

**KAUNO TECHNOLOGIJOS UNIVERSITETAS**

**MATEMATIKOS IR GAMTOS MOKSLŲ FAKULTETAS**

**Studijų modulio**

**DIDŽIŲJŲ DUOMENŲ RINKINIŲ TYRYBOS METODAI**

Kursinis darbas

Atliko: MGDVDA-9 grup. stud.

Edgaras Dvariškis

Martynas Jonaitis

Tikrino: lekt. Darius Aliulis

**KAUNAS, 2019**

**Turinys**

[1. Duomenų paruošimas 3](#_Toc27277557)

[1.1. Duomenų apžvalga 3](#_Toc27277558)

[1.2. Duomenų kokybės gerinimas 5](#_Toc27277559)

[1.3. DUOMENŲ AGREGAVIMAS IR APJUNGIMAS 6](#_Toc27277560)

[2. DUOMENŲ KLASTERIZAVIMAS 6](#_Toc27277561)

# Duomenų paruošimas

## Duomenų apžvalga

Projekte buvo naudojami duomenys apie mobilaus ryšio operatoriaus vartotojus. Duomenų faile yra 65 kintamieji, kurie kiekvienas turi 13844 įrašus. Kiekvieną unikalų vartotoją apibūdina trys kintamieji: *<user\_account\_id>*, *<year>* ir *<month>.* Šie kintamieji nurodo kurio periodo ir kurio vartotojo duomenys pateikti. Likę 62 kintamieji apibūdina periodo rodmenis. Juos galima suskirstyti taip:

|  |  |  |
| --- | --- | --- |
| Kintamieji | Apibūdinimas | Skaitinė reikšmė |
| calls\_outgoing\_count | Išeinančių skambučių kiekis | interger |
| calls\_outgoing\_duration | Išeinančių skambučių trukmė | numeric |
| calls\_outgoing\_duration\_max | Ilgiausias skambutis per periodą | numeric |
| calls\_outgoing\_inactive\_days | Dienų, be išeinančių skambučių, kiekis | interger |
| calls\_outgoing\_spendings | Išlaidos išeinantiems skambučiams | numeric |
| calls\_outgoing\_spendings\_max | Brangiausias skambutis per periodą | numeric |
| calls\_outgoing\_to\_abroad\_count | Skambučių į kitas šalis kiekis | interger |
| calls\_outgoing\_to\_abroad\_duration | Trukmė skambučių į uždienį | numeric |
| calls\_outgoing\_to\_abroad\_inactive\_days | Dienų, be išeinančių skambučių į kitas šalis, kiekis | interger |
| calls\_outgoing\_to\_abroad\_spendings | Išlaidos skambučiams į užsienį | numeric |
| calls\_outgoing\_to\_offnet\_count | Skambučių įkitus tinklus, kiekis | interger |
| calls\_outgoing\_to\_offnet\_duration | Išeinančių skambučių į kitus tinklus trukmė | interger |
| calls\_outgoing\_to\_offnet\_inactive\_days | Dienų, be išeinančių skambučių į kitus tinklus, kiekis | interger |
| calls\_outgoing\_to\_offnet\_spendings | Kaina, išleista skambučiams į kitus tinklus | interger |
| calls\_outgoing\_to\_onnet\_count | Skambučių, tinklo viduje, kiekis | interger |
| calls\_outgoing\_to\_onnet\_duration | Trukmė skambučių tinklo viduje | numeric |
| calls\_outgoing\_to\_onnet\_inactive\_days | Dienų, be išeinančių skambučių tinklo viduje, kiekis | interger |
| calls\_outgoing\_to\_onnet\_spendings | Išlaidos skambučiams tinklo viduje | numeric |
| gprs\_inactive\_days | Dienų be duomenų naudojimo kiekis | interger |
| gprs\_session\_count | Prisijungimų skaičius | interger |
| gprs\_spendings | Išlaidos už duomenų naudojimą | numeric |
| gprs\_usage | Išnaudotų kb kiekis | numeric |
| last\_100\_calls\_outgoing\_duration | Paskutinių 100 dienų išeinančių skambučių trukmė | numeric |
| last\_100\_calls\_outgoing\_to\_abroad\_duration | Paskutinių 100 dienų išeinančių skambučių š kitas šalis trukmė | numeric |
| last\_100\_calls\_outgoing\_to\_offnet\_duration | Paskutinių 100 dienų išeinančių skambučių į kitus tinklus trukmė | numeric |
| last\_100\_calls\_outgoing\_to\_onnet\_duration | Paskutinių 100 dienų išeinančių skambučių tiklo viduje trukmė | numeric |
| last\_100\_gprs\_usage | Paskutinių 100 dienų naudotas kb kiekis | numeric |
| last\_100\_reloads\_count | Paskutinių 100 dienų papildymų skaičius | interger |
| last\_100\_reloads\_sum | Paskutinių 100 dienų papildymų suma | numeric |
| last\_100\_sms\_outgoing\_count | Paskutinių 100 dienų SMS žinučių kiekis | interger |
| last\_100\_sms\_outgoing\_to\_abroad\_count | Paskutinių 100 dienų SMS žinučių kiekis į kitas šalis | interger |
| last\_100\_sms\_outgoing\_to\_offnet\_count | Paskutinių 100 dienų SMS žinučių kiekis į kitus tinklus | interger |
| last\_100\_sms\_outgoing\_to\_onnet\_count | Paskutinių 100 dienų SMS žinučių kiekis tinklo viduje | interger |
| month | Mėnesis, kada pradėjo klientas naudotis telefono kompanijos paslaugomis | interger |
| reloads\_count | Papildymų kiekis | interger |
| reloads\_inactive\_days | Dienų, be papildymų, skaičius | interger |
| reloads\_sum | Papildymų bendra suma | numeric |
| sms\_incoming\_count | Įeinančių SMS žinučių skaičius | interger |
| sms\_incoming\_from\_abroad\_count | Įeinančių SMS žinučių iš kitų tinklų skaičius | interger |
| sms\_incoming\_from\_abroad\_spendings | Išlaidos įeinančioms SMS žintumės iš kitų šalių | numeric |
| sms\_incoming\_spendings | Išlaidos įeinančioms SMS žinutėms | numeric |
| sms\_outgoing\_count | Išeinančių SMS žinučių kiekis | interger |
| sms\_outgoing\_inactive\_days | Dienų, be išeinančių SMS žinučių, kiekis | interger |
| sms\_outgoing\_spendings | Išlaidos išeinančios SMS žinutėms | numeric |
| sms\_outgoing\_spendings\_max | Brangiausia SMS žinutė | numeric |
| sms\_outgoing\_to\_abroad\_count | Išeinančių SMS žinučių į kitas šalis skaičius | interger |
| sms\_outgoing\_to\_abroad\_inactive\_days | Dienų, be išeinančių SMS žinučių į kitas šalis, skaičius | interger |
| sms\_outgoing\_to\_abroad\_spendings | Išlaidos išeinančioms SMS žinutėms į kitas šalis | numeric |
| sms\_outgoing\_to\_offnet\_count | Išeinančių SMS žinučių į kitus tinklus kiekis | interger |
| sms\_outgoing\_to\_offnet\_inactive\_days | Dienų, be išeinančių SMS žinučių į kitus tinklus, skaičius | interger |
| sms\_outgoing\_to\_offnet\_spendings | Išlaidos išeinančioms SMS žinutėms į kitus tinklus | numeric |
| sms\_outgoing\_to\_onnet\_count | Išeinančių SMS žinučių tinklo viduje kiekis | interger |
| sms\_outgoing\_to\_onnet\_inactive\_days | Dienų, be išeinančių SMS žinučių tinklo viduje, kiekis | interger |
| sms\_outgoing\_to\_onnet\_spendings | Išlaidos išeinančioms SMS žinutėms tinklo viduje | numeric |
| user\_account\_balance\_last | Kliento sąskaitos likutis periodo gale | numeric |
| user\_account\_id | Unikalus kliento indentifikavimo numeris | interger |
| user\_does\_reload | Klientas padarė bent vieną papildymą | interger |
| user\_has\_outgoing\_calls | Klientas padarė bent vieną skambutį | interger |
| user\_has\_outgoing\_sms | Klientas išsiuntė bent vieną SMS žinutę | interger |
| user\_intake | Naujas klientas | interger |
| user\_lifetime | Kliento buvimo laikas dienomis | interger |
| user\_no\_outgoing\_activity\_in\_days | Dienų, kai vartotojas neatliko jokių operacijų, kiekis | interger |
| user\_spendings | Išlaidos per periodą | numeric |
| user\_use\_gprs | Klientas panaudojo duomenis bent vieną kartą | interger |
| year | Metai, kada pradėjo klientas naudotis telefono kompanijos paslaugomis | interger |

28 *numeric* (realieji skaičiai) tipo kintamieji

34 *interger* (sveikieji skaičiai) tipo kintamieji

## **Duomenų kokybės gerinimas**

Prieš pradėdami duomenų paruošimo modeliavimui žingnius (istornių paslaugų vartojimo duomenų agregravimą pagal vartotojo ID) ir modeliavimą, turime atlikti žvalgomają analizę ir duomenų kokybės gerinimą. Duomenų kokybės gerinimas buvo vykdomas tiukrinant kiekvieną kintamąjį ir ieškant nekorektiškų reikšmių. Randant tokias rekšmes jas šaliname iš imties ir tolimesnį darbą darome su likusia imtimi. Tai darysime naudodmai python funkciją „Describe“, kuri padės matyti: minimumo ir maksimumo taškus, kvantilius, standartinį nuokrypį ir vidurkį ; naudosime histogramas, kuriuose matysis, kaip pasiskirstę duomenys. Toliau aprašysime tuos kintamuosius, kuriuos reikėjo sutvarkyti:

***user\_lifetime*** - šis kintamasis apibūdina kiek dienų klientas jau yra vartotojas. Kadangi turi kelių periodų (tiksliai trijų mėnesių) duomenis, matom, jog po kiekvieno periodo (mėnesio) šis kintamasis padidėja 31 vnt. Todėl teigiame, jog kintamasis yra matuojamas dienomis ir kiekvienas periodas (mėnesis) laikomas 31 dienos ilgio. Nusibrėžus šio kintamojo histogramą (1 pav.) matome, jog dalis reikšmių yra didesnės negu 15000. Laikantis ankstesnių prielaidų gauname, jog tokie vartotojai yra klientai jau daugiau negu 40 metų t.y. nuo ankščiau negu 1973 metai (mūsų nagrinėjame duomenys yra 2013 metų). Kadangi pirmasis komercinis mobilusis ryšys pradėtas teikti tik 1979 metais, galite teigti, jog visi šie įrašai yra klaidingi. Panaikinta 16550 įrašai.

***user\_intake*** - šis kintamasis yra binarinis ir nurodo ar klientas yra naujas. Pagal histogramą (2 pav.) matome, jog tokių yra mažuma – tiksliai 1344. Tačiau didžioji dalis, tiksliai 1322, yra nekorektiški, nes jiems kintamojo *<user\_lifetime>* reikšmė yra nelygi nuliui t.y. nors klientai ir nauji, jie jau yra vartotojai iš ankščiau. Kadangi atmetus šiuos įrašus lieka 20 naujų klientų įrašų, panaikiname ir juos, laikydami jog tolimesniam klaisifikavimui jie neturės įtakos. Bendrai pašaliname 1344 įrašus.

## Duomenų agregavimas ir sujungimas

Atlikus duomenų kokybės gerinimo žingsnį, paslaugų vartojimo duomenys suagreguojami pagal vartotojo ID, kad kiekvienam vartotojui turėtume po vieną stebėjimą. Vidutiniškai vienam vartotojui tenka po 2.1 įrašo (104009 įrašai 49693 vartotojams).

Grupavimas atliekamas tolydiems kintamiesiems skaičiuojant vidutinę reikšmę o diskretiems randant didžiausią reikšmę. Vartotojai su agreguotais įrašais sujungiami per vartotojo ID stulpelį. Pastebima, kad po šių veiksmų lieka 38806 eilutės. Tai indikuoja, kad duomenų tvarkymo etape, buvo rasta vartotojų, kurių visi įrašų kokybė buvo įvertinta kaip prasta ir jie buvo panaikinti. Sujungtos eilutės saugomos *Parquet* formatu.

Programos kodas naudotas duomenų agregavimui pateikiamas Programos kodas 1, o apjungimui Programos kodas 2.

# Duomenų klasterizavimas

## Kintamųjų atranka

## K-vidurkių algoritmas

Duomenų klasterizavimui buvo pasirinktas K-vidurkių algoritmas. Klasterių (centroidų) skaičiui nustatyti buvo taikomas iteracinis algoritmas, kurio metu visos aktualios metrikos skaičiuojamos iteruojant per skirtingą klasterių skaičių. Optimaliam klasterių skaičių nustatymui buvo panaudota SSE bei f(K) metrikos, kurių priklausomybė nuo klasterių skaičiaus pateikiama Pav. 1 bei Pav. 2.

|  |  |
| --- | --- |
| A close up of a map  Description automatically generated | A close up of a map  Description automatically generated |
| **Pav. 1.** SSE metrikos priklausomybė nuo klasterių skaičiaus | **Pav. 2. f**(K) metrikos priklausomybė nuo klasterių skaičiaus |

Gautos metrikų priklausomybės rodo, kad optimaliausias sprendinys (klasterių skaičius) yra 7. Tokia išvada daroma dėl to, kad f(K) šioje reikšmėje yra mažiausia, o SSE pagal alkūnės principą įgauna kritinę vertę (lūžio taškas).

## Klasterių interpretavimas

Klasterizavimo metu gauti 7 klasteriai, jų pasiskirstymas pateikiamas Lentelė 1. Iš pateiktų pagal dydį galime išskirti kelias klasterių grupes:

* Klasteris Nr. 1 bei Nr. 4 yra didžiausieji ir sumoje sudaro 67.45 % nuo visų klientų.
* Klasteris Nr. 2 bei Nr. 5 yra vidutinio dydžio ir sumoje sudaro 26,80 % nuo visų klientų.
* Kiti likę klasteriai yra mažo dydžio ir sudaro likusius 5.75 %.

Rezultatuose taip pat pastebima, kad klasteriuose Nr. 2, Nr. 3 bei Nr. 4 yra didžiausia išėjusių klientų dalis. Kadangi ši informacija yra aktualiausia, kiti klasteriai analizuojami nebus.

Lentelė . Klientų pasiskirstymas klasteriuose

|  |  |  |  |
| --- | --- | --- | --- |
| Klasteris | Vartotojų skaičius | Procentinė | Išėjusių klientų dalis |
| 1 | 15 599 | 40.20% | 5.03% |
| 2 | 6 136 | 15.81% | 76.42% |
| 3 | 615 | 1.58% | 44.55% |
| 4 | 10 603 | 27.32% | 25.44% |
| 5 | 3 861 | 9.95% | 2.93% |
| 6 | 1 288 | 3.32% | 5.82% |
| 7 | 704 | 1.81% | 3.27% |

* Klasteris Nr. 2:

Tai klasteris, kurio didžiausia dalis klientų nutraukia. Šį klasterį sudaro seni neaktyvūs arba mažai aktyvūs klientai. Aktyvumas vertinamas per paskutines 100 dienų. Vidutiniškai tik 28.7 % klientų atlieka skambučius, ir tik 3 % rašo žinutes. Visi neaktyvumo rodikliai yra patys didžiausi būtent šiame klasteryje. Taip pat pastebima, kad klientai beveik nesinaudojo skambučiais į užsienį. Iš to darome išvada, kad tai yra vartotojai, kurie dėl kažkokios priežasties ilgą nebesinaudoja operatoriaus paslaugomis arba naudojasi jomis labai mažai ir dėl to pasirenka nutraukti sutartį.

* Klasteris Nr. 3:

Tai labai mažas klasteris kuris išsiskiria tuo, kad jam priskirti klientai yra gana jauni ir jie vieni seniausiai iš visų kitų nepildė kliento sąskaitos. Jų aktyvumo rodikliai yra gan prasti. Pastebima, kad vidutiniškai 50 % klientų rašo žinutes ir visi naudojasi skambinimo paslauga. Taip pat pastebima, kad per paskutines 100 dienų tik 50 % proc. žmonių pildė sąskaita. Kadangi klasteris labai mažas, tikėtina, kad tai yra panašios išskirtys, kurios nepapuola į didelį klasterį.

* Klasteris Nr. 4:

Tai klasteris, kurio klientai ilgiausiai iš visų nepildė sąskaitos. Vidutinis neaktyvumas – 1,5 sav. Vidutiniškai 50 % klientų rašo žinutes ir visi naudojasi skambinimo paslauga. Nors neaktyvumo rodikliai nėra labai blogi, pastebima, kad per paskutines 100 dienų šie klientai paslaugomis naudojosi santykinai mažai. Taip pat svarbu paminėti, kad lyginant su Klasteriais Nr.2 bei Nr. 3 šio klasterio klientai žymiai daugiau naudojasi GPRS paslauga. Galime daryti išvada, kad į šį klasterį patenka smulkūs klientai, iš kurių sutartį nutraukę žmonės tiesiog pasirinko kitą operatorių.

# Programos kodas

Programos kodas . Duomenų agregavimas

|  |
| --- |
| **import** logging  **import** os  **import** glob  **import** pandas **as** pd  **from** pathlib **import** Path  **from** dotenv **import** find\_dotenv**,** load\_dotenv  **import** shutil  **import** click  **import** pyspark  LOG\_FORMAT **=** **(**  "%(asctime)s.%(msecs)03d - %(levelname)s - "  "%(filename)s - %(lineno)s - %(message)s"**)**  logging**.**basicConfig**(**  level**=**logging**.**INFO**,**  format**=**LOG\_FORMAT**,**  datefmt**=**"%Y-%m-%d,%H:%M:%S"**)**  TMP\_DIR **=** 'aggregated\_customer\_usage\_TMP'  *@click***.**command**()**  *@click***.**argument**(**'path\_usage'**,** type**=**click**.**Path**(**exists**=True))**  *@click***.**argument**(**'path\_output'**,** type**=**click**.**Path**())**  **def** main**(**path\_usage**,** path\_output**):**  logger **=** logging**.**getLogger**(**\_\_name\_\_**)**  tmp\_dir\_path **=** os**.**path**.**join**(**path\_output**,** TMP\_DIR**)**  spark **=** **(**  pyspark**.**sql**.**SparkSession  **.**builder  **.**appName**(**"Python Spark SQL aggregation example"**)**  **.**enableHiveSupport**()**  **.**getOrCreate**()**  **)**  os**.**makedirs**(**path\_output**,** exist\_ok**=True)**  usage\_df **=** spark**.**read**.**csv**(**path\_usage**,** header**=True,** inferSchema**=True,** sep**=**','**)**  usage\_df**.**createOrReplaceTempView**(**"customer\_usage"**)**  date\_columns **=** **[**"year"**,** "month"**]**  id\_columns **=** **[**"user\_account\_id"**]**  binary\_columns **=** **[**  "user\_intake"**,**  "user\_has\_outgoing\_calls"**,** "user\_has\_outgoing\_sms"**,**  "user\_use\_gprs"**,** "user\_does\_reload"  **]**  categorical\_columns **=** date\_columns **+** binary\_columns **+** id\_columns  continuous\_columns **=** **[**c **for** c **in** usage\_df**.**columns **if** c **not** **in** categorical\_columns**]**  sql\_expressions\_avg **=** **[**"AVG({0}) AS {0}"**.**format**(**c**)** **for** c **in** continuous\_columns**]**  sql\_expressions\_max **=** **[**"MAX({0}) AS {0}"**.**format**(**c**)** **for** c **in** binary\_columns**]**  sql\_expressions\_count **=** **[**"COUNT(\*) AS n\_months"**]**  sql\_expressions\_aggregation **=** sql\_expressions\_avg **+** sql\_expressions\_max **+** sql\_expressions\_count  sql\_query\_aggregate\_by\_user\_id **=** """  SELECT user\_account\_id, {}  FROM customer\_usage  GROUP BY user\_account\_id  """**.**format**(**"\n , "**.**join**(**sql\_expressions\_aggregation**))**  aggregate\_usage\_df **=** spark**.**sql**(**sql\_query\_aggregate\_by\_user\_id**)**  aggregate\_usage\_df**.**write**.**csv**(**tmp\_dir\_path**)**  **with** open**(**os**.**path**.**join**(**path\_output**,** TMP\_DIR**,** "header\_\_aggregated\_customer\_usage.txt"**),** "w"**)** **as** f**:**  f**.**write**(**","**.**join**(**aggregate\_usage\_df**.**columns**)** **+** "\n"**)**  **with** open**(**os**.**path**.**join**(**path\_output**,** TMP\_DIR**,** "header\_\_aggregated\_customer\_usage.txt"**))** **as** f**:**  columns **=** f**.**read**().**rstrip**().**split**(**","**)**  combined\_df **=** join\_csv\_files**(**tmp\_dir\_path**,** columns**)**  combined\_df**.**to\_csv**(**os**.**path**.**join**(**tmp\_dir\_path**,** ".."**,** "aggregated\_customer\_usage.csv"**),** index**=False,**  encoding**=**'utf-8-sig'**)**  **try:**  shutil**.**rmtree**(**tmp\_dir\_path**)**  **except** OSError **as** e**:**  **print(**"Error: %s - %s." **%** **(**e**.**filename**,** e**.**strerror**))**  **def** join\_csv\_files**(**file\_dir**,** coll\_names**):**  extension **=** 'csv'  all\_filenames **=** **[**i **for** i **in** glob**.**glob**(**file\_dir **+** '/\*.{}'**.**format**(**extension**))]**  combined\_csv **=** pd**.**concat**([**pd**.**read\_csv**(**f**,** sep**=**','**,** names**=**coll\_names**)** **for** f **in** all\_filenames**])**  **return** combined\_csv  **if** \_\_name\_\_ **==** '\_\_main\_\_'**:**  log\_fmt **=** '%(asctime)s - %(name)s - %(levelname)s - %(message)s'  logging**.**basicConfig**(**level**=**logging**.**INFO**,** format**=**log\_fmt**)**  # not used in this stub but often useful for finding various files  project\_dir **=** Path**(**\_\_file\_\_**).**resolve**().**parents**[**2**]**  # find .env automagically by walking up directories until it's found, then  # load up the .env entries as environment variables  load\_dotenv**(**find\_dotenv**())**  main**()** |

Programos kodas . Duomenų sujungimas

|  |
| --- |
| **import** logging  **from** pathlib **import** Path  **from** dotenv **import** find\_dotenv**,** load\_dotenv  **import** click  **import** pyspark  LOG\_FORMAT **=** **(**  "%(asctime)s.%(msecs)03d - %(levelname)s - "  "%(filename)s - %(lineno)s - %(message)s"**)**  logging**.**basicConfig**(**  level**=**logging**.**INFO**,**  format**=**LOG\_FORMAT**,**  datefmt**=**"%Y-%m-%d,%H:%M:%S"**)**  *@click***.**command**()**  *@click***.**argument**(**'path\_usage\_csv'**,** type**=**click**.**Path**(**exists**=True))**  *@click***.**argument**(**'path\_churn\_csv'**,** type**=**click**.**Path**(**exists**=True))**  *@click***.**argument**(**'path\_output'**,** type**=**click**.**Path**())**  **def** main**(**path\_usage\_csv**,** path\_churn\_csv**,** path\_output**):**  logger **=** logging**.**getLogger**(**\_\_name\_\_**)**  spark **=** **(**  pyspark**.**sql**.**SparkSession  **.**builder  **.**appName**(**"Python Spark SQL aggregation with join"**)**  **.**enableHiveSupport**()**  **.**getOrCreate**()**  **)**  usage\_df **=** spark**.**read**.**csv**(**  path\_usage\_csv**,**  header=True,  inferSchema=True)  churn\_df = spark.read.csv(  path\_churn\_csv,  header=True,  inferSchema=True)  agg\_usage\_churn\_df = (  usage\_df  .join(churn\_df, "user\_account\_id")  )  agg\_usage\_churn\_df.write.parquet(path\_output)  if \_\_name\_\_ == '\_\_main\_\_':  log\_fmt = '%(asctime)s - %(name)s - %(levelname)s - %(message)s'  logging.basicConfig(level=logging.INFO, format=log\_fmt)  # not used in this stub but often useful for finding various files  project\_dir = Path(\_\_file\_\_).resolve().parents[2]  # find .env automagically by walking up directories until it's found, then  # load up the .env entries as environment variables  load\_dotenv(find\_dotenv())  main() |

Programos kodas . Duomenų klasterizavimas

|  |
| --- |
| **import** logging  **from** pathlib **import** Path  **from** dotenv **import** find\_dotenv**,** load\_dotenv  **import** click  **import** os  **import** operator  **import** jsonlines  **import** pandas **as** pd  **import** pyspark  **from** pyspark**.**ml**.**feature **import** VectorAssembler**,** StandardScaler  **from** pyspark**.**ml**.**pipeline **import** Pipeline  **from** pyspark**.**ml**.**clustering **import** KMeans**,** GaussianMixture  **import** json  **import** pyspark  LOG\_FORMAT **=** **(**  "%(asctime)s.%(msecs)03d - %(levelname)s - "  "%(filename)s - %(lineno)s - %(message)s"**)**  logging**.**basicConfig**(**  level**=**logging**.**INFO**,**  format**=**LOG\_FORMAT**,**  datefmt**=**"%Y-%m-%d,%H:%M:%S"**)**  *@click***.**command**()**  *@click***.**argument**(**'params\_json'**,** type**=**click**.**Path**(**exists**=True))**  **def** main**(**params\_json**):**  logger **=** logging**.**getLogger**(**\_\_name\_\_**)**  **with** open**(**params\_json**)** **as** json\_file**:**  settings\_json **=** json**.**load**(**json\_file**)**  spark **=** **(**  pyspark**.**sql**.**SparkSession  **.**builder  **.**appName**(**"Python Spark K-means"**)**  **.**enableHiveSupport**()**  **.**getOrCreate**()**  **)**  path\_aggregated\_df **=** settings\_json**[**'path\_aggregated\_df'**]**  clustering\_df **=** spark**.**read**.**parquet**(**path\_aggregated\_df**)**  columns\_clustering\_features **=** columns\_clustering\_features **=** **[**  'user\_lifetime'**,**  'user\_no\_outgoing\_activity\_in\_days'**,**  'user\_account\_balance\_last'**,**  'user\_spendings'**,**  'reloads\_inactive\_days'**,**  'reloads\_count'**,**  'calls\_outgoing\_count'**,**  'calls\_outgoing\_spendings\_max'**,**  'calls\_outgoing\_inactive\_days'**,**  'calls\_outgoing\_to\_onnet\_count'**,**  'calls\_outgoing\_to\_onnet\_spendings'**,**  'calls\_outgoing\_to\_abroad\_count'**,**  'calls\_outgoing\_to\_abroad\_duration'**,**  'sms\_outgoing\_count'**,**  'sms\_outgoing\_spendings\_max'**,**  'sms\_outgoing\_inactive\_days'**,**  'sms\_outgoing\_to\_onnet\_count'**,**  'sms\_outgoing\_to\_abroad\_count'**,**  'gprs\_session\_count'**,**  'gprs\_spendings'**,**  'gprs\_inactive\_days'**,**  **]**  vector\_assembler **=** VectorAssembler**(**  inputCols**=**columns\_clustering\_features**,**  outputCol**=**"initial\_features"**)**  standard\_scaler **=** StandardScaler**(**  inputCol**=**"initial\_features"**,**  outputCol**=**"features"**,**  withStd**=True,**  withMean**=True)**  vectorized\_df **=** vector\_assembler**.**transform**(**clustering\_df**)**  model\_scaler **=** standard\_scaler**.**fit**(**vectorized\_df**)**  featurized\_clustering\_df **=** model\_scaler**.**transform**(**vectorized\_df**)**  featurization\_pipeline **=** Pipeline**(**stages**=[**vector\_assembler**,** standard\_scaler**])**  featurization\_pipeline\_model **=** featurization\_pipeline**.**fit**(**clustering\_df**)**  model\_scaler **=** featurization\_pipeline\_model**.**stages**[-**1**]**  featurized\_clustering\_df **=** featurization\_pipeline\_model**.**transform**(**clustering\_df**)**  **for** k **in** settings\_json**[**'k\_values'**]:**  kmeans **=** KMeans**(**featuresCol**=**"features"**,** k**=**k**)**  model\_kmeans **=** kmeans**.**fit**(**featurized\_clustering\_df**)**  path\_metrics\_kmeans\_sse **=** settings\_json**[**'path\_metrics\_kmeans\_sse'**]**  sse **=** model\_kmeans**.**computeCost**(**featurized\_clustering\_df**)**  metrics\_row **=** **{**"k"**:** k**,** "sse"**:** sse**}**  **with** jsonlines**.**open**(**path\_metrics\_kmeans\_sse**,** "a"**)** **as** f**:**  f**.**write**(**metrics\_row**)**  normalized\_cluster\_centers **=** model\_kmeans**.**clusterCenters**()**  scaler\_mean **=** model\_scaler**.**mean  scaler\_std **=** model\_scaler**.**std  cluster\_sizes **=** model\_kmeans**.**summary**.**clusterSizes  n\_obs **=** clustering\_df**.**count**()**  denormalized\_cluster\_centers **=** **[**  **(**cluster\_id**,)** **+** **(**size**,** 100 **\*** size **/** n\_obs**)** **+** tuple**(**center **\*** scaler\_std **+** scaler\_mean**)**  **for** cluster\_id**,** **(**size**,** center**)** **in**  enumerate**(**zip**(**cluster\_sizes**,** normalized\_cluster\_centers**))**  **]**  cluster\_centers\_pddf **=** pd**.**DataFrame**.**from\_records**(**denormalized\_cluster\_centers**)**  cluster\_centers\_pddf**.**columns **=** **(**  **[**"cluster\_id"**,** "cluster\_size"**,** "cluster\_size\_pct"**]** **+**  columns\_clustering\_features  **)**  pd**.**set\_option**(**"max\_columns"**,** 999**)**  path\_cluster\_centers **=** settings\_json**[**'path\_cluster\_centers'**]** **+** "cluster\_centers\_kmeans\_\_k\_{}.csv"**.**format**(**k**)**  cluster\_centers\_pddf**.**to\_csv**(**path\_cluster\_centers**,** index**=False)**  clustered\_kmeans\_df **=** model\_kmeans**.**transform**(**featurized\_clustering\_df**)**  path\_clustered\_df **=** settings\_json**[**'path\_cluster\_centers'**]** **+** "clustered\_kmeans\_\_k\_{}\_parquet"**.**format**(**k**)**  clustered\_kmeans\_df**.**write**.**parquet**(**path\_clustered\_df**)**  **if** \_\_name\_\_ **==** '\_\_main\_\_'**:**  log\_fmt **=** '%(asctime)s - %(name)s - %(levelname)s - %(message)s'  logging**.**basicConfig**(**level**=**logging**.**INFO**,** format**=**log\_fmt**)**  # not used in this stub but often useful for finding various files  project\_dir **=** Path**(**\_\_file\_\_**).**resolve**().**parents**[**2**]**  # find .env automagically by walking up directories until it's found, then  # load up the .env entries as environment variables  load\_dotenv**(**find\_dotenv**())**  main**()** |

Programos kodas . f(K) metrikos apskaičiavimas

|  |
| --- |
| #!/usr/bin/env python  # coding: utf-8  # # $f(K)$ metrikos naudojimo pavyzdys K-vidurkių klasterizavimo rezultatams  # In[10]:  get\_ipython**().**run\_line\_magic**(**'load\_ext'**,** 'autoreload'**)**  get\_ipython**().**run\_line\_magic**(**'autoreload'**,** '2'**)**  get\_ipython**().**run\_line\_magic**(**'matplotlib'**,** 'inline'**)**  # In[11]:  **from** IPython**.**display **import** Image  **import** plotnine **as** gg  **import** pandas **as** pd  # `pyspark.ml.clustering.KMeansModel` metodas `computeCost` apskaičiuoja stebėjimų Euklido atstumų nuo savo klasterių centrų sumą $S\_K$ (angl. \_Within Set Sum of Squared Error (WSSSE)\_):  #  # $I\_k = \sum\_{\mathbf{x}\_i \in C\_k} \| \mathbf{x}\_i - \mathbf{\overline{x}}\_k \|$  #  # $S\_K = \sum\_{k}^{K} I\_k$  #  # čia  #  # $k$ - klasterio indeksas,  #  # $C\_k$ - $k$-asis klasteris  #  # $K$ - klasterių skaičius,  #  # $N\_k$ - $k$-jam klasteriui priklausančių stebėjimų skaičius,  #  # $\mathbf{x\_i}$ - $i$-tojo stebėjimo vektorius,  #  # $\mathbf{\overline{x}}\_k$ - $k$-otojo klasterio vidurinio taško (centro) vektorius,  #  # $\|\mathbf{x}\|$ - vektoriaus Euklido norma, t.y. kvadratinė šaknis iš jo komponenčių kvadratų sumos.  # $f(K)$ yra naudojama nustatyti optimalią $K$ reikšmę ir yra aprašyta [čia](http://www.ee.columbia.edu/~dpwe/papers/PhamDN05-kmeans.pdf) ir [čia](https://datasciencelab.wordpress.com/2014/01/21/selection-of-k-in-k-means-clustering-reloaded/). Jos reikšmė $f(K)$ apskaičiuojama tokiu būdu:  # In[12]:  # Image("../data/img/fk.png")  # Optimilaus $K$ yra ties mažiausia $f(K)$ reikšme.  # Realizuojame formulės išraišką.  # In[13]:  **def** compute\_fk**(**k**,** sse**,** prev\_sse**,** dim**):**  **if** k **==** 1 **or** prev\_sse **==** 0**:**  **return** 1  weight **=** weight\_factor**(**k**,** dim**)**  **return** sse **/** **(**weight **\*** prev\_sse**)**  # calculating alpha\_k in functional style with tail recursion -- which is not optimized in Python :(  **def** weight\_factor**(**k**,** dim**):**  **if** **not** k **>** 1**:**  **raise** ValueError**(**"k must be greater than 1"**)**    **def** weigth\_factor\_accumulator**(**acc**,** k**):**  **if** k **==** 2**:**  **return** acc  **return** weigth\_factor\_accumulator**(**acc **+** **(**1 **-** acc**)** **/** 6**,** k **-** 1**)**    weight\_k2 **=** 1 **-** 3 **/** **(**4 **\*** dim**)**  **return** weigth\_factor\_accumulator**(**weight\_k2**,** k**)**  # Aprašome funkciją, kuri iš $K$ ir $S\_K$ reikšmių porų `list`'o pateikia galimas įvertinti $f(K)$ reikšmes.  # In[14]:  **def** compute\_fk\_from\_k\_sse\_pairs**(**k\_sse\_pairs**,** dimension**):**  triples **=** make\_fk\_triples**(**k\_sse\_pairs**)**  k\_fk\_pairs **=** **[**  **(**k**,** compute\_fk**(**k**,** sse**,** prev\_sse**,** dimension**))**  **for** **(**k**,** sse**,** prev\_sse**)** **in** triples**]**  **return** sorted**(**k\_fk\_pairs**,** key**=lambda** pair**:** pair**[**0**])**  **def** make\_fk\_triples**(**k\_sse\_pairs**):**  sorted\_pairs **=** sorted**(**k\_sse\_pairs**,** reverse**=True)**  candidates **=** list**(**zip**(**sorted\_pairs**,** sorted\_pairs**[**1**:]** **+** **[(**0**,** 0.0**)]))**  triples **=** **[**  **(**k**,** sse**,** prev\_sse**)**  **for** **((**k**,** sse**),** **(**prev\_k**,** prev\_sse**))** **in** candidates  **if** k **-** prev\_k **==** 1  **]**  **return** triples  # Naudojimo pavyzdys:  # In[15]:  get\_ipython**().**system**(**' cat ../data/examples\_io/metrics\_\_k\_means\_\_sse.jsonl'**)**  # Tarkime, iš disko nuskaitome tokią $K$ ir $S\_K$ reikšmių lentelę.  # In[16]:  metrics\_pddf **=** pd**.**read\_json**(**  "../data/examples\_io/metrics\_\_k\_means\_\_sse.jsonl"**,**  orient**=**"records"**,**  lines**=True)**  metrics\_pddf  # Pakeičiame stulpelių tvarką.  # In[17]:  k\_sse\_pddf **=** metrics\_pddf**[[**"k"**,** "sse"**]]**  k\_sse\_pddf  # Skačiuojant $f(K)$ metriką reikia žinoti duomenų dimensiją, t.y. klasterizavimui naudotų požymių skaičių. Tarkime, kad šiuo atveju naudojome du požymius.  # In[18]:  dimension **=** 2  k\_sse\_pairs **=** **[**tuple**(**r**)** **for** r **in** k\_sse\_pddf**.**to\_records**(**index**=False)]**  k\_sse\_pairs  # In[19]:  k\_fk\_pairs **=** compute\_fk\_from\_k\_sse\_pairs**(**k\_sse\_pairs**,** dimension**)**  k\_fk\_pairs  # In[20]:  k\_fk\_pddf **=** pd**.**DataFrame**.**from\_records**(**k\_fk\_pairs**,** columns**=[**"k"**,** "fk"**])**  k\_fk\_pddf  # In[21]:  plot\_k\_sse **=** **(**  gg**.**ggplot**(**gg**.**aes**(**x**=**"k"**,** y**=**"sse"**),** data**=**k\_sse\_pddf**)** **+**  gg**.**geom\_line**()** **+**  gg**.**xlab**(**"K"**)** **+**  gg**.**ylab**(**"SSE"**)** **+**  gg**.**ggtitle**(**"SSE pagal klasterių skaičių K"**)** **+**  gg**.**theme\_bw**()**  **)**  plot\_k\_fk **=** **(**  gg**.**ggplot**(**gg**.**aes**(**x**=**"k"**,** y**=**"fk"**),** data**=**k\_fk\_pddf**)** **+**  gg**.**geom\_line**()** **+**  gg**.**xlab**(**"K"**)** **+**  gg**.**ylab**(**"f(K)"**)** **+**  gg**.**ggtitle**(**"f(K) pagal klasterių skaičių K"**)** **+**  gg**.**theme\_bw**()**  **)**  # In[22]:  **print(**plot\_k\_sse**)**  **print(**plot\_k\_fk**)** |