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

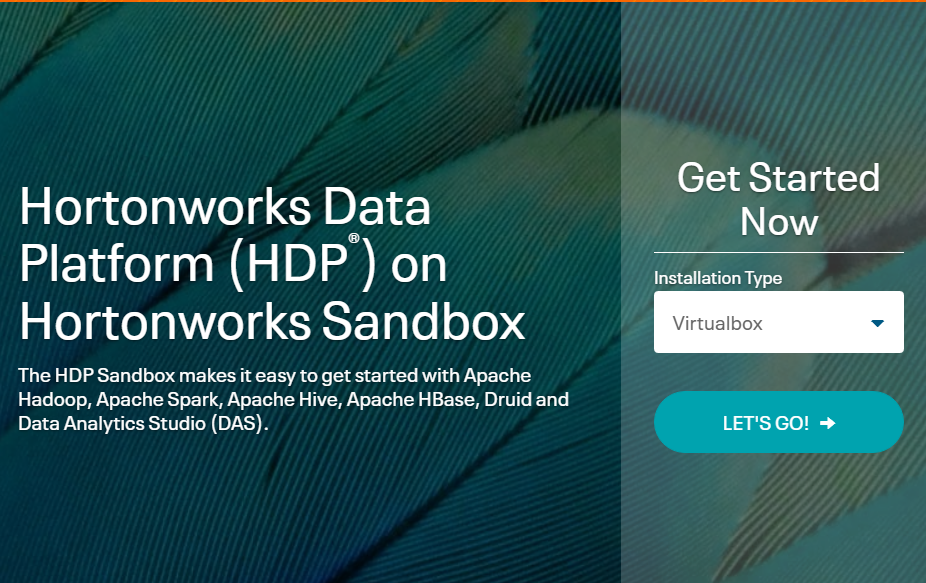
...

...

# Pirma užduotis

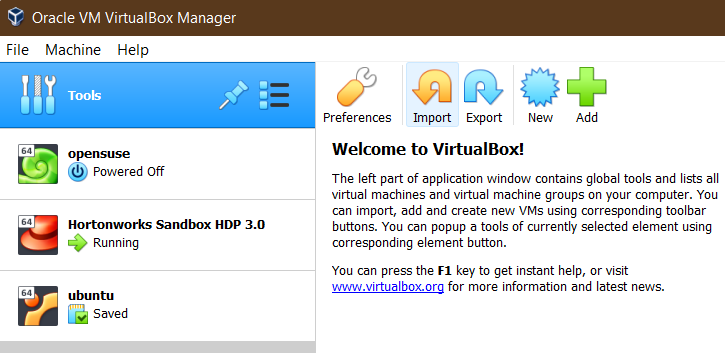
# Aplinkos pasiruošimas

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)



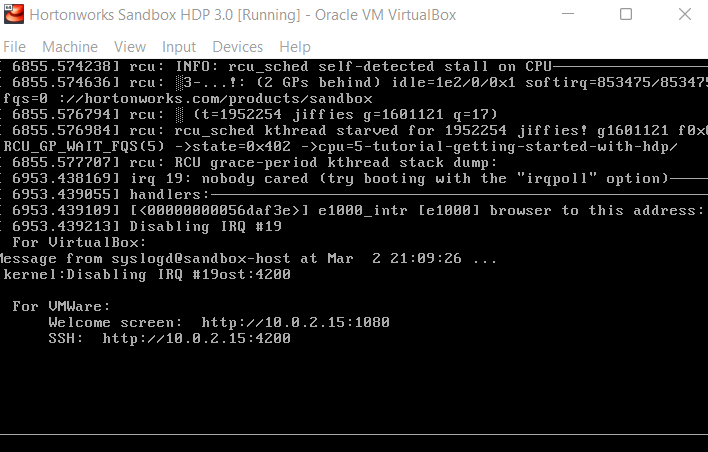
**Pav. 1** "Hortonworks HDP" puslapio vaizdas

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



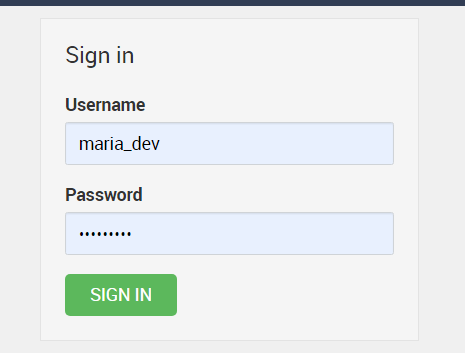
**Pav. 2** "VirtualBox" pradinis vaizdas

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



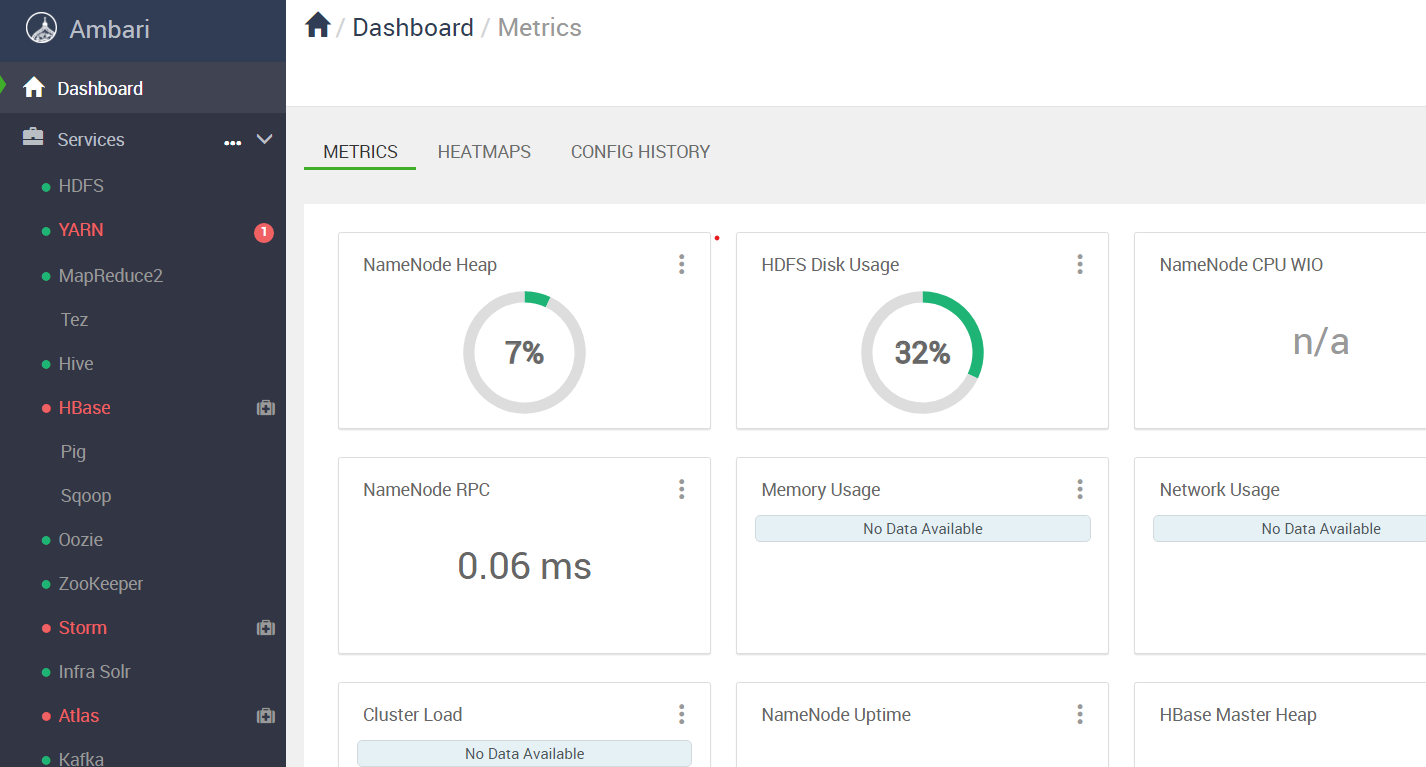
**Pav. 3** "Hortonworks HDP" vaizdas paleidus per "VirtualBox"

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



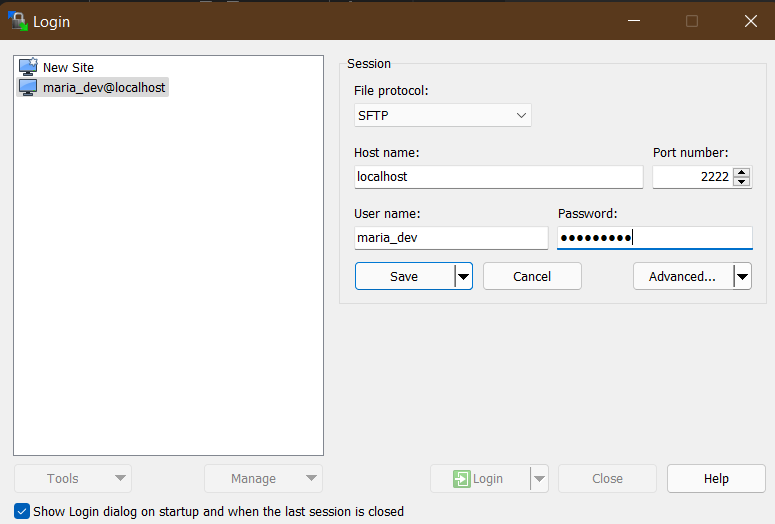
**Pav. 4** prisijungimas prie "Hortonworks HDP"

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



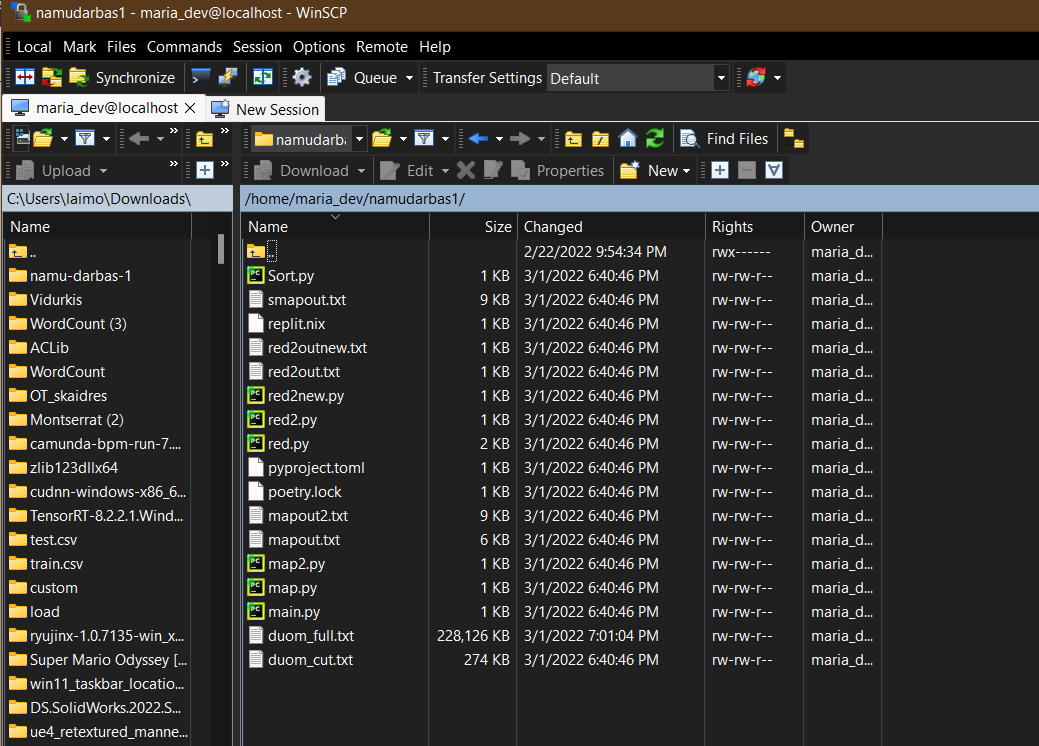
**Pav. 5** "Hortonworks HDP" pagrindinis langas

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



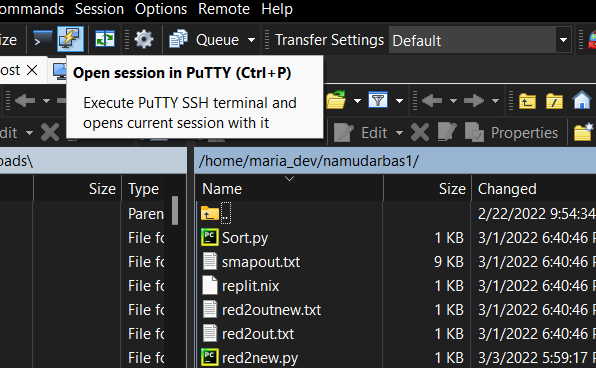
**Pav. 6** prisijungimas su "WinScp"

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



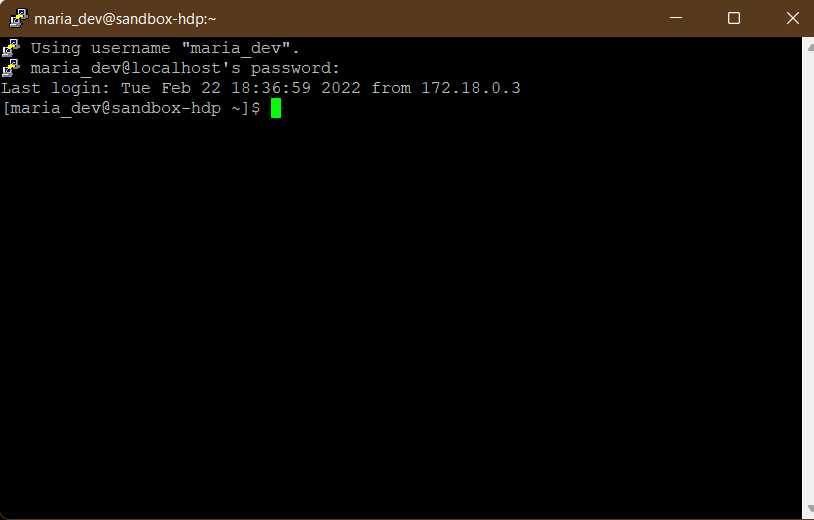
**Pav. 7** failų perkėlimas per "WinScp"

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



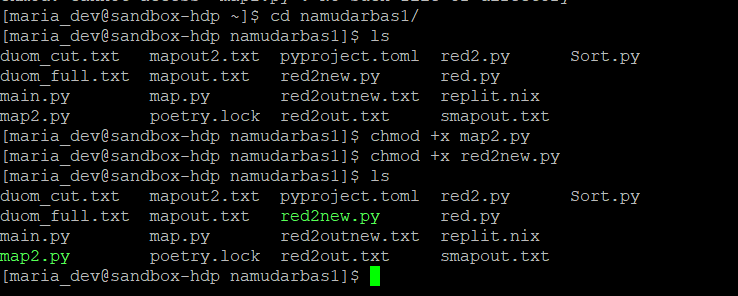
Pav. 8 "PuTTY" atidarymas per "WinScp"

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



**Pav. 9** 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.



**Pav. 10** esamų failų vaizdas terminale

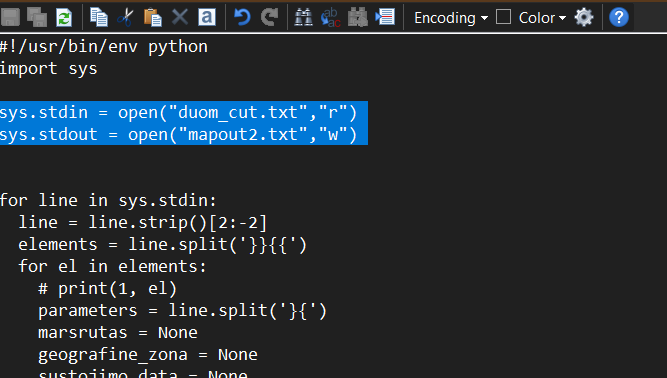
Tada failus duomenų failus reikia perkelti į HDFS duomenų bazę. Pirmiausia yra sukuriama direktorija kur tie failai bus saugomi su komanda „mkdir”, o kad tai būtų įvykdoma HDFS aplinkoje reikia tai nurodyti su „hadoop fs” komanda pvz.: „hadoop fs -mkdir namudarbas”. O duomenys yra perkeliami su „hadoop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





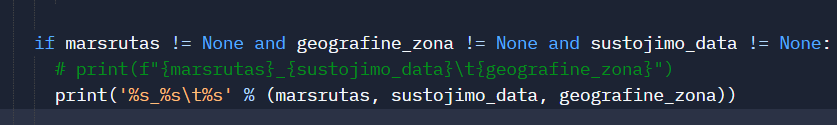
**Pav. 11** HDFS direktorijų kūrimo komandos

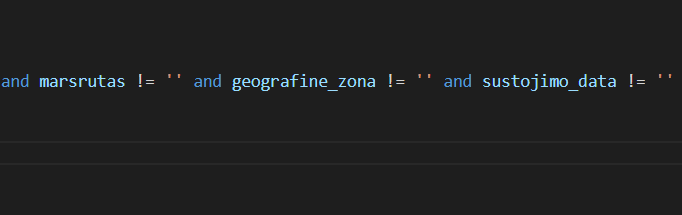
Prieš paleidžiant programinį MapReduce kodą reikia įsitikinti ar jame nėra atidarinėjami failai, nes tai išmes klaidos pranešimą Hadoop sistemoje.



**Pav. 12** 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.





**Pav. 13** 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 \

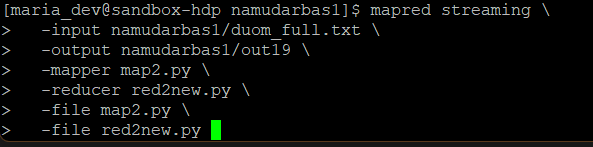
-mapper map2.py \

-reducer red2new.py \

-file map2.py \

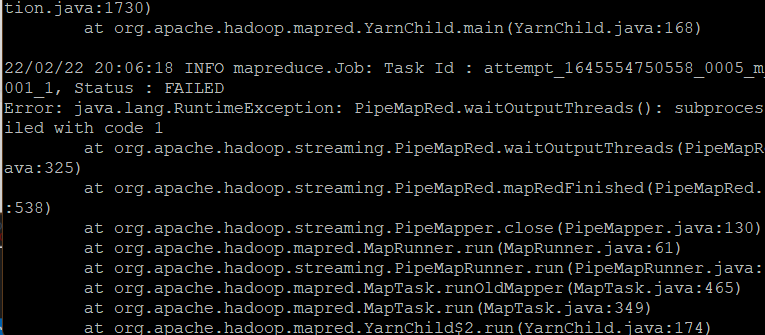
-file red2new.py

”



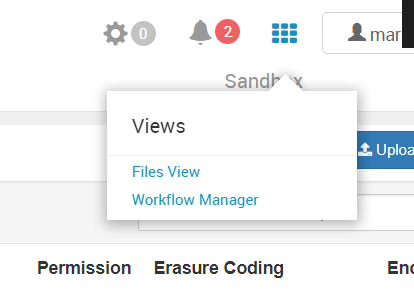
**Pav. 14** MapReduce komanda

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.



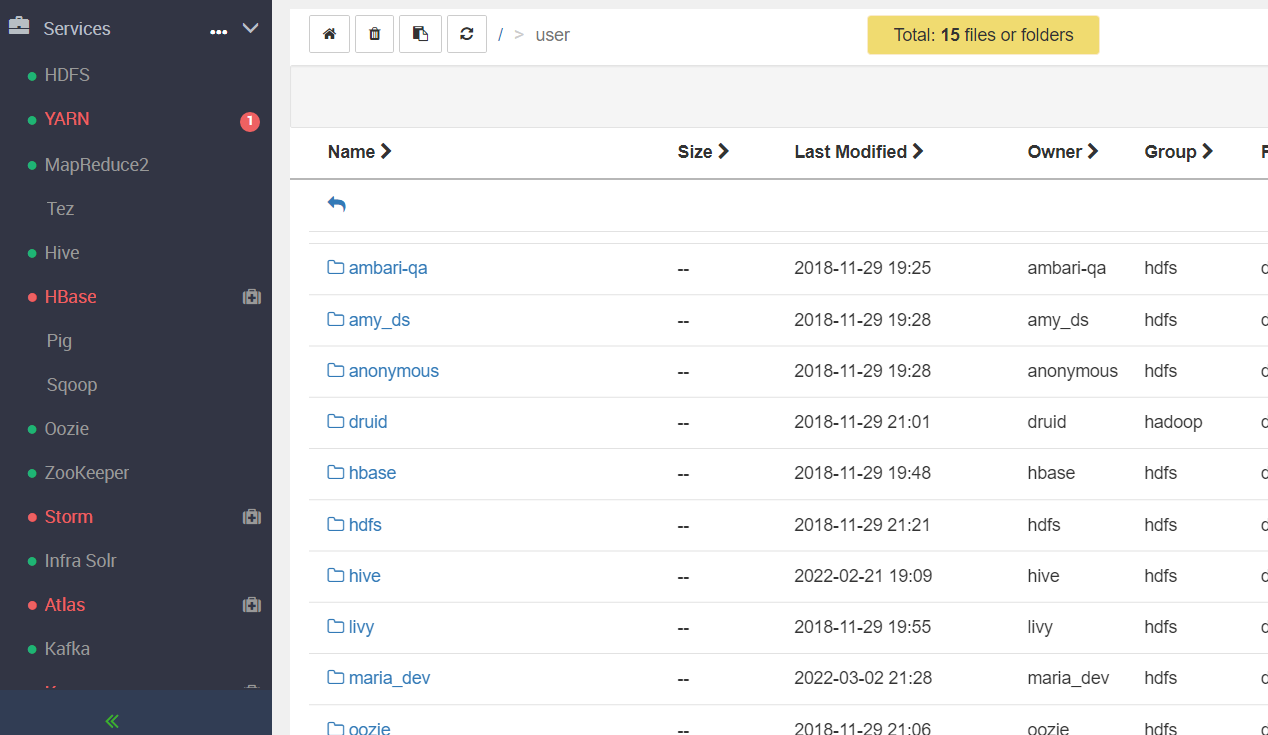
**Pav. 15** "Hadoop" klaidos pranešimas

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



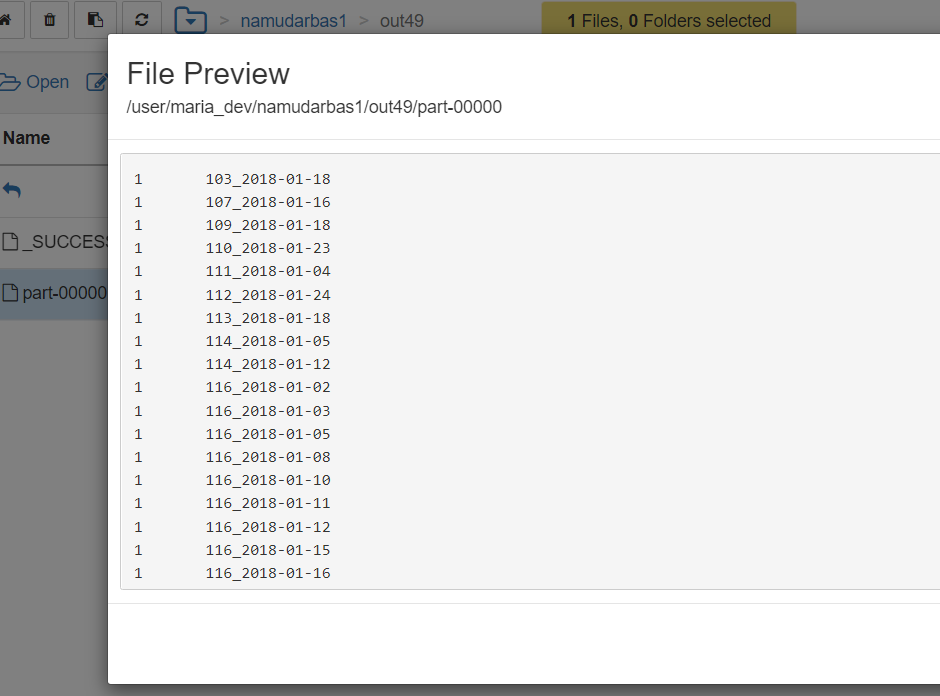
**Pav. 16** "File View" nuoroda

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



**Pav. 17** "Hortonworks HDP" failų sistema

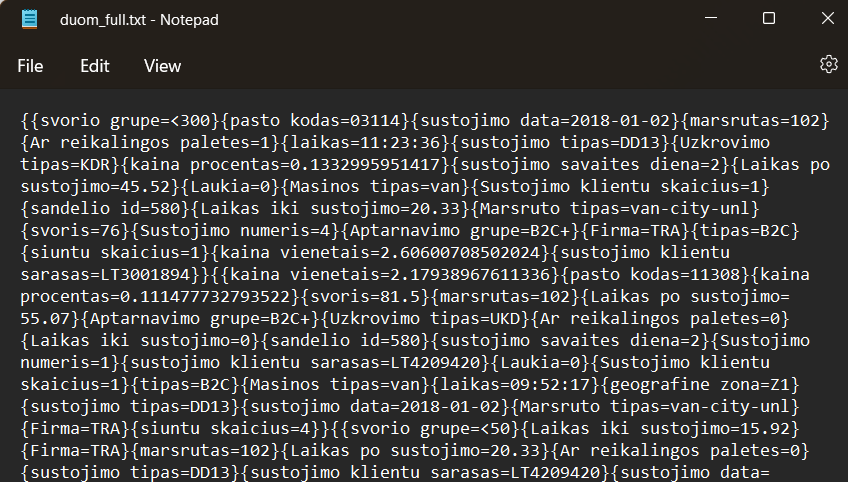
Direktorijos viduje bus failas su išvesties duomenimis, kurį galima peržiūrėti.



**Pav. 18** išvestas programos rezultatas

# Pirmojo namų darbo užduotis

Namų 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”.



**Pav. 19** duomenų failo fragmentas

Namų 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.

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 maršruto, geografines zonos ir datos. Kadangi reikia rasti maršrutus, kurie aplanko daugiau nei viena 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':

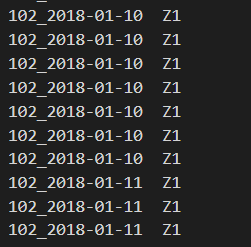
sustojimo\_data = value

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 kiekviena 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 negalima išvesti šitų duomenų ir iteruojama 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:



**Pav. 20** "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:

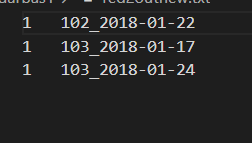
if len(current\_set) > 1:

emit(1, current\_key)

current\_set = set(zona)

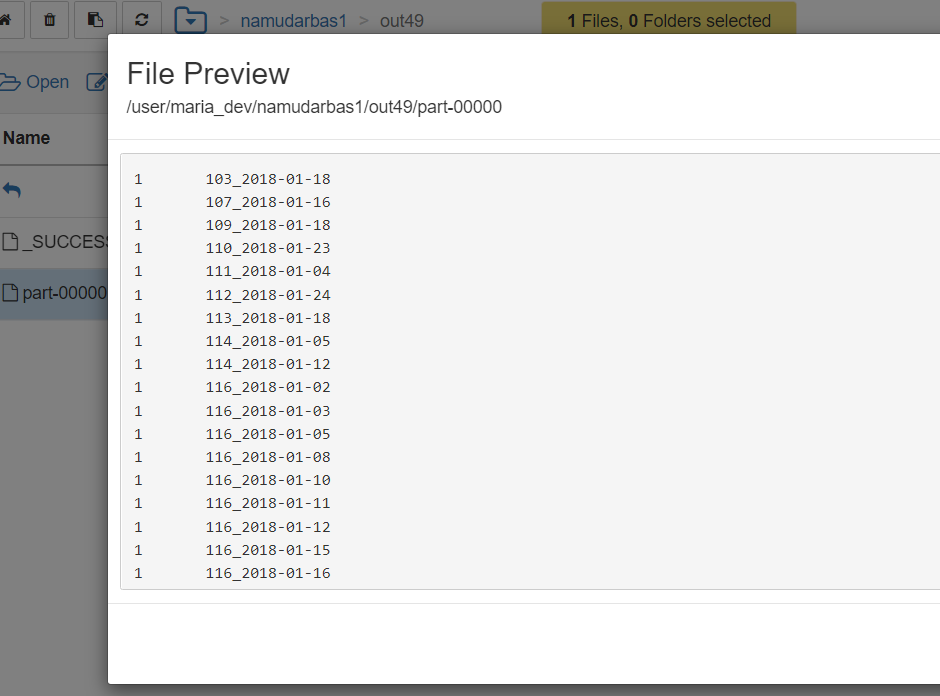
current\_key = marsrutas\_data

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. 21** "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:

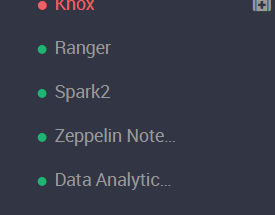


**Pav. 22** "MapReduce" rezultatas "Hortonworks HDP" sistemoje

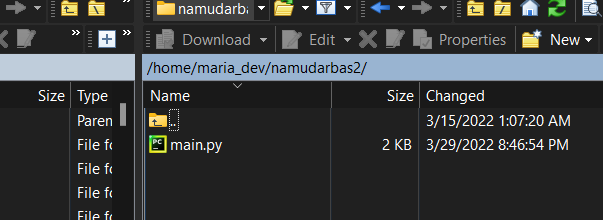
Taip pat, šiame uždavinyje būtų galima pritaikyti „Combiner“. „Combiner“ leidžia vykdyti kodą iškart po „Map“, tame pačiame mazge, su tais duomenimis, kurios išvedė „Map“. Pavyzdžiui, šio uždavinio atveju, galėtume po „Map“ pritaikyti „Combiner“, tam kad paliktume tik unikalius rakto ir reikšmės variantus. Kadangi mums reikia sužinoti maršrutus, tą dieną, kurie aplankė kelias geografines zonas, tai mes su „Combiner“ galėtume pašalinti visas pasikartojančias reikšmes (kaip pav. 20) ir tokiu būdu sumažinti perduodamą kiekį duomenų į „Reduce“ bei galimai pagreitinant visą skaičiavimo procesą, nes mazguose skaičiavimai būtų atliekami su mažesniu skaičiumi duomenų. Deja, „Combiner“ netinka visoms problemoms spręsti, todėl ne visada galima jį pritaikyti.

# Antra užduotis

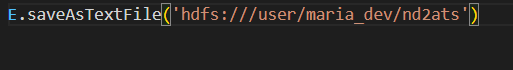
Šiam darbui bus naudojama ne „Hadoop“, o „Spark“ programinė įranga. „Spark“ veikimo principas yra panašus į Hadoop, tik skirtumas, kad Spark viskas veikia operatyviojoje atmintyje, veikia greičiau ir pateikia patogesnius įrankius duomenims išgauti, ir supaprastina visą procesą. Prieš pradedant darbą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



Tada per WinScp yra ikeliamas programinis Python kodas.



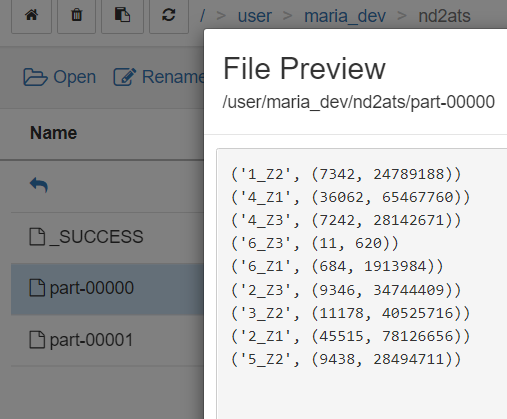
Programiniame kode yra nurodoma kur išsaugomi bus failai.



Programinis kodas yra paleidžiamas su spark-submit.



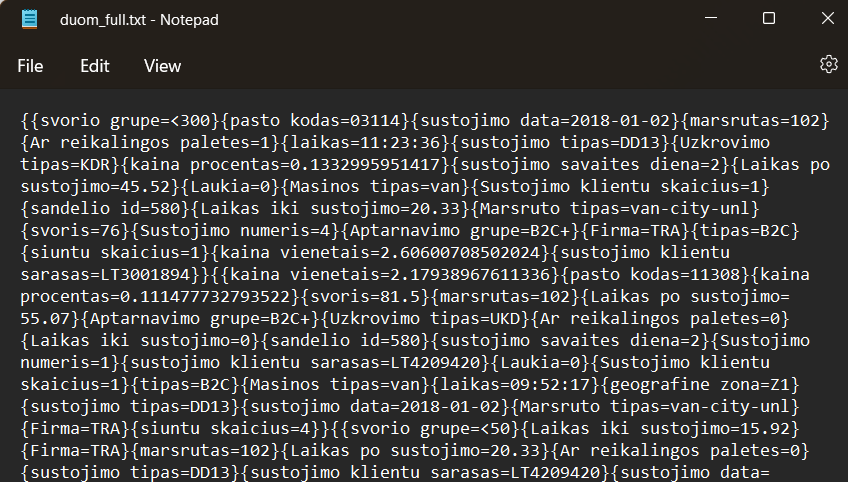
O gautą rezultatą galime matyti HDFS sistemoje.



# 

# Namų darbo užduotis

Namų 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”.



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

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 reiskia 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'):

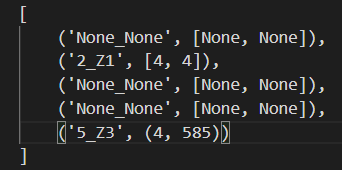
klientu\_skaicius = reiksme

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

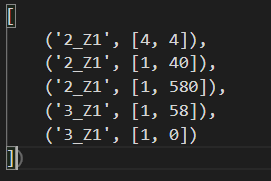
return(key, [siuntos, klientu\_skaicius])

B = A.map(MapF)

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 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 geografine zona kaip raktas (sujungiant šiuos parametrus per brūkšnelį), o reikšmė būtų siuntų ir klientų skaičius, pateikiamas masyvu. Surinkti duomenys atrodo taip:



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:



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, reiškias, mums iš kiekvieno mazgo gautus duomenis, pagal raktą, reikia sudėti, todėl naudosime funkciją reduceByKey, kuri kiekvieno rakto reikšmę sudės atitinkamai.

Pseudo kodas:

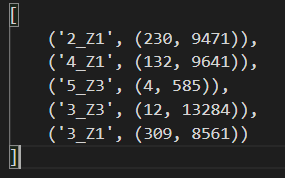
def red (a,b):

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

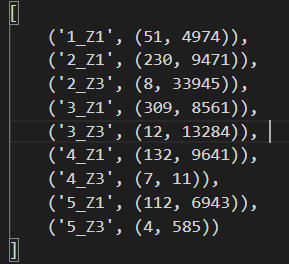
D = C.reduceByKey(red)

print(D.take(5))

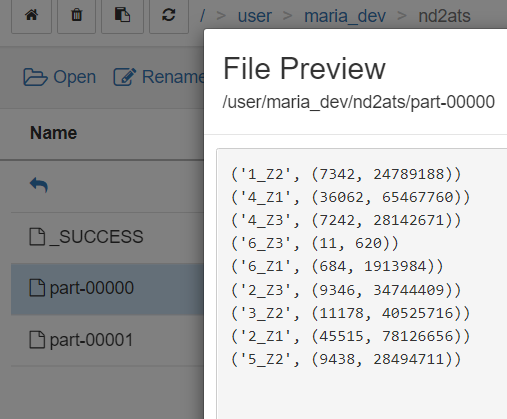
Reduce kodas tiesiog sudeda atitinkamas reikšmes. Kadangi mūsų reikšmė yra masyvas, mes kiekvienos reikšmės pirmąją reikšmę sudedame su kitos reikšmės pirmąją reikšmę ir tą patį darome su antrąja reikšme. Gautas rezultatas yra toks:



Dar paskutiniame etape galime šias reikšmes išrikiuoti, kad duomenys būtų tvarkingesni. Galutinis rezultatas:



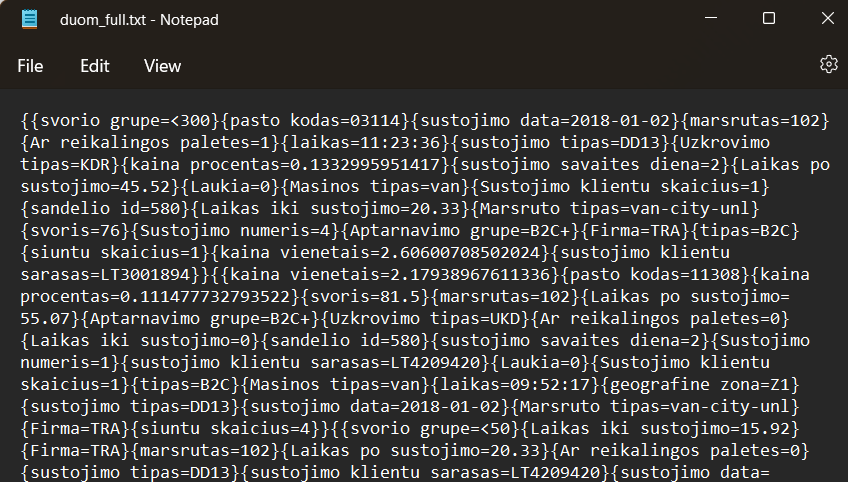
Su pilnais duomenimis paleistas kodas buvo išvestas Hortonworks HDP sistemoje:



# 

# Trečia užduotis

Namų 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”.



Namų 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.

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 maršruto, geografines zonos ir datos. Kadangi reikia rasti maršrutus, kurie aplanko daugiau nei viena geografine zona tą pačią dieną, reiškia raktas yra pagal du parametrus: maršrutą ir dieną, tai reiskia reikia isvesti tokį rakta kuris jungia siuos 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':

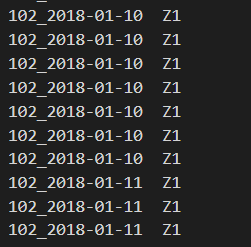
sustojimo\_data = value

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 kiekviena 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 negalima išvesti šitų duomenų ir iteruojama 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:



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:

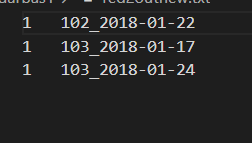
if len(current\_set) > 1:

emit(1, current\_key)

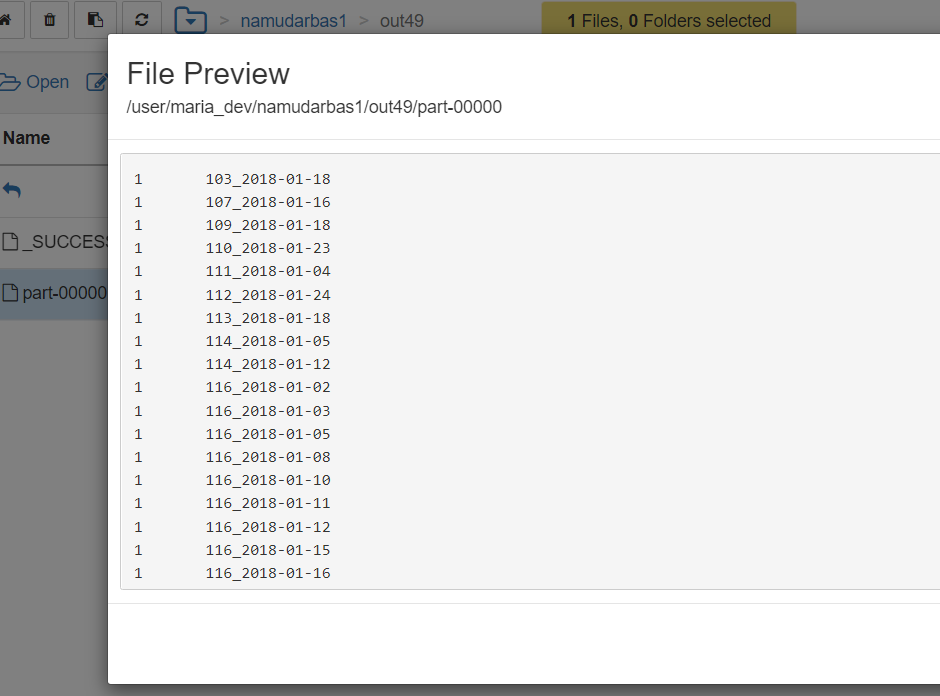
current\_set = set(zona)

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:



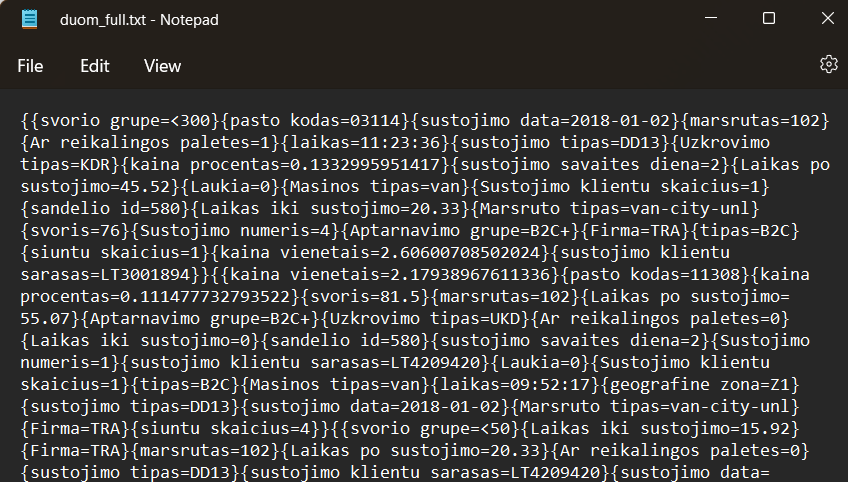
Dar papildomai buvo grąžintas vienetas, nes Reduce turi išvesti raktas reikšmė tipo duomenis, bet šiuo atveju jis nereikalingas. Taigi, naudojantis MapReduce buvo gauti maršrutai aplankę kelias geografines zonas tą pačią dieną. Su pilnais duomenimis paleistas kodas buvo išvestas Hortonworks HDP sistemoje:



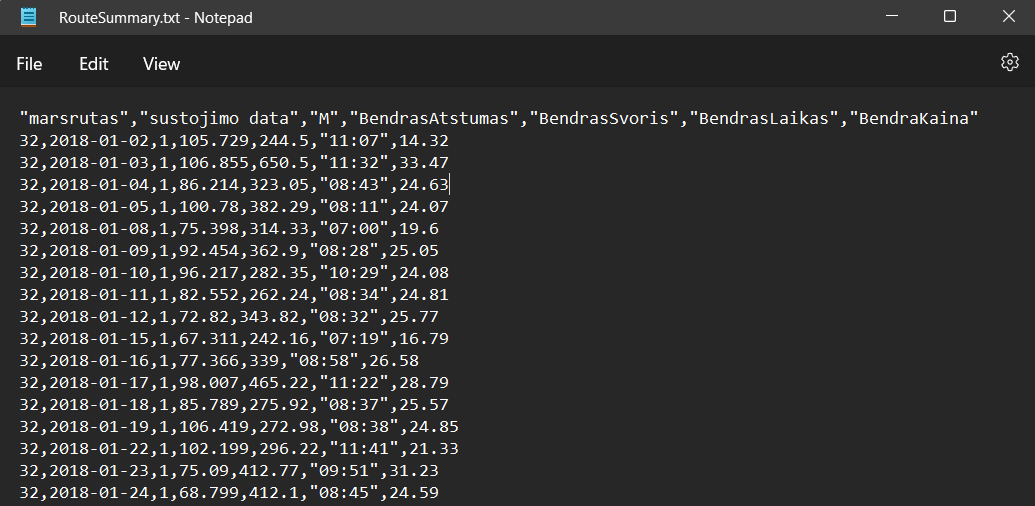
# 

# Namų darbo užduotis

Namų 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”.



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



Namų 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".

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 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 kiekviena 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', "outer")

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(maxIter=10, regParam=0.3, elasticNetParam=0.8,

featuresCol = "features", labelCol = "BendraKaina")

lrModel = lr.fit(vhouse\_df)

Nurodome tiesinės regresijos mokymo parametrus ir pradedame mokymą. Šią operaciją būtų galima atlikti naudojantis GPU arba perduoti operaciją į klasterį. 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)

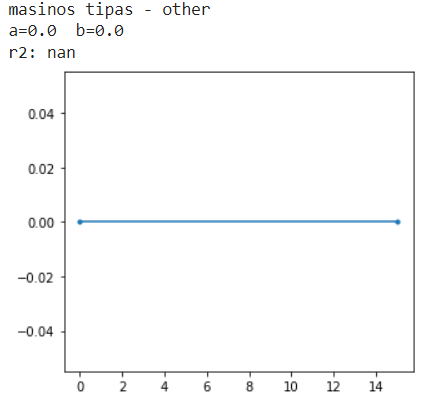
lineX = [min(x), max(x)]

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

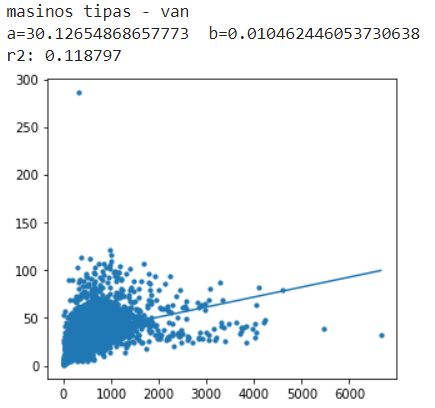
axes[2].plot(lineX, lineY)

plt.show()

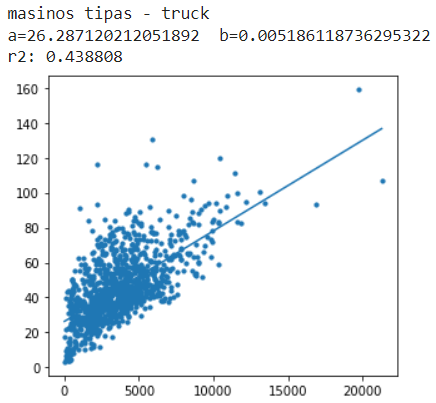
Gauti rezultatai rodo, kad su mašinos tipu “other” koeficientai yra nulis, o determinacijos koeficientas yra “nan”, kas reiškia, kad nebuvo duomenų, kad galėtų sukurti tiesinės regresijos modelį



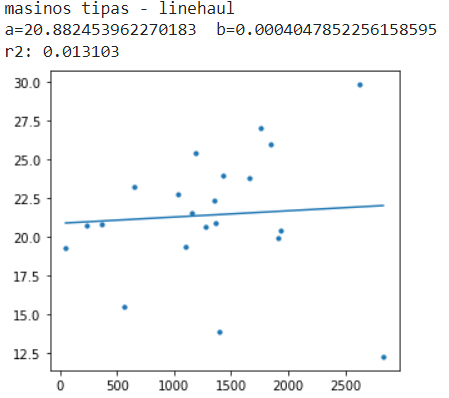
Toliau mašinos tipas “van”, determinacijos koeficientas rodo 0.11. Labai abejotina, kad yra tiesinė priklausomybė tarp svorio ir bendros kainos šiam mašinos tipui. Taip pat, tai patvirtina sklaidos diagrama, galima matyti, kad reikšmės yra išsibarsčiusios toli vienas nuo kito, todėl sunku teigti, kad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



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.



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.



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.

