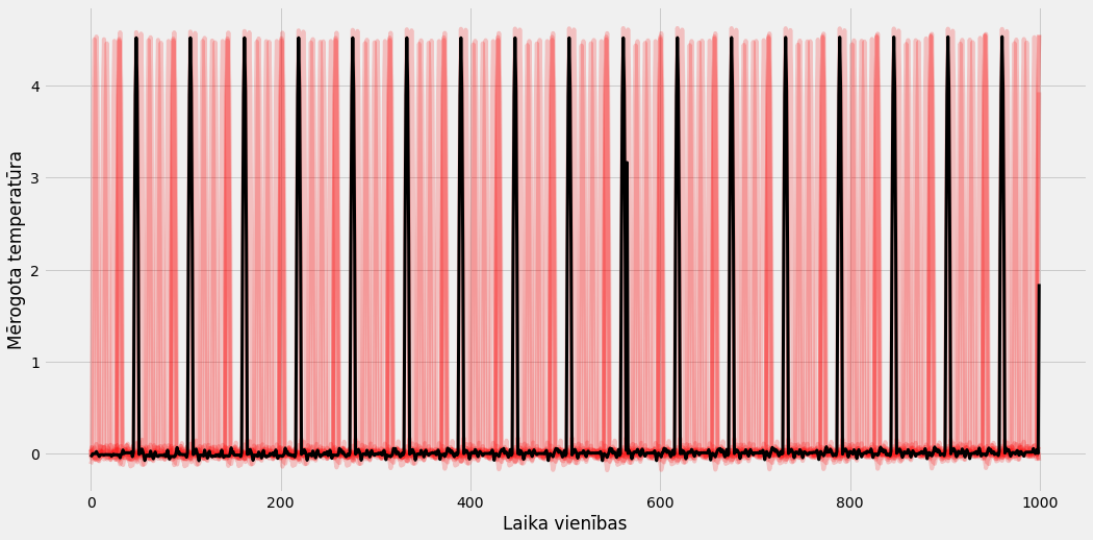
[26]:
```
array([2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1,
       1, 1, 1, 1, 1, 1, 1], dtype=int64)
```

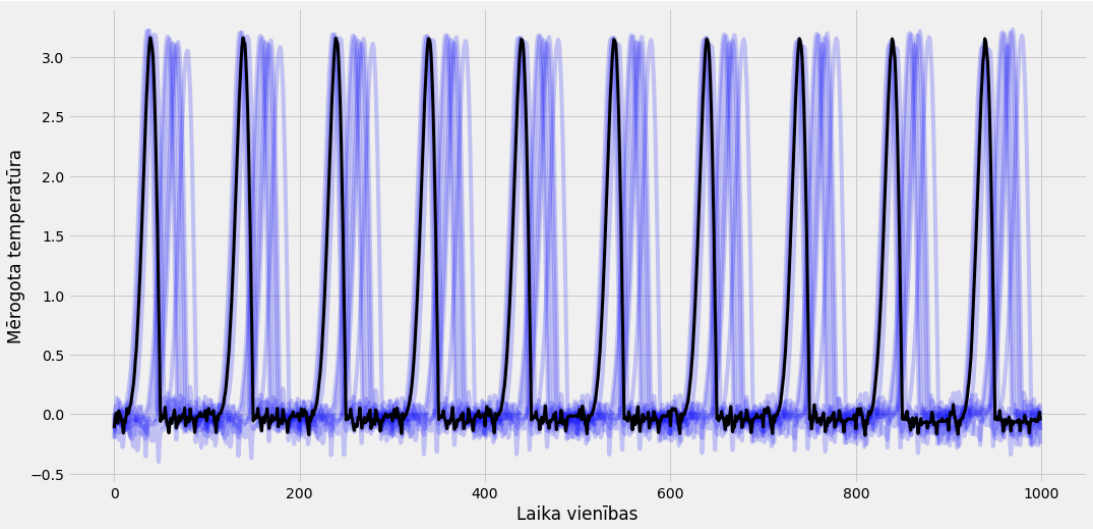
[117]:
```
# Aprēķinam klāsterēšanas preciziāti!
max_prec = 0
for i in permutations(set(km.labels_), len(set(km.labels_))):
    c = []
    for j in i:
        c.extend([j]*10)
    prec = sum(km.labels_ == c)/len(km.labels_)*100
    if prec > max_prec:
        max_prec = prec
print(f'Klāsterēšanas precizitāte {round(max_prec, 2)} %')
```

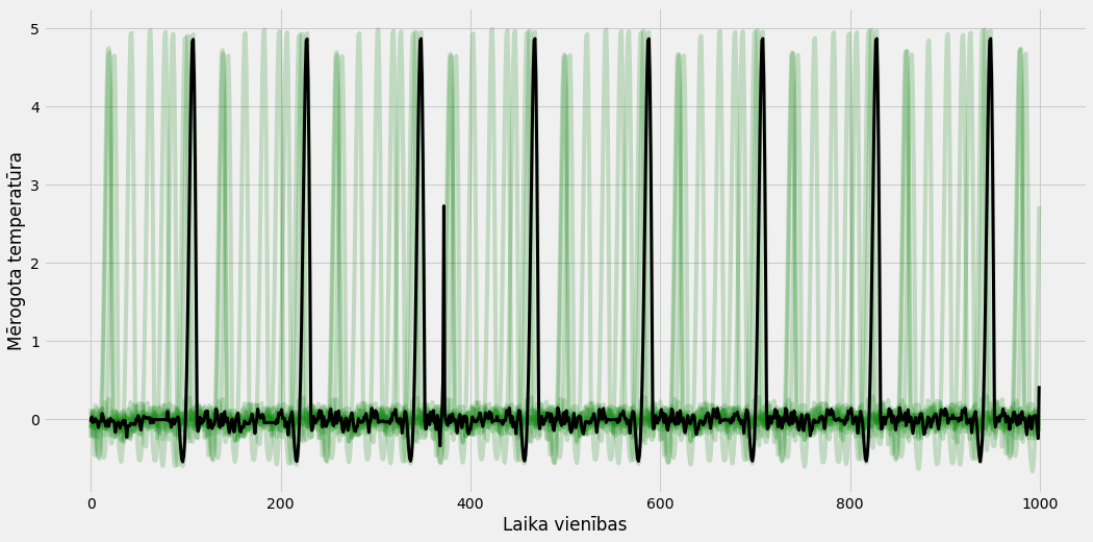
```
Klāsterēšanas precizitāte 100.0 %
```

[40]:
```
#clust_plot(km, df_train)
#plt.savefig('images/klasteresana/time_series_clusters.png')
```

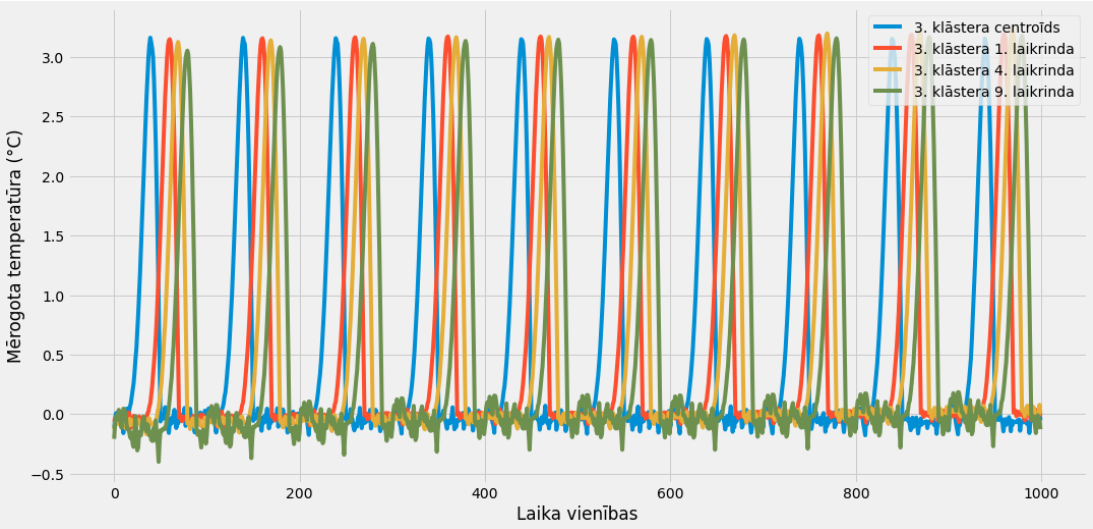
[124]:
```
# Zīmē atsevišķis klāsteru grafikus un saglabā tos
clust_sep_plot(km, df_train, method_name='DTW')
```







[71]:
```
plt.plot(km.cluster_centers_[1])
plt.plot(df_SS.loc[:, ['g3_ts1', 'g3_ts4', 'g3_ts9']])
plt.legend(['3. klāstera centroīds', '3. klāstera 1. laikrinda', '3. klāstera 4.
 ↪ laikrinda', '3. klāstera 9. laikrinda'], loc='upper right');
plt.xlabel('Laika vienības');
plt.ylabel(u'Mērogota temperatūra (\u00B0C)');
plt.savefig('images/klasteresana/group_3_ts_with_centroid.png')
```



[72]:
```
# Baricentru aprēķins
```

[146]:
```
df_train[0, :, :].shape
```

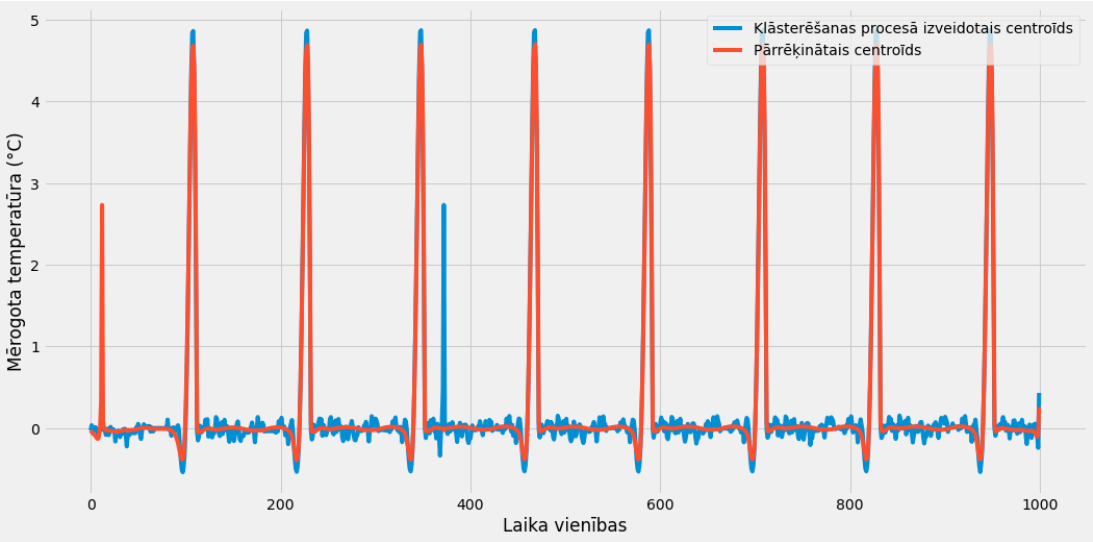
[146]:
```
(1000, 1)
```

!!! Nomainīt init uz attiecīgās laikrindas klāstera centroīda vērtībām !!!

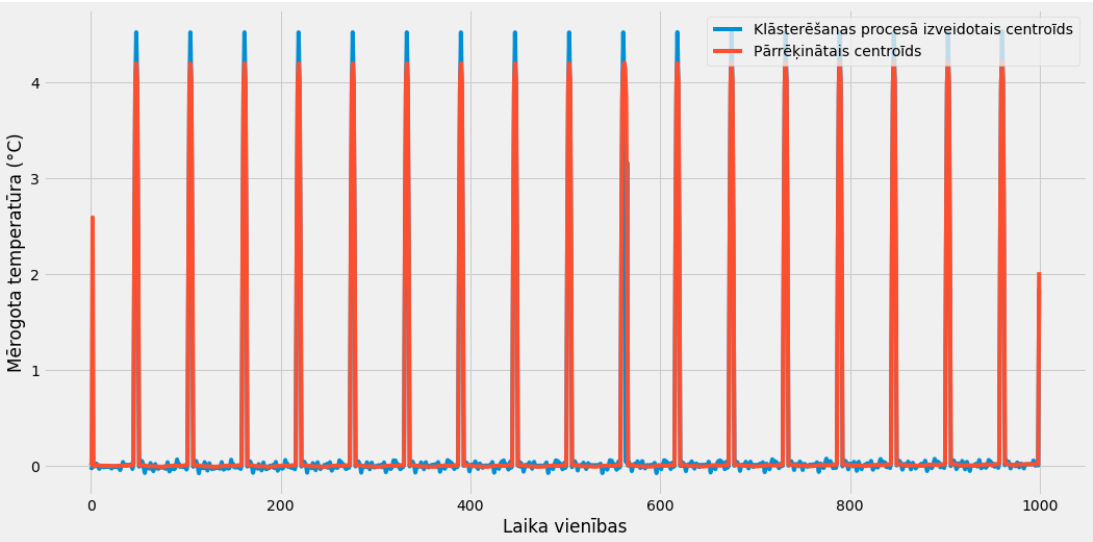
[158]:
```
sd_g1 = softdtw_barycenter(df_train[0:10, :, :], gamma=1., max_iter=200,␣
 ↪tol=1e-5, init=km.cluster_centers_[2])
print('1. done!')
sd_g2 = softdtw_barycenter(df_train[10:20, :, :], gamma=1., max_iter=200,␣
 ↪tol=1e-5, init=km.cluster_centers_[0])
print('2. done!')
sd_g3 = softdtw_barycenter(df_train[20:30, :, :], gamma=1., max_iter=200,␣
 ↪tol=1e-5, init=km.cluster_centers_[1])
print('3. done!')
```

```
1. done!
2. done!
3. done!
```

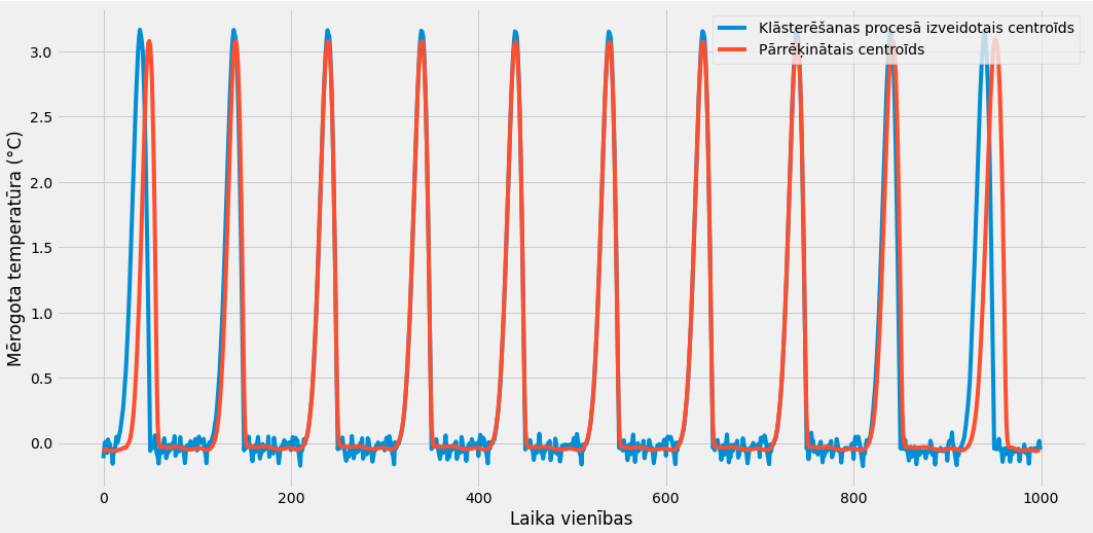
[167]:
```
plt.plot(km.cluster_centers_[2])
plt.plot(sd_g1)
plt.xlabel('Laika vienības');
plt.ylabel(u'Mērogota temperatūra (\u00B0C)');
plt.legend(['Klāsterēšanas procesā izveidotais centroīds', 'Pārrēķinātais␣
 ↪centroīds'], loc='upper right')
plt.savefig('images/klasteresana/cluster_1_centroid.png')
```



[168]:
```
plt.plot(km.cluster_centers_[0])
plt.plot(sd_g2)
plt.xlabel('Laika vienības');
plt.ylabel(u'Mērogota temperatūra (\u00B0C)');
plt.legend(['Klāsterēšanas procesā izveidotais centroīds', 'Pārrēķinātais␣
 ↪centroīds'], loc='upper right')
plt.savefig('images/klasteresana/cluster_2_centroid.png')
```



[171]:
```
plt.plot(km.cluster_centers_[1])
plt.plot(sd_g3)
plt.xlabel('Laika vienības');
plt.ylabel(u'Mērogota temperatūra (\u00B0C)');
plt.legend(['Klāsterēšanas procesā izveidotais centroīds', 'Pārrēķinātais␣
 ↪centroīds'], loc='upper right')
plt.savefig('images/klasteresana/cluster_3_centroid.png')
```



2. Eiklīda attālums

[77]:
```
# Trenējam k-means metodi ar dtw distances mēru
start = time.time()
euc_km.fit_predict(df_train);
print(f'Modelis trenējās {round((time.time()-start)/60,2)} minūtes!')
```

```
Modelis trenējās 0.0 minūtes!
```

[78]:
```
unique, counts = np.unique(euc_km.labels_, return_counts=True)
for i, j in zip(unique, counts):
    print(f'{i} - {j} laikrindas')
```

```
0 - 8 laikrindas
1 - 10 laikrindas
2 - 12 laikrindas
```

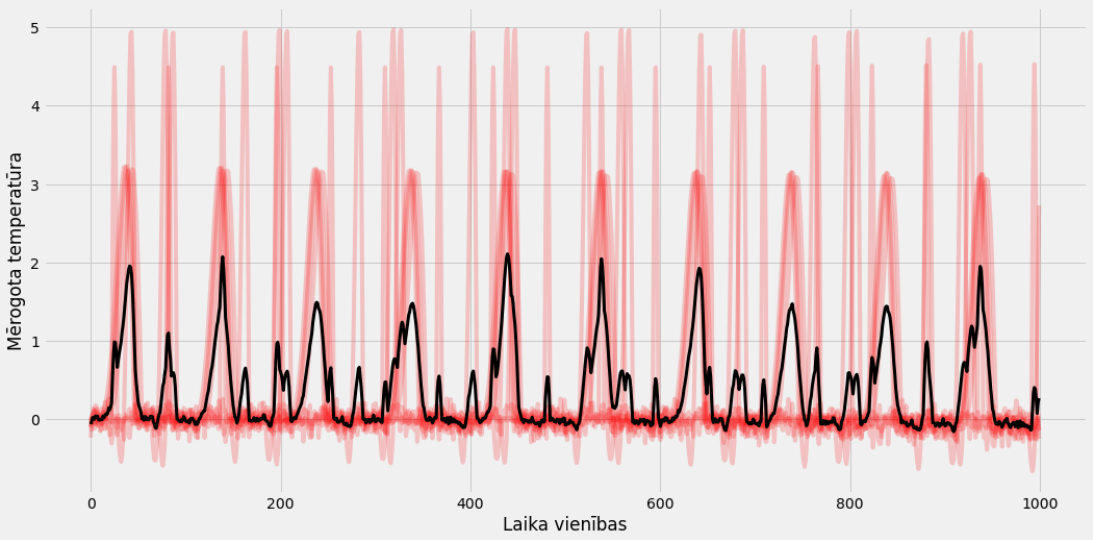
[79]:
```
# Modeļa saglabāšana!
# joblib.dump(km, '../models/KM_Clustering_model.pkl')
# km = joblib.load('../models/KM_Clustering_model.pkl')
```

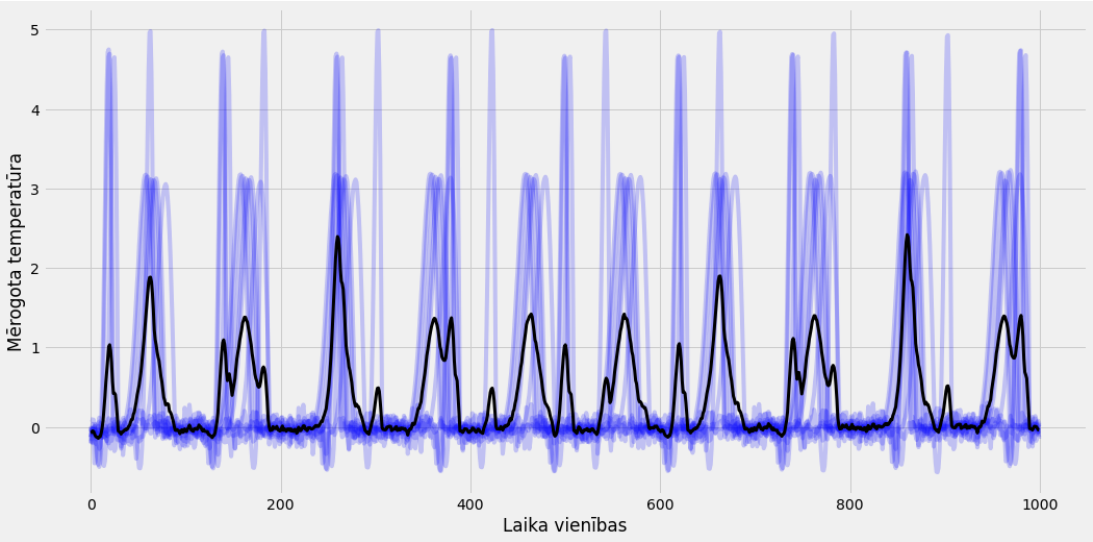
[113]:
```
max_prec = 0
for i in permutations(set(euc_km.labels_), len(set(euc_km.labels_))):
    c = []
    for j in i:
        c.extend([j]*10)
    prec = sum(euc_km.labels_ == c)/len(euc_km.labels_)*100
    if prec > max_prec:
        max_prec = prec
print(f'Klāsterēšanas precizitāte {round(max_prec, 2)} %')
```

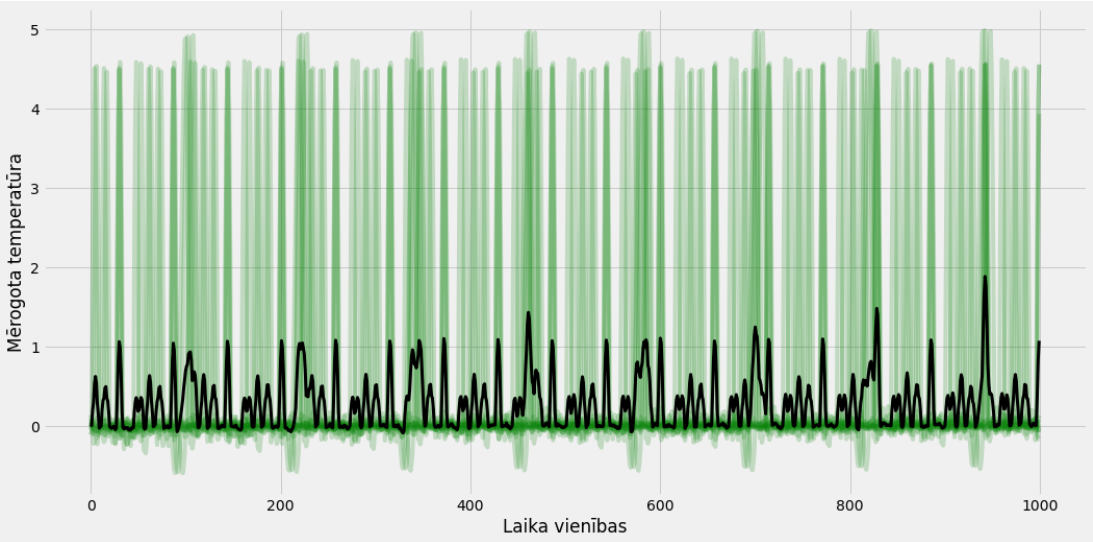
```
Klāsterēšanas precizitāte 60.0 %
```

[81]:
```
#print(f'Klāsterēšanas precizitāte {sum(euc_km.labels_ == c)/len(euc_km.
↪labels_)*100} %')
```

[125]:
```
#clust_plot(euc_km, df_train)
#plt.savefig('images/klasteresana/time_series_euc_clusters.png')
clust_sep_plot(euc_km, df_train, method_name='EUC')
```







3. k-shape klāsterēšanas

[127]:
```
start = time.time()
ks = KShape(n_clusters=3, n_init=3, verbose=True)
ks.fit_predict(df_train)
print(f'Modelis trenējās {round((time.time()-start)/60,2)} minūtes!')
```

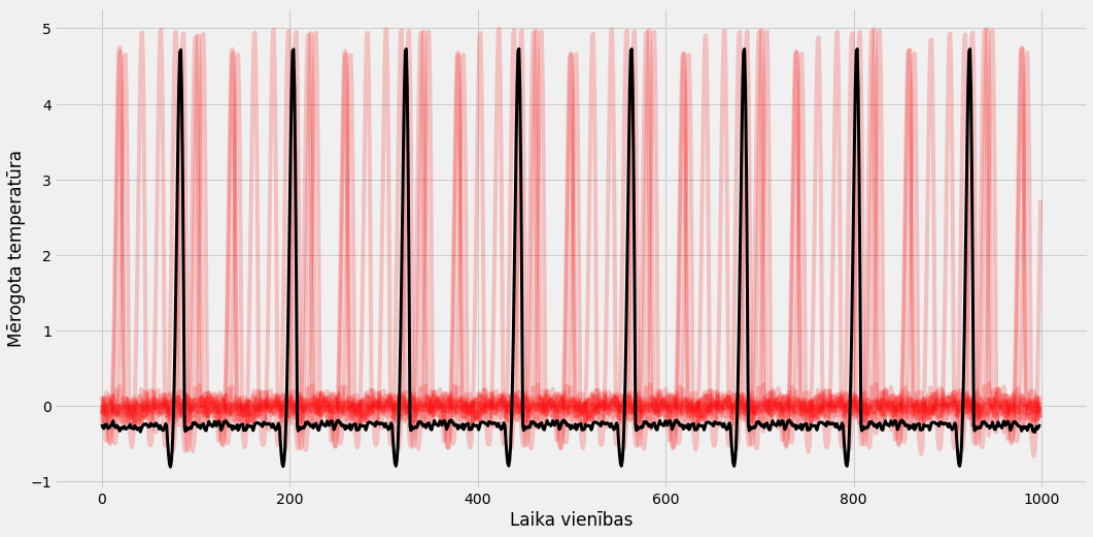
```
Init 1
0.006 --> 0.006 --> 0.006 -->
Init 2
0.120 --> 0.081 --> 0.078 --> 0.075 --> 0.075 --> 0.075 --> 0.075 --> 0.075 -->
```

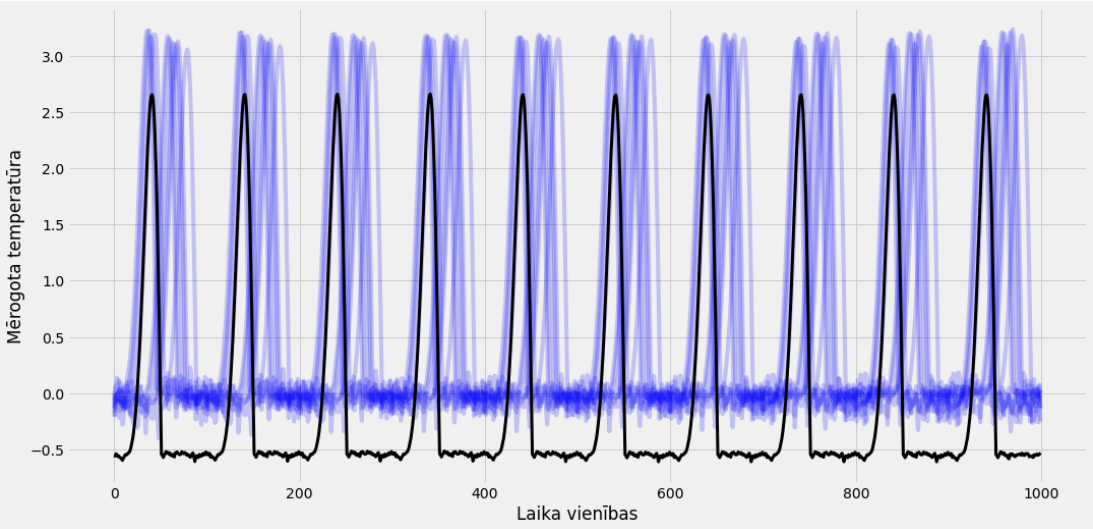
```
Init 3
0.006 --> 0.006 --> 0.006 --> 0.006 -->
Modelis trenējās 0.42 minūtes!
```

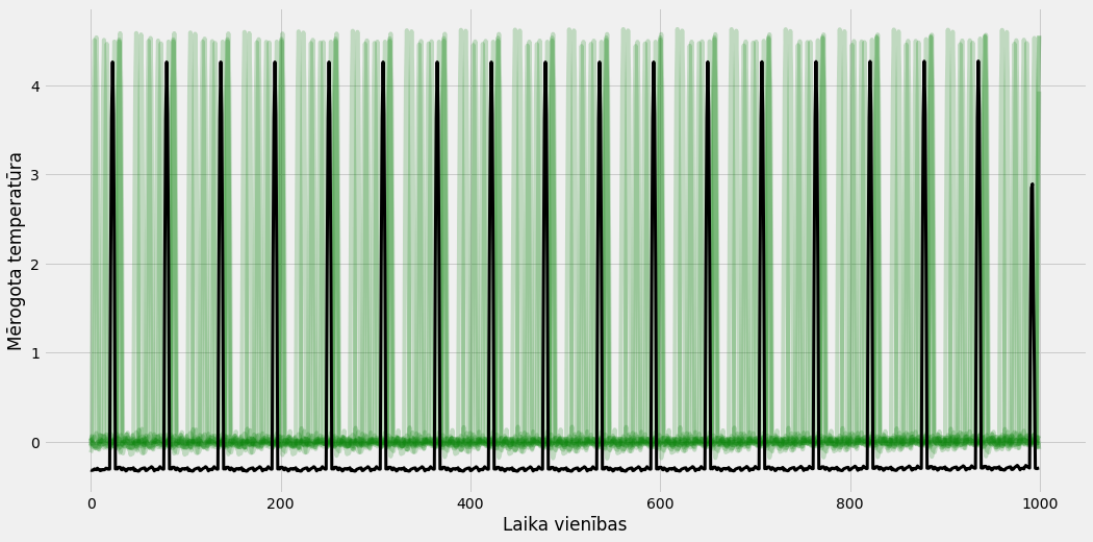
[118]:
```python
# Aprēķinam klāsterēšanas preciziāti!
max_prec = 0
for i in permutations(set(ks.labels_), len(set(ks.labels_))):
    c = []
    for j in i:
        c.extend([j]*10)
    prec = sum(ks.labels_ == c)/len(ks.labels_)*100
    if prec > max_prec:
        max_prec = prec
print(f'Klāsterēšanas precizitāte {round(max_prec, 2)} %')
```

```
Klāsterēšanas precizitāte 100.0 %
```

[126]:
```python
#clust_plot(ks, df_train)
clust_sep_plot(ks, df_train, method_name='SBC')
```

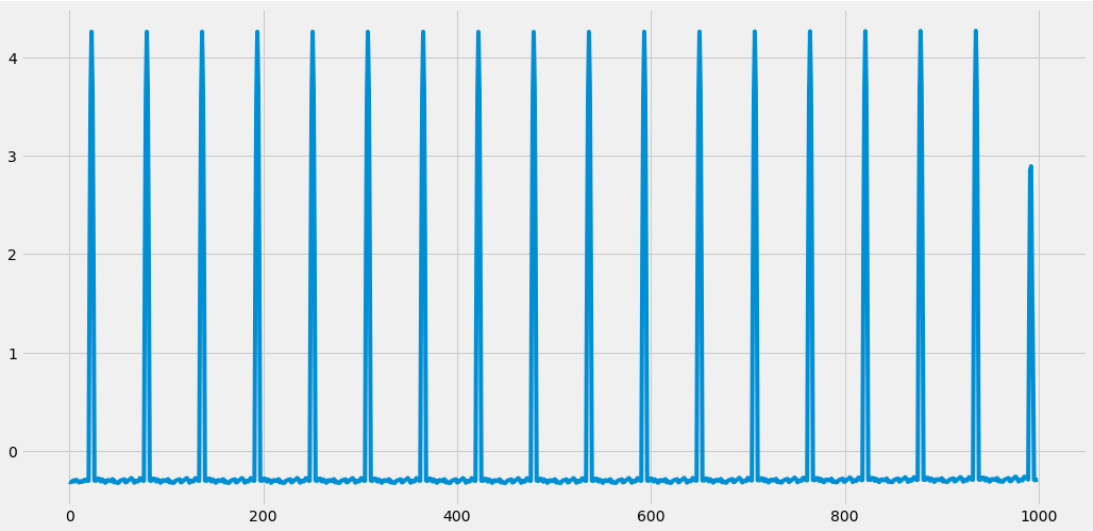






[121]:
```
plt.plot(ks.cluster_centers_[2,:,0])
```

[121]:
```
[<matplotlib.lines.Line2D at 0x28a28295a00>]
```



4. klāsterēšana izmantojot GAK

[1]:
```
# gak_km = KernelKMeans(n_clusters=3,
#                       kernel="gak",
#                       kernel_params={"sigma": "auto"},
#                       n_init=3,
#                       verbose=True,
#                       n_jobs=5)
# gak_km.fit_predict(df_train)
```

[2]:
```
# gak_km.labels_
```

GAK klāsterēšanas metode mūsu apskatītajam piemēram nestrādā, jo visas laikrindas tiek saklāsterēsas vienā klāsterī!

[44]:
```
#clust_plot(gak_km, df_train)
```

6 Apakšperiodu klāsterēšana

[16]:
```
# Nodefinējam loga garumu
window_size = 20

# Nodefinējam klāsteru skaitu (Piezīme: apakšperiodu klāsteru skaits var nebūt␣
 ↪vienāds ar laikrindu klāsteru skaitu)
clusters_sp = 2

# Izvēlamies laikrindu kuru gribam analizēt
df_ts = df_SS[['g1_ts1']].values
```

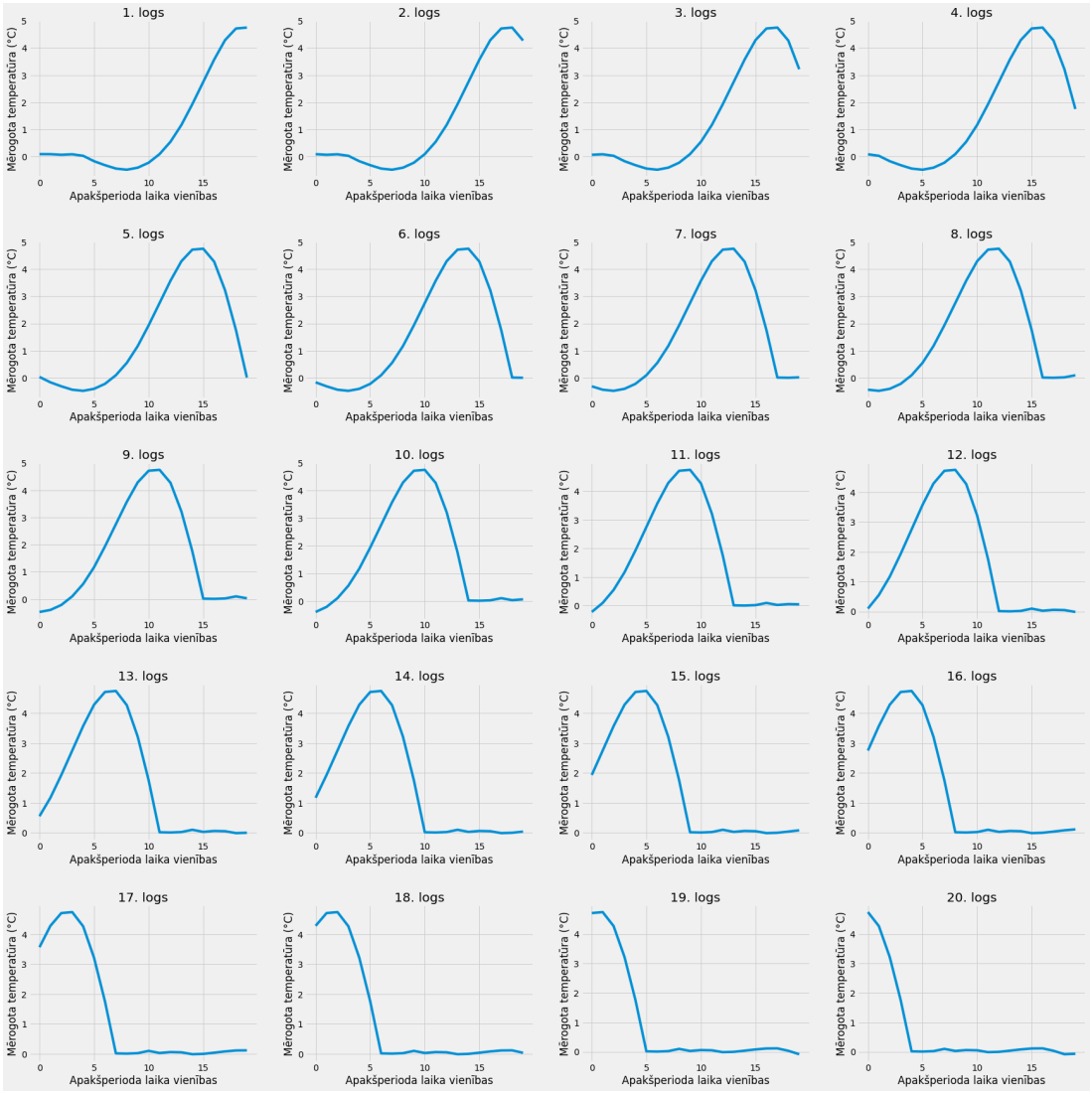
[54]:
```
def window_split(data, window_size=20, step=1):
    i = 0
    res = list()
    while i + window_size <= len(data):
        res.append(data[i:i+window_size])
        i += step
    res = np.array(res)
    return res.reshape(res.shape[0], res.shape[1], -1)
```

[18]:
```
df_sp = window_split(df_ts)
```

[19]:
```
df_sp.shape
```

[19]:
```
(981, 20, 1)
```

[20]:
```
plt.figure(figsize=(25,25))

for i, ts in enumerate(df_sp):

    if i == 20:
        break

    plt.subplot(5,4, i+1)
    #plt.plot(smooth, linewidth=3, color='blue')
    plt.plot(ts)
    #plt.fill_between(range(len(true)), low[i], up[i], alpha=0.3)
    plt.title(f"{i+1}. logs")
    plt.xlabel('Apakšperioda laika vienības');
    plt.ylabel(u'Mērogota temperatūra (\u00B0C)');
plt.tight_layout(pad=3.0)
plt.savefig('images/klasteresana/subperiods.png')
```

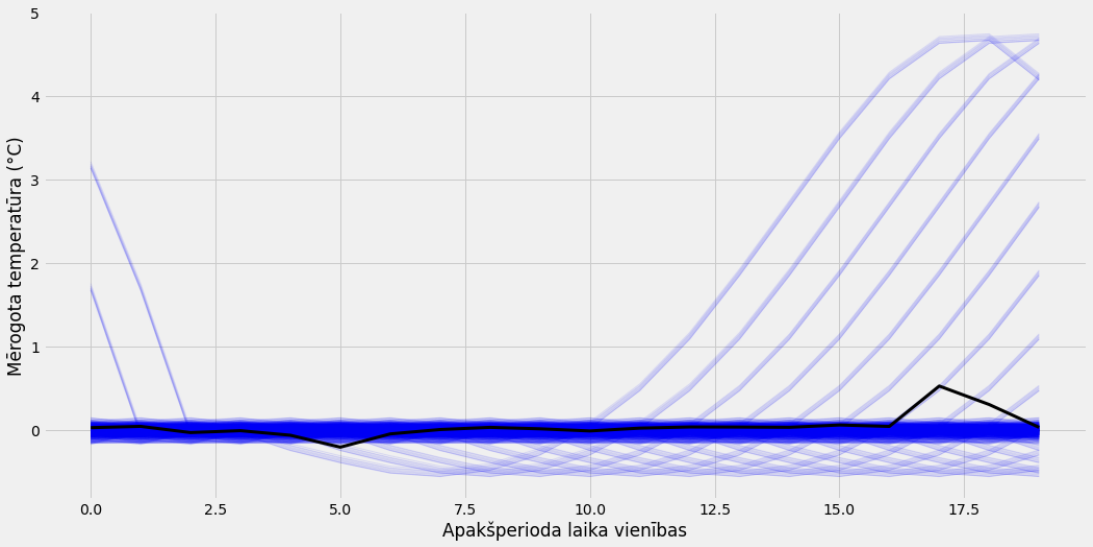


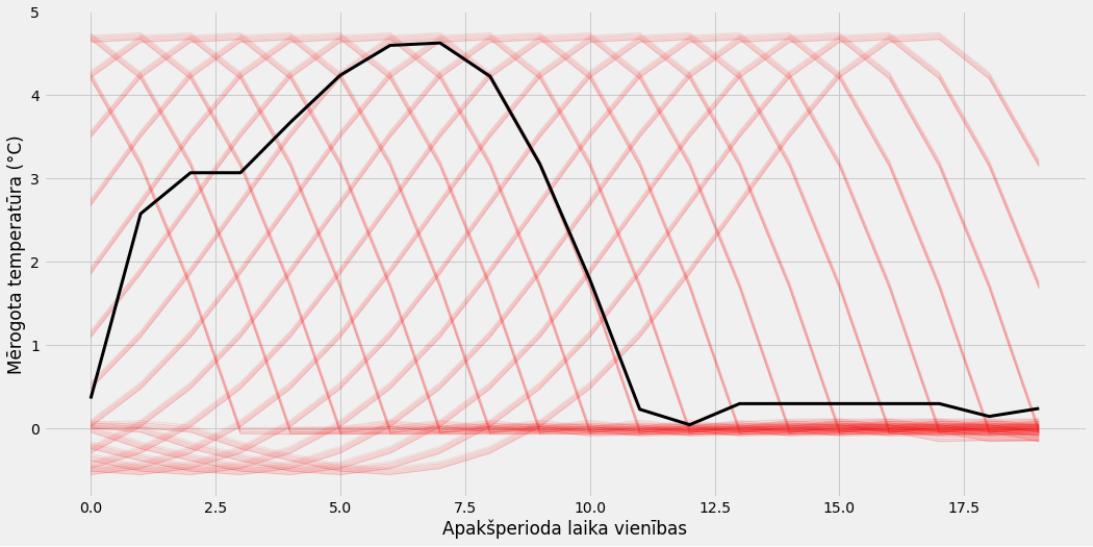
[21]:
```
# Cluster smoothed windows
km_sp = TimeSeriesKMeans(n_clusters=clusters_sp,
                         metric="dtw",
                         max_iter=20,
                         random_state=42)
km_sp.fit(df_sp)
```

[21]:
```
TimeSeriesKMeans(max_iter=20, metric='dtw', n_clusters=2, random_state=42)
```

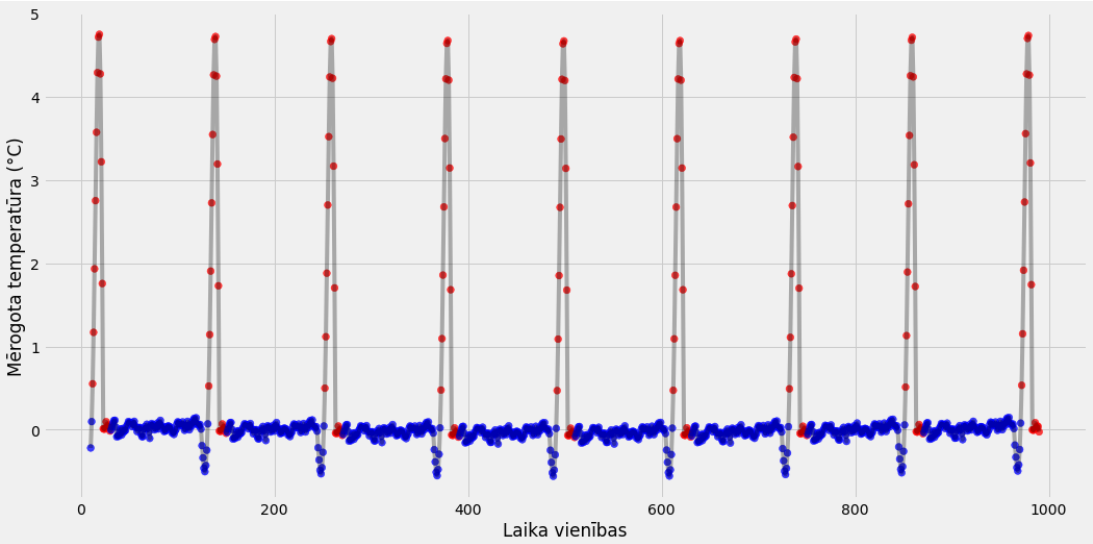
[22]:
```
# Plotting resulting clusters
colors = {0:'blue',1:'red',2:'green',3:'orange'}

for c in range(km_sp.n_clusters):
```

```
    plt.plot(np.squeeze(df_sp[km_sp.labels_ == c],-1).T, c=colors[c], alpha=0.
↪1, linewidth=1)
    #plt.plot(np.squeeze(km_sp.cluster_centers_, -1)[c], c=colors[c],␣
↪linewidth=5)
    plt.plot(np.squeeze(km_sp.cluster_centers_, -1)[c], c='black', linewidth=3)
    #plt.title(f"{c+1}. klāsteris")
    plt.xlabel('Apakšperioda laika vienības');
    plt.ylabel(u'Mērogota temperatūra (\u00B0C)');

    plt.savefig(f'images/klasteresana/subperiod_cluster_{c}.png')
    plt.show()
```





[162]:
```
# Ploting clusters on raw data
plt.figure(figsize=(16,8))

nobide = math.floor(df_sp.shape[1]/2)
# nepieciešama, jo klāsterētā intervāla garums ir n vienības un šajā gadījumā␣
 ↪vizualizācijas nolūkiem tiek izvēlēts
# attēlot vidējo perioda vērtību perioda klāstera krāsā otrs variants būtu␣
 ↪nosvērt un aprēķināt pie kura klāstera periodiem
# konkrētais punkt pieder visvairāk (rezultātiem vajadzētu būt diezgn līdzīgiem)

plt.plot(df_SS['g1_ts1'][nobide:len(km_sp.labels_)+nobide], c='black', alpha=0.
 ↪3)
plt.scatter(range(nobide, len(df_SS['g1_ts1'][:len(km_sp.labels_)])+nobide),␣
 ↪df_SS['g1_ts1'][nobide:len(km_sp.labels_)+nobide],
            c=[colors[c] for c in km_sp.labels_], s=45, alpha=.75)
plt.xlabel('Laika vienības');
plt.ylabel(u'Mērogota temperatūra (\u00B0C)');

# plt.xticks(range(0,len(df.timestamp.unique())-window_shape, 100),
#            df.timestamp.dt.date.unique()[window_shape::100], rotation=70)
np.set_printoptions(False)
plt.savefig('images/klasteresana/ts_with_colored_subclusters.png')
```



[44]:
```
# saglabājam mērogošanas parametrus pirmās grupas pirmajai laikrindai!
sp = [df_smooth.median(axis=0).values[0], df_smooth.std(axis=0).values[0]]
```

```
with open('data/g1_t1_scaling.pickle', 'wb') as f: # Pickling!
    pickle.dump(sp, f)
```

[52]:
```
(ts_g1.iloc[:,[0]]-sp[0])/sp[1]
```

[52]:
```
       g1_ts1
0    0.035242
1    0.070964
2    0.110307
3    0.085433
4   -0.053472
..        …
995  0.276029
996 -0.062656
997 -0.280598
998 -0.026162
999 -0.151899

[1000 rows x 1 columns]
```

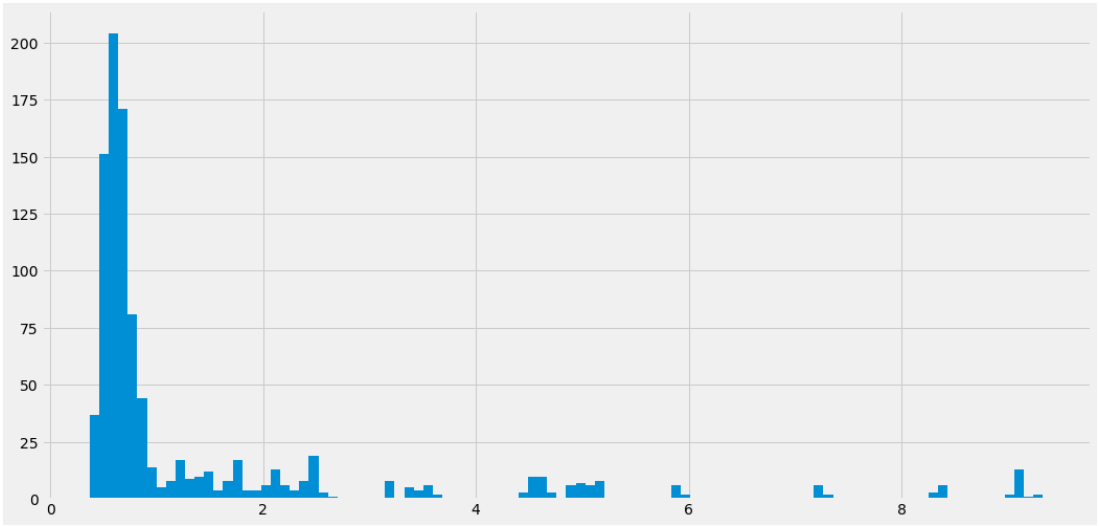
[57]:
```
df_ts1_g1_org = (ts_g1.iloc[:,[0]]-sp[0])/sp[1]
```

[64]:
```
df_ts_split = window_split(df_ts1_g1_org.values)
```

[ ]:
```
cluster_0 = df_ts_split[km_sp.labels_ == 0,:,:]
cluster_1 = df_ts_split[km_sp.labels_ == 1,:,:]
```

[119]:
```
# Aprēķinam katra apakšperioda attālumu līdz klāstera centram!
# Un tiek izprintēti apakšperiodi DTW attālumu lielāku par 8!!!
km_sp_res = []
for i,j in zip(df_ts_split, km_sp.labels_):
    dist = dtw(i.reshape(-1), km_sp.cluster_centers_[j])
    km_sp_res.append(dist)
    if dist > 8:
        plt.plot(i.reshape(-1))
        plt.plot(km_sp.cluster_centers_[j])
        plt.show()
```

[115]:
```
plt.hist(km_sp_res, bins = 100);
```



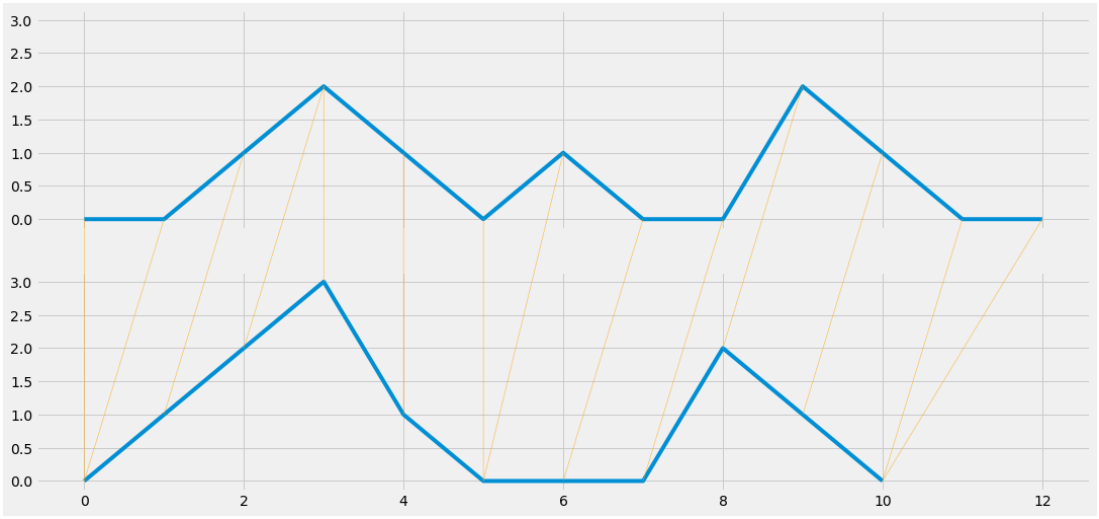
[121]:
```
# Saglabājam klasterizācija atlikumus un klasteru centroīdu vērtības!
with open('data/km_sp_res.pickle', 'wb') as f: # Pickling!
    pickle.dump(km_sp_res, f)

with open('data/cluster_centers.pickle', 'wb') as f: # Pickling!
    pickle.dump(km_sp.cluster_centers_, f)
```

6.0.1 Sandbox

[22]:
```
# km_2 = KMeans(k=4, max_it=10, max_dba_it=10, dists_options={"window": 40})
# cluster_idx, performed_it = km_2.fit(x, use_c=True, use_parallel=False)
```

[20]:
```
x = df_train.reshape(30, 1000)
```

[39]:
```
from dtaidistance import dtw
from dtaidistance import dtw_visualisation as dtwvis
import numpy as np
s1 = np.array([0, 0, 1, 2, 1, 0, 1, 0, 0, 2, 1, 0, 0])
s2 = np.array([0, 1, 2, 3, 1, 0, 0, 0, 2, 1, 0])
path = dtw.warping_path(s1, s2)
dtwvis.plot_warping(s1, s2, path);
plt.tight_layout(pad=3.0)
```



[17]:
```
#dtw.distance_matrix_fast(df_train)
```

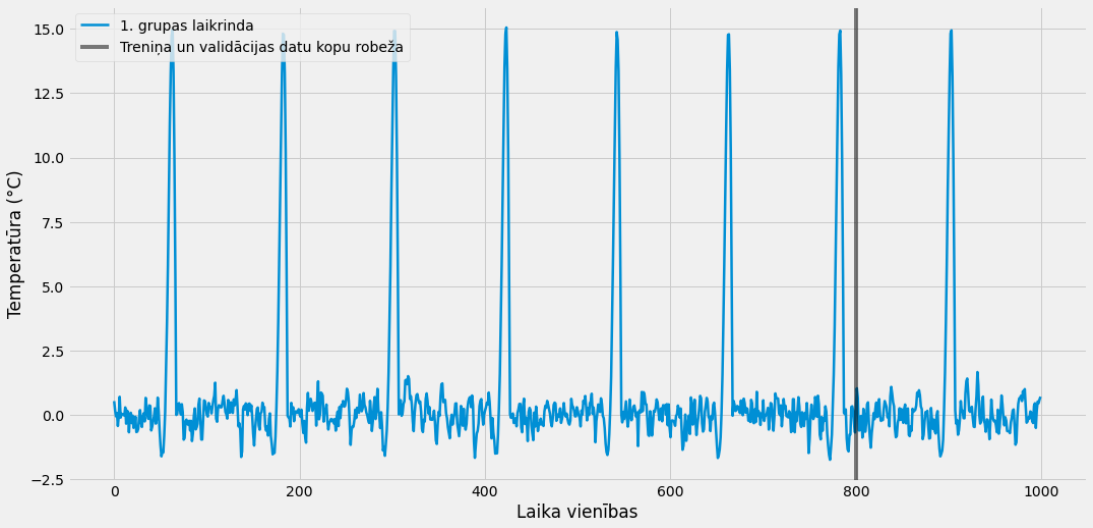
B.5. Prognozēšanas darba grāmata

1 Prognozēšana

[147]:
```
# Pakotņu ielāde un iestatījumi
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import TimeSeriesSplit # Treniņa un testa datu␣
↪kopas dalījums
from scipy.optimize import minimize # priekš funkcijas minimizēšanas
import itertools as it
from tqdm.notebook import tqdm
import statsmodels.formula.api as smf            # statistics and econometrics
import statsmodels.tsa.api as smt
import statsmodels.api as sm
from statsmodels.tsa.statespace.sarimax import SARIMAX
from math import sqrt
from prophet import Prophet
import time
import seaborn as sns
from keras.utils.vis_utils import plot_model
import pickle

from statsmodels.tsa.forecasting.stl import STLForecast
from statsmodels.tsa.arima.model import ARIMA
from statsmodels.tsa.forecasting.theta import ThetaModel

from lightgbm import LGBMRegressor
from lightgbm import plot_importance as lightboost_plot_importance
from xgboost import XGBRegressor, plot_importance
from catboost import CatBoostRegressor
from sklearn import svm
from sklearn.datasets import make_regression
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import RepeatedKFold
from lightgbm import plot_tree

# Novērtējumu metrikas
from sklearn.metrics import r2_score, median_absolute_error, mean_absolute_error
from sklearn.metrics import mean_squared_error, mean_squared_log_error
def mean_absolute_percentage_error(y_true, y_pred):
    return np.mean(np.abs((y_true - y_pred) / y_true)) * 100


%matplotlib inline
plt.style.use('fivethirtyeight') # Grafiku stils!
plt.rcParams["figure.figsize"] = (16,8) # Grafiku izmērs
```

[2]:
```
# Simulētie temperatūras mērījumu dati
ts_g1 = pd.read_csv('data/sim_ts_g1.csv', index_col=0)
ts_g2 = pd.read_csv('data/sim_ts_g2.csv', index_col=0)
ts_g3 = pd.read_csv('data/sim_ts_g3.csv', index_col=0)

df_g1 = ts_g1.iloc[:,3]
df_g2 = ts_g2.iloc[:,1]
df_g3 = ts_g3.iloc[:,1]
```

[3]:
```
df_g1.plot(linewidth=2.5)
#df_g2.plot(linewidth=2.5)
#df_g3.plot(linewidth=2.5)
plt.axvline(x=800, color='black', alpha=.5)
plt.legend(['1. grupas laikrinda',
            #'2. grupas laikrinda',
            #'3. grupas laikrinda',
            'Treniņa un validācijas datu kopu robeža'
           ], loc='upper left')
plt.xlabel('Laika vienības')
plt.ylabel(u'Temperatūra (\u00B0C)')
plt.savefig('images/prognozesana/all_3_ts_plot_w_threshold.png');
```



1.0.1 Laikrindu sadalīšana treniņa un testa/validācijas datu kopās

[4]:
```
# def train_test_split(series, n_train):
#     test = series[:-n_train]
#     train = series[-n_train:]
#     return test, train
```

```
def train_test_split(data, n_test):
    return np.array(data[:-n_test]), np.array(data[-n_test:])
# TimeSeriesSplit()
```

1.0.2 Modeļu prognožu novērtējuma metrikas

- R^2 : $R^2 = 1 - \frac{SS_{res}}{SS_{tot}}$
- MAE : $MAE = \frac{\sum_{i=1}^{n} |y_i - \hat{y}_i|}{n}$
- MedAE : $MedAE = median(|y_1 - \hat{y}_1|, ..., |y_n - \hat{y}_n|)$
- MSE : $MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$
- MSLE (nav piemērots apskatāmajai problēmāi) : $MSLE = \frac{1}{n}\sum_{i=1}^{n}(log(1 + y_i) - log(1 + \hat{y}_i))^2$
- MAPE : $MAPE = \frac{100}{n}\sum_{i=1}^{n}\frac{|y_i - \hat{y}_i|}{y_i}$

1.0.3 0. Prognožu novērtēšanas funkcija

[5]:
```
# r2_score, median_absolute_error, mean_absolute_error, median_absolute_error,␣
↪mean_squared_error, mean_squared_log_error, mean_absolute_percentage_error
```

[6]:
```
def pred_evaluation(y_true, y_pred, model, exec_time = 0, model_type = '(n/a)',␣
↪comment = ''):
    res = {
        'Modelis': model,
        'Modeļa tips': model_type,
        'Izpildes laiks, sekundes': round(exec_time,2),
        'R^2': round(r2_score(y_true, y_pred),2),
        'MAE': round(mean_absolute_error(y_true, y_pred),2),
        'MedAE': round(median_absolute_error(y_true, y_pred),2),
        'MSE': round(mean_squared_error(y_true, y_pred),2),
        #'MSLE': round(mean_squared_log_error(y_true, y_pred),2),
        'MAPE': round(mean_absolute_percentage_error(np.array(y_true), np.
↪array(y_pred)),2),
        'Komentārs': comment
    }
    return res
```

1.0.4 1. Naivie modeļi

[7]:
```
train, test = train_test_split(df_g1, 200)
```

Par naivajiem nodeļiem tiek saukti vienkārši modeļi ar mazu parametru skaitu, kas neprasa sarežģītus aprēķinus un ir viegli interpretējami. Tie pamatā ir uz nosacījumiem/noteikumiem balstīti modeļi.

a) Pēdējās vērtības prognoze! $\hat{y}_t = y_{t-1}$

[8]:
```
def pred_last_step(series, pred_steps):
    pred = np.array([])
    pred_series = series.copy()
    for i in range(pred_steps):
        pred = np.append(pred, pred_series[-1])
        pred_series = np.append(pred_series, pred_series[-1])
    return pred
```

b) Vidējās vērtības prognoze! $\hat{y}_t = \overline{y}$

[9]:
```
def pred_avg(series, pred_steps):
    pred = np.array([])
    y_avg = np.average(series)
    for i in range(pred_steps):
        pred = np.append(pred, y_avg)
    return pred
```

c) Pēdējo k vērtību vidējās vērtības prognoze $\hat{y}_t = \frac{1}{k}\sum_{n=1}^{k} y_{t-n}$

[10]:
```
def pred_avg_k_step(series, k, pred_steps):
    pred = np.array([])
    pred_series = series.copy()
    for i in range(pred_steps):
        pred = np.append(pred, sum(pred_series[-k:])/k)
        pred_series = np.append(pred_series, sum(pred_series[-k:])/k)
    return pred
```

d) Pēdējās vērtības m laika vienības atpakaļ prongoze $\hat{y}_t = y_{t-m}$

[11]:
```
def pred_last_seasona_step(series, m, pred_steps):
    pred = np.array([])
    pred_series = series.copy()
    for i in range(pred_steps):
        pred = np.append(pred, pred_series[-m])
        pred_series = np.append(pred_series, pred_series[-m])
    return pred
```

e) Pēdējo k vērtību ik pēc m laika vienībā atpakaļ prognoze $\hat{y}_t = \frac{1}{k}\sum_{n=1}^{k} y_{t-nm}$

[12]:
```
def pred_avg_k_seasonal_step(series, m, k, pred_steps):
    pred = np.array([])
    pred_series = series.copy()
```

```
    for i in range(pred_steps):
        s = 0
        for j in range(k):
            s += pred_series[-((j+1)*m)]
        pred = np.append(pred, s/k)
        pred_series = np.append(pred_series, s/k)
    return pred
```

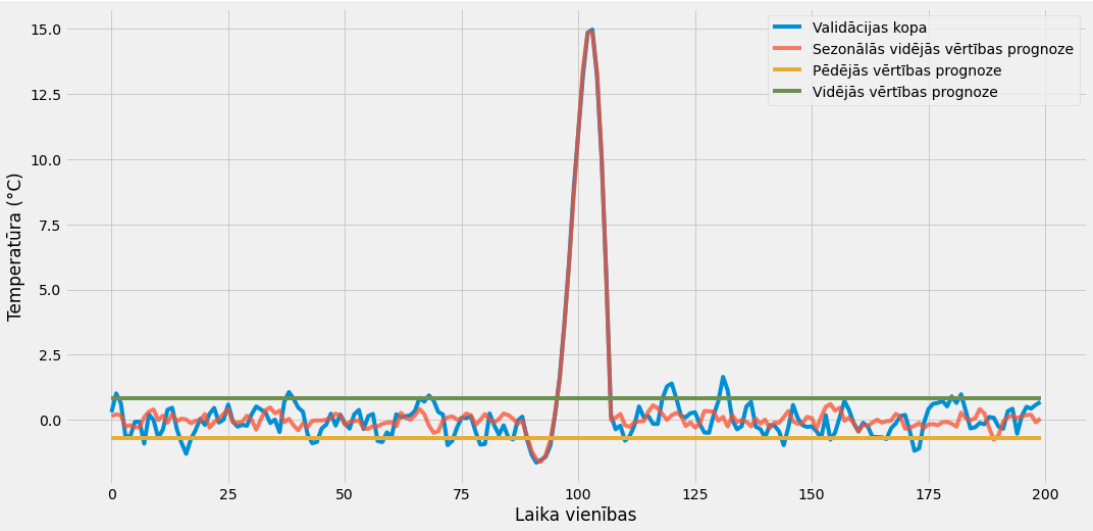
Teorētiski varam izmantot visas četras, iepriekš definētās, naivās metodes, bet tā kā zinam, ka mūsu laikrindas ir ar sezonālu struktūru varam izvēlēties divas piemērotākās metodes: **"Pēdējās vērtības prognoze"** un **"Pēdējo k vērtību ik pēc m laika vienībā atpakaļ prognoze"** metodes.

[13]:
```
start_time = time.time()
one_step_pred = pred_last_step(train, 200)
one_step_et = time.time() - start_time
```

[14]:
```
start_time = time.time()
average_pred = pred_avg(train, 200)
average_et = time.time() - start_time
```

[15]:
```
start_time = time.time()
avg_k_seasonal_step_pred = pred_avg_k_seasonal_step(train, 120, 5, 200)
avg_k_seasonal_step_et = time.time() - start_time
```

[17]:
```
plt.plot(test)
plt.plot(avg_k_seasonal_step_pred, alpha=.75)
plt.plot(one_step_pred)
plt.plot(average_pred)
plt.xlabel('Laika vienības')
plt.ylabel(u'Temperatūra (\u00B0C)')
plt.legend(['Validācijas kopa',
            'Sezonālās vidējās vērtības prognoze',
            'Pēdējās vērtības prognoze',
            'Vidējās vērtības prognoze'
           ])
plt.savefig('images/prognozesana/naive_model_pred.png');
```



[18]:
```
# pred_evaluation(test, one_step_pred, 'Last value')
```

[19]:
```
# pred_evaluation(test, average_pred, 'Average value')
```

[20]:
```
# pred_evaluation(test, avg_k_seasonal_step_pred, 'Avg k seaonal step')
```

[21]:
```
# plt.plot(test-avg_k_seasonal_step_pred)
```

1.0.5 2. statistiskās metodes

a) Trīskāršās eksponenciālās gludināšanas modelis

Netiek apskatīta vienkāršās un divkāršās eksponenciālā gludināšana modeļi, jo tie neņem vērā sezonalitāti.

Trīskāršās eksponenciālās gludināšanas modelis:
$\ell_x = \alpha(y_x - s_{x-L}) + (1-\alpha)(\ell_{x-1} + b_{x-1})$
$b_x = \beta(\ell_x - \ell_{x-1}) + (1-\beta)b_{x-1}$
$s_x = \gamma(y_x - \ell_x) + (1-\gamma)s_{x-L}$
$\hat{y}_{x+m} = \ell_x + mb_x + s_{x-L+1+(m-1)modL}$

Ticamības intervāli tiek rēķināti pēc Brutlag metodes:
$\hat{y}_{max_x} = \ell_{x-1} + b_{x-1} + s_{x-T} + m\ d_{t-T}$
$\hat{y}_{min_x} = \ell_{x-1} + b_{x-1} + s_{x-T} - m\ d_{t-T}$
$d_t = \gamma\ y_t - \hat{y}_t\ + (1-\gamma)d_{t-T},$

[156]:
```
class HoltWinters:

    """
    Holt-Winters model with the anomalies detection using Brutlag method
```

```
    # series - initial time series
    # slen - length of a season
    # alpha, beta, gamma - Holt-Winters model coefficients
    # n_preds - predictions horizon
    # scaling_factor - sets the width of the confidence interval by Brutlag␣
↪(usually takes values from 2 to 3)

    """


    def __init__(self, series, slen, alpha, beta, gamma, n_preds,␣
↪scaling_factor=1.96):
        self.series = series
        self.slen = slen
        self.alpha = alpha
        self.beta = beta
        self.gamma = gamma
        self.n_preds = n_preds
        self.scaling_factor = scaling_factor


    def initial_trend(self):
        sum = 0.0
        for i in range(self.slen):
            sum += float(self.series[i+self.slen] - self.series[i]) / self.slen
        return sum / self.slen

    def initial_seasonal_components(self):
        seasonals = {}
        season_averages = []
        n_seasons = int(len(self.series)/self.slen)
        # let's calculate season averages
        for j in range(n_seasons):
            season_averages.append(sum(self.series[self.slen*j:self.slen*j+self.
↪slen])/float(self.slen))
        # let's calculate initial values
        for i in range(self.slen):
            sum_of_vals_over_avg = 0.0
            for j in range(n_seasons):
                sum_of_vals_over_avg += self.series[self.
↪slen*j+i]-season_averages[j]
            seasonals[i] = sum_of_vals_over_avg/n_seasons
        return seasonals


    def triple_exponential_smoothing(self):
        self.result = []
```

```
        self.Smooth = []
        self.Season = []
        self.Trend = []
        self.PredictedDeviation = []
        self.UpperBond = []
        self.LowerBond = []

        seasonals = self.initial_seasonal_components()

        for i in range(len(self.series)+self.n_preds):
            if i == 0: # components initialization
                smooth = self.series[0]
                trend = self.initial_trend()
                self.result.append(self.series[0])
                self.Smooth.append(smooth)
                self.Trend.append(trend)
                self.Season.append(seasonals[i%self.slen])

                self.PredictedDeviation.append(0)

                self.UpperBond.append(self.result[0] +
                                      self.scaling_factor *
                                      self.PredictedDeviation[0])

                self.LowerBond.append(self.result[0] -
                                      self.scaling_factor *
                                      self.PredictedDeviation[0])
                continue

            if i >= len(self.series): # predicting
                m = i - len(self.series) + 1
                self.result.append((smooth + m*trend) + seasonals[i%self.slen])

                # when predicting we increase uncertainty on each step
                self.PredictedDeviation.append(self.PredictedDeviation[-1]*1.01)

            else:
                val = self.series[i]
                last_smooth, smooth = smooth, self.alpha*(val-seasonals[i%self.
↪slen]) + (1-self.alpha)*(smooth+trend)
                trend = self.beta * (smooth-last_smooth) + (1-self.beta)*trend
                seasonals[i%self.slen] = self.gamma*(val-smooth) + (1-self.
↪gamma)*seasonals[i%self.slen]
                self.result.append(smooth+trend+seasonals[i%self.slen])

                # Deviation is calculated according to Brutlag algorithm.
```

```
                self.PredictedDeviation.append(self.gamma * np.abs(self.
↪series[i] - self.result[i])
                                               + (1-self.gamma)*self.
↪PredictedDeviation[-1])

            self.UpperBond.append(self.result[-1] +
                                  self.scaling_factor *
                                  self.PredictedDeviation[-1])

            self.LowerBond.append(self.result[-1] -
                                  self.scaling_factor *
                                  self.PredictedDeviation[-1])

            self.Smooth.append(smooth)
            self.Trend.append(trend)
            self.Season.append(seasonals[i%self.slen])
```

[157]:
```
# Laikrindas krosvalidācijas funkcija priekš Holt-Winters modeļa parametru␣
↪atrašanas
def timeseriesCVscore(params, series, loss_function=mean_squared_error,␣
↪slen=24):
    """
        Returns error on CV

        params - vector of parameters for optimization
        series - dataset with timeseries
        slen - season length for Holt-Winters model
    """
    # errors array
    errors = []

    values = series.values
    alpha, beta, gamma = params

    # set the number of folds for cross-validation
    tscv = TimeSeriesSplit(n_splits=3)

    # iterating over folds, train model on each, forecast and calculate error
    for train, test in tscv.split(values):

        model = HoltWinters(series=values[train], slen=slen,
                            alpha=alpha, beta=beta, gamma=gamma,␣
↪n_preds=len(test))
        model.triple_exponential_smoothing()

        predictions = model.result[-len(test):]
        actual = values[test]
```

```
        error = loss_function(predictions, actual)
        errors.append(error)

    return np.mean(np.array(errors))
```

[158]:
```
def plotHoltWinters(series, plot_intervals=False, plot_anomalies=False):
    """
        series - dataset with timeseries
        plot_intervals - show confidence intervals
        plot_anomalies - show anomalies
    """

    plt.figure(figsize=(20, 10))
    plt.plot(model.result, label = "Model")
    plt.plot(series.values, label = "Actual")
    error_mape = mean_absolute_percentage_error(series.values, model.result[:
↪len(series)])
    error_r2 = r2_score(series.values, model.result[:len(series)])
    error_mse = mean_squared_error(series.values, model.result[:len(series)])
    plt.title("R^2: {0:.2f}, MSE: {1:.2f}, MAPE: {2:.2f}%".format(error_r2,␣
↪error_mse, error_mape))

    if plot_anomalies:
        anomalies = np.array([np.NaN]*len(series))
        anomalies[series.values<model.LowerBond[:len(series)]] = \
            series.values[series.values<model.LowerBond[:len(series)]]
        anomalies[series.values>model.UpperBond[:len(series)]] = \
            series.values[series.values>model.UpperBond[:len(series)]]
        plt.plot(anomalies, "o", markersize=10, label = "Anomalies")

    if plot_intervals:
        plt.plot(model.UpperBond, "r--", alpha=0.5, label = "Up/Low confidence")
        plt.plot(model.LowerBond, "r--", alpha=0.5)
        plt.fill_between(x=range(0,len(model.result)), y1=model.UpperBond,
                         y2=model.LowerBond, alpha=0.2, color = "grey")

    plt.vlines(len(series), ymin=min(model.LowerBond), ymax=max(model.
↪UpperBond), linestyles='dashed')
    plt.axvspan(len(series)-20, len(model.result), alpha=0.3, color='lightgrey')
    plt.grid(True)
    plt.axis('tight')
    plt.legend(loc="best", fontsize=13);
```

[159]:
```
%%time
train = df_g1#[:-120]
#train, test = train_test_split(df_g1, 200)
season = 120
```

```
n_pred = 300

# initializing model parameters alpha, beta and gamma
x = [0, 0, 0]

start_time = time.time()
# Minimizing the loss function
opt = minimize(timeseriesCVscore, x0=x,
               args=(train, mean_squared_error, season),
               method="TNC", bounds = ((0, 1), (0, 1), (0, 1))
              )

# Take optimal values...
alpha_final, beta_final, gamma_final = opt.x
# print(alpha_final, beta_final, gamma_final)

# ...and train the model with them, forecasting for the next 50 hours
model = HoltWinters(train, slen = season,
                    alpha = alpha_final,
                    beta = beta_final,
                    gamma = gamma_final,
                    n_preds = n_pred, scaling_factor = 3)
HW_et = time.time() - start_time

model.triple_exponential_smoothing()
```

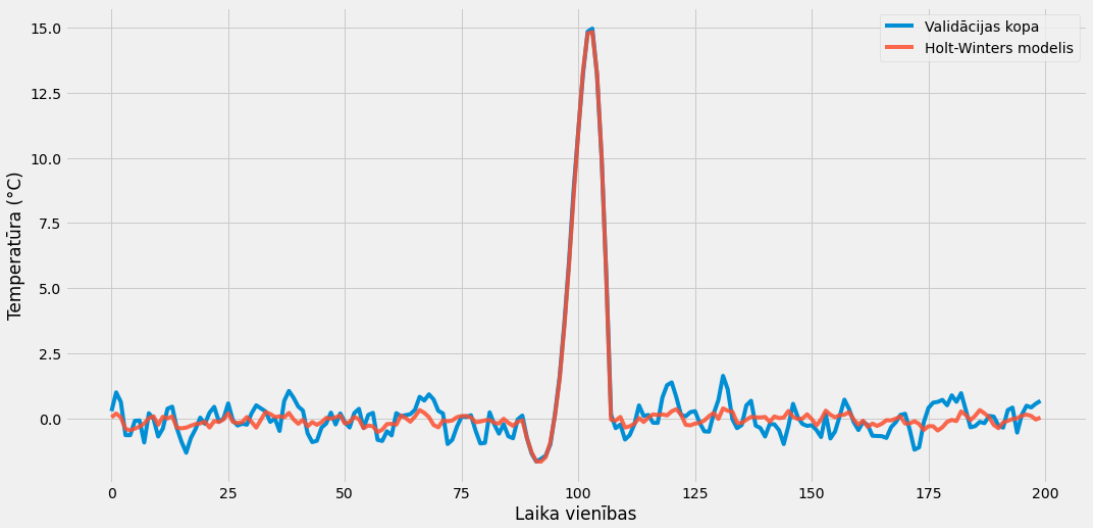
```
Wall time: 2.15 s
```

[163]:
```
# saglabājam HW modeli!
with open('models/HW_model.pickle', 'wb') as f: # Pickling!
    pickle.dump(model, f)
```

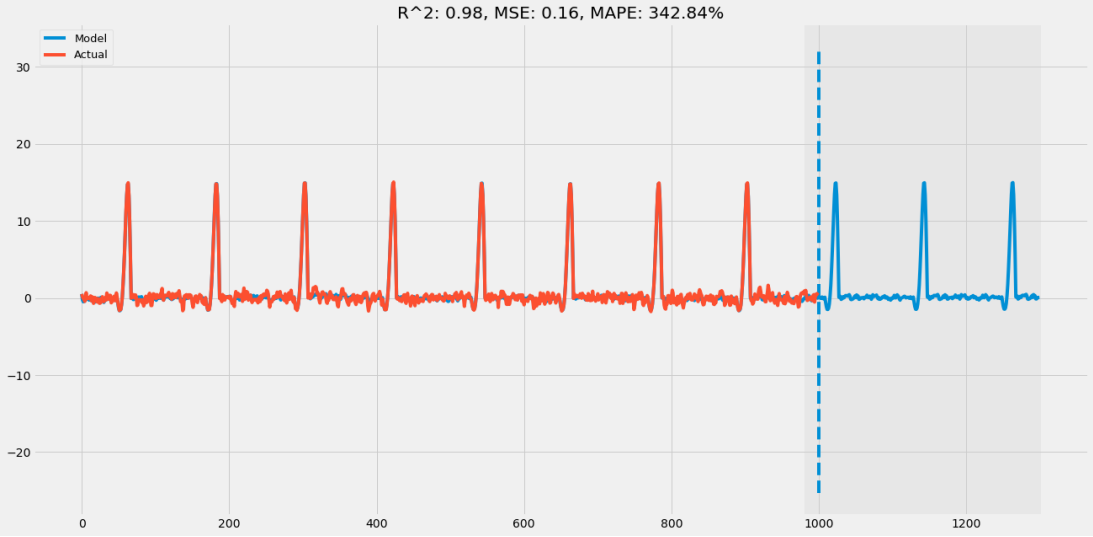
[26]:
```
HW_pred = model.result[800:1000]

pred_evaluation(test, HW_pred, 'Holt-Winters')
```

[26]:
```
{'Modelis': 'Holt-Winters',
 'Modeļa tips': '(n/a)',
 'Izpildes laiks, sekundes': 0,
 'R^2': 0.96,
 'MAE': 0.37,
 'MedAE': 0.31,
 'MSE': 0.22,
 'MAPE': 158.75,
 'Komentārs': ''}
```

[27]:
```
plt.plot(test)
plt.plot(HW_pred, alpha= .85)
plt.xlabel('Laika vienības')
plt.ylabel(u'Temperatūra (\u00B0C)')
plt.legend(['Validācijas kopa',
            'Holt-Winters modelis',
           ])
plt.savefig('images/prognozesana/HW_model.png');
```



[28]:
```
plotHoltWinters(df_g1, plot_intervals=False, plot_anomalies=False)
```



[34]:
```
from statsmodels.tsa.holtwinters import ExponentialSmoothing, SimpleExpSmoothing
from statsmodels.tools.eval_measures import rmse
from statsmodels.stats.diagnostic import acorr_ljungbox as ljung
from scipy.stats import jarque_bera as jb
```

[35]:
```
# m
```

2. ETS modelis

[36]:
```
train, test = train_test_split(df_g1, 200)
```

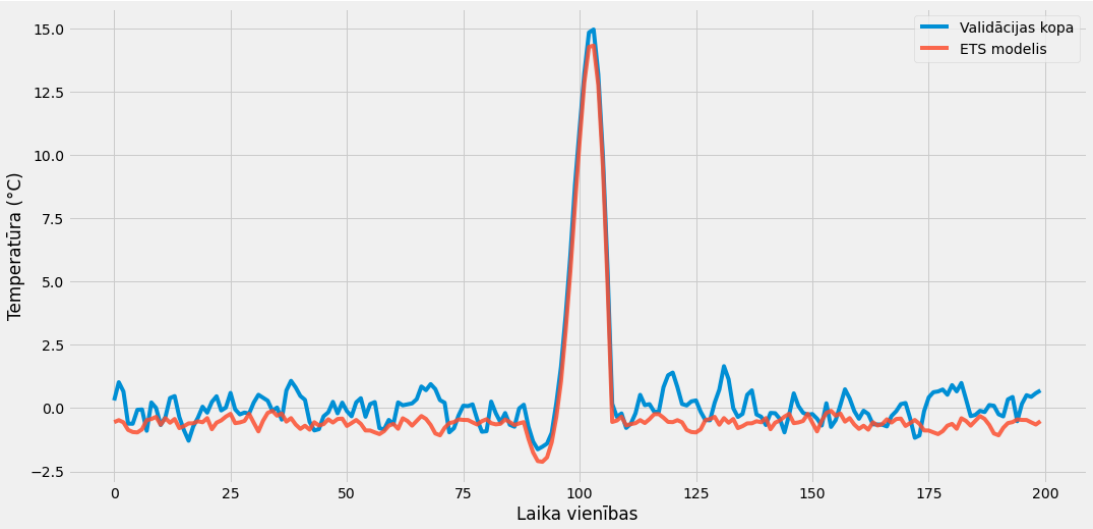
[37]:
```
%%time

start_time = time.time()
ets_model = sm.tsa.statespace.ExponentialSmoothing(train,
                                                   # trend=False,
                                                   initialization_method=␣
 ↪'concentrated', # estimated
                                                   seasonal=120,
                                                   # damped_trend=False
                                                  ).fit()
ets_et = time.time() - start_time
ets_pred = ets_model.forecast(200)

#predictions["LAdA"]=fc_LAdA
```

```
Wall time: 15.6 s
```

[38]:
```
pred_evaluation(test, ets_pred, 'ETS', exec_time=ets_et)
```

[38]:
```
{'Modelis': 'ETS',
 'Modeļa tips': '(n/a)',
 'Izpildes laiks, sekundes': 15.46,
 'R^2': 0.89,
 'MAE': 0.65,
 'MedAE': 0.56,
 'MSE': 0.63,
 'MAPE': 354.15,
 'Komentārs': ''}
```

[39]:
```
plt.plot(test)
plt.plot(ets_pred, alpha= .85)
plt.xlabel('Laika vienības')
plt.ylabel(u'Temperatūra (\u00B0C)')
plt.legend(['Validācijas kopa',
            'ETS modelis',
           ])
plt.savefig('images/prognozesana/ETS_model.png');
```



3. SARIMA

[40]:
```
#train, test = train_test_split(df_g1, 200)
n_season = 120
n_test = 200
```

[41]:
```
# # Izveidojam sarakstu ar parametriem, kas tik pārbaudīti!
# ps = range(0, 3)
# #ps = range(0, 1)
# d=1
# qs = range(0, 3)
# #qs = range(0, 1)
# Ps = range(0, 3)
# #Ps = range(0, 1)
# D=1
# Qs = range(0, 3)
# #Qs = range(0, 1)
# s =  120

# parameters = it.product(ps, qs, Ps, Qs)
# parameters_list = list(parameters)
# print(f'Kopā tiks pārbaudītdas {len(parameters_list)} dažādas kombinācijas')
```

[42]:
```
# # SARIMA optimizācijas funkcija!
# def optimizeSARIMA(series, parameters_list, d, D, s):
#     """
#         Return dataframe with parameters and corresponding AIC

#         parameters_list - list with (p, q, P, Q) tuples
#         d - integration order in ARIMA model
```

```
#         D - seasonal integration order
#         s - length of season
#     """

#     results = []
#     best_aic = float("inf")

#     for param in tqdm(parameters_list):
#         # we need try-except because on some combinations model fails to converge
#         try:
#             model=sm.tsa.statespace.SARIMAX(series, order=(param[0], d, param[1]),
#                                             seasonal_order=(param[2], D, param[3], s)).fit(disp=-1)
#         except:
#             continue
#         aic = model.aic
#         # saving best model, AIC and parameters
#         if aic < best_aic:
#             best_model = model
#             best_aic = aic
#             best_param = param
#         results.append([param, model.aic])

#     result_table = pd.DataFrame(results)
#     result_table.columns = ['parameters', 'aic']
#     # sorting in ascending order, the lower AIC is - the better
#     result_table = result_table.sort_values(by='aic', ascending=True).reset_index(drop=True)

#     return result_table
```

[43]:
```
# %%time
# result_table = optimizeSARIMA(train, parameters_list, d, D, s)
```

[44]:
```
# result_table
```

Dēļ garās sezonas pilnā pārlase aizņem ļoti daudz laika!

[45]:
```
# SARIMA funkcija!
# split a univariate dataset into train/test sets
def train_test_split(data, n_test):
    return np.array(data[:-n_test]), np.array(data[-n_test:])

# root mean squared error or rmse
def measure_rmse(actual, predicted):
```

```
    return sqrt(mean_squared_error(actual, predicted))

# one-step sarima forecast
def sarima_forecast(history, config, n_forward):
    order, sorder, trend = config
    # define model
    model = SARIMAX(history, order=order, seasonal_order=sorder, trend=trend,
                    enforce_stationarity=False, enforce_invertibility=False)
    # fit model
    model_fit = model.fit(disp=False)
    # make one step forecast
    fcast = model_fit.get_forecast(n_forward)
    yhat = fcast.predicted_mean
    confint = fcast.conf_int()

    return yhat, confint, model_fit

def SARIMA_n_forward(data, n_test, cfg, n_train_short = 0, plot=True):
    predictions = list()
    # split dataset
    train, test = train_test_split(data, n_test)
    # seed history with training dataset
    history = [x for x in train]
    # fit model and make forecast for history
    yhat, confint, model = sarima_forecast(history, cfg, n_test)
    # store forecast in list of predictions
    predictions = [i for i in yhat]

    # estimate prediction error
    error = measure_rmse(test, predictions)
    error_mape = mean_absolute_percentage_error(test, predictions)
    error_r2 = r2_score(test, predictions)
    error_mse = mean_squared_error(test, predictions)

    if plot==True:
        # calculate indexes for forecast
        index_of_fc = [i + len(train) for i in range(len(test))]
        index_of_train = [i for i in range(len(train))]

        # make series for plotting purpose
        train_series = pd.Series(train.reshape(-1), index=index_of_train)
        test_series = pd.Series(test.reshape(-1), index=index_of_fc)
        fitted_series = pd.Series(predictions, index=index_of_fc)
        lower_series = pd.Series(confint[:, 0], index=index_of_fc)
        upper_series = pd.Series(confint[:, 1], index=index_of_fc)

        # plot results and print error
```

```
        plt.plot(train_series)
        plt.plot(test_series, color = 'g', alpha=.75)
        plt.plot(fitted_series, 'r') # plotting t, b separately
        plt.fill_between(index_of_fc,
                         lower_series,
                         upper_series,
                         color = 'k',
                         alpha = .20)
        plt.title('')
        plt.show()

        if n_train_short > 0:
            plt.plot(train_series[-n_train_short:])
            plt.plot(test_series, color = 'g', alpha=.75)
            plt.plot(fitted_series, 'r') # plotting t, b separately
            plt.legend(['Treniņa datu kopa', 'Testa datu kopa', 'Modeļa␣
↪prognoze'], loc='upper right')
            plt.fill_between(index_of_fc,
                             lower_series,
                             upper_series,
                             color = 'k',
                             alpha = .20)
            plt.title("R^2: {0:.2f}, MSE: {1:.2f}, MAPE: {2:.2f}%".
↪format(error_r2, error_mse, error_mape))
            plt.show()

    print(f'Error: {round(error,2)}')

    return model
```

[46]:
```
%%time
start_time = time.time()
fc = SARIMA_n_forward(df_g1, n_test, [(1, 0, 0), (1, 0, 1, n_season), 'n'],␣
 ↪200, plot=False)
sarima_et = time.time() - start_time
```

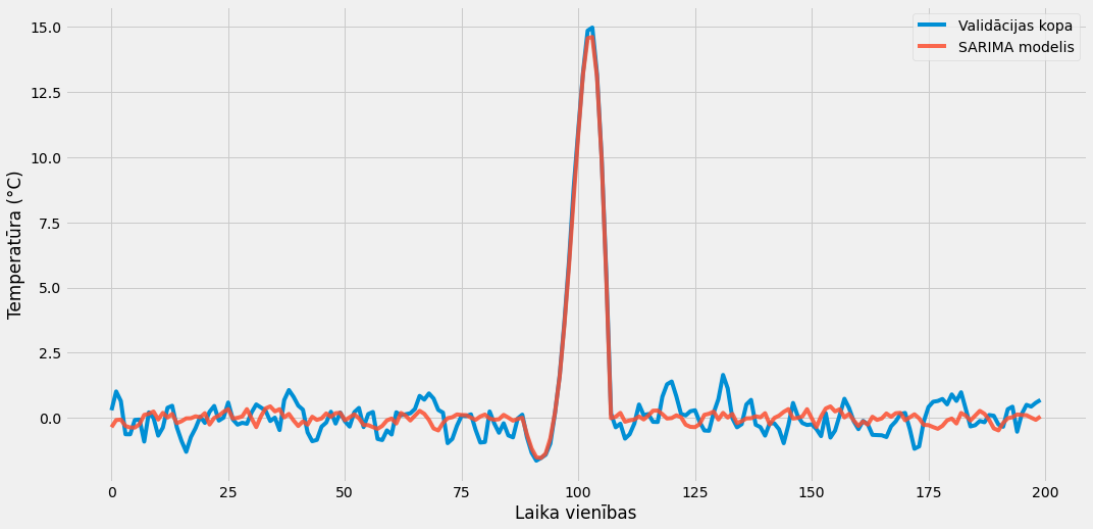
```
Error: 0.56
Wall time: 33.8 s
```

[47]:
```
sarima_pred = fc.get_forecast(200).predicted_mean
pred_evaluation(test, sarima_pred, 'SARIMA', sarima_et)
```

[47]:
```
{'Modelis': 'SARIMA',
 'Modeļa tips': '(n/a)',
 'Izpildes laiks, sekundes': 33.76,
 'R^2': 0.95,
 'MAE': 0.44,
```

```
 'MedAE': 0.38,
 'MSE': 0.31,
 'MAPE': 169.5,
 'Komentārs': ''}
```

[49]:
```
plt.plot(test)
plt.plot(sarima_pred, alpha= .85)
plt.xlabel('Laika vienības')
plt.ylabel(u'Temperatūra (\u00B0C)')
plt.legend(['Validācijas kopa',
            'SARIMA modelis',
           ])
plt.savefig('images/prognozesana/sarima_model.png');
```



4. FBProphet

[50]:
```
n_test = 200
df_fbp = pd.DataFrame({'ds': pd.date_range(start='2021.01.01', freq='10min',␣
 ↪periods=len(df_g1)),
                       'y': df_g1.values
                      })
train, test = df_fbp.iloc[:-n_test], df_fbp.iloc[-n_test:]
```

[51]:
```
# # Fit the model to train
# fb1_model=Prophet(n_changepoints=0,
#             seasonality_mode="additive").fit(train)

# #Prophet results are saved to a dataframe using make_future_dataframe()
```

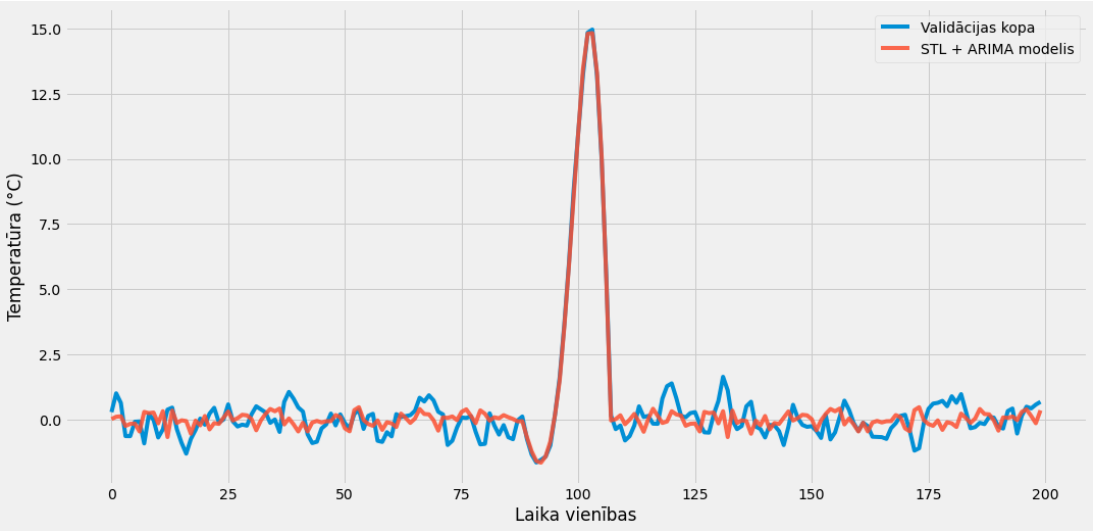
```
# fb1_df=fb1_model.make_future_dataframe(6, freq='10min') #set the freq␣
 ↪argument to 'Q' for quarterly data

# #We only need "ds" and "yhat" columns.. "ds" is the date column and "yhat"␣
 ↪are predictions

# fb1_fc_df=fb1_model.predict(fb1_df)[["ds","yhat"]]

# fb1_fc__=fb1_model.predict(fb1_df)

# #Residuals
# fb1_resid = train["Sales"].values - fb1_fc_df['yhat'].iloc[:len(train)]
# fb1_fc = fb1_fc_df.iloc[-len(test):]

# predictions["fb1"] = fb1_fc["yhat"].values
```

5. STL + ARIMA

[52]:
```
train, test = train_test_split(df_g1, 200)
```

[53]:
```
stlf = STLForecast(train, ARIMA, model_kwargs=dict(order=(1,0,1)), period=120,␣
 ↪seasonal_deg=0, trend_deg=1, low_pass_deg=1, robust=True)

start_time = time.time()
stlf_res = stlf.fit()
stlf_et = time.time() - start_time

stlf_pred = stlf_res.forecast(len(test))
```

[54]:
```
plt.plot(test)
plt.plot(stlf_pred, alpha= .85)
plt.xlabel('Laika vienības')
plt.ylabel(u'Temperatūra (\u00B0C)')
plt.legend(['Validācijas kopa',
            'STL + ARIMA modelis',
           ])
plt.savefig('images/prognozesana/stlf_model.png');
```



[55]:
```
pred_evaluation(test, stlf_pred, 'STLF', stlf_et)
```

[55]:
```
{'Modelis': 'STLF',
 'Modeļa tips': '(n/a)',
 'Izpildes laiks, sekundes': 1.25,
 'R^2': 0.95,
 'MAE': 0.45,
 'MedAE': 0.34,
 'MSE': 0.33,
 'MAPE': 213.73,
 'Komentārs': ''}
```

6. Theta

[56]:
```
train, test = train_test_split(df_g1, 200)
```

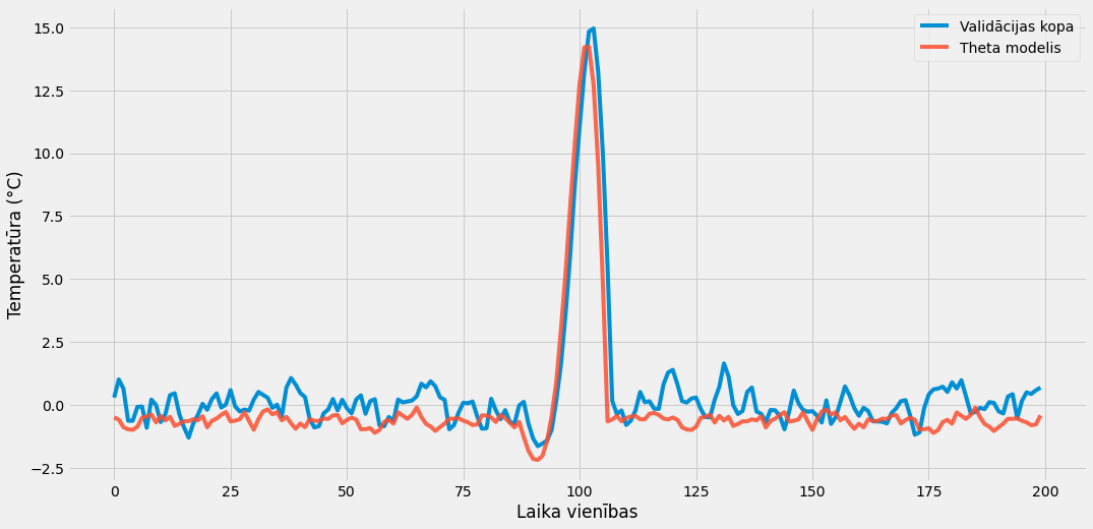
[57]:
```
m_theta = ThetaModel(train, period=120, deseasonalize=True, difference=True)

start_time = time.time()
theta_res = m_theta.fit()
theta_et = time.time() - start_time

print(theta_res.summary())
```

```
                              ThetaModel Results
==============================================================================
Dep. Variable:                   endog   No. Observations:                  800
Method:                        OLS/SES   Deseasonalized:                   True
Date:                Sun, 20 Jun 2021   Deseas. Method:               Additive
Time:                        00:09:20   Period:                            120
```

```
Sample:                              0
                                   800

    Parameter Estimates
============================
            Parameters
----------------------------
b0    -7.399830123964071e-05
alpha      0.5584193148957477
----------------------------
```

[58]:
```
theta_1_pred = theta_res.forecast(200, theta=1)
theta_pred = theta_res.forecast(200, theta=2)
theta_4_pred = theta_res.forecast(200, theta=3)
theta_inf_pred = theta_res.forecast(200, theta=np.inf)
```

[60]:
```
plt.plot(test)
plt.plot(range(len(theta_pred)), theta_pred, alpha= .85)
plt.xlabel('Laika vienības')
plt.ylabel(u'Temperatūra (\u00B0C)')
plt.legend(['Validācijas kopa',
            'Theta modelis',
           ])
plt.savefig('images/prognozesana/theta_model.png');
```



[61]:
```
pred_evaluation(test, theta_pred, 'theta', theta_et)
```

[61]:
```
{'Modelis': 'theta',
 'Modeļa tips': '(n/a)',
 'Izpildes laiks, sekundes': 0.04,
```

```
 'R^2': 0.81,
 'MAE': 0.78,
 'MedAE': 0.64,
 'MSE': 1.16,
 'MAPE': 468.61,
 'Komentārs': ''}
```

[62]:
```
# ax = theta_res.plot_predict(200, theta=100)
```

1.0.6 2. ML modeļi

[63]:
```
df = pd.DataFrame(df_g1)
train, test = train_test_split(df_g1, 200)
```

[64]:
```
df_test = df.iloc[:-200,:].rename({'g1_ts4': 'temp_lag_0'}, axis='columns')
```

[65]:
```
df_train = df.iloc[:-200,:].rename({'g1_ts4': 'temp_lag_0'}, axis='columns')
```

[66]:
```
df_train['temp_lag_1'] = df_train['temp_lag_0'].shift(1)
df_train['temp_lag_2'] = df_train['temp_lag_0'].shift(2)
df_train['temp_lag_3'] = df_train['temp_lag_0'].shift(3)
df_train['temp_lag_4'] = df_train['temp_lag_0'].shift(4)

df_train['temp_lag_60'] = df_train['temp_lag_0'].shift(60)
df_train['temp_lag_118'] = df_train['temp_lag_0'].shift(118)
df_train['temp_lag_119'] = df_train['temp_lag_0'].shift(119)
df_train['temp_lag_120'] = df_train['temp_lag_0'].shift(120)

df_train['temp_rolling_10_mean'] = df_train['temp_lag_0'].rolling(window=10).
 ↪mean()
df_train['temp_rolling_10_min'] = df_train['temp_lag_0'].rolling(window=10).
 ↪min()
df_train['temp_rolling_10_max'] = df_train['temp_lag_0'].rolling(window=10).
 ↪max()
df_train['temp_rolling_10_median'] = df_train['temp_lag_0'].rolling(window=10).
 ↪median()

df_train['y'] = df_train['temp_lag_0'].shift(-1)
```

[67]:
```
df_train.dropna(inplace=True)
```

[68]:
```
df_train[:5].to_excel('excel_to_convert/ML_dataset.xlsx', index=False)
```

[69]:
```
trainX, trainY = df_train.loc[:, df_train.columns != 'y'], df_train.loc[:,␣
 ↪df_train.columns == 'y']
```

[70]:
```
#X, y = make_classification(n_samples=1000, n_features=10, n_informative=5,␣
↪n_redundant=5, random_state=1)
```

LightBoost

[71]:
```
# # define dataset
# X, y = make_regression(n_samples=1000, n_features=10, n_informative=5,␣
↪random_state=1)
# # evaluate the model
# model = LGBMRegressor()
# cv = RepeatedKFold(n_splits=10, n_repeats=3, random_state=1)
# n_scores = cross_val_score(model, X, y, scoring='neg_mean_squared_error',␣
↪cv=cv, n_jobs=-1, error_score='raise')
# print('MAE: %.3f (%.3f)' % (np.mean(n_scores), np.std(n_scores)))
# # fit the model on the whole dataset
# model = LGBMRegressor()
# model.fit(X, y)
```

[72]:
```
# evaluate the model
#model = LGBMRegressor(learning_rate=0.1, num_leaves=20)
model = LGBMRegressor(learning_rate=0.1, num_leaves=20)

start_time = time.time()
cv = RepeatedKFold(n_splits=5, n_repeats=10, random_state=12)
n_scores = cross_val_score(model, trainX, trainY,␣
↪scoring='neg_mean_squared_error', cv=cv, n_jobs=-1, error_score='raise')
print('MSE: %.3f (%.3f)' % (np.mean(n_scores), np.std(n_scores)))
# fit the model on the whole dataset
#model = LGBMRegressor()
model.fit(trainX, trainY)
lightboost_et = time.time() - start_time
```

```
MSE: -0.208 (0.027)
```

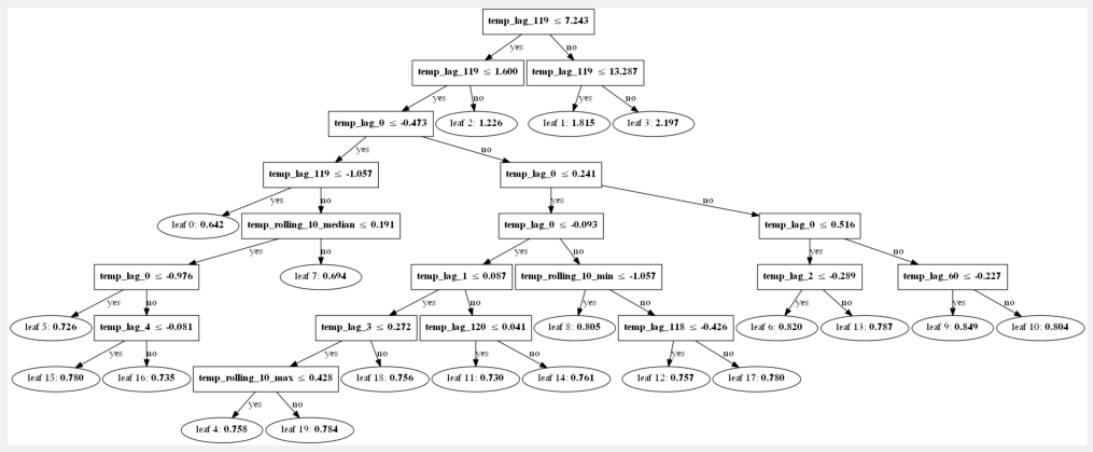
[73]:
```
df_test = df.iloc[:-200,:].rename({'g1_ts4': 'temp_lag_0'}, axis='columns')
n_pred = 200
lightboost_pred = np.array([])

for i in range(n_pred):
    df_tmp = df_test.copy()

    # Neefektīva implementācija - jāpārtaisa korektāk!!!
    df_tmp['temp_lag_1'] = df_tmp['temp_lag_0'].shift(1)
    df_tmp['temp_lag_2'] = df_tmp['temp_lag_0'].shift(2)
    df_tmp['temp_lag_3'] = df_tmp['temp_lag_0'].shift(3)
    df_tmp['temp_lag_4'] = df_tmp['temp_lag_0'].shift(4)

    df_tmp['temp_lag_60'] = df_tmp['temp_lag_0'].shift(60)
```

```
    df_tmp['temp_lag_118'] = df_tmp['temp_lag_0'].shift(118)
    df_tmp['temp_lag_119'] = df_tmp['temp_lag_0'].shift(119)
    df_tmp['temp_lag_120'] = df_tmp['temp_lag_0'].shift(120)

    df_tmp['temp_rolling_10_mean'] = df_tmp['temp_lag_0'].rolling(window=10).
↪mean()
    df_tmp['temp_rolling_10_min'] = df_tmp['temp_lag_0'].rolling(window=10).
↪min()
    df_tmp['temp_rolling_10_max'] = df_tmp['temp_lag_0'].rolling(window=10).
↪max()
    df_tmp['temp_rolling_10_median'] = df_tmp['temp_lag_0'].rolling(window=10).
↪median()

    X = df_tmp.loc[:, df_tmp.columns != 'temp'][-1:].values.tolist()

    yhat = model.predict(X)
    df_test = df_test.append({'temp_lag_0': yhat[0]}, ignore_index=True)
    lightboost_pred = np.append(lightboost_pred, yhat)
```

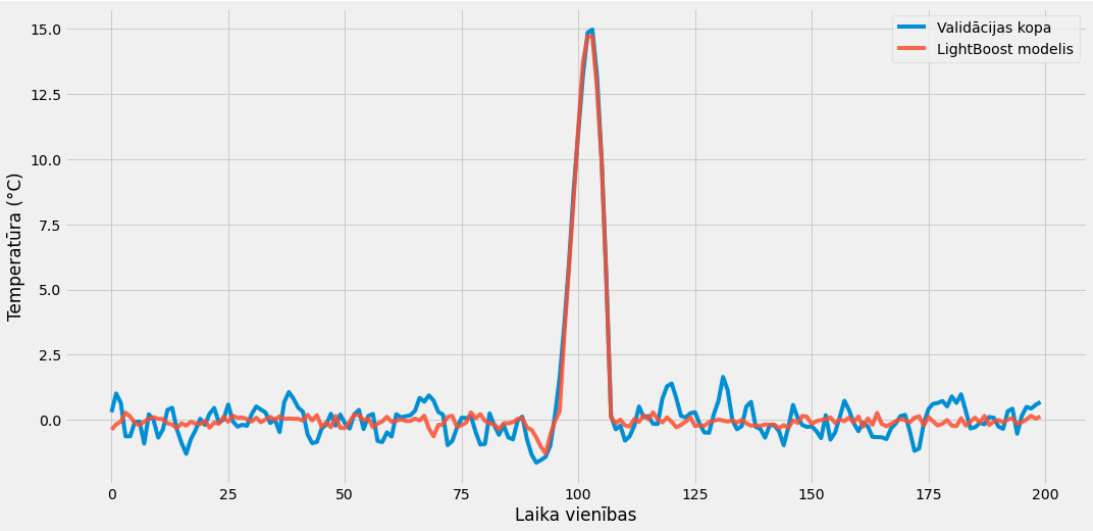
[74]:
```
pred_evaluation(test, lightboost_pred, 'LightGBM', lightboost_et)
```

[74]:
```
{'Modelis': 'LightGBM',
 'Modeļa tips': '(n/a)',
 'Izpildes laiks, sekundes': 3.13,
 'R^2': 0.95,
 'MAE': 0.45,
 'MedAE': 0.36,
 'MSE': 0.32,
 'MAPE': 113.75,
 'Komentārs': ''}
```

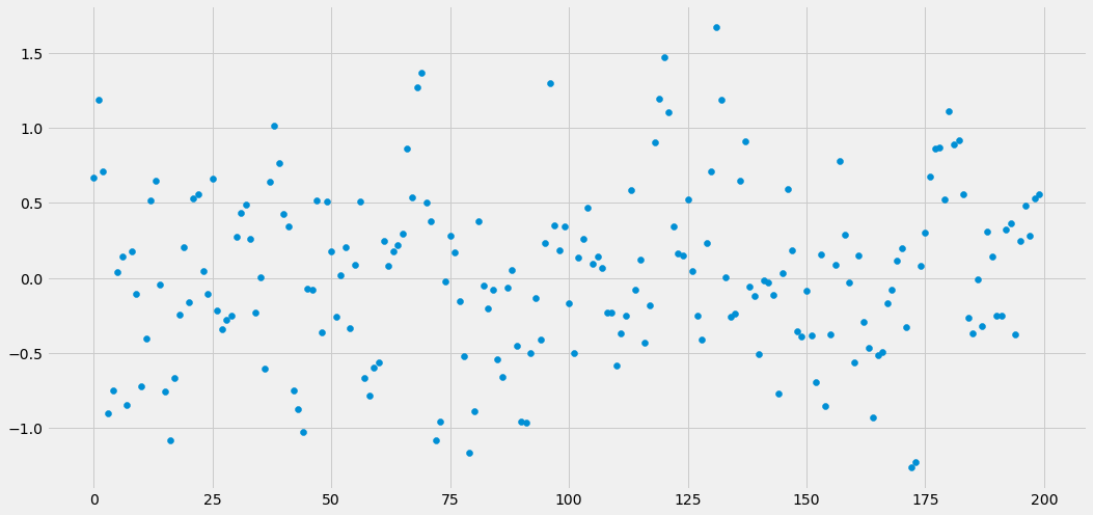
[75]:
```
plot_tree(model, tree_index=0, orientation='vertical')
plt.savefig('images/prognozesana/lightboost_model_tree_0.png');
```



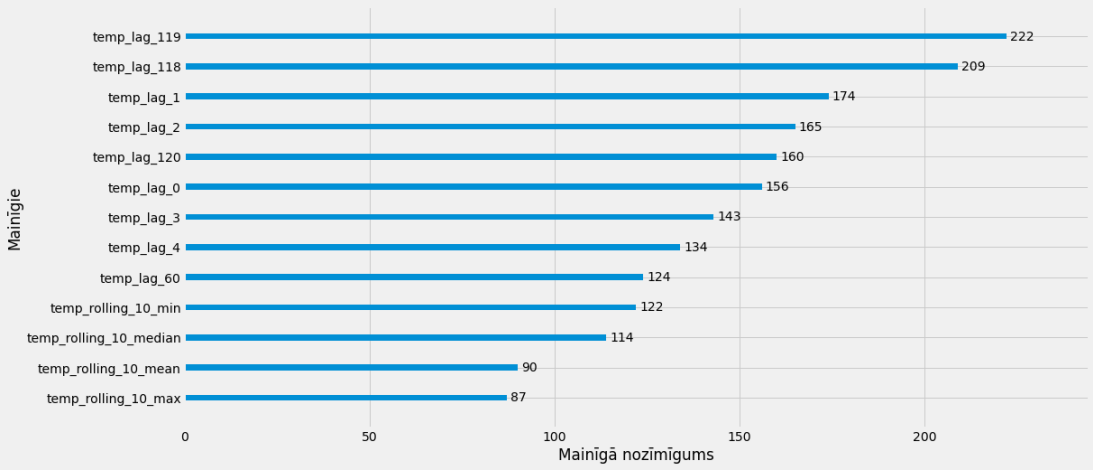
[76]:
```
plt.plot(test)
plt.plot(lightboost_pred, alpha= .85)
plt.xlabel('Laika vienības')
plt.ylabel(u'Temperatūra (\u00B0C)')
plt.legend(['Validācijas kopa',
            'LightBoost modelis',
           ])
plt.savefig('images/prognozesana/lightboost.png');
```



[79]:
```
plt.scatter([i for i in range(len(test))], test- lightboost_pred);
```



[80]:
```
lightboost_plot_importance(model, title='', ylabel='Mainīgie', xlabel='Mainīgā␣
↪nozīmīgums')
plt.savefig('images/prognozesana/lightboost_importance.png',␣
↪bbox_inches='tight');
```



XGBoost

[81]:
```
start_time = time.time()
model = XGBRegressor(objective='reg:squarederror')
cv = RepeatedKFold(n_splits=10, n_repeats=3, random_state=42)
n_scores = cross_val_score(model, trainX, trainY,␣
↪scoring='neg_mean_squared_error', cv=cv, n_jobs=-1, error_score='raise')
print('MSE: %.3f (%.3f)' % (np.mean(n_scores), np.std(n_scores)))
# fit the model on the whole dataset
#model = XGBRegressor(objective='reg:squarederror')
model.fit(trainX, trainY)
xgboost_et = time.time() - start_time
```

```
MSE: -0.190 (0.032)
```

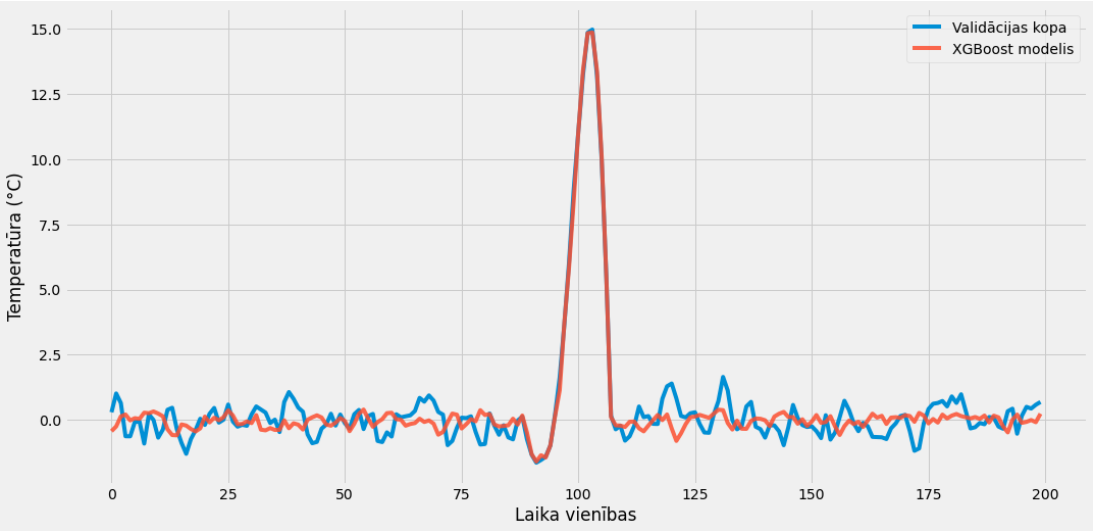
[82]:
```
df_test = df.iloc[:-200,:].rename({'g1_ts4': 'temp_lag_0'}, axis='columns')
n_pred = 200
xgboost_pred = np.array([])

for i in range(n_pred):
    df_tmp = df_test.copy()

    # Neefektīva implementācija - jāpārtaisa korektāk!!!
    df_tmp['temp_lag_1'] = df_tmp['temp_lag_0'].shift(1)
```

```
    df_tmp['temp_lag_2'] = df_tmp['temp_lag_0'].shift(2)
    df_tmp['temp_lag_3'] = df_tmp['temp_lag_0'].shift(3)
    df_tmp['temp_lag_4'] = df_tmp['temp_lag_0'].shift(4)

    df_tmp['temp_lag_60'] = df_tmp['temp_lag_0'].shift(60)
    df_tmp['temp_lag_118'] = df_tmp['temp_lag_0'].shift(118)
    df_tmp['temp_lag_119'] = df_tmp['temp_lag_0'].shift(119)
    df_tmp['temp_lag_120'] = df_tmp['temp_lag_0'].shift(120)

    df_tmp['temp_rolling_10_mean'] = df_tmp['temp_lag_0'].rolling(window=10).
↪mean()
    df_tmp['temp_rolling_10_min'] = df_tmp['temp_lag_0'].rolling(window=10).
↪min()
    df_tmp['temp_rolling_10_max'] = df_tmp['temp_lag_0'].rolling(window=10).
↪max()
    df_tmp['temp_rolling_10_median'] = df_tmp['temp_lag_0'].rolling(window=10).
↪median()

    X = df_tmp.loc[:, df_tmp.columns != 'temp'][-1:].values

    yhat = model.predict(X)
    df_test = df_test.append({'temp_lag_0': yhat[0]}, ignore_index=True)
    xgboost_pred = np.append(xgboost_pred, yhat)
```

[83]:
```
pred_evaluation(test, xgboost_pred, 'XGBoost', xgboost_et)
```

[83]:
```
{'Modelis': 'XGBoost',
 'Modeļa tips': '(n/a)',
 'Izpildes laiks, sekundes': 2.96,
 'R^2': 0.94,
 'MAE': 0.46,
 'MedAE': 0.39,
 'MSE': 0.34,
 'MAPE': 224.62,
 'Komentārs': ''}
```

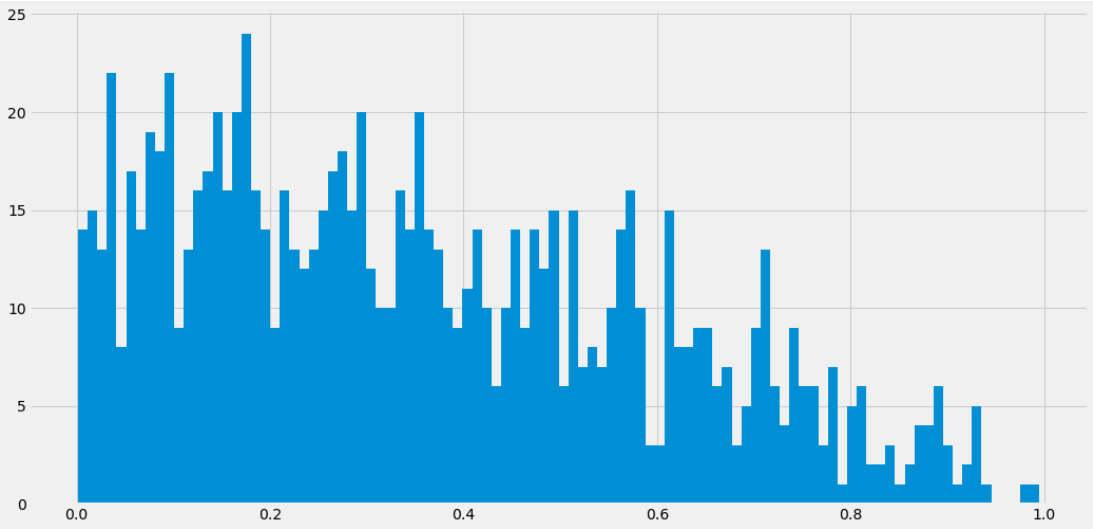
[84]:
```
plt.plot(test)
plt.plot(xgboost_pred, alpha= .85)
plt.xlabel('Laika vienības')
plt.ylabel(u'Temperatūra (\u00B0C)')
plt.legend(['Validācijas kopa',
            'XGBoost modelis',
           ])
plt.savefig('images/prognozesana/xgboost.png');
```



[85]:
```
from scipy import stats
data = test- xgboost_pred
```

[86]:
```
shap_p = []
# data = np.random.uniform(size= 1000)

for i in range(1000):
    tmp = np.random.choice(data, 100)
    shapiro_test = stats.shapiro(tmp)
    shap_p.append(shapiro_test.pvalue)

#plt.hist(data, bins=20)
```

[87]:
```
plt.hist(shap_p, bins=100);
```



[88]:
```
np.mean(shap_p)
```

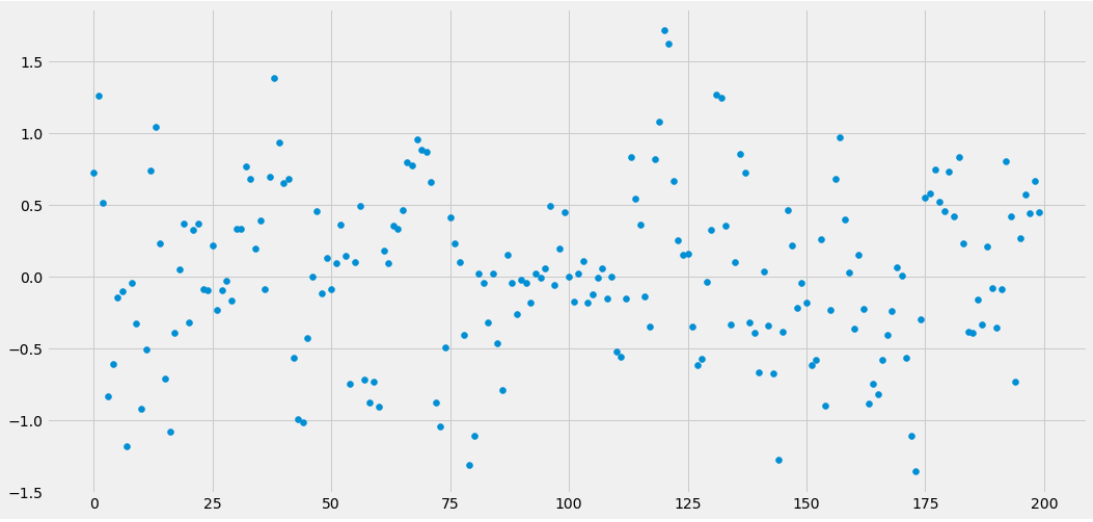
[88]: 0.3577647758719977

[89]:
```
stats.shapiro(data).pvalue
```

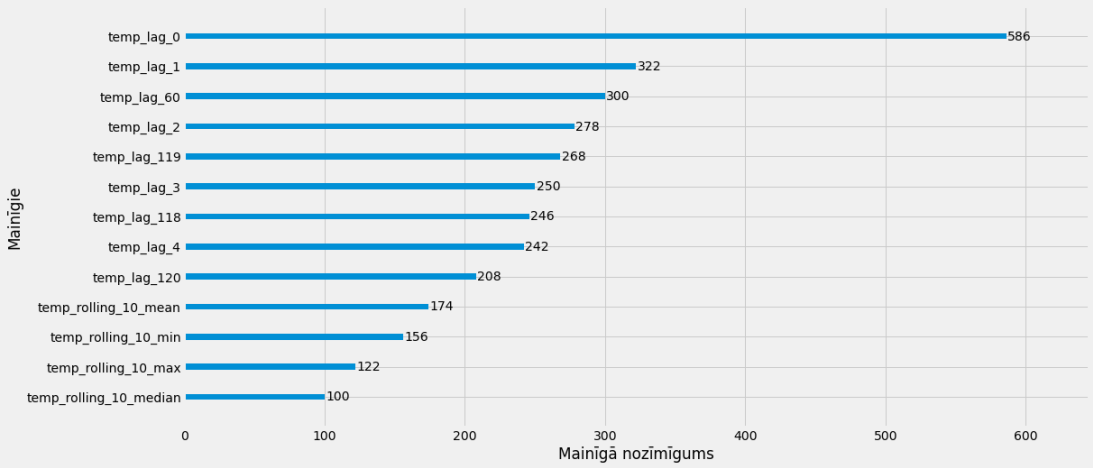
[89]: 0.8092828392982483

[90]:
```
plt.scatter([i for i in range(len(test))], test- xgboost_pred)
```

[90]: <matplotlib.collections.PathCollection at 0x13f117abac0>



[91]:
```
plot_importance(model, title='', ylabel='Mainīgie', xlabel='Mainīgā nozīmīgums')
plt.savefig('images/prognozesana/xgboost_importance.png', bbox_inches='tight');
```



CatBoost

[92]:
```
start_time = time.time()
model = CatBoostRegressor(verbose=0, n_estimators=100)
cv = RepeatedKFold(n_splits=10, n_repeats=3, random_state=1)
n_scores = cross_val_score(model, trainX, trainY,␣
 ↪scoring='neg_mean_squared_error', cv=cv, n_jobs=-1, error_score='raise')
print('MSE: %.3f (%.3f)' % (np.mean(n_scores), np.std(n_scores)))
# fit the model on the whole dataset
model = CatBoostRegressor(verbose=0, n_estimators=100)
model.fit(trainX, trainY)
catboost_et = time.time() - start_time
```

```
MSE: -0.201 (0.031)
```

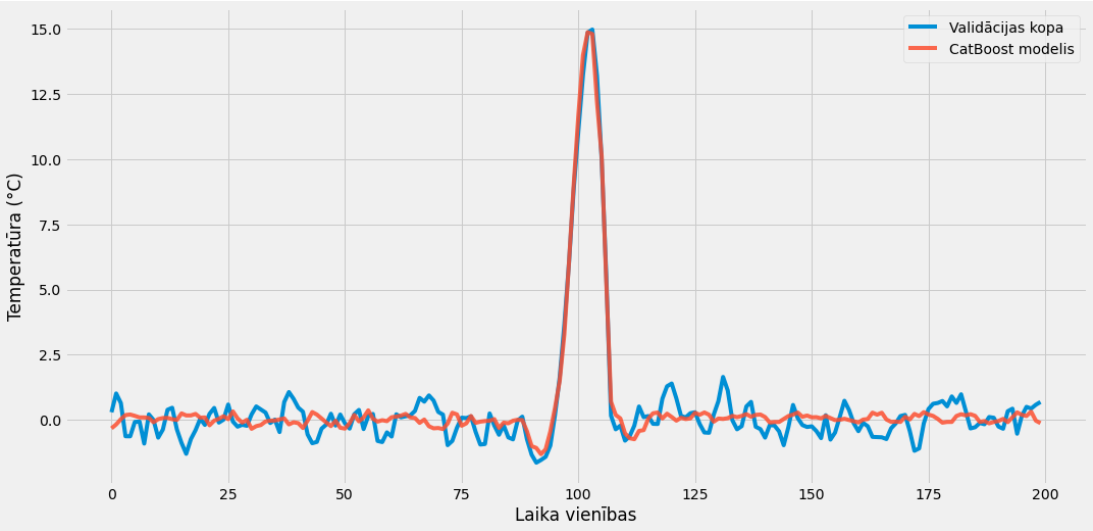
[93]:
```
df_test = df.iloc[:-200,:].rename({'g1_ts4': 'temp_lag_0'}, axis='columns')
n_pred = 200
catboost_pred = np.array([])

for i in range(n_pred):
    df_tmp = df_test.copy()

    # Neefektīva implementācija - jāpārtaisa korektāk!!!
    df_tmp['temp_lag_1'] = df_tmp['temp_lag_0'].shift(1)
    df_tmp['temp_lag_2'] = df_tmp['temp_lag_0'].shift(2)
    df_tmp['temp_lag_3'] = df_tmp['temp_lag_0'].shift(3)
    df_tmp['temp_lag_4'] = df_tmp['temp_lag_0'].shift(4)

    df_tmp['temp_lag_60'] = df_tmp['temp_lag_0'].shift(60)
```

```
    df_tmp['temp_lag_118'] = df_tmp['temp_lag_0'].shift(118)
    df_tmp['temp_lag_119'] = df_tmp['temp_lag_0'].shift(119)
    df_tmp['temp_lag_120'] = df_tmp['temp_lag_0'].shift(120)

    df_tmp['temp_rolling_10_mean'] = df_tmp['temp_lag_0'].rolling(window=10).
↪mean()
    df_tmp['temp_rolling_10_min'] = df_tmp['temp_lag_0'].rolling(window=10).
↪min()
    df_tmp['temp_rolling_10_max'] = df_tmp['temp_lag_0'].rolling(window=10).
↪max()
    df_tmp['temp_rolling_10_median'] = df_tmp['temp_lag_0'].rolling(window=10).
↪median()

    X = df_tmp.loc[:, df_tmp.columns != 'temp'][-1:].values

    yhat = model.predict(X)
    df_test = df_test.append({'temp_lag_0': yhat[0]}, ignore_index=True)
    catboost_pred = np.append(catboost_pred, yhat)
```

[94]:
```
pred_evaluation(test, catboost_pred, 'CatBoost', catboost_et)
```

[94]:
```
{'Modelis': 'CatBoost',
 'Modeļa tips': '(n/a)',
 'Izpildes laiks, sekundes': 12.82,
 'R^2': 0.94,
 'MAE': 0.47,
 'MedAE': 0.43,
 'MSE': 0.34,
 'MAPE': 155.01,
 'Komentārs': ''}
```

[95]:
```
plt.plot(test)
plt.plot(catboost_pred, alpha= .85)
plt.xlabel('Laika vienības')
plt.ylabel(u'Temperatūra (\u00B0C)')
plt.legend(['Validācijas kopa',
            'CatBoost modelis',
           ])
plt.savefig('images/prognozesana/catboost.png');
```



SVM

[96]:
```
trainY_svm = trainY.iloc[:, 0].values.tolist()
trainX_svm = trainX.values.tolist()
```

[97]:
```
start_time = time.time()
SVM_model = svm.SVR(kernel='rbf') # linear', 'poly', 'rbf', 'sigmoid',␣
 ↪'precomputed'
SVM_model.fit(trainX_svm, trainY_svm)
svm_et = time.time() - start_time
```

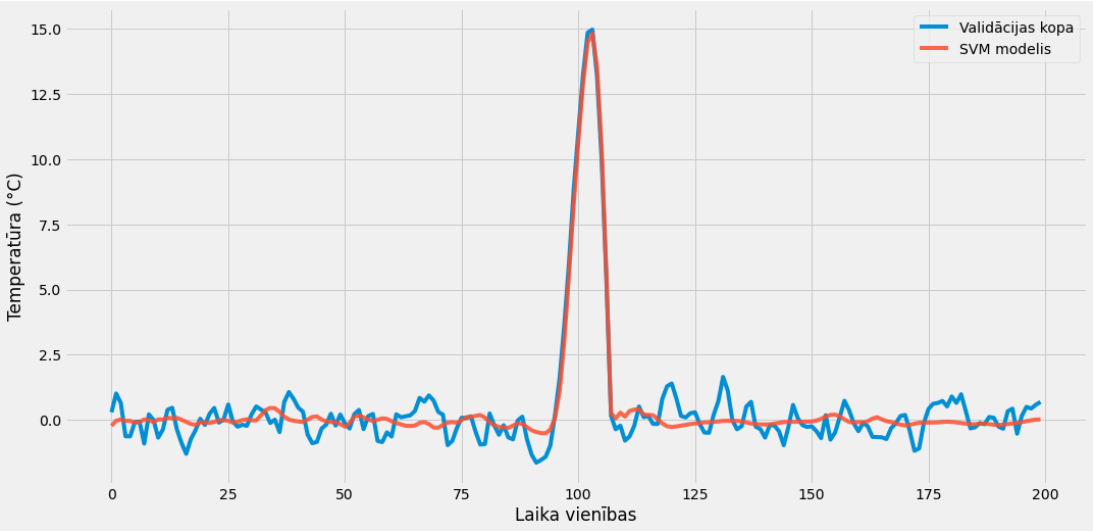
[98]:
```
df_test = df.iloc[:-200,:].rename({'g1_ts4': 'temp_lag_0'}, axis='columns')
n_pred = 200
svm_pred = np.array([])

for i in range(n_pred):
    df_tmp = df_test.copy()

    # Neefektīva implementācija - jāpārtaisa korektāk!!!
    df_tmp['temp_lag_1'] = df_tmp['temp_lag_0'].shift(1)
    df_tmp['temp_lag_2'] = df_tmp['temp_lag_0'].shift(2)
    df_tmp['temp_lag_3'] = df_tmp['temp_lag_0'].shift(3)
    df_tmp['temp_lag_4'] = df_tmp['temp_lag_0'].shift(4)

    df_tmp['temp_lag_60'] = df_tmp['temp_lag_0'].shift(60)
    df_tmp['temp_lag_118'] = df_tmp['temp_lag_0'].shift(118)
    df_tmp['temp_lag_119'] = df_tmp['temp_lag_0'].shift(119)
    df_tmp['temp_lag_120'] = df_tmp['temp_lag_0'].shift(120)
```

```
    df_tmp['temp_rolling_10_mean'] = df_tmp['temp_lag_0'].rolling(window=10).
↪mean()
    df_tmp['temp_rolling_10_min'] = df_tmp['temp_lag_0'].rolling(window=10).
↪min()
    df_tmp['temp_rolling_10_max'] = df_tmp['temp_lag_0'].rolling(window=10).
↪max()
    df_tmp['temp_rolling_10_median'] = df_tmp['temp_lag_0'].rolling(window=10).
↪median()

    X = df_tmp.loc[:, df_tmp.columns != 'temp'][-1:].values

    yhat = SVM_model.predict(X)
    df_test = df_test.append({'temp_lag_0': yhat[0]}, ignore_index=True)
    svm_pred = np.append(svm_pred, yhat)
```

[99]:
```
pred_evaluation(test, svm_pred, 'SVM', svm_et)
```

[99]:
```
{'Modelis': 'SVM',
 'Modeļa tips': '(n/a)',
 'Izpildes laiks, sekundes': 0.03,
 'R^2': 0.94,
 'MAE': 0.47,
 'MedAE': 0.4,
 'MSE': 0.34,
 'MAPE': 122.29,
 'Komentārs': ''}
```

[100]:
```
plt.plot(test)
plt.plot(svm_pred, alpha= .85)
plt.xlabel('Laika vienības')
plt.ylabel(u'Temperatūra (\u00B0C)')
plt.legend(['Validācijas kopa',
            'SVM modelis',
           ])
plt.savefig('images/prognozesana/SVM.png');
```



1.0.7 3. Dziļie neironu tīklu modeļi

[101]:
```
def convert2matrix(data_arr, look_back):
    X, Y =[], []
    for i in range(len(data_arr)-look_back):
        d=i+look_back
        X.append(data_arr[i:d,0])
        Y.append(data_arr[d,0])
    return np.array(X), np.array(Y)
```

[102]:
```
df = pd.DataFrame(df_g1)
```

LSTM

[113]:
```
from sklearn.preprocessing import MinMaxScaler, StandardScaler
df = pd.DataFrame(df_g1)

#create numpy.ndarray
df_arr= df.values
# df.values = df_arr.astype('float32')
df_arr = np.reshape(df_arr, (-1, 1)) #LTSM requires more input features␣
 ↪compared to RNN or DNN
scaler = StandardScaler()#LTSM is senstive to the scale of features
df_arr = scaler.fit_transform(df_arr)
```

[115]:
```
def model_loss(history, loss_name='RMSE'):
    # plt.figure(figsize=(8,4))
    plt.plot(history.history['loss'], label=f'Treniņa kopas {loss_name}')
```

```
    plt.plot(history.history['val_loss'], label=f'Validācijas kopas␣
↪{loss_name}')
    # plt.title('model loss')
    plt.ylabel(loss_name)
    plt.xlabel('Epohi')
    plt.legend(loc='upper right')
    plt.show();
```

[116]:
```
train_size = 750
look_back = 120

test_size = len(df_arr) - train_size
train, test = df_arr[0:train_size,:], df_arr[train_size:len(df_arr),:]
trainX, trainY = convert2matrix(train, look_back)
testX, testY = convert2matrix(test, look_back)
# reshape input to be [samples, time steps, features]
trainX = np.reshape(trainX, (trainX.shape[0], 1, trainX.shape[1]))
testX = np.reshape(testX, (testX.shape[0], 1, testX.shape[1]))
```

[117]:
```
from keras.models import Sequential
from keras.layers import Dense, LSTM, Dropout, TimeDistributed, Flatten
from keras.callbacks import EarlyStopping
from tensorflow.keras import regularizers
from keras.layers.convolutional import Conv1D, MaxPooling1D

def model_lstm(look_back):
    model=Sequential()
    #model.add(LSTM(20, input_shape=(1, look_back), activation='relu',␣
↪return_sequences=True)) # 200
    model.add(LSTM(100, input_shape=(1, look_back), activation='relu'))
    #model.add(Dropout(0.20))
    #model.add(LSTM(100, activation='relu')) # 100
    model.add(Dense(100, activation='relu')) # 100
    #model.add(Dropout(0.20))
    model.add(Dense(50, activation='relu'))
    #model.add(Dropout(0.3))
    #model.add(Dense(50, activation='relu')) # 50
    #model.add(Dense(25, activation='relu')) # 25
    model.add(Dense(1))
    model.compile(loss='mean_squared_error',  optimizer='adam', metrics =␣
↪['mse', 'mae'])
    return model
```

[118]:
```
%%time
model=model_lstm(look_back)
model.summary()
```

```
start_time = time.time()
history = model.fit(trainX,
                    trainY,
                    epochs=1000,
                    batch_size=20,
                    validation_data=(testX, testY),
                    #callbacks=[EarlyStopping(monitor='val_loss', patience=10)],
                    verbose=0,
                    workers=5,
                    shuffle=False
                   )
lstm_et = time.time() - start_time
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm (LSTM)                  (None, 100)               88400
_________________________________________________________________
dense (Dense)                (None, 100)               10100
_________________________________________________________________
dense_1 (Dense)              (None, 50)                5050
_________________________________________________________________
dense_2 (Dense)              (None, 1)                 51
=================================================================
Total params: 103,601
Trainable params: 103,601
Non-trainable params: 0
_________________________________________________________________
Wall time: 1min 17s
```

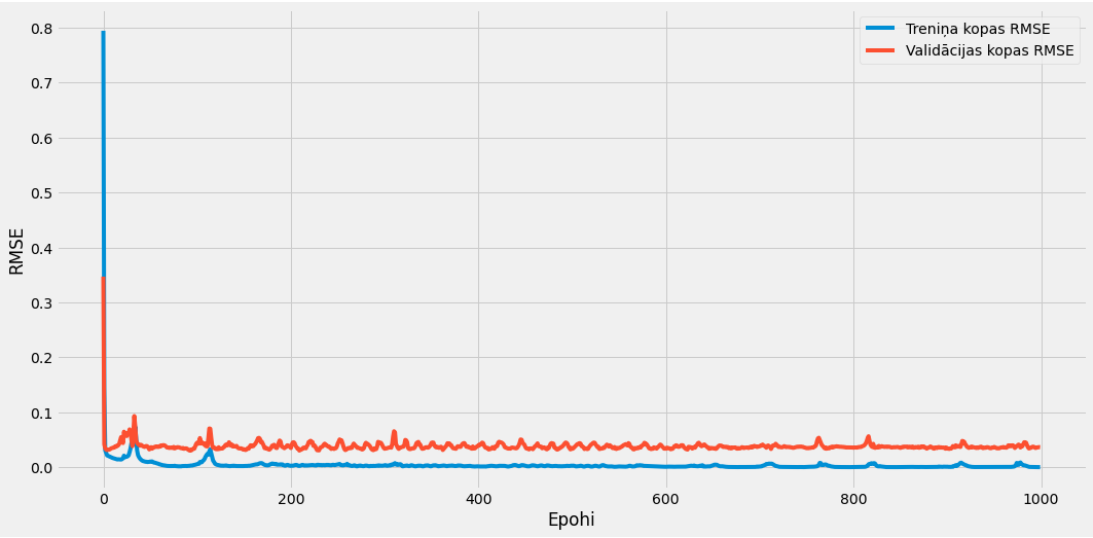
[119]:
```
# plot_model() -
#     model: A Keras model instance
#     to_file: File name of the plot image.
#     show_shapes: whether to display shape information.
#     show_dtype: whether to display layer dtypes.
#     show_layer_names: whether to display layer names.
#     rankdir: rankdir argument passed to PyDot, a string specifying the format␣
 ↪of the plot: 'TB' creates a vertical plot; 'LR' creates a horizontal plot.
#     expand_nested: Whether to expand nested models into clusters.
#     dpi: Dots per inch.


plot_model(model,
           to_file = 'images/prognozesana/lstm_arhitektūra.png',
           show_shapes = True,
           show_dtype = False,
```

```
            show_layer_names = True,
            rankdir = 'LR',
            expand_nested = True,
            dpi = 300
           )
```

```
('You must install pydot (`pip install pydot`) and install graphviz (see
instructions at https://graphviz.gitlab.io/download/) ', 'for
plot_model/model_to_dot to work.')
```

[122]:
```
#from keras_visualizer import visualizer
#model.summary()
# visualizer(model)
```

[123]:
```
train_predict = model.predict(trainX)
test_predict = model.predict(testX)
# invert predictions
train_predict = scaler.inverse_transform(train_predict)
trainY = scaler.inverse_transform([trainY])
test_predict = scaler.inverse_transform(test_predict)
testY = scaler.inverse_transform([testY])
#print('Train Root Mean Squared Error(RMSE): %.2f; Train Mean Absolute␣
 ↪Error(MAE) : %.2f '% (np.sqrt(mean_squared_error(trainY[0], train_predict[:
 ↪,0])),(mean_absolute_error(trainY[0], train_predict[:,0]))))
#print('Test Root Mean Squared Error(RMSE): %.2f; Test Mean Absolute Error(MAE)␣
 ↪: %.2f '% (np.sqrt(mean_squared_error(testY[0], test_predict[:
 ↪,0])),(mean_absolute_error(testY[0], test_predict[:,0]))))
model_loss(history)
```

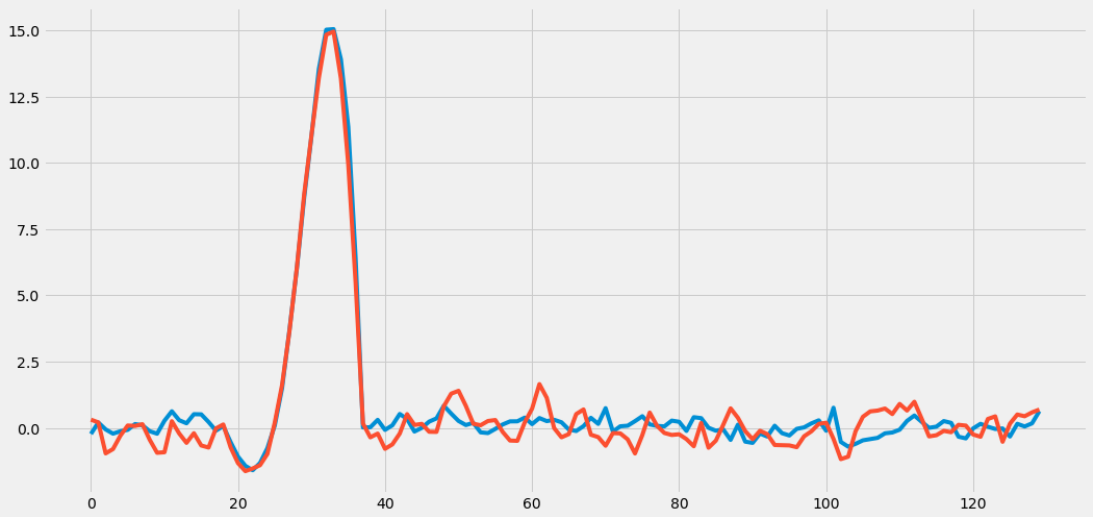


[124]:
```
pred_evaluation(testY.reshape(-1), test_predict, 'LSTM', lstm_et)
```

[124]:
```
{'Modelis': 'LSTM',
 'Modeļa tips': '(n/a)',
 'Izpildes laiks, sekundes': 77.65,
 'R^2': 0.96,
 'MAE': 0.47,
 'MedAE': 0.38,
 'MSE': 0.33,
 'MAPE': 758.05,
 'Komentārs': ''}
```

[128]:
```
plt.plot(test_predict)
plt.plot(testY.reshape(-1))
```

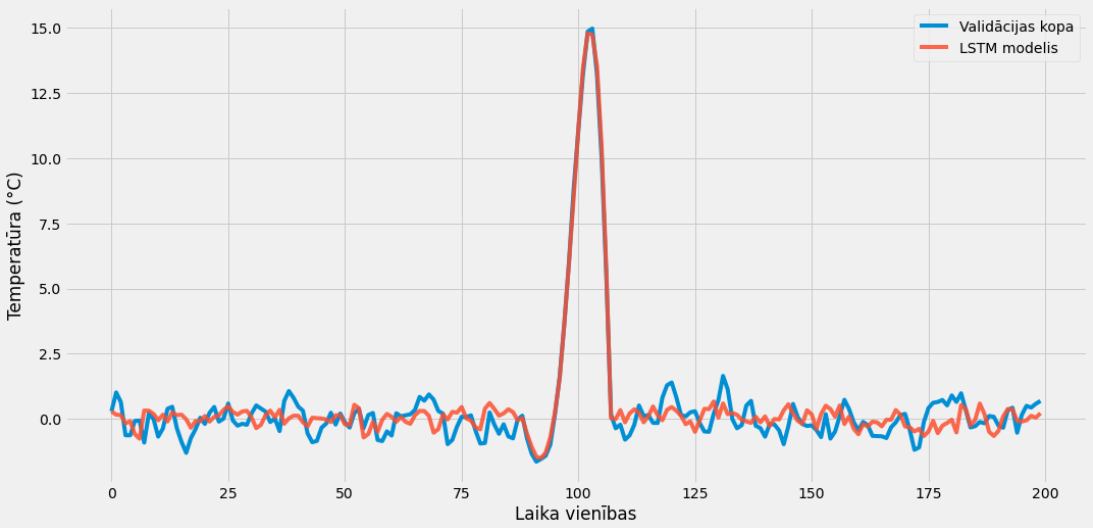
[128]:
```
[<matplotlib.lines.Line2D at 0x13f527b5820>]
```



[129]:
```
original_train, original_test = train_test_split(df_g1, 200)
```

[130]:
```
start_ts = scaler.transform(df_g1[-320:-200].values.reshape([-1,1]))
lstm_pred = np.array([])
new_ts = start_ts
for i in range(200):
    pred = model.predict(new_ts[-120:].reshape([1,1,120]))[0][0]
    new_ts = np.append(new_ts, pred)
    lstm_pred = np.append(lstm_pred, pred)
#model.predict(testX[0:1,:,:])
lstm_pred_org = scaler.inverse_transform(lstm_pred.reshape([-1,1]))
```

[131]:
```
plt.plot(original_test)
plt.plot(lstm_pred_org, alpha= .85)
plt.xlabel('Laika vienības')
plt.ylabel(u'Temperatūra (\u00B0C)')
plt.legend(['Validācijas kopa',
            'LSTM modelis',
           ])
plt.savefig('images/prognozesana/LSTM_model.png');
```



[132]:
```
pred_evaluation(original_test, lstm_pred_org, 'LSTM')
```

[132]:
```
{'Modelis': 'LSTM',
 'Modeļa tips': '(n/a)',
 'Izpildes laiks, sekundes': 0,
 'R^2': 0.95,
 'MAE': 0.46,
 'MedAE': 0.4,
 'MSE': 0.31,
 'MAPE': 580.15,
 'Komentārs': ''}
```

1.0.8 n. Modeļu prognožu novērtējumi

[140]:
```
#tst = pred_evaluation(test, one_step_pred, 'Last value')
```

[141]:
```
train, test = train_test_split(df_g1, 200)
```

[142]:
```
evalutaion_results = pd.DataFrame([
    pred_evaluation(test, one_step_pred, 'Pēdējās vērtības prognoze',␣
 ↪one_step_et, 'Naivais', ''),
    pred_evaluation(test, average_pred, 'Vidējās vērtības prognoze',␣
 ↪average_et, 'Naivais', ''),
    pred_evaluation(test, avg_k_seasonal_step_pred, 'Sezonālās vidējās vērtības␣
 ↪prognoze', avg_k_seasonal_step_et, 'Naivais', ''),
    pred_evaluation(test, sarima_pred, 'SARIMA', sarima_et, 'Stat', ''),
    pred_evaluation(test, HW_pred, 'Holt-Winters', HW_et, 'Stat', 'Aprēķina␣
 ↪laikā iekļauta krosvalidācija parametru novērtēšanai.'),
    pred_evaluation(test, stlf_pred, 'ETS + ARIMA', stlf_et, 'Stat', ''),
    pred_evaluation(test, theta_pred, 'Theta', theta_et, 'Stat', ''),
    pred_evaluation(test, lstm_pred_org, 'LSTM', lstm_et, 'DL', ''),
    pred_evaluation(test, lightboost_pred, 'LightGBM', lightboost_et, 'ML',␣
 ↪'Aprēķina laikā iekļauta krosvalidācija parametru novērtēšanai.'),
    pred_evaluation(test, xgboost_pred, 'XGBoost', xgboost_et, 'ML', 'Aprēķina␣
 ↪laikā iekļauta krosvalidācija parametru novērtēšanai.'),
    pred_evaluation(test, catboost_pred, 'CatBoost', catboost_et, 'ML',␣
 ↪'Aprēķina laikā iekļauta krosvalidācija parametru novērtēšanai..'),
    pred_evaluation(test, svm_pred, 'SVM', svm_et, 'ML', 'Aprēķina laikā␣
 ↪iekļauta krosvalidācija parametru novērtēšanai.')
]).sort_values(by=['R^2', 'MAPE', 'MSE'], ascending=[False, True, True]).
 ↪reset_index(drop=True)
```

[143]:
```
evalutaion_results.to_excel('excel_to_convert/model_eval_results.xlsx',␣
 ↪index=False)
```

[145]:
```
evalutaion_results
```

[145]:

	Modelis	Modeļa tips	Izpildes laiks, sekundes	R^2	MAE	MedAE	MSE	MAPE
0	Holt-Winters	Stat	2.12	0.96	0.37	0.31	0.22	158.75
1	LightGBM	ML	3.13	0.95	0.45	0.36	0.32	113.75
2	SARIMA	Stat	33.76	0.95	0.44	0.38	0.31	169.50
3	Sezonālās vidējās vērtības prognoze	Naivais	0.01					
4	ETS + ARIMA	Stat	1.25					
5	LSTM	DL	77.65					
6	SVM	ML	0.03					
7	CatBoost	ML	12.82					
8	XGBoost	ML	2.96					
9	Theta	Stat	0.04					
10	Vidējās vērtības prognoze	Naivais	0.00					
11	Pēdējās vērtības prognoze	Naivais	0.01					

```
3   0.95  0.44   0.36  0.31  195.67
4   0.95  0.45   0.34  0.33  213.73
5   0.95  0.46   0.40  0.31  580.15
6   0.94  0.47   0.40  0.34  122.29
7   0.94  0.47   0.43  0.34  155.01
8   0.94  0.46   0.39  0.34  224.62
9   0.81  0.78   0.64  1.16  468.61
10 -0.02  1.34   0.97  6.10  603.96
11 -0.23  1.24   0.72  7.32  493.00
```

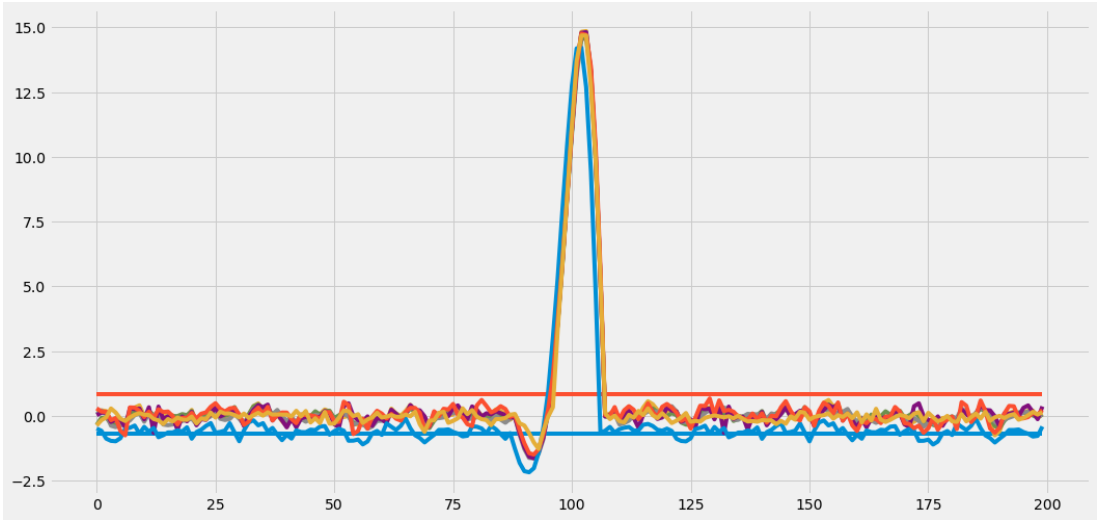
	Komentārs
0	Aprēķina laikā iekļauta krosvalidācija paramet…
1	Aprēķina laikā iekļauta krosvalidācija paramet…
2	
3	
4	
5	
6	Aprēķina laikā iekļauta krosvalidācija paramet…
7	Aprēķina laikā iekļauta krosvalidācija paramet…
8	Aprēķina laikā iekļauta krosvalidācija paramet…
9	
10	
11	

[164]:
```
# Saglabājam 3 labāko modeļu prognozes!
with open('data/HW_pred.pickle', 'wb') as f: # Pickling!
    pickle.dump(HW_pred, f)

with open('data/lstm_pred_org.pickle', 'wb') as f: # Pickling!
    pickle.dump(lstm_pred_org, f)

with open('data/lightboost_pred.pickle', 'wb') as f: # Pickling!
    pickle.dump(lightboost_pred, f)

with open('data/test.pickle', 'wb') as f: # Pickling!
    pickle.dump(test, f)

# with open("data/HW_pred.pickle", "rb") as fp:   # Unpickling!
#     HW_pred = pickle.load(fp)
```

[144]:
```
plt.plot(one_step_pred)
plt.plot(average_pred)
plt.plot(avg_k_seasonal_step_pred)
plt.plot(sarima_pred)
plt.plot(HW_pred)
plt.plot(stlf_pred)
plt.plot(range(len(theta_pred)), theta_pred)
```

```
plt.plot(lstm_pred_org)
plt.plot(lightboost_pred)
```

[144]: [<matplotlib.lines.Line2D at 0x13f54585580>]



B.6. Anomāliju noteikšanas darba grāmata

1 Anomāliju noteikšana

[201]:
```
# Pakotņu ielāde un iestatījumi
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import pickle
from scipy import stats
from tslearn.metrics import dtw

%matplotlib inline
plt.style.use('fivethirtyeight') # Grafiku stils!
plt.rcParams["figure.figsize"] = (16,8) # Grafiku izmērs
```

[4]:
```
class HoltWinters:

    """
    Holt-Winters model with the anomalies detection using Brutlag method

    # series - initial time series
    # slen - length of a season
    # alpha, beta, gamma - Holt-Winters model coefficients
    # n_preds - predictions horizon
    # scaling_factor - sets the width of the confidence interval by Brutlag␣
 ↪(usually takes values from 2 to 3)

    """


    def __init__(self, series, slen, alpha, beta, gamma, n_preds,␣
 ↪scaling_factor=1.96):
        self.series = series
        self.slen = slen
        self.alpha = alpha
        self.beta = beta
        self.gamma = gamma
        self.n_preds = n_preds
        self.scaling_factor = scaling_factor


    def initial_trend(self):
        sum = 0.0
        for i in range(self.slen):
            sum += float(self.series[i+self.slen] - self.series[i]) / self.slen
        return sum / self.slen

    def initial_seasonal_components(self):
```

```
        seasonals = {}
        season_averages = []
        n_seasons = int(len(self.series)/self.slen)
        # let's calculate season averages
        for j in range(n_seasons):
            season_averages.append(sum(self.series[self.slen*j:self.slen*j+self.
↪slen])/float(self.slen))
        # let's calculate initial values
        for i in range(self.slen):
            sum_of_vals_over_avg = 0.0
            for j in range(n_seasons):
                sum_of_vals_over_avg += self.series[self.
↪slen*j+i]-season_averages[j]
            seasonals[i] = sum_of_vals_over_avg/n_seasons
        return seasonals


    def triple_exponential_smoothing(self):
        self.result = []
        self.Smooth = []
        self.Season = []
        self.Trend = []
        self.PredictedDeviation = []
        self.UpperBond = []
        self.LowerBond = []

        seasonals = self.initial_seasonal_components()

        for i in range(len(self.series)+self.n_preds):
            if i == 0: # components initialization
                smooth = self.series[0]
                trend = self.initial_trend()
                self.result.append(self.series[0])
                self.Smooth.append(smooth)
                self.Trend.append(trend)
                self.Season.append(seasonals[i%self.slen])

                self.PredictedDeviation.append(0)

                self.UpperBond.append(self.result[0] +
                                      self.scaling_factor *
                                      self.PredictedDeviation[0])

                self.LowerBond.append(self.result[0] -
                                      self.scaling_factor *
                                      self.PredictedDeviation[0])
                continue
```

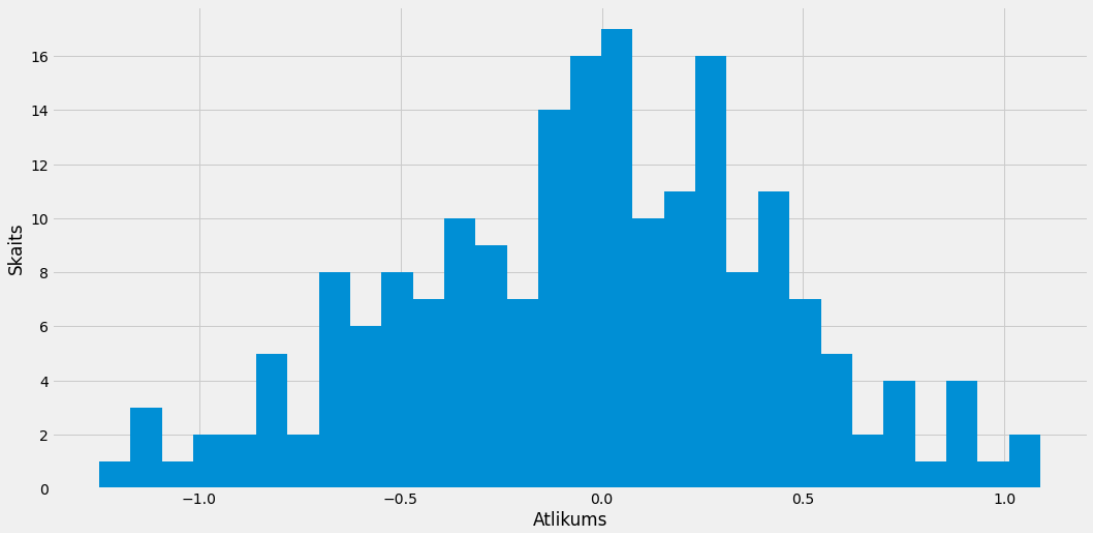
```
            if i >= len(self.series): # predicting
                m = i - len(self.series) + 1
                self.result.append((smooth + m*trend) + seasonals[i%self.slen])

                # when predicting we increase uncertainty on each step
                self.PredictedDeviation.append(self.PredictedDeviation[-1]*1.01)

            else:
                val = self.series[i]
                last_smooth, smooth = smooth, self.alpha*(val-seasonals[i%self.
↪slen]) + (1-self.alpha)*(smooth+trend)
                trend = self.beta * (smooth-last_smooth) + (1-self.beta)*trend
                seasonals[i%self.slen] = self.gamma*(val-smooth) + (1-self.
↪gamma)*seasonals[i%self.slen]
                self.result.append(smooth+trend+seasonals[i%self.slen])

                # Deviation is calculated according to Brutlag algorithm.
                self.PredictedDeviation.append(self.gamma * np.abs(self.
↪series[i] - self.result[i])
                                               + (1-self.gamma)*self.
↪PredictedDeviation[-1])

            self.UpperBond.append(self.result[-1] +
                                  self.scaling_factor *
                                  self.PredictedDeviation[-1])

            self.LowerBond.append(self.result[-1] -
                                  self.scaling_factor *
                                  self.PredictedDeviation[-1])

            self.Smooth.append(smooth)
            self.Trend.append(trend)
            self.Season.append(seasonals[i%self.slen])
```

[238]:
```
# Datu ielāde
with open('models/HW_model.pickle', 'rb') as fp: # Unpickling!
    HW_model = pickle.load(fp)

with open('data/HW_pred.pickle', 'rb') as fp: # Unpickling!
    HW_pred = pickle.load(fp)

with open('data/lstm_pred_org.pickle', 'rb') as fp: # Unpickling!
    lstm_pred_org = pickle.load(fp)

with open('data/lightboost_pred.pickle', 'rb') as fp: # Unpickling!
    lightboost_pred = pickle.load(fp)
```

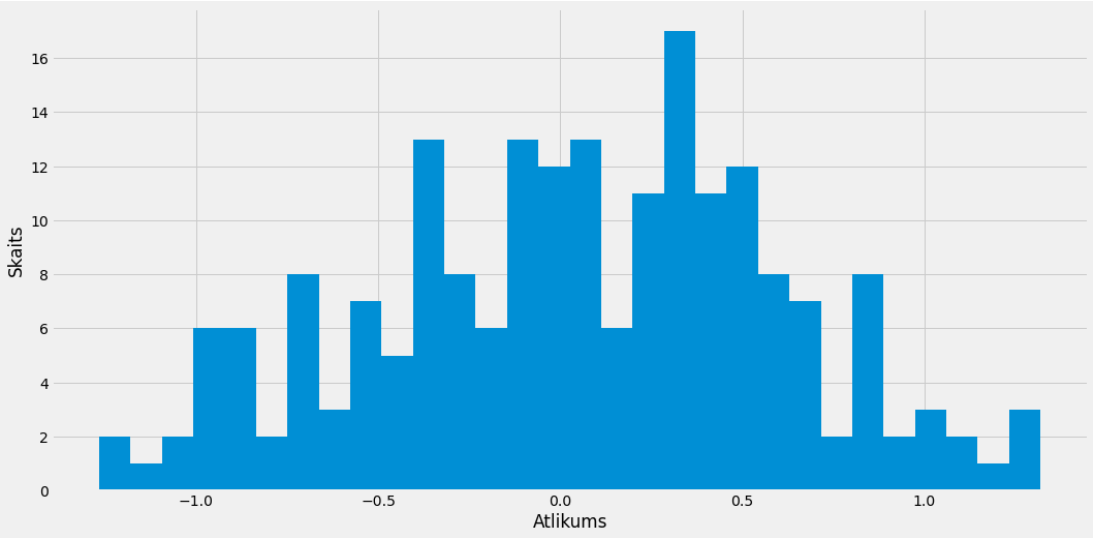
```
with open('data/test.pickle', 'rb') as fp: # Unpickling!
    test = pickle.load(fp)

with open('data/km_sp_res.pickle', 'rb') as fp: # Unpickling!
    km_sp_res = pickle.load(fp)

with open('data/cluster_centers.pickle', 'rb') as fp: # Unpickling!
    cluster_centers = pickle.load(fp)

with open('data/g1_t1_scaling.pickle', 'rb') as fp: # Unpickling!
    scaling_factors = pickle.load(fp)
```

1.0.1 1. Atlikumu un attālumu analīze

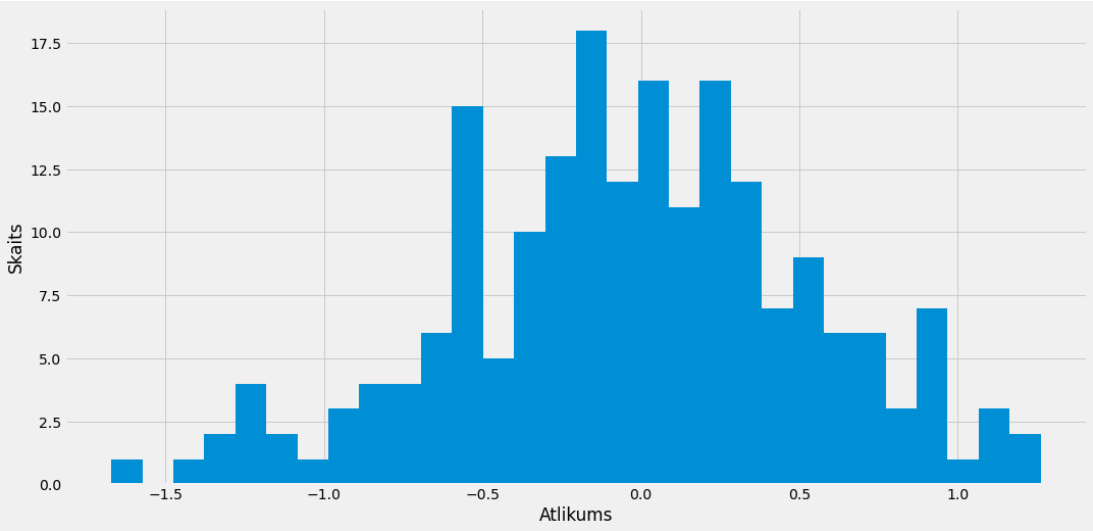
[154]:
```
HW_res = HW_pred-test
plt.hist(HW_res, bins=30)
plt.xlabel('Atlikums')
plt.ylabel('Skaits');
plt.savefig('images/anomalijas/HW_res.png');
```



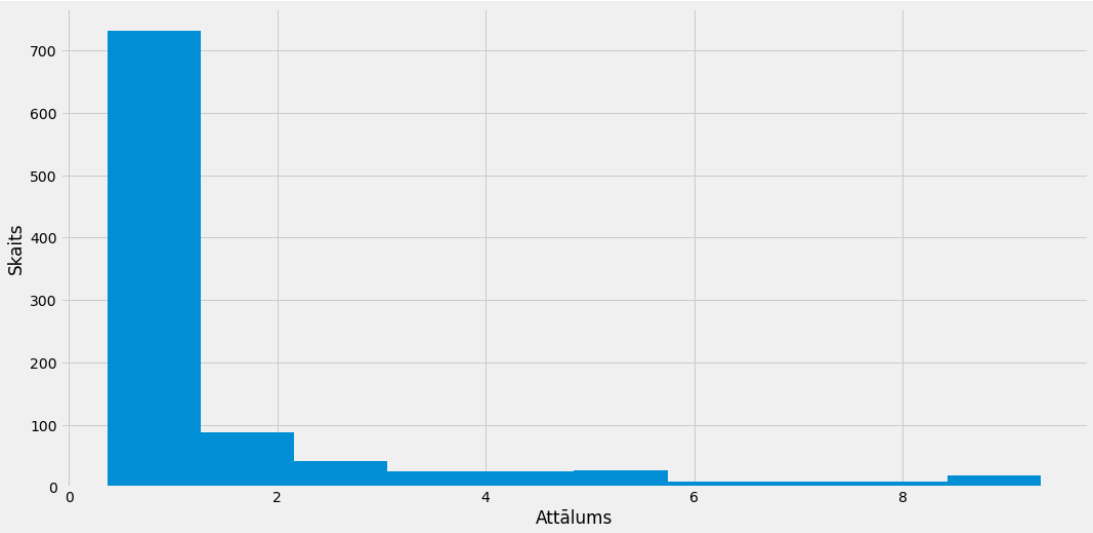
[155]:
```
lstm_res = lstm_pred_org.reshape(-1)-test
plt.hist(lstm_res, bins=30)
plt.xlabel('Atlikums')
plt.ylabel('Skaits');
plt.savefig('images/anomalijas/lstm_res.png');
```



[157]:
```
lightgbm_res = lightboost_pred-test
plt.hist(lightgbm_res, bins=30)
plt.xlabel('Atlikums')
plt.ylabel('Skaits');
plt.savefig('images/anomalijas/lightgbm_res.png');
```



[158]:
```
plt.hist(km_sp_res)
plt.xlabel('Attālums')
plt.ylabel('Skaits');
plt.savefig('images/anomalijas/km_dist.png');
```



[62]:
```
lstm_shap_p = stats.shapiro(lstm_res).pvalue
lightgbm_shap_p = stats.shapiro(lightgbm_res).pvalue
HW_shap_p = stats.shapiro(HW_res).pvalue

lstm_shap_w = stats.shapiro(lstm_res).statistic
lightgbm_shap_w = stats.shapiro(lightgbm_res).statistic
HW_shap_w = stats.shapiro(HW_res).statistic

lstm_shap_kurt = stats.kurtosis(lstm_res)
lightgbm_shap_kurt = stats.kurtosis(lightgbm_res)
HW_shap_kurt = stats.kurtosis(HW_res)

lstm_shap_skew = stats.skew(lstm_res)
lightgbm_shap_skew = stats.skew(lightgbm_res)
HW_shap_skew = stats.skew(HW_res)
```

[69]:
```
df_shap = pd.DataFrame({
    'Modelis': ['LSTM', 'LightGBM', 'Holt-Winters'],
    'W statistika': [round(lstm_shap_w,4), round(lightgbm_shap_w,4),␣
 ↪round(HW_shap_w,4)],
    'p-vērtības': [round(lstm_shap_p,2), round(lightgbm_shap_p,2),␣
 ↪round(HW_shap_p,2)],
    'Nobīde': [round(lstm_shap_skew,2), round(lightgbm_shap_skew,2),␣
 ↪round(HW_shap_skew,2)],
    'Pīķainība': [round(lstm_shap_kurt,2), round(lightgbm_shap_kurt,2),␣
 ↪round(HW_shap_kurt,2)],
    'Vidējā vērtība': [round(np.mean(lstm_res),2), round(np.
 ↪mean(lightgbm_res),2), round(np.mean(HW_res),2)],
```

```
    'Standartnovirze': [round(np.std(lstm_res),2), round(np.
 ↪std(lightgbm_res),2), round(np.std(HW_res),2)]
})
```

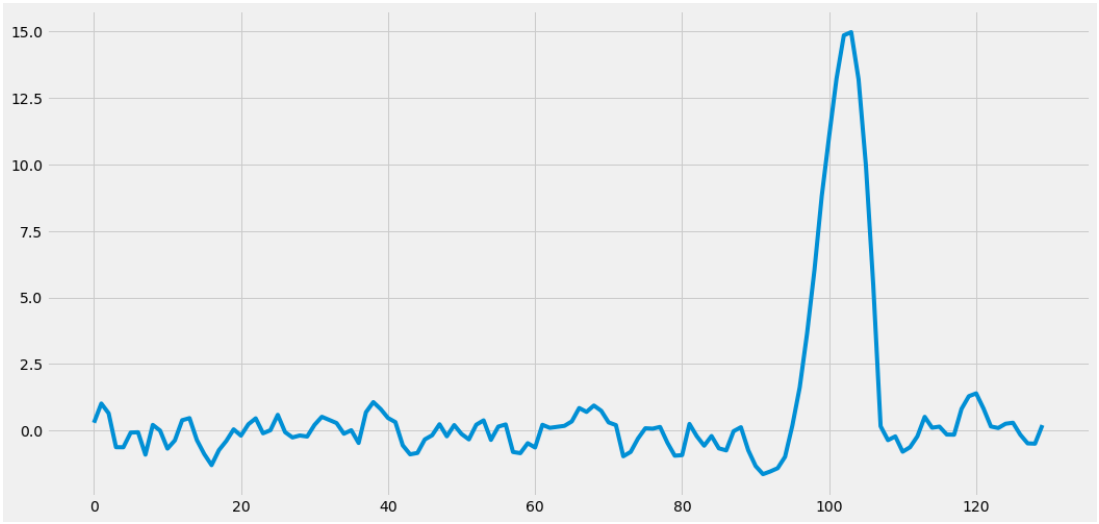
[70]:
```
df_shap.to_excel('excel_to_convert/normality_test_res.xlsx', index=False)
df_shap
```

[70]:

	Modelis	W statistika	p-vērtības	Nobīde	Pīķainība	Vidējā vērtība	Standartnovirze
0	LSTM	0.9906	0.22	-0.11	-0.52	0.05	0.56
1	LightGBM	0.9940	0.60	-0.19	-0.08	-0.03	0.56
2	Holt-Winters	0.9935	0.53	-0.16	-0.25	-0.03	0.47

[79]:
```
plt.plot(test[:130])
```

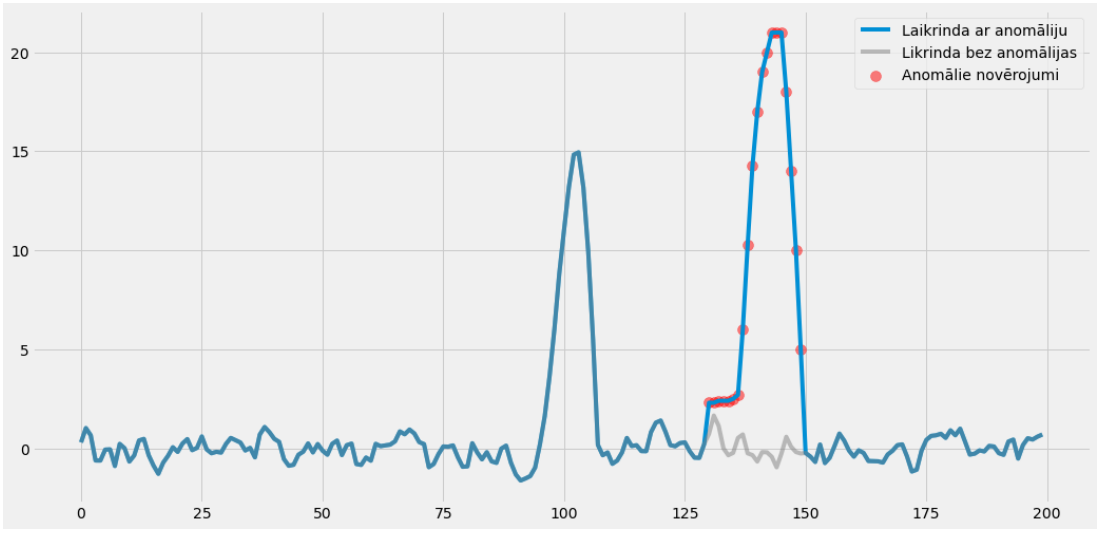
[79]:
```
[<matplotlib.lines.Line2D at 0x228cd2a0910>]
```



[143]:
```
sz = [0] * 130
anomaly = [2.3, 2.3, 2.4, 2.4, 2.4, 2.5, 2.7, 6, 10.25, 14.25, 17, 19, 20, 21,␣
 ↪21, 21, 18, 14, 10, 5]
bz = [0] * 50

test_w_anomaly = sz + anomaly + bz
```

[161]:
```
test_transformed = []

for i, j in zip(test, test_w_anomaly):
    if j == 0:
        test_transformed.append(i)
    else:
        test_transformed.append(j)

df_anomalies = pd.DataFrame({'ts': [i for i in range(len(test_w_anomaly))],
                             'anomalies': test_w_anomaly,
                             'obs': test_transformed
                            })
```

[162]:
```
#alpha = []
plt.plot(test_transformed)
plt.scatter(x=df_anomalies.loc[df_anomalies['anomalies']!= 0, 'ts'],␣
 ↪y=df_anomalies.loc[df_anomalies['anomalies']!= 0, 'anomalies'], color='red',␣
 ↪s=100, alpha=0.5)
plt.plot(test, color='grey', alpha=0.5)
#plt.vlines(x=130, ymin=0, ymax=20)
plt.legend(['Laikrinda ar anomāliju', 'Likrinda bez anomālijas', 'Anomālie␣
 ↪novērojumi'])
plt.savefig('images/anomalijas/ts_w_anomalies.png');
```



[246]:
```
HW_anomaly_id = []
for obs, up, low in zip(df_anomalies['obs'].values.tolist(), HW_model.
 ↪UpperBond[800:1000], HW_model.LowerBond[800:1000]):
    if obs <= up and obs >= low:
        HW_anomaly_id.append(False)
```

```
    else:
        HW_anomaly_id.append(True)
```

[196]:
```
lightboost_anomaly_id = []
lightboost_std = np.std(lightgbm_res)
for obs, pred in zip(df_anomalies['obs'].values.tolist(), lightboost_pred):
    if obs <= pred + 3*lightboost_std and obs >= pred - 3*lightboost_std:
        lightboost_anomaly_id.append(False)
    else:
        lightboost_anomaly_id.append(True)
```

[199]:
```
lstm_anomaly_id = []
lstm_std = np.std(lstm_res)
for obs, pred in zip(df_anomalies['obs'].values.tolist(), lstm_pred_org):
    if obs <= pred + 3*lstm_std and obs >= pred - 3*lstm_std:
        lstm_anomaly_id.append(False)
    else:
        lstm_anomaly_id.append(True)
```

[204]:
```
max(km_sp_res)
```

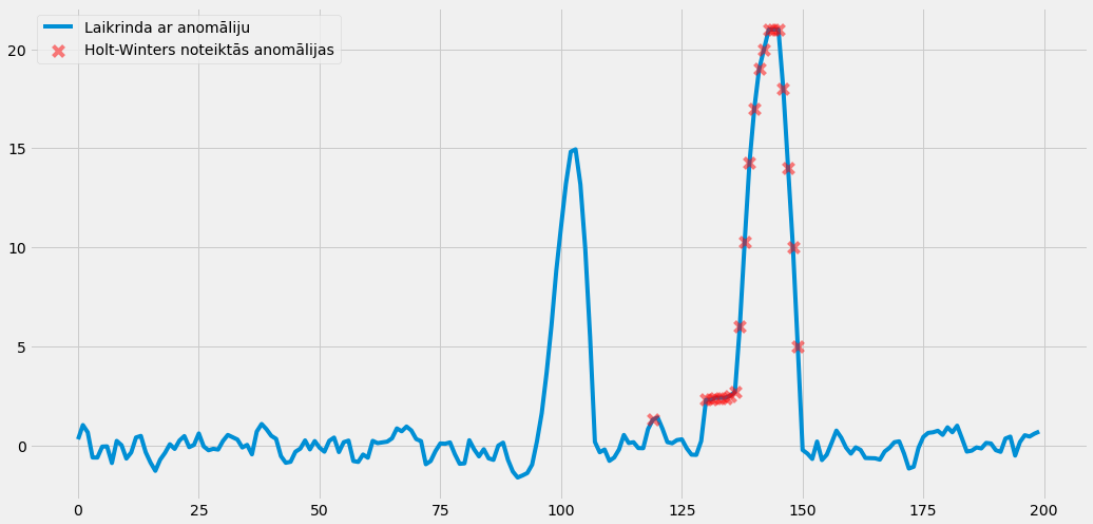
[204]:
```
9.327183688428928
```

[270]:
```
threshold = 10 # Nosaka balstoties uz max(km_sp_res)!
n_step = len(cluster_centers[0])
obs_list = (df_anomalies['obs'].values.tolist()-scaling_factors[0])/
 ↪scaling_factors[1]
sp_anomaly_id = []

for i in range(len(obs_list)):
    flag = False
    c1=0
    c2=0
    if i>=n_step-1:
        c1 = dtw(cluster_centers[0],obs_list[i-n_step+1:i+1])
        c2 = dtw(cluster_centers[1],obs_list[i-n_step+1:i+1])
        dist = min(c1, c2)
        #print(obs_list[i-n_step+1:i+1])
        if dist >= threshold:
            flag = True
            #plt.plot(obs_list[i-n_step+1:i+1])
    sp_anomaly_id.append(flag)
    #print(f'Anomalija:{flag}, attālums: {round(dist,2)}, ({round(c1,2)},␣
 ↪{round(c2,2)})')

#cluster_centers
```

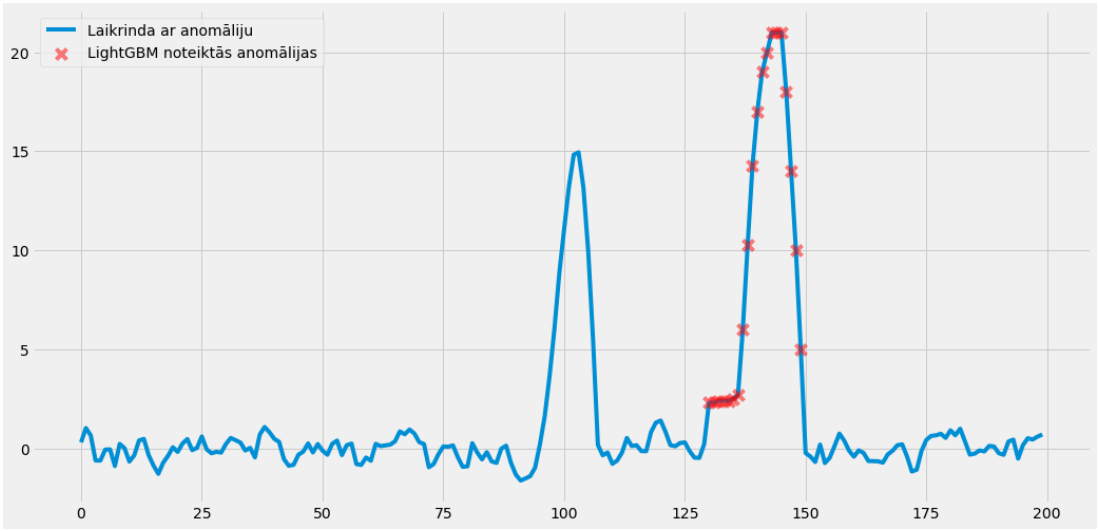
[271]:
```
#HW_anomaly_id
```

[272]:
```
df_anomalies['HW_anomaly_id'] = HW_anomaly_id
df_anomalies['lightboost_anomaly_id'] = lightboost_anomaly_id
df_anomalies['lstm_anomaly_id'] = lstm_anomaly_id
df_anomalies['sp_anomaly_id'] = sp_anomaly_id
```

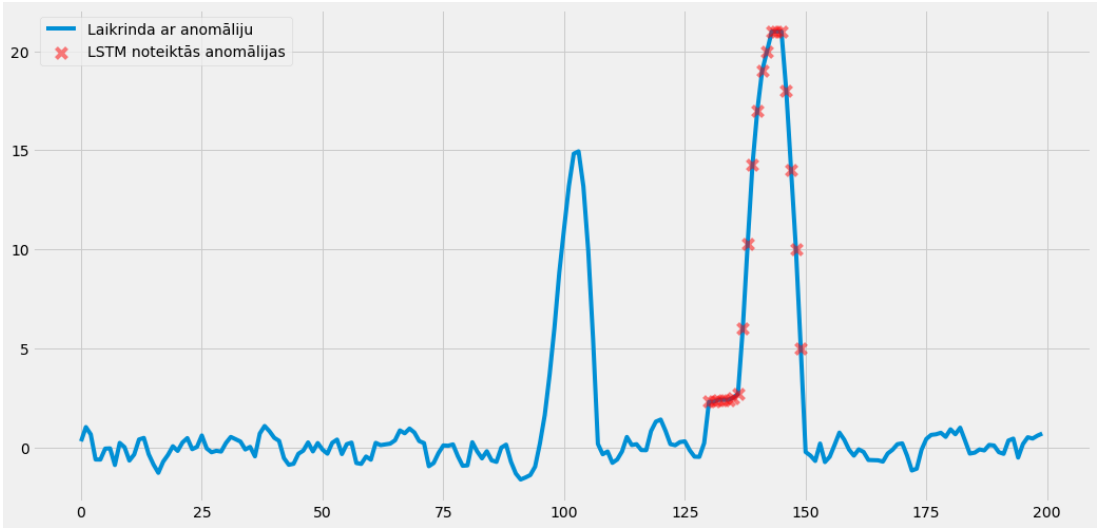
[286]:
```
plt.plot(df_anomalies['obs'], zorder=1)
#plt.scatter(x=df_anomalies['ts'], y=df_anomalies['obs'], color='black',␣
 ↪alpha=0.5, zorder=2)
plt.scatter(x=df_anomalies.loc[df_anomalies['HW_anomaly_id']==True,'ts'],
            y=df_anomalies.loc[df_anomalies['HW_anomaly_id']==True,'obs'],␣
 ↪color='red', s=100, alpha=.5, marker='x',zorder=2)
plt.legend(['Laikrinda ar anomāliju', 'Holt-Winters noteiktās anomālijas'],␣
 ↪loc='upper left')
plt.savefig('images/anomalijas/HW_detected_anomalies.png');
```



[287]:
```
plt.plot(df_anomalies['obs'], zorder=1)
plt.scatter(x=df_anomalies.
 ↪loc[df_anomalies['lightboost_anomaly_id']==True,'ts'],
            y=df_anomalies.
 ↪loc[df_anomalies['lightboost_anomaly_id']==True,'obs'], color='red', s=100,␣
 ↪alpha=.5, marker='x',zorder=2)
plt.legend(['Laikrinda ar anomāliju', 'LightGBM noteiktās anomālijas'],␣
 ↪loc='upper left')
plt.savefig('images/anomalijas/lightgbm_detected_anomalies.png');
```

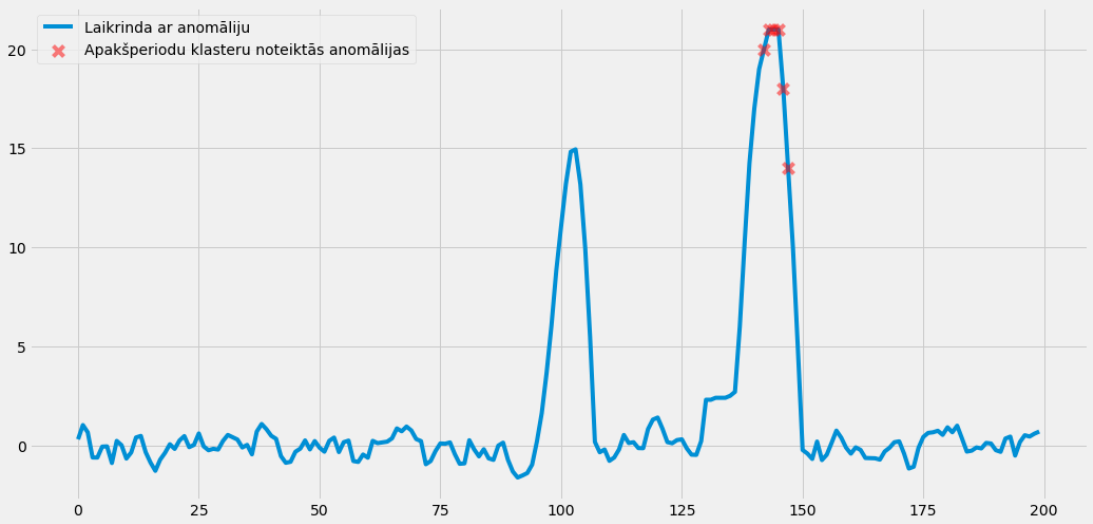


[288]:
```
plt.plot(df_anomalies['obs'], zorder=1)
plt.scatter(x=df_anomalies.loc[df_anomalies['lstm_anomaly_id']==True,'ts'],
            y=df_anomalies.loc[df_anomalies['lstm_anomaly_id']==True,'obs'],␣
 ↪color='red', s=100, alpha=.5, marker='x',zorder=2)
plt.legend(['Laikrinda ar anomāliju', 'LSTM noteiktās anomālijas'], loc='upper␣
 ↪left')
plt.savefig('images/anomalijas/LSTM_detected_anomalies.png');
```



[289]:
```
plt.plot(df_anomalies['obs'], zorder=1)
plt.scatter(x=df_anomalies.loc[df_anomalies['sp_anomaly_id']==True,'ts'],
```

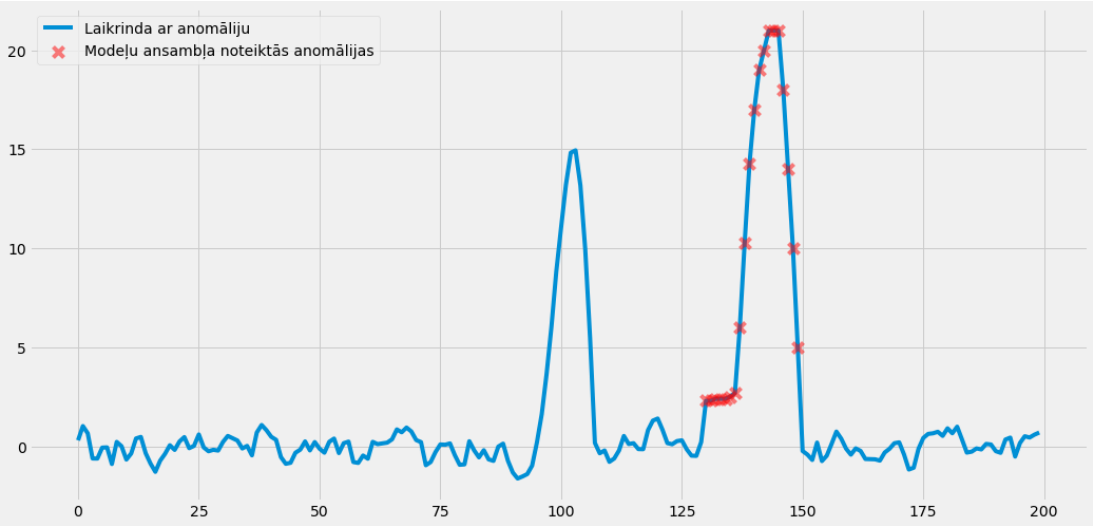
```
            y=df_anomalies.loc[df_anomalies['sp_anomaly_id']==True,'obs'],␣
 ↪color='red', s=100, alpha=.5, marker='x',zorder=2)
plt.legend(['Laikrinda ar anomāliju', 'Apakšperiodu klasteru noteiktās␣
 ↪anomālijas'], loc='upper left')
plt.savefig('images/anomalijas/sp_cluster_detected_anomalies.png');
```



[294]:
```
ensemble = []
for i,j,o,p in zip (HW_anomaly_id, lightboost_anomaly_id, lstm_anomaly_id,␣
 ↪sp_anomaly_id):
    s = (i+j+o+p)/4
    ensemble.append(round(s))
```

[296]:
```
df_anomalies['is_anomaly'] = ensemble
```

[297]:
```
plt.plot(df_anomalies['obs'], zorder=1)
plt.scatter(x=df_anomalies.loc[df_anomalies['is_anomaly']==1,'ts'],
            y=df_anomalies.loc[df_anomalies['is_anomaly']==1,'obs'],␣
 ↪color='red', s=100, alpha=.5, marker='x',zorder=2)
plt.legend(['Laikrinda ar anomāliju', 'Modeļu ansambļa noteiktās anomālijas'],␣
 ↪loc='upper left')
plt.savefig('images/anomalijas/is_anomaly.png');
```


Laikrinda ar anomāliju
Modeļu ansambļa noteiktās anomālijas
20
15
10
5
0
0
25
50
75
100
125
150
175
200

B.7. Python YMAL fails

```
name: anomaly-detection
channels:
  - conda-forge
  - defaults
dependencies:
  - anyio=2.2.0=py39haa95532_2
  - argon2-cffi=20.1.0=py39h2bbff1b_1
  - arviz=0.11.2=pyhd8ed1ab_1
  - async_generator=1.10=pyhd3eb1b0_0
  - attrs=21.2.0=pyhd3eb1b0_0
  - babel=2.9.1=pyhd3eb1b0_0
  - backcall=0.2.0=pyhd3eb1b0_0
  - blas=1.0=mkl
  - bleach=3.3.0=pyhd3eb1b0_0
  - brotlipy=0.7.0=py39h2bbff1b_1003
  - ca-certificates=2021.5.25=haa95532_1
  - certifi=2021.5.30=py39haa95532_0
  - cffi=1.14.5=py39hcd4344a_0
  - cftime=1.5.0=py39h5d4886f_0
  - chardet=4.0.0=py39haa95532_1003
  - colorama=0.4.4=pyhd3eb1b0_0
  - convertdate=2.3.2=pyhd8ed1ab_0
  - cryptography=3.4.7=py39h71e12ea_0
  - curl=7.77.0=h789b8ee_0
  - cycler=0.10.0=py39haa95532_0
  - cython=0.29.23=py39h415ef7b_0
  - decorator=5.0.7=pyhd3eb1b0_0
  - defusedxml=0.7.1=pyhd3eb1b0_0
  - entrypoints=0.3=py39haa95532_0
  - ephem=4.0.0.2=py39hb82d6ee_0
  - freetype=2.10.4=hd328e21_0
  - graphviz=2.38=hfd603c8_2
```

- hdf4=4.2.13=h712560f_2
- hdf5=1.10.6=nompi_h5268f04_1114
- hijri-converter=2.1.2=pyhd8ed1ab_0
- holidays=0.11.1=pyhd8ed1ab_0
- icc_rt=2019.0.0=h0cc432a_1
- icu=58.2=ha925a31_3
- idna=2.10=pyhd3eb1b0_0
- importlib-metadata=3.10.0=py39haa95532_0
- importlib_metadata=3.10.0=hd3eb1b0_0
- intel-openmp=2021.2.0=haa95532_616
- ipykernel=5.3.4=py39h7b7c402_0
- ipython=7.22.0=py39hd4e2768_0
- ipython_genutils=0.2.0=pyhd3eb1b0_1
- jedi=0.17.2=py39haa95532_1
- jinja2=2.11.3=pyhd3eb1b0_0
- joblib=1.0.1=pyhd8ed1ab_0
- jpeg=9b=hb83a4c4_2
- json5=0.9.5=py_0
- jsonschema=3.2.0=py_2
- jupyter-packaging=0.7.12=pyhd3eb1b0_0
- jupyter_client=6.1.12=pyhd3eb1b0_0
- jupyter_core=4.7.1=py39haa95532_0
- jupyter_server=1.4.1=py39haa95532_0
- jupyterlab=3.0.14=pyhd3eb1b0_1
- jupyterlab_pygments=0.1.2=py_0
- jupyterlab_server=2.4.0=pyhd3eb1b0_0
- kiwisolver=1.3.1=py39hd77b12b_0
- korean_lunar_calendar=0.2.1=pyh9f0ad1d_0
- krb5=1.19.1=hbae68bd_0
- libcurl=7.77.0=h789b8ee_0
- libnetcdf=4.6.1=hf59b723_3
- libpng=1.6.37=h2a8f88b_0
- libpython=2.0=py39hcbf5309_1
- libsodium=1.0.18=h62dcd97_0

- libssh2=1.9.0=h680486a_6
- libtiff=4.2.0=hd0e1b90_0
- llvmlite=0.36.0=py39ha0cd8c8_0
- lunarcalendar=0.0.9=py_0
- lz4-c=1.9.3=h2bbff1b_0
- m2w64-binutils=2.25.1=5
- m2w64-bzip2=1.0.6=6
- m2w64-crt-git=5.0.0.4636.2595836=2
- m2w64-gcc=5.3.0=6
- m2w64-gcc-ada=5.3.0=6
- m2w64-gcc-fortran=5.3.0=6
- m2w64-gcc-libgfortran=5.3.0=6
- m2w64-gcc-libs=5.3.0=7
- m2w64-gcc-libs-core=5.3.0=7
- m2w64-gcc-objc=5.3.0=6
- m2w64-gmp=6.1.0=2
- m2w64-headers-git=5.0.0.4636.c0ad18a=2
- m2w64-isl=0.16.1=2
- m2w64-libiconv=1.14=6
- m2w64-libmangle-git=5.0.0.4509.2e5a9a2=2
- m2w64-libwinpthread-git=5.0.0.4634.697f757=2
- m2w64-make=4.1.2351.a80a8b8=2
- m2w64-mpc=1.0.3=3
- m2w64-mpfr=3.1.4=4
- m2w64-pkg-config=0.29.1=2
- m2w64-toolchain=5.3.0=7
- m2w64-toolchain_win-64=2.4.0=0
- m2w64-tools-git=5.0.0.4592.90b8472=2
- m2w64-windows-default-manifest=6.4=3
- m2w64-winpthreads-git=5.0.0.4634.697f757=2
- m2w64-zlib=1.2.8=10
- markupsafe=1.1.1=py39h2bbff1b_0
- matplotlib=3.3.4=py39haa95532_0
- matplotlib-base=3.3.4=py39h49ac443_0

- mistune=0.8.4=py39h2bbff1b_1000
- mkl=2021.2.0=haa95532_296
- mkl-service=2.3.0=py39h2bbff1b_1
- mkl_fft=1.3.0=py39h277e83a_2
- mkl_random=1.2.1=py39hf11a4ad_2
- msys2-conda-epoch=20160418=1
- nbclassic=0.2.6=pyhd3eb1b0_0
- nbclient=0.5.3=pyhd3eb1b0_0
- nbconvert=6.0.7=py39haa95532_0
- nbformat=5.1.3=pyhd3eb1b0_0
- nest-asyncio=1.5.1=pyhd3eb1b0_0
- netcdf4=1.5.6=py39hb33177a_0
- notebook=6.3.0=py39haa95532_0
- numba=0.53.1=py39h69f9ab1_0
- olefile=0.46=py_0
- openssl=1.1.1k=h2bbff1b_0
- packaging=20.9=pyhd3eb1b0_0
- pandas=1.2.4=py39hd77b12b_0
- pandoc=2.12=haa95532_0
- pandocfilters=1.4.3=py39haa95532_1
- parso=0.7.0=py_0
- pickleshare=0.7.5=pyhd3eb1b0_1003
- pillow=8.2.0=py39h4fa10fc_0
- pip=21.1.2=pyhd8ed1ab_0
- prometheus_client=0.10.1=pyhd3eb1b0_0
- prompt-toolkit=3.0.17=pyh06a4308_0
- prophet=1.0.1=pyhd8ed1ab_0
- pycparser=2.20=py_2
- pygments=2.8.1=pyhd3eb1b0_0
- pyopenssl=20.0.1=pyhd3eb1b0_1
- pyparsing=2.4.7=pyhd3eb1b0_0
- pyqt=5.9.2=py39hd77b12b_6
- pyrsistent=0.17.3=py39h2bbff1b_0
- pysocks=1.7.1=py39haa95532_0

- pystan=2.19.1.1=py39h0e69a74_3
- python=3.9.4=h6244533_0
- python-dateutil=2.8.1=pyhd3eb1b0_0
- python-graphviz=0.16=pyhd3eb1b0_1
- python_abi=3.9=1_cp39
- pytz=2021.1=pyhd3eb1b0_0
- pywin32=228=py39he774522_0
- pywinpty=0.5.7=py39haa95532_0
- pyzmq=20.0.0=py39hd77b12b_1
- qt=5.9.7=vc14h73c81de_0
- requests=2.25.1=pyhd3eb1b0_0
- scikit-learn=0.24.2=py39hf11a4ad_1
- send2trash=1.5.0=pyhd3eb1b0_1
- sip=4.19.13=py39hd77b12b_0
- six=1.15.0=py39haa95532_0
- sniffio=1.2.0=py39haa95532_1
- sqlite=3.35.4=h2bbff1b_0
- terminado=0.9.4=py39haa95532_0
- testpath=0.4.4=pyhd3eb1b0_0
- threadpoolctl=2.1.0=pyh5ca1d4c_0
- tk=8.6.10=he774522_0
- tornado=6.1=py39h2bbff1b_0
- tqdm=4.61.1=pyhd8ed1ab_0
- traitlets=5.0.5=pyhd3eb1b0_0
- tslearn=0.5.1.0=py39h5d4886f_0
- tzdata=2020f=h52ac0ba_0
- urllib3=1.26.4=pyhd3eb1b0_0
- vc=14.2=h21ff451_1
- vs2015_runtime=14.27.29016=h5e58377_2
- wcwidth=0.2.5=py_0
- webencodings=0.5.1=py39haa95532_1
- wheel=0.36.2=pyhd3eb1b0_0
- win_inet_pton=1.1.0=py39haa95532_0
- wincertstore=0.2=py39h2bbff1b_0

```
- winpty=0.4.3=4
- xarray=0.18.2=pyhd8ed1ab_0
- xz=5.2.5=h62dcd97_0
- zeromq=4.3.3=ha925a31_3
- zipp=3.4.1=pyhd3eb1b0_0
- zlib=1.2.11=h62dcd97_4
- zstd=1.4.5=h04227a9_0
- pip:
  - absl-py==0.13.0
  - astunparse==1.6.3
  - cachetools==4.2.2
  - catboost==0.26
  - datetime==4.3
  - dtaidistance==2.2.5
  - et-xmlfile==1.1.0
  - flatbuffers==1.12
  - gast==0.4.0
  - google-auth==1.31.0
  - google-auth-oauthlib==0.4.4
  - google-pasta==0.2.0
  - grpcio==1.34.1
  - h5py==3.1.0
  - ipywidgets==7.6.3
  - jupyterlab-widgets==1.0.0
  - keras-nightly==2.5.0.dev2021032900
  - keras-preprocessing==1.1.2
  - lightgbm==3.2.1
  - markdown==3.3.4
  - numpy==1.19.5
  - oauthlib==3.1.1
  - openpyxl==3.0.7
  - opt-einsum==3.3.0
  - patsy==0.5.1
  - plotly==4.14.3
```

- protobuf==3.17.3
- pyasn1==0.4.8
- pyasn1-modules==0.2.8
- pymeeus==0.5.11
- requests-oauthlib==1.3.0
- retrying==1.3.3
- rsa==4.7.2
- scipy==1.6.3
- seaborn==0.11.1
- setuptools==57.0.0
- setuptools-git==1.2
- simdkalman==1.0.1
- statsmodels==0.12.2
- tensorboard==2.5.0
- tensorboard-data-server==0.6.1
- tensorboard-plugin-wit==1.8.0
- tensorflow==2.5.0
- tensorflow-estimator==2.5.0
- termcolor==1.1.0
- tsmoothie==1.0.2
- typing-extensions==3.7.4.3
- werkzeug==2.0.1
- widgetsnbextension==3.5.1
- wrapt==1.12.1
- xgboost==1.4.2
- xlrd==2.0.1
- zope-interface==5.4.0

Maģistra darbs "Reāllaika laikrindu analīze prognozēšanai un anomāliju detektēšanai" izstrādāts LU Fizikas, matemātikas un optometrijas fakultātē.

Ar savu parakstu apliecinu, ka pētījums veikts patstāvīgi, izmantoti tikai tajā norādītie informācijas avoti un iesniegtā darba elektroniskā kopija atbilst izdrukai.

Autors: Artis Alksnis

_____ _____

(paraksts) (datums)

Rekomendēju darbu aizstāvēšanai:
Vadītājs: profesors Jānis Valeinis

_____ _____

(paraksts) (datums)

Recenzents: pētniece Māra Delesa-Vēliņa

_____ _____

(paraksts) (datums)

Darbs iesniegts Matemātikas nodaļā 2021.gada ____. jūnijā.

_____

(darbu pieņēma)

Darbs aizstāvēts maģistra gala parbaudījuma komisijas sēdē

2021.gada ____. jūnijā.