Pradžia

VILNIAUS GEDIMINO TECHNIKOS UNIVERSITETAS

FUNDAMENTINIŲ MOKSLŲ FAKULTETAS

INFORMACINIŲ SISTEMŲ KATEDRA

**DIDŽIŲJŲ DUOMENŲ APDOROJIMAS**

**Kursinis darbas**

Darbą atliko: DGTfm-21 stud.

Laimonas Janutėnas

Priėmė: Doc. Dr. Andrej Bugajev

Vilnius 2022

# 

# Įvadas

Šiame kursiniame darbe bus analizuojamos didžiųjų duomenų apdorojimo programinės sistemos bei išbandomos šios sistemos praktiškai per laboratorinius darbus. Didžiųjų duomenų sistemos veikia skirtingai nuo įprastų duomenų bazių. Didieji duomenys yra per dideli ir per sudėtingi, kad juos būtų galima apdoroti įprastomis sistemomis. Šios sistemos yra pritaikytos apdoroti didelius kiekius duomenų greičiau, palyginus su įprastomis duomenų bazių sistemomis. Laboratoriniuose darbuose bus nagrinėjami duomenys, aprašantys siuntų pristatymo bei logistikos įmonės veiklą. Su šiais duomenimis bei naudojantis „Hadoop“ ir „Spark“ sistemomis, priklausomai nuo laboratorinio darbo užduoties, pagal „MapReduce“ principą, bus atrenkami reikalingi duomenys ir suskaičiuojamas reikiamas rezultatas.

„Hadoop“ yra labiausiai paplitusi „MapReduce“ realizacija, kuri leidžia lengvai atlikti paskirstytus skaičiavimus, paskirstyti duomenis ir užtikrinti duomenų atsparumą, programuotojui net nežinant, kaip tie paskirstyti procesai vyksta viduje. O „Spark“ yra naujesnė sistema, atliekanti skaičiavimus dar greičiau, nes visus duomenis saugo operatyviojoje atmintyje, taip leidžiant juos pasiekti ir apdoroti efektyviau. Taip pat, „Spark“ suteikia patogesnę programavimo sąsaja bei papildomų paketų, kurie dažnai yra naudingi duomenų gavybos procese. Šios sistemos leidžia atlikti skaičiavimus, kurie anksčiau buvo per sudėtingi ir per ilgai užtrunkantys.

**Darbo tikslas** – susipažinti su didžiųjų duomenų apdorojimo sistemomis ir mokėti jomis naudotis

**Darbo uždaviniai:**

1. Išanalizuoti „MapReduce“ programavimo modelį ir išspręsti pirmą laboratorinį darbą naudojantis „Hadoop“ programine sistema.
2. Išanalizuoti „DAG“ programavimo modelį ir išspręsti antrą laboratorinį darbą naudojantis „Spark“ programine sistema.
3. Naudojantis „Spark“ programiniais papildiniais, trečiame laboratoriniame darbe atlikti tiesinę regresijos analizę.

# Pirma užduotis

# Pirmojo laboratorinio užduotis

Laboratoriniame darbe reikia išnagrinėti duomenis, aprašančius siuntų pristatymo bei logistikos įmonės veiklą. Duotajame faile pateikti dalinai struktūrizuoti duomenys apie siuntų išvežiotojų sustojimus. Naudojamas failas „duom\_full.txt” ir sutrumpintas duomenų failas „duom\_cut.txt” (paveikslas 1).

A screenshot of a computer

Description automatically generated

**Pav. 1** duomenų failo fragmentas

Laboratoriniame darbe atliksiu **2 užduotį,** joje reikia iš duomenų rinkinio rasti maršrutus, kurie aplanko daugiau nei vieną geografinę zoną ir kai tai daroma tą pačią dieną.

Kadangi naudojame „Hadoop“ sistemą, mes naudosime „MapReduce“ programavimo modelį. „MapReduce“ procesas yra pavaizduotas paveiksle 2.

Diagram

Description automatically generated

**pav. 2** „MapReduce“ programavimo modelis

Visų pirmiausia visi duomenys yra saugomi HDFS failų sistemoje, kuri suskirsto duomenis į atskirus blokus. Dėl šios priežasties, galima lygiagrečiai atlikti tam tikrus skaičiavimus, duomenų atrinkimus. Pirmiausia šie paskirstyti duomenys turi būti apdoroti naudojantis „Map“. „Map“ atlieka veiksmus kiekviename mazge atskirai ir nepriklausomai nuo kitų mazgų. Toks atskyrimas leidžia didelius duomenų kiekius apdoroti greičiau, nes kiekvienas mazgas gali dirbti lygiagrečiai su savo duomenimis. Su „Map“, iš to duomenų bloko, atsirenkame tik mums reikalingus duomenis ir išvedame, pagal poreikį, reikalingas raktų reikšmių poras, tokių kokių reikės apdorojant su „Reduce“. Po „Map“ veiksmų vyksta „Shuffle“ ir „Sort“. „Shuffle“ perduoda duomenis į „Reduce“, o „Sort“ padaro, kad reikšmės būtų surikiuojamos pagal raktą. Kadangi jau iš visur yra surinkti duomenys, mums reikia iš jų išgauti tam tikrą informacija. Tai kitame veiksme, su „Reduce“, mes jau atliekame tuos skaičiavimus, pagal kuriuos, iš duomenų gauname reikiamą rezultatą.

Taigi, pirmiausia reikia sukurti „Map“ programinį kodą, kuris iš duomenų mums atrinktų tik reikiamus parametrus, kuriuos analizuosime. Šiame duomenų rinkinyje yra pateikiama daug informacijos apie pristatymus, bet mums reikia tiktai maršruto, geografines zonos ir datos. Kadangi reikia rasti maršrutus, kurie aplanko daugiau nei vieną geografine zona tą pačią dieną, reiškia raktas yra pagal du parametrus: maršrutą ir dieną, tai reiškia reikia išvesti tokį rakta kuris jungia šiuos abu parametrus.

Pseudo kodas:

data = input()

for element in data:

parameters = element.split()

for param in parameters:

key, value = param.split()

if key == 'marsrutas':

marsrutas = value

elif key == 'geografine zona':

geografine\_zona = value

elif key == 'sustojimo data':

sustojimo\_data = value

if marsrutas and geografine\_zona and sustojimo\_data are not Empty:

key = marsrutas + '\_' + sustojimo\_data

emit(key, geografine\_zona)

Šis pseudo kodas nuskaito duomenų failą, išskiria atskirus elementus, kuriuose yra parametrai apie tam tikrą siuntą ar maršrutą ir t.t. Tada einama per kiekvieną elemento parametrą ir tikriname ar tas elementas turi mums reikiamus duomenis (maršrutą, sustojimo datą ir geografinę zoną). Jei bent vieno iš trijų parametrų nėra elemente, mes negalime išvesti šitų duomenų ir iteruojame toliau. Jei yra visi duomenys, išvedamas maršrutas ir sustojimo data kaip raktas (sujungiant šiuos parametrus per brūkšnelį), o reikšmė būtų geografinė zona. Surinkti duomenys atrodo taip:

A screenshot of a computer

Description automatically generated with low confidence

**Pav. 3** „Map“ operacijos surinkti duomenys

Reiktų paminėti, kad po „Map“ funkcijos įvyksta rikiavimas pagal raktą, todėl išvesti duomenys jau būna surikiuoti. Toliau reikia sukurti „Reduce“ programinį kodą. Kadangi jau turime mums reikiamus duomenis, naudodamiesi „Reduce“ kodu mes iš duomenų išgauname mums reikalingą informaciją. Šiuo atveju mes norime sužinoti ar su mūsų turimu raktu (maršrutas ir sustojimo data) buvo aplankyta skirtingų geografinių zonų, t. y. ar yra skirtingų reikšmių su tokiu raktu.

Pseudo kodas:

current\_key = None

current\_set = set()

lines = input()

for line in lines:

marsrutas\_data, zona = line.split()

if current\_key == marsrutas\_data:

current\_set.add(zona)

else:

if current\_key != None:

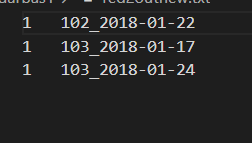
if len(current\_set) > 1:

emit(1, current\_key)

current\_set = set(zona)

current\_key = marsrutas\_data

„Reduce“ kodas nuskaito raktą ir reikšmę. Kadangi žinome, kad duomenys yra surikiuoti, išsaugome dabartinį raktą ir tikriname ar jis nepasikeitė. Jei raktas nepasikeitė nuskaitytą reikšmę dedame į rinkinį, kuriame yra saugomos unikalios geografinių zonų reikšmės. Jei raktas pasikeitė, tikriname ar rinkinyje yra daugiau nei viena reikšmė (pagal sąlygą reikia rasti maršrutus, kurie aplankė kelias geografines zonas tą pačią dieną) ir išvedame tą maršrutą su data. Ir taip einama per visus duomenis. Gautas rezultatas yra toks:



**Pav. 4** "Reduce" rezultato išvestis

Dar papildomai buvo grąžintas vienetas, nes „Reduce“ turi išvesti „raktas-reikšmė“ tipo duomenis, bet šiuo atveju jis nereikalingas. Taigi, naudojantis „MapReduce“ buvo gauti maršrutai aplankę kelias geografines zonas tą pačią dieną. Su pilnais duomenimis paleistas kodas buvo išvestas „Hortonworks HDP“ sistemoje:

Graphical user interface, table

Description automatically generated

**Pav. 5** „MapReduce“ rezultatas „Hortonworks HDP“ sistemoje

Taip pat, šiame uždavinyje būtų galima pritaikyti „Combiner“. „Combiner“ leidžia vykdyti kodą iškart po „Map“, tame pačiame mazge, su tais duomenimis, kurios išvedė „Map“. Pavyzdžiui, šio uždavinio atveju, galėtume po „Map“ pritaikyti „Combiner“, tam kad paliktume tik unikalius rakto ir reikšmės variantus. Kadangi mums reikia sužinoti maršrutus, tą dieną, kurie aplankė kelias geografines zonas, tai mes su „Combiner“ galėtume pašalinti visas pasikartojančias reikšmes ir tokiu būdu sumažinti perduodamą kiekį duomenų į „Reduce“. Galimai pagreitintume visą skaičiavimo procesą, nes mazguose skaičiavimai būtų atliekami su mažesniu skaičiumi duomenų bei nereiktu perduoti daug duomenų tarp mazgų. Deja, „Combiner“ netinka visoms problemoms spręsti, todėl ne visada galima jį pritaikyti, nors šiuo atveju tinka.

# „Hadoop“ aplinkos pasiruošimas ir komandos

Prieš pradedant darbą reikia įsirašyti „Hortonworks HDP“ programinę įrangą, kurią reikia pasileisti per „VirtualBox“ platformą. „Hortonworks HDP“ galima atsisiųsti per šią nuorodą - [Hortonworks Data Platform (HDP) on Sandbox (cloudera.com)](https://www.cloudera.com/downloads/hortonworks-sandbox/hdp.html)

Graphical user interface

Description automatically generated

**Pav. 6** "Hortonworks HDP" puslapio vaizdas

Atsisiųsta programinė įranga ateina su visa operacine sistema ir yra sukonfigūruota veikti iš karto per „VirtualBox“, kuri sukuria virtualią mašiną.

Graphical user interface, text, application

Description automatically generated

**Pav. 7** "VirtualBox" pradinis vaizdas

Importavus virtualios mašinos failą, programa pasileidžia. „Hortonworks HPD“ tada galima pasiekti per naršyklę su nuoroda: „<https://localhost:8080/>“.

Text

Description automatically generated

**Pav. 8** "Hortonworks HDP" vaizdas paleidus per "VirtualBox"

Atsidarius nuorodą reikia prisijungti prie sistemos su naudotojo vardu ir slaptažodžiu “maria\_dev”.

Graphical user interface, application

Description automatically generated

**Pav. 9** prisijungimas prie "Hortonworks HDP"

Prisijungus atsidaro „Hortonworks HDP“ valdymo langas, kuriame reikia įjungti norimus servisus.

Graphical user interface

Description automatically generated

**Pav. 10** "Hortonworks HDP" pagrindinis langas

Taip pat reikia įsirašyti „WinScp“ programinę įrangą, kuri per FTP protokolą, leis perkelti norimus failus ir duomenis į „Hortonworks HDP“. Tai galima padaryti prisijungus adresu „localhost:2222“ ir su prisijungimu bei slaptažodžiu “maria\_dev”.

Graphical user interface

Description automatically generated

**Pav. 11** prisijungimas su "WinScp"

Atsidarius „WinScp“, į sistemą reikia perkelti visus duomenis ir programinius failus į „maria\_dev“ direktoriją.

Graphical user interface, text

Description automatically generated

**Pav. 12** failų perkėlimas per "WinScp"

Kai failai jau sukelti, reikia atsidaryti „PuTTY“, kuris leidžia per SSH protokolą pasiekti sistemą ir naudotis joje esančia „Linux“ operacine sistema.

Graphical user interface, text

Description automatically generated

Pav. 13 "PuTTY" atidarymas per "WinScp"

Atsidarius „PuTTY“ terminalo langą taip pat reikia suvesti “maria\_dev” slaptažodį.

Text

Description automatically generated

**Pav. 14** prisijungimas prie sistemos per terminalą

Prisijungus prie sistemos, pirmiausia reikia nueiti į direktoriją kur yra sukelti įkopijuoti failai, tai galima padaryti su „cd” komanda ir nurodant direktorijos kelią, pvz.: „cd namudarbas”. Tada su komanda „ls” galima pamatyti visus direktorijoje esančius failus. Tam, kad failus būtų galima paleisti aplinkoje, reikia jiems suteikti paleidimo teises, tai galima padaryti naudojantis „chmod +x” ir nurodant failo pavadinimą. Tada dar kartą paleidus “ls” galima pamatyti, kad tie failai nuspalvinami žaliai, tai reiškia juos galima paleisti.

Text

Description automatically generated

**Pav. 15** esamų failų vaizdas terminale

Tada duomenų failus reikia perkelti į HDFS failų sistemą. Pirmiausia yra sukuriama direktorija kur tie failai bus saugomi su komanda „mkdir”, o kad tai būtų įvykdoma HDFS aplinkoje, reikia tai nurodyti su „hadoop fs” komanda pvz.: „hadoop fs -mkdir namudarbas”. O duomenys yra perkeliami su „hadoop fs -put duomenys direktorija/failas”





**Pav. 16** HDFS direktorijų kūrimo komandos

Prieš paleidžiant programinį „MapReduce“ kodą reikia įsitikinti ar jame nėra atidarinėjami failai, nes tai išmes klaidos pranešimą „Hadoop“ sistemoje.

Graphical user interface, text

Description automatically generated

**Pav. 17** Galimos klaidos „MapReduce“ programiniame kode

Taip pat reikia įsitikinti, kad programinis kodas veikia teisingai ir neturi jokio pašalinio poveikio, kaip tuščios eilutes ar tušti tekstiniai žodžiai.

A screenshot of a computer

Description automatically generated

Graphical user interface, text

Description automatically generated

**Pav. 18** duomenų patikrinimas „MapReduce“ programiniame kode

Paleisti programinį kodą „Hadoop“ sistemoje galima naudojantis „mapred streaming”,

nurodant įvesties failą, išvesties vietą, „mapper“ programinio kodo failą, „reducer“ programinio kodo failą bei juos paduodant su „file“ komanda. Pilnos komandos pavyzdys:

“

mapred streaming \

-input namudarbas1/duom\_full.txt \

-output namudarbas1/out40 \

-mapper map2.py \

-reducer red2new.py \

-file map2.py \

-file red2new.py

”

Text

Description automatically generated

**Pav. 19** „MapReduce“ komanda

Jei kažkur yra klaida, „Hadoop“ išmes klaidos pranešimą „waitOutputThreads()” ir neišmes konkrečios klaidos kur yra problema, todėl programinį kodą reiktų gerai ištestuoti ne „Hadoop“ aplinkoje.

Text

Description automatically generated

**Pav. 20** „Hadoop“ klaidos pranešimas

Jei komanda sėkmingai suveikė, per „Hortonworks HDP“ sistemą reikia susirasti išvesties failą. Failų sistemą galima pasiekti per „Files View” nuorodą.

Graphical user interface, application, chat or text message

Description automatically generated

**Pav. 21** "File View" nuoroda

Atsidarius failų sistemos langui reikia nueiti į „user/maria\_dev” direktoriją. Joje bus sukurta nurodyta išvesties direktorija.

Graphical user interface, table

Description automatically generated

**Pav. 22** "Hortonworks HDP" failų sistema

Direktorijos viduje bus failas su išvesties duomenimis, kurį galima peržiūrėti.

Graphical user interface, table

Description automatically generated

**Pav. 23** išvestas programos rezultatas

# Antra užduotis

# Antro laboratorinio užduotis

Laboratoriniame darbe reikia išnagrinėti duomenis, aprašančius siuntų pristatymo bei logistikos įmonės veiklą. Duotajame faile pateikti dalinai struktūrizuoti duomenys apie siuntų išvežiotojų sustojimus. Failas “duom\_full.txt” ir sutrumpintas duomenų failas “duom\_cut.txt” (paveikslas 24).

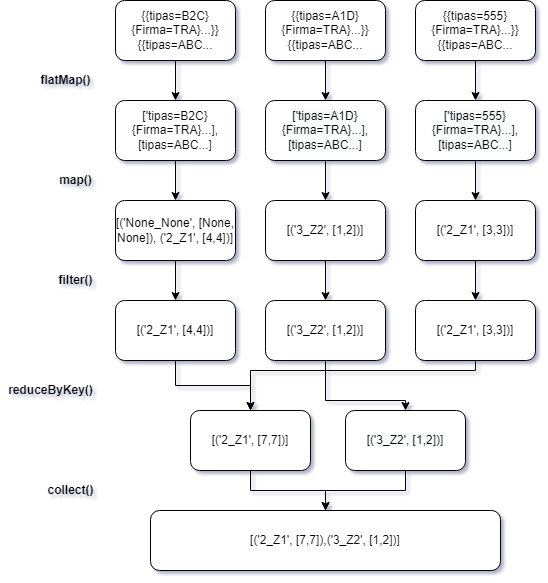
A screenshot of a computer

Description automatically generated

**Pav. 24** duomenų failo ištrauka

Laboratoriniame darbe atliksiu **3 užduotį,** joje reikia iš duomenų rinkinio rasti siuntų skaičių bei aptarnautų klientų skaičių, kiekvienoje geografinėje zonoje, kiekvieną savaitės dieną.

Šiame laboratoriniame darbe jau naudojama ne „Hadoop“, o „Spark“. „Spark“ nebenaudoja „MapReduce“, bet naudoja savo paskirstytą vykdymo variklį, kur naudojamas DAG bei suteikia patogesnę vartotojo sąsają. DAG, tai grafas kuris yra sudarytas iš viršūnių ir briaunų, kur viršūnės atitinka specialius duomenų vienetus RDD, o briaunos parodo atliekamas operacijas. DAG yra „MapReduce“ programavimo modelio apibendrinimas. „Hadoop“ turi trūkumą, kad įrašo tarpinius rezultatus į kietąjį diską, kai dažnai nėra tam poreikio, nes norime atlikti papildomas operacijas. DAG optimizuoja ir vykdo veiksmus vienas po kito, pasirūpina tuo kur duomenys yra saugomi, juos surenka kiekvienam „MapReduce“, atsižvelgia kokios yra operacijos ir pagal tai nusprendžia kaip efektyviausiai vykdyti kodą. DAG naudoja „lazy evaluation“ – grafas saugo apdorojimo priklausomybes, ir kai reikia, tik tada atlieka skaičiavimus. Nustatoma grandinė veiksmų, kurią reikės atlikti, nepriklausomai nuo to kiek programuotojas yra prirašęs programinio kodo. Šio darbo DAG vizualizacija pavaizduota paveiksle 25.



**Pav. 25** Laboratorinio darbo DAG vizualizacija

Pirmiausia reikia sukurti „Map“ programinį kodą, kuris iš duomenų mums atrinktu tik reikiamus parametrus, kuriuos analizuosime. Šiame duomenų rinkinyje yra pateikiama daug informacijos apie pristatymus, bet mums reikia tiktai siuntų skaičiaus, sustojimo klientų skaičiaus, geografine zoną ir sustojimo datą. Kadangi reikia rasti geografines zonas skirtingomis savaites dienomis, reiškia raktas yra pagal du parametrus: geografines zoną ir savaitės dieną, tai reiškia reikia išvesti tokį rakta kuris jungia šiuos abu parametrus.

Pseudo kodas:

lines = sc.textFile()

def FM(line):

substring = line.split()

return substring

A=lines.flatMap(FM)

def MapF(stopas):

parstrings = stopas.split()

for parstring in parstrings:

(vardas, reiksme) = parstring.split()

if(reiksme != '' and vardas == 'sustojimo savaites diena'):

diena = reiksme

if(reiksme != '' and vardas == 'siuntu skaicius'):

siuntos = reiksme

if(reiksme != '' and vardas == 'geografine zona'):

geografine\_zona = reiksme

if(reiksme != '' and vardas == 'sustojimo klientu skaicius'):

klientu\_skaicius = reiksme

key = str(diena) + '\_' + str(geografine\_zona)

return(key, [siuntos, klientu\_skaicius])

B = A.map(MapF)

C = B.filter(value != None)

Šis pseudo kodas nuskaito duomenų failą, išskiria atskirus elementus su „flatMap“ funkcija, kuriuose yra parametrai apie tam tikrą siuntą ar maršrutą ir t.t. Tada su „Map“ einama per kiekviena elemento parametrą ir išvedame mums reikalingus duomenis: sustojimo savaitės dieną, siuntų skaičių, geografine zoną ir sustojimo klientų skaičių. Išvedamas sustojimo diena ir geografinė zona kaip raktas (sujungiant šiuos parametrus per brūkšnelį), o reikšmė būtų siuntų ir klientų skaičius, pateikiamas masyvu. Surinkti duomenys atrodo taip:

Text

Description automatically generated

**Pav. 26** "Map" išvestis

Kadangi parašytas kodas nepilnai patikrina ar visos reikšmės yra teisingos, todėl galima pasinaudoti „Filter“ funkcija, kuria būtų galima išfiltruoti visas „None“ reikšmes. Tada išvestis atrodo taip:

Text

Description automatically generated

**Pav. 27** "Filter" išvestis

RDD palaiko dviejų rūšių transformacijas, tai siaurąsias ir plačiąsias. Siaurosios, tai tokios transformacijos kurios nereikalauja duomenų rūšiavimo bei grupavimo. Tai yra tokios operacijos kaip: „FlatMap“, „Map“ ir „Filter“. Šios transformacijos yra atliekamos lokaliai, tame pačiame mazge. O plačiosios transformacijos tai yra tokios kaip „ReduceByKey“, „Join“ ir kitos, kurioms reikia sugrupuotų ir surūšiuotų duomenų. Šioms operacijoms jau reikia perduoti duomenis tarp mazgų. Taigi, čia panaudojome „FlatMap“, „Map“ ir „Filter“, kurios yra siaurosios transformacijos.

Toliau reikia sukurti „Reduce“ programinį kodą. Kadangi jau turime mums reikiamus duomenis, naudodamiesi „Reduce“ kodu mes iš duomenų išgauname mums reikalingą informaciją. Šiuo atveju mums reikia žinoti kiek siuntų buvo pristatyta ir kiek klientų aptarnauta. Iš kiekvieno mazgo, gautus duomenis reikia sudėti pagal raktą, todėl naudosime funkciją „ReduceByKey“, kuri kiekvieno rakto reikšmę sudės atitinkamai.

Pseudo kodas:

def red (a,b):

return (a[0] + b[0], a[1] + b[1])

D = C.reduceByKey(red)

„Reduce“ kodas tiesiog sudeda atitinkamas reikšmes. Kadangi mūsų reikšmė yra masyvas, mes kiekvienos reikšmės pirmąją reikšmę sudedame su kitos reikšmės pirmąją reikšmę ir tą patį darome su antrąja reikšme. Gautas rezultatas yra toks:

Text

Description automatically generated

**Pav. 28** "Reduce" išvestis

„ReduceByKey“ yra plačioji transformacija, nes šiai operacijai reikia surūšiuotų bei sugrupuotų duomenų. Dar paskutiniame etape galime šias reikšmes išrikiuoti, kad duomenys būtų tvarkingesni. Galutinis rezultatas:

Text

Description automatically generated

**Pav. 29** išrikiuotas rezultatas

Su pilnais duomenimis paleistas kodas buvo išvestas „Hortonworks HDP“ sistemoje:

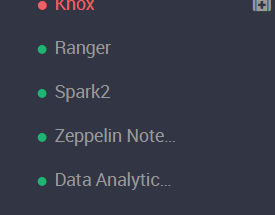
Graphical user interface, text, application

Description automatically generated

**Pav. 30** rezultatas "Hortonworks HDP" sistemoje

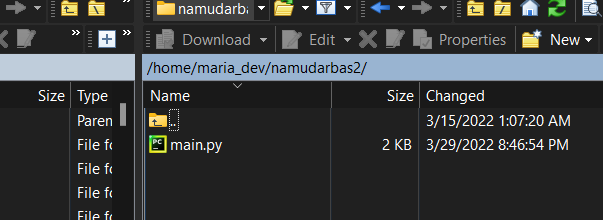
# „Spark“ aplinkos pasiruošimas ir komandos

Šiam darbui bus naudojama ne „Hadoop“, o „Spark“ programinė įranga. „Spark“ veikimo principas yra panašus į „Hadoop“, tik skirtumas, kad „Spark“ viskas veikia operatyviojoje atmintyje, veikia greičiau ir pateikia patogesnius įrankius duomenims išgauti, ir supaprastina visą procesą. Prieš pradedant darbą reikia įsitikinti, kad yra įjungtas „Spark“ modulis „Hortonworks HDP“ sistemoje.



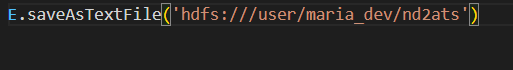
**Pav. 31** "Spark" modulio įjungimas per "Hortonworks HDP"

Tada per „WinScp“ yra įkeliamas programinis „Python“ kodas.



**Pav. 32** per "WinScp" įkeliamas kodas

Programiniame kode yra nurodoma kur išsaugomi bus failai.



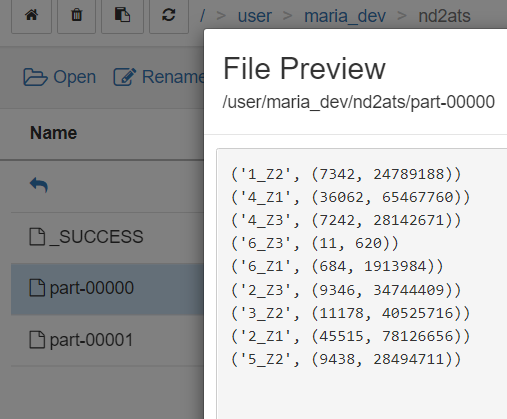
**Pav. 33** per "Spark" nurodoma kur bus saugomi failai

Programinis kodas yra paleidžiamas su „spark-submit“.



**Pav. 34** "spark-submit" komanda

O gautą rezultatą galime matyti HDFS sistemoje.



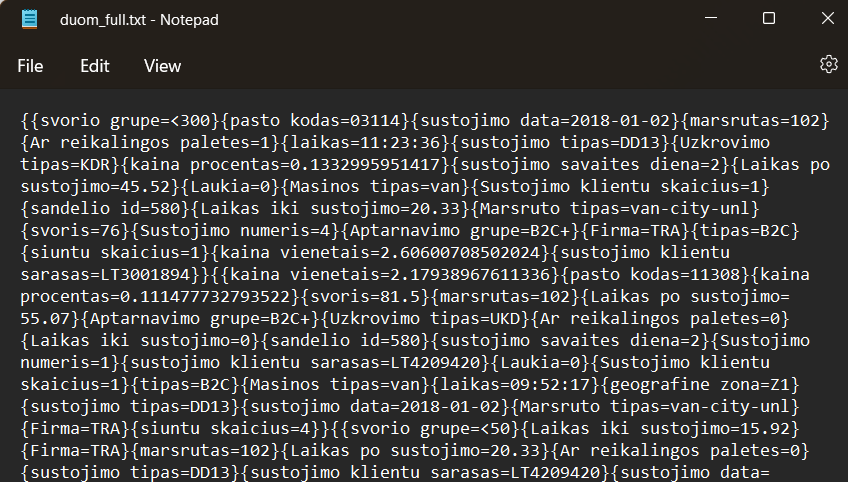
**Pav. 35** rezultatas HDFS sistemoje

# 

# Trečia užduotis

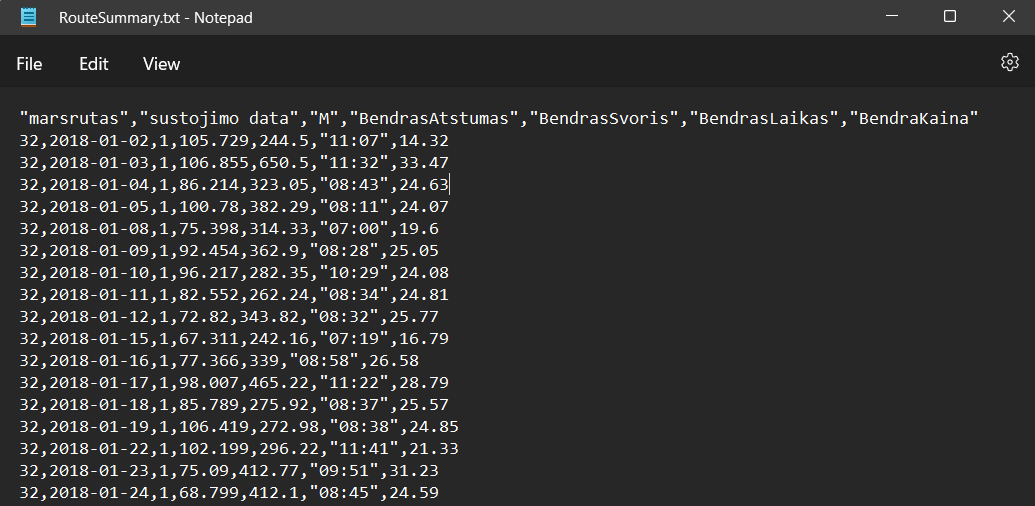
# Trečio laboratorinio užduotis

Laboratoriniame darbe reikia išnagrinėti duomenis, aprašančius siuntų pristatymo bei logistikos įmonės veiklą. Duotajame faile pateikti dalinai struktūrizuoti duomenys apie siuntų išvežiotojų sustojimus. Failas „duom\_full.txt“ ir sutrumpintas duomenų failas „duom\_cut.txt“ (paveikslas 36).



**Pav. 36** pirmojo duomenų failo fragmentas

Taip pat, šiuos duomenis papildysime duomenimis kitokio formato, iš failo „RoutesSummary.txt“ (paveikslas 37). Šiame faile dar yra pridedami nauji parametrai.



**Pav. 37** antrojo duomenų failo fragmentas

Laboratoriniame darbe atliksiu **2 užduotį,** joje reikia iš abiejų duomenų rinkinių kartu sudėjus, ištirti tiesinę priklausomybę parametro "BendraKaina" nuo parametro "svoris" (agreguojant pagal maršrutą ir datą), kai nagrinėjami duomenys tik su viena ta pačia reikšme "Masinos tipas".

Skirtingai nuo praeitų laboratorinių darbų, šiame darbe reikės kelių papildomų bibliotekų, kad įgyvendintume norimą rezultatą. Tos bibliotekos yra: „Spark SQL”, „MLlib“ ir „matplotlib“.

* „Spark SQL” – tai biblioteka skirta apdoroti struktūrizuotus duomenis, tokius kaip SQL lenteles bei „DataFrame“. Tai lentelės tipo duomenys, kur yra stulpeliai ir eilutės. Su biblioteka galima ištrinti, pridėti naujus stulpelius, sujungti lenteles bei atlikti visas kitas lentelių transformacijas.
* „MLlib“ – suteikia prieigą prie mašininio mokymosi įrankių „Spark“ sistemoje ir priderina skaičiavimus paskirstytai sistemai. Tai tokie įrankiai kaip: statistika, požymių apdorojimas, klasifikacija, regresija, klasterizavimas ir kita.
* „matplotlib“ – tai yra atviro kodo biblioteka skirta įvairių matematinių ir statistinių grafikų vizualizavimui „Python“ aplinkoje.

Pirma reikia sukurti „Map“ programinį kodą, kuris iš duomenų mums atrinktu tik reikiamus parametrus, kuriuos analizuosime. Šiame duomenų rinkinyje yra pateikiama daug informacijos apie pristatymus, bet mums reikia tiktai maršruto, datos, mašinos tipo, svorio ir bendros kainos. Duomenys bus jungiami pagal datą ir maršrutą, o bendra kaina bus gaunama sujungus papildomus duomenis iš antro failo.

Pseudo kodas:

lines = sc.textFile()

def fm(line):

return line.split()

def mapF(line):

objs = line.split()

for at in objs:

key, val=at.split()

if(key == 'marsrutas'):

marsrutas = val

if(key == 'sustojimo data'):

data = val

if(key == 'Masinos tipas'):

tipas = val

if(key == 'svoris'):

svoris = val

return (marsrutas + "\_" + data, (svoris, tipas))

fmap = lines.flatMap(fm)

mmap = fmap.map(mapF)\

.filter(lambda l : l[1][1] == tipas)\

.reduceByKey(lambda a, b: (a[0] + b[0], a[1]))\

.map(lambda t: (t[0], t[1][0]))

Šis pseudo kodas nuskaito pirmąjį duomenų failą, išskiria atskirus elementus su „FlatMap“ funkcija, kuriuose yra parametrai apie tam tikrą siuntą ar maršrutą ir t.t. Tada einama per kiekvieną elemento parametrą ir išvedame mums reikalingus duomenis: maršrutas, sustojimo data, mašinos tipas ir svoris. Išvedamas maršrutas ir data kaip raktas (sujungiant šiuos parametrus per brūkšnelį), o reikšmė būtų svoris ir mašinos tipas, pateikiamas masyvu. Tada naudojantis „Filter“ funkcija filtruojame reikšmes pagal mašinos tipą. Tada naudojantis „ReduceByKey“ sudedame svorio reikšmes pagal raktą ir galiausiai panaudojame vėl „Map“ funkciją, kuri duomenyse palieka tiktai raktą ir svorį (mašinos tipas nebereikalingas, nes žinome, kad visos reikšmės jau išfiltruotos pagal mašinos tipą). Surinkti duomenys atrodo taip:

[

('102\_2018-01-09', 1619.3),

('102\_2018-01-18', 1017.0500000000001),

('102\_2018-01-19', 1114.2500000000005),

('102\_2018-01-26', 680.1999999999999),

('102\_2018-01-31', 1204.6499999999999)

]

Taip pat, panaudojant šį kodą, galima sužinoti kokie yra skirtingi mašinų tipai:

tipai = fmap.map(parsinam2)\

.map(lambda t:(t[1][1],0))\

.reduceByKey(lambda a,b : a)\

.map(lambda t:t[0])

Šiuo kodu išfiltruojami visi skirtingi mašinų tipai. Mums reikia sužinoti visus mašinų tipus, kad galėtume su jais visais atlikti tiesinę regresijos analizę. Pirmiausia su „Map“ funkcija paliekame tiktai mašinos tipo pavadinimus kaip raktą, tada su „ReduceByKey“ paliekame tik pavienes reikšmes, o tada su paprastu „Map“ sudedame visas reikšmes į vieną masyvą ir gauname visus mašinos tipus:

[

'other', 'van', 'truck', 'linehaul', 'depot'

]

Tada mums reikia gauti kitus duomenis iš antrojo failo. Kadangi tai yra „csv“ failas, tai mums leidžia jį atitinkamai nuskaityti kaip “Spark SQL” duomenų bazės lentelės failą, kas leidžia lengvai manipuliuoti duomenimis, išimti ir atimti stulpelius.

Pseudo kodas:

routes = spark.read.csv("RouteSummary.txt")

routes = routes.drop("BendrasAtstumas","BendrasLaikas","BendrasSvoris")

routes2 = routes.withColumn('ID', makeID("marsrutas", "sustojimo data"))\

.drop("marsrutas", "sustojimo data")

Šiuo kodu nuskaitome failą, išimame nereikalingus stulpelius ir vietoje maršruto ir sustojimo datos, mes pasidarome vieną stulpelį jungiantį šias reikšmes. Šis reikšmių sujungimas yra padaromas naudojant “udf”, kas leidžia atlikti tam tikrą operaciją visoms lentelės reikšmėms.

Kadangi jau nusiskaitėme ir susitvarkėme abu duomenų failus, mes juos dabar galime sujungti į vieną kintamąjį:

routes = aggregated\_rdd.toDF(["ID", "svoris"])

training = routes.join(routes2, 'ID', "inner")

Iš pirmojo duomenų failo nuskaitytus duomenis suvienodiname, paverčiame į „DataFrame“ duomenų tipą, kas paverčia duomenis į tą patį lentelės duomenų tipą ir sujungiame šiuos duomenų failus pagal raktą: maršrutą ir sustojimo datą.

Toliau belieka paruošti duomenis tiesinės regresijos sudarymui naudojant “VectorAssembler” iš “MLlib” paketo:

vectorAssembler = VectorAssembler(inputCols = ['svoris'], outputCol = 'features')

vhouse\_df = vectorAssembler.transform(training)

vhouse\_df = vhouse\_df.select(['features', 'BendraKaina'])

Pasirenkame, kad parametras pagal kurį sudarinėsime tiesinę regresijos analizę yra svoris. Šiuos duomenis sudedame į “features”, o kintamasis kurį norime prognozuoti yra bendra kaina. Toliau belieka paleisti mokymą, naudojant “LinearRegression” iš “MLlib” paketo:

lr = LinearRegression(featuresCol = "features", labelCol = "BendraKaina")

lrModel = lr.fit(vhouse\_df)

Ir galiausiai iš gautų duomenų galime sužinoti tiesinės regresijos lygties koeficientus, determinacijos koeficientą bei nubraižyti grafikus, naudojant “matplotlib”:

trainingSummary = lrModel.summary

a=lrModel.intercept

b=lrModel.coefficients[0]

print(f'a={a} b={b}')

print("r2: %f" % trainingSummary.r2)

fig, axes = plt.subplots(nrows=1, ncols=3, figsize=(20,5))

axes[2].scatter(x,y, s = 10)

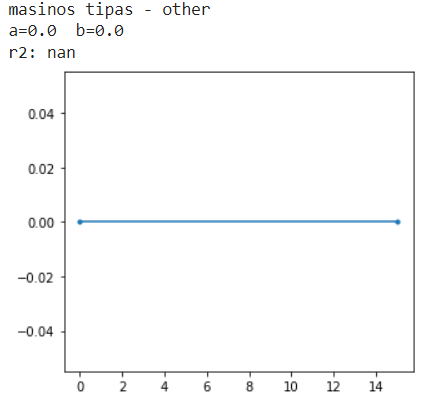
lineX = [min(x), max(x)]

lineY = [a+b\*lineX[0],a+b\*lineX[1]]

axes[2].plot(lineX, lineY)

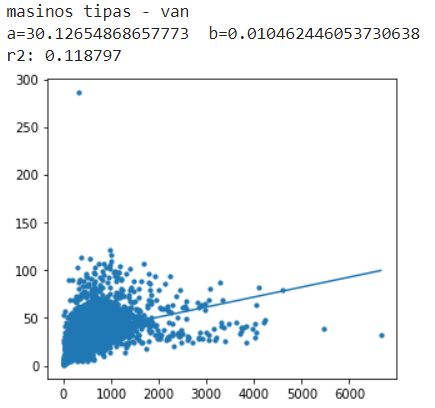
plt.show()

Gauti rezultatai rodo, kad su mašinos tipu “other” koeficientai yra nuliai, o determinacijos koeficientas yra “nan”, kas reiškia, kad nebuvo duomenų, kad galėtų sukurti tiesinės regresijos modelį.



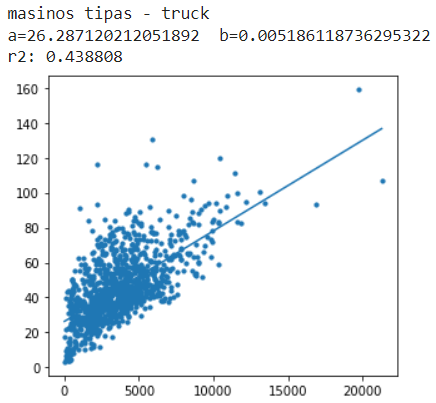
**Pav. 38** „other“ mašinos tipo regresijos lygtis su sklaidos diagrama

Toliau mašinos tipas “van”, determinacijos koeficientas rodo 0.11. Labai abejotina, kad yra tiesinė priklausomybė tarp svorio ir bendros kainos šiam mašinos tipui. Taip pat, tai patvirtina sklaidos diagrama, galima matyti, kad reikšmės yra išsibarsčiusios toli vienas nuo kito, todėl sunku teigti, kad yra kažkokia priklausomybė.



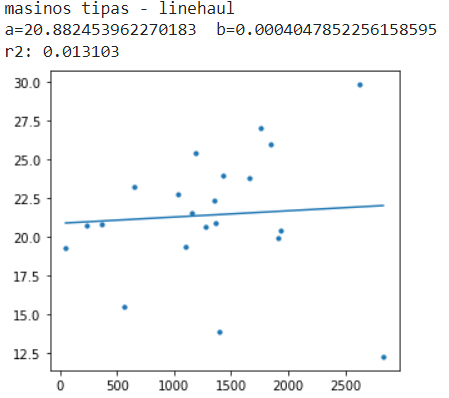
**Pav. 39** „van“ mašinos tipo regresijos lygtis su sklaidos diagrama

Toliau “truck” determinacijos koeficientas yra 0.43, reiškias šis tiesinės regresijos modelis gali būti naudojamas prognozavimui ir yra kažkokia priklausomybė tarp svorio ir bendros kainos. Pagal grafiką galima matyti, kad didėjant svoriui, didėja ir kaina. Nors pagal grafiką ir pagal determinacijos koeficientą nebūtų galima teigti, kad tiesinė regresijos lygtis yra tinkama tiksliam kainos prognozavimui, bet tai parodo, kad yra priklausomybė tarp svorio ir bendros kainos.



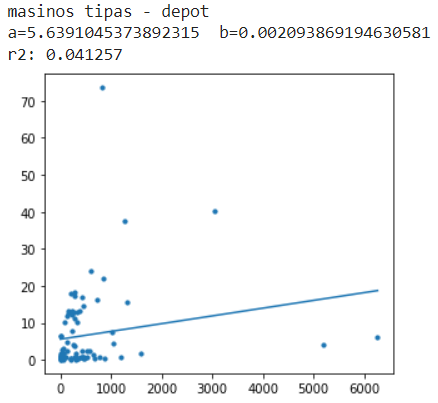
**Pav. 40 „**truck“ mašinos tipo regresijos lygtis su sklaidos diagrama

Kitas mašinos tipas “linehaul” turi determinacijos koeficientą 0.01, kas reiškia, kad nėra jokios priklausomybės. O iš grafiko galime matyti, kad reikšmės yra išsibarsčiusios aplinkui ir nerodo jokių priklausomybės požymių. Taip pat, yra per mažai duomenų, kad būtų tinkamai sudaryti tiesinė regresiją.



**Pav. 41** „linehaul“ mašinos tipo regresijos lygtis su sklaidos diagrama

Ir paskutinis mašinos tipas “depot”, taip pat turi labai žemą determinacijos koeficientą 0.04. Kaip praeitas mašinos tipas, svoris nerodo jokios priklausomybės nuo bendros kainos, tą patvirtina ir grafikas.



**Pav. 42** „depot“ mašinos tipo regresijos lygtis su sklaidos diagrama

Jeigu šis kodas būtų vykdomas klasteryje visos siaurosios transformacijos kaip „Map“ ir „Filter“ būtų vykdomos lokaliai mazguose, o plačiosios transformacijos reikalautų siuntimo tarp mazgų klasteryje. Taip pat, failo nuskaitymas veikia su serveriu.

# Priedai

# Pirmas laboratorinis

# „Map“

#!/usr/bin/env python

import sys

sys.stdin = open("duom\_cut.txt","r")

sys.stdout = open("mapout2.txt","w")

for line in sys.stdin:

  line = line.strip()[2:-2]

  elements = line.split('}}{{')

  for el in elements:

    parameters = line.split('}{')

    marsrutas = None

    geografine\_zona = None

    sustojimo\_data = None

    for param in parameters:

      k, v = param.split('=')

      k = k.replace('}','').replace('{','')

      v = v.replace('}','').replace('{','')

      if k == 'marsrutas':

        marsrutas = int(v)

      elif k == 'geografine zona':

        geografine\_zona = v

      elif k == 'sustojimo data':

        sustojimo\_data = v

    if marsrutas != None and geografine\_zona != None and sustojimo\_data != None and marsrutas != '' and geografine\_zona != '' and sustojimo\_data != '' and marsrutas != ' ' and geografine\_zona != ' ' and sustojimo\_data != ' ':

      key = str(marsrutas) + '\_' + str(sustojimo\_data)

      print('%s\t%s' % (key, geografine\_zona))

# „Reduce“

#!/usr/bin/env python

import sys

sys.stdin = open("smapout.txt","r")

sys.stdout = open("red2outnew.txt","w")

current\_key = None

current\_set = {}

for line in sys.stdin:

    line = line.strip()

    marsrutas\_data, zona = line.split('\t')

    if current\_key == marsrutas\_data:

        current\_set.add(zona)

    else:

        if current\_key != None:

            if len(current\_set) > 1:

                print('%s\t%s' % (1, current\_key))

        current\_set = {zona}

        current\_key = marsrutas\_data

if current\_key != None:

    if len(current\_set) > 1:

        print('%s\t%s' % (1, current\_key))

# Antras laboratorinis

from pyspark import SparkContext, SparkConf

conf = SparkConf().setAppName('MyFirstStandaloneApp')

sc = SparkContext(conf=conf)

lines = sc.textFile("namudarbas1/duom\_full.txt")

def FM(line):

  line = line.strip()

  line = line[2:-2]

  susstring = line.split('}}{{')

  return susstring

A=lines.flatMap(FM)

print(A.take(3))

def MapF(stopas):

  parstrings = stopas.split('}{')

  diena = None

  siuntos = None

  geografine\_zona = None

  klientu\_skaicius = None

  for parstring in parstrings:

    (vardas, reiksme) = parstring.split('=')

    if(reiksme != '' and vardas == 'sustojimo savaites diena'):

      diena = reiksme

    if(reiksme != '' and vardas == 'siuntu skaicius'):

      siuntos = reiksme

    if(reiksme != '' and vardas == 'geografine zona'):

      geografine\_zona = reiksme

    if(reiksme != '' and vardas == 'sustojimo klientu skaicius'):

      klientu\_skaicius = reiksme

  try:

    diena = int(diena)

    siuntos = int(siuntos)

    klientu\_skaicius = int(reiksme)

  except:

    diena = None

    siuntos = None

    geografine\_zona = None

    klientu\_skaicius = None

  key = str(diena) + '\_' + str(geografine\_zona)

  return(key, [siuntos, klientu\_skaicius])

B = A.map(MapF)

test = B.take(5)

C = B.filter(lambda pair : pair[1][0] != None and pair[1][1] != None and pair[0].find('None') == -1)

def red (a,b):

  return (a[0] + b[0], a[1] + b[1])

D = C.reduceByKey(red)

E = D.sortByKey()

E.saveAsTextFile('hdfs:///user/maria\_dev/nd2ats')

# Trečias laboratorinis

from pyspark import SparkContext, SparkConf

conf = SparkConf().setAppName('MyFirstStandaloneApp')

sc = SparkContext(conf=conf)

from pyspark.sql import SparkSession

spark = SparkSession.builder.appName('Laboras3').getOrCreate()

from pyspark.sql.functions import udf, log

from pyspark.sql.types import \*

# vyksta duomenų siuntimas tarp mazgu nuskaitant faila

text\_file = spark.sparkContext.textFile("duom\_full.txt")

def parsinam(line):

    return line[2:len(line)-2].split('}}{{')

def parsinam2(line):

    objs = line.split('}{')

    k1=None

    k3=None

    tipas= None

    svoris=None

    for at in objs:

        temp = at.split('=')

        if(len(temp)<2):

            break

        key,val=at.split('=')

        if(key == 'marsrutas'):

            k1=val

        if(key == 'sustojimo data'):

            k3=val

        if(key == 'Masinos tipas'):

            tipas=val

        if(key == 'svoris'):

            svoris=val

    if(k1!=None and k3!=None and tipas!=None and svoris!=None):

        return (k1+"\_"+k3, (float(svoris), tipas))

    else:

        return ("0", (1, "blogai"))

fmap = text\_file.flatMap(parsinam)

# vyksta duomenų siuntimas tarp mazgu naudojant reduceByKey

tipai = fmap.map(parsinam2).map(lambda t:(t[1][1],0))\

    .reduceByKey(lambda a,b : a).map(lambda t:t[0])

from pyspark.sql.functions import col, udf

from pyspark.sql.types import StringType

# vyksta duomenų siuntimas tarp mazgu nuskaitant faila

routes = spark.read.option("header",True).csv("RouteSummary.txt", inferSchema=True)

routes.printSchema()

routes = routes.drop("M", "BendrasAtstumas","BendrasLaikas","BendrasSvoris")

def makeID(str1, str2):

    return str(str1)+"\_"+str2

makeID\_UDF = udf(lambda z1,z2: makeID(z1,z2),StringType())

routes2 = routes.withColumn('ID', makeID\_UDF("marsrutas", "sustojimo data")).drop("marsrutas", "sustojimo data")

from pyspark.ml.feature import VectorAssembler

from pyspark.ml.regression import LinearRegression

import matplotlib.pyplot as plt

from matplotlib import cm

from matplotlib.ticker import LinearLocator

from mpl\_toolkits.mplot3d import Axes3D

for tipas in tipai.collect():

    if tipas == "blogai":

        continue

    print(tipas)

    print(tipas)

    print(tipas)

    print(tipas)

    print(tipas)

# vyksta duomenų siuntimas tarp mazgu naudojant reduceByKey

    aggregated\_rdd = fmap.map(parsinam2) \

        .filter(lambda l : l[1][1] == tipas) \

        .reduceByKey(lambda a, b: (a[0] + b[0], a[1]))\

        .map(lambda t: (t[0], t[1][0]))

    # vyksta siuntimas tarp serverio mazgu kai naudojamas toDF

    routes = aggregated\_rdd.toDF(["ID", "svoris"])

    routes2 = routes2.withColumn("BendraKaina",routes2.BendraKaina.cast('float'))

    # vyksta siuntimas tarp serverio mazgu kai naudojamas join

    training = routes.join(routes2, 'ID', "outer")

    training = training.na.drop("any")

    vectorAssembler = VectorAssembler(inputCols = ['svoris'], outputCol = 'features')

    # vyksta siuntimas tarp serverio mazgu kai transformuojame ir darome select

    vhouse\_df = vectorAssembler.transform(training)

    vhouse\_df = vhouse\_df.select(['features', 'BendraKaina'])

    lr = LinearRegression(maxIter=10, regParam=0.3, elasticNetParam=0.8, featuresCol = "features", labelCol = "BendraKaina")

    # vyksta siuntimas tarp serverio mazgu kai naudojamas fit

    lrModel = lr.fit(vhouse\_df)

    trainingSummary = lrModel.summary

    a=lrModel.intercept

    b=lrModel.coefficients[0]

    print(f'masinos tipas - {tipas}')

    print(f'a={a}  b={b}')

    print("r2: %f" % trainingSummary.r2)

    # vyksta siuntimas tarp serverio mazgu kai konvertuojame i pandas

pandasDF = training.toPandas()

    x = pandasDF['svoris'].to\_list()

    y = pandasDF['BendraKaina'].to\_list()

    fig, axes = plt.subplots(nrows=3, ncols=1, figsize=(5,15))

    axes[0].scatter(x,y, s = 10)

    lineX = [min(x), max(x)]

    lineY = [a+b\*lineX[0],a+b\*lineX[1]]

    axes[0].plot(lineX, lineY)

    plt.show()