EECS6893 Big Data Analytics HW2

Lijie Huang
lh3158

Oct 22, 2022

Part I
Save the ipynb file as a pdf format, containing your code and result.

[1]:
```
from pyspark.sql import SparkSession
import matplotlib.pyplot as plt
%matplotlib inline
import numpy as np
```

[2]:
```
!pyspark --version
```

```
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /___/ .__/\_,_/_/ /_/\_\   version 3.1.3
      /_/

Using Scala version 2.12.14, OpenJDK 64-Bit Server VM, 1.8.0_332
Branch HEAD
Compiled by user  on 2022-06-29T09:07:05Z
Revision ac3ee790108b118a30777ac494863af244584e75
Url https://bigdataoss-internal.googlesource.com/third_party/apache/spark
Type --help for more information.
```

1. Data loading

[3]:
```
#Read csv file to dataframe
#=====your code here==========
spark = SparkSession \
    .builder \
    .appName("Read CSV to DataFrame") \
    .config("spark.some.config.option", "some-value") \
    .getOrCreate()

schema = """`age` DOUBLE,
`workclass` STRING,
`fnlwgt` DOUBLE,
`education` STRING,
`education_num` DOUBLE,
`marital_status` STRING,
`occupation` STRING,
`relationship` STRING,
`race` STRING,
`sex` STRING,
`capital_gain` DOUBLE,
`capital_loss` DOUBLE,
`hours_per_week` DOUBLE,
`native_country` STRING,
`income` STRING"""

data = spark.read.csv("/input/adult.csv", schema = schema)


#===============================
data.show(3)
```

```
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use
setLogLevel(newLevel).
22/10/22 00:05:44 INFO org.apache.spark.SparkEnv: Registering MapOutputTracker
22/10/22 00:05:44 INFO org.apache.spark.SparkEnv: Registering BlockManagerMaster
22/10/22 00:05:44 INFO org.apache.spark.SparkEnv: Registering
BlockManagerMasterHeartbeat
22/10/22 00:05:45 INFO org.apache.spark.SparkEnv: Registering
OutputCommitCoordinator
[Stage 0:>                                                          (0 + 1) / 1]

+----+-----------------+--------+----------+-------------+-------------------+--
----------------+--------------+------+-----+------------+------------+---------
-----+--------------+------+
| age|        workclass|  fnlwgt| education|education_num|     marital_status|
occupation|  relationship|  race|
sex|capital_gain|capital_loss|hours_per_week|native_country|income|
+----+-----------------+--------+----------+-------------+-------------------+--
----------------+--------------+------+-----+------------+------------+---------
-----+--------------+------+
|39.0|        State-gov| 77516.0| Bachelors|         13.0|      Never-married|
Adm-clerical| Not-in-family| White| Male|      2174.0|         0.0|
40.0| United-States| <=50K|
|50.0| Self-emp-not-inc| 83311.0| Bachelors|         13.0| Married-civ-spouse|
Exec-managerial|       Husband| White| Male|         0.0|         0.0|
13.0| United-States| <=50K|
|38.0|          Private|215646.0|   HS-grad|          9.0|           Divorced|
Handlers-cleaners| Not-in-family| White| Male|         0.0|         0.0|
40.0| United-States| <=50K|
+----+-----------------+--------+----------+-------------+-------------------+--
----------------+--------------+------+-----+------------+------------+---------
-----+--------------+------+
only showing top 3 rows
```

[4]:
```python
from functools import reduce
```

[5]:
```python
data.printSchema()
data.show(2)
df = data
dataset = df
```

```
root
 |-- age: double (nullable = true)
 |-- workclass: string (nullable = true)
 |-- fnlwgt: double (nullable = true)
 |-- education: string (nullable = true)
 |-- education_num: double (nullable = true)
 |-- marital_status: string (nullable = true)
 |-- occupation: string (nullable = true)
 |-- relationship: string (nullable = true)
 |-- race: string (nullable = true)
 |-- sex: string (nullable = true)
 |-- capital_gain: double (nullable = true)
 |-- capital_loss: double (nullable = true)
 |-- hours_per_week: double (nullable = true)
 |-- native_country: string (nullable = true)
 |-- income: string (nullable = true)

[Stage 1:>                                                          (0 + 1) / 1]

+----+----------------+-------+----------+-------------+-------------------+---
-------------+--------------+------+-----+------------+------------+------------
--+--------------+------+
| age|       workclass| fnlwgt| education|education_num|     marital_status|
occupation|  relationship|  race|
sex|capital_gain|capital_loss|hours_per_week|native_country|income|
+----+----------------+-------+----------+-------------+-------------------+---
-------------+--------------+------+-----+------------+------------+------------
--+--------------+------+
|39.0|       State-gov|77516.0| Bachelors|         13.0|      Never-married|
Adm-clerical| Not-in-family| White| Male|      2174.0|         0.0|
40.0| United-States| <=50K|
|50.0| Self-emp-not-inc|83311.0| Bachelors|         13.0| Married-civ-spouse|
Exec-managerial|       Husband| White| Male|         0.0|         0.0|
13.0| United-States| <=50K|
+----+----------------+-------+----------+-------------+-------------------+---
-------------+--------------+------+-----+------------+------------+------------
--+--------------+------+
only showing top 2 rows
```

2. Data preprocessing

[6]:

```
from pyspark.ml import Pipeline
from pyspark.ml.feature import OneHotEncoder, StringIndexer, VectorAssembler
```

[7]:

```
#stages in our Pipeline
stages = []
categoricalColumns =␣
↪["workclass","education","marital_status","occupation","relationship","race","sex","native_
```

[8]:

```
for categoricalCol in categoricalColumns:
    # Category Indexing with StringIndexer
    stringIndexer = StringIndexer(inputCol=categoricalCol,␣
↪outputCol=categoricalCol + "Index")
    # Use OneHotEncoder to convert categorical variables into binary␣
↪SparseVectors
    encoder = OneHotEncoder(inputCols=[stringIndexer.getOutputCol()],␣
↪outputCols=[categoricalCol + "classVec"])
    # Add stages.  These are not run here, but will run all at once later on.
    stages += [stringIndexer, encoder]
```

[9]:

```
# Convert label into label indices using the StringIndexer
label_stringIdx = StringIndexer(inputCol="income", outputCol="label")
stages += [label_stringIdx]
```

[10]:

```
# Transform all features into a vector using VectorAssembler
numericCols = ["age", "fnlwgt", "education_num", "capital_gain",␣
↪"capital_loss", "hours_per_week"]
assemblerInputs = [c + "classVec" for c in categoricalColumns] + numericCols
assembler = VectorAssembler(inputCols=assemblerInputs, outputCol="features")
stages += [assembler]
```

[11]:

```
pipeline = Pipeline(stages=stages)
pipelineModel = pipeline.fit(dataset)
preppedDataDF = pipelineModel.transform(dataset)
```

[12]:

```
preppedDataDF.take(3)
```

```
22/10/22 00:06:43 WARN org.apache.spark.sql.catalyst.util.package: Truncated the
string representation of a plan since it was too large. This behavior can be
adjusted by setting 'spark.sql.debug.maxToStringFields'.
```

```
[12]: [Row(age=39.0, workclass=' State-gov', fnlwgt=77516.0, education=' Bachelors',
      education_num=13.0, marital_status=' Never-married', occupation=' Adm-clerical',
      relationship=' Not-in-family', race=' White', sex=' Male', capital_gain=2174.0,
      capital_loss=0.0, hours_per_week=40.0, native_country=' United-States', income='
      <=50K', workclassIndex=4.0, workclassclassVec=SparseVector(8, {4: 1.0}),
      educationIndex=2.0, educationclassVec=SparseVector(15, {2: 1.0}),
      marital_statusIndex=1.0, marital_statusclassVec=SparseVector(6, {1: 1.0}),
      occupationIndex=3.0, occupationclassVec=SparseVector(14, {3: 1.0}),
      relationshipIndex=1.0, relationshipclassVec=SparseVector(5, {1: 1.0}),
      raceIndex=0.0, raceclassVec=SparseVector(4, {0: 1.0}), sexIndex=0.0,
      sexclassVec=SparseVector(1, {0: 1.0}), native_countryIndex=0.0,
      native_countryclassVec=SparseVector(41, {0: 1.0}), label=0.0,
      features=SparseVector(100, {4: 1.0, 10: 1.0, 24: 1.0, 32: 1.0, 44: 1.0, 48: 1.0,
      52: 1.0, 53: 1.0, 94: 39.0, 95: 77516.0, 96: 13.0, 97: 2174.0, 99: 40.0})),
       Row(age=50.0, workclass=' Self-emp-not-inc', fnlwgt=83311.0, education='
      Bachelors', education_num=13.0, marital_status=' Married-civ-spouse',
      occupation=' Exec-managerial', relationship=' Husband', race=' White', sex='
      Male', capital_gain=0.0, capital_loss=0.0, hours_per_week=13.0, native_country='
      United-States', income=' <=50K', workclassIndex=1.0,
      workclassclassVec=SparseVector(8, {1: 1.0}), educationIndex=2.0,
      educationclassVec=SparseVector(15, {2: 1.0}), marital_statusIndex=0.0,
      marital_statusclassVec=SparseVector(6, {0: 1.0}), occupationIndex=2.0,
      occupationclassVec=SparseVector(14, {2: 1.0}), relationshipIndex=0.0,
      relationshipclassVec=SparseVector(5, {0: 1.0}), raceIndex=0.0,
      raceclassVec=SparseVector(4, {0: 1.0}), sexIndex=0.0,
      sexclassVec=SparseVector(1, {0: 1.0}), native_countryIndex=0.0,
      native_countryclassVec=SparseVector(41, {0: 1.0}), label=0.0,
      features=SparseVector(100, {1: 1.0, 10: 1.0, 23: 1.0, 31: 1.0, 43: 1.0, 48: 1.0,
      52: 1.0, 53: 1.0, 94: 50.0, 95: 83311.0, 96: 13.0, 99: 13.0})),
       Row(age=38.0, workclass=' Private', fnlwgt=215646.0, education=' HS-grad',
      education_num=9.0, marital_status=' Divorced', occupation=' Handlers-cleaners',
      relationship=' Not-in-family', race=' White', sex=' Male', capital_gain=0.0,
      capital_loss=0.0, hours_per_week=40.0, native_country=' United-States', income='
      <=50K', workclassIndex=0.0, workclassclassVec=SparseVector(8, {0: 1.0}),
      educationIndex=0.0, educationclassVec=SparseVector(15, {0: 1.0}),
      marital_statusIndex=2.0, marital_statusclassVec=SparseVector(6, {2: 1.0}),
      occupationIndex=9.0, occupationclassVec=SparseVector(14, {9: 1.0}),
      relationshipIndex=1.0, relationshipclassVec=SparseVector(5, {1: 1.0}),
      raceIndex=0.0, raceclassVec=SparseVector(4, {0: 1.0}), sexIndex=0.0,
      sexclassVec=SparseVector(1, {0: 1.0}), native_countryIndex=0.0,
      native_countryclassVec=SparseVector(41, {0: 1.0}), label=0.0,
      features=SparseVector(100, {0: 1.0, 8: 1.0, 25: 1.0, 38: 1.0, 44: 1.0, 48: 1.0,
      52: 1.0, 53: 1.0, 94: 38.0, 95: 215646.0, 96: 9.0, 99: 40.0}))]
```

```
[13]: # Keep relevant columns
      cols = dataset.columns
      selectedcols = ["label", "features"] + cols
```

```
dataset = preppedDataDF.select(selectedcols)
display(dataset)
```

```
DataFrame[label: double, features: vector, age: double, workclass: string,␣
 ↪fnlwgt: double, education: string, education_num: double, marital_status:␣
 ↪string, occupation: string, relationship: string, race: string, sex: string,␣
 ↪capital_gain: double, capital_loss: double, hours_per_week: double,␣
 ↪native_country: string, income: string]
```

[14]:

```
### Randomly split data into training and test sets. set seed for␣
 ↪reproducibility
#=====your code here==========

trainingData, testData = dataset.randomSplit([0.8, 0.2], seed=42)

#===============================
print(trainingData.count())
print(testData.count())
```

```
26076

[Stage 33:>                                                         (0 + 1) / 1]

6485
```

3. Modeling

[22]:

```
modelNames = []
accuracies = []
```

[23]:

```
# Fit model to prepped data

#LogisticRegression model, maxIter=10
#=====your code here==========

from pyspark.ml.evaluation import MulticlassClassificationEvaluator
from pyspark.ml.classification import LogisticRegression

lr = LogisticRegression(
    maxIter = 10,
    regParam = 0.05,
    labelCol="label",
    featuresCol="features"
)
lrModel = lr.fit(trainingData)
```

```python
#===============================

# select example rows to display.
predictions = lrModel.transform(testData)
predictions.show(5)

# compute accuracy on the test set
evaluator = MulticlassClassificationEvaluator(labelCol="label",␣
 ↪predictionCol="prediction", metricName="accuracy")
accuracy = evaluator.evaluate(predictions)
print("Test set accuracy = " + str(accuracy))

modelNames.append("Linear Regression")
accuracies.append(accuracy)
```

```
+-----+--------------------+----+---------+--------+---------+-------------+----
---------------+---------------+------------+------+-----+------------+---------
---+--------------+--------------+------+--------------------+------------------
--+----------+
|label|            features| age|workclass|  fnlwgt|education|education_num|
marital_status|     occupation|relationship|  race|
sex|capital_gain|capital_loss|hours_per_week|native_country|income|
rawPrediction|         probability|prediction|
+-----+--------------------+----+---------+--------+---------+-------------+----
---------------+---------------+------------+------+-----+------------+---------
---+--------------+--------------+------+--------------------+------------------
--+----------+
|  0.0|(100,[0,8,23,29,4…|32.0| Private|130304.0| HS-grad|          9.0|
Married-civ-spouse| Prof-specialty|     Husband| White| Male|          0.0|
1485.0|          48.0| United-States|
<=50K|[-0.0983474588228…|[0.47543293361429…|       1.0|
|  0.0|(100,[0,8,23,29,4…|29.0| Private| 40295.0| HS-grad|          9.0|
Married-civ-spouse| Prof-specialty|     Husband| White| Male|          0.0|
0.0|          40.0| United-States|
<=50K|[0.58889836353206…|[0.64311233923242…|       0.0|
|  0.0|(100,[0,8,23,29,4…|31.0| Private| 62374.0| HS-grad|          9.0|
Married-civ-spouse| Prof-specialty|     Husband| White| Male|          0.0|
0.0|          50.0| United-States|
<=50K|[0.45419914487542…|[0.61163715317751…|       0.0|
|  0.0|(100,[0,8,23,29,4…|37.0| Private|282951.0| HS-grad|          9.0|
Married-civ-spouse| Prof-specialty|     Husband| White| Male|          0.0|
0.0|          40.0| United-States|
<=50K|[0.53440082959582…|[0.63050894939002…|       0.0|
|  0.0|(100,[0,8,23,29,4…|50.0| Private| 81548.0| HS-grad|          9.0|
Married-civ-spouse| Prof-specialty|     Husband| White| Male|          0.0|
0.0|          40.0| United-States|
<=50K|[0.39400172043404…|[0.59724565971324…|       0.0|
+-----+--------------------+----+---------+--------+---------+-------------+----
---------------+---------------+------------+------+-----+------------+---------
---+--------------+--------------+------+--------------------+------------------
--+----------+
only showing top 5 rows

Test set accuracy = 0.8425597532767926
```

[24]:

```
#Random Forest
from pyspark.ml.classification import RandomForestClassifier
rf = RandomForestClassifier(labelCol = "label", featuresCol = "features")
rfModel = rf.fit(trainingData)
predictions = rfModel.transform(testData)
# compute accuracy on the test set
evaluator = MulticlassClassificationEvaluator(labelCol="label",␣
 ↪predictionCol="prediction", metricName="accuracy")
accuracy = evaluator.evaluate(predictions)
print("Test set accuracy = " + str(accuracy))

modelNames.append("Random Forest")
accuracies.append(accuracy)
```

```
Test set accuracy = 0.823284502698535
```

[26]:

```
#NaiveBayes
#=====your code here==========
from pyspark.ml.classification import NaiveBayes
nb = NaiveBayes(labelCol = "label", featuresCol = "features")
nbModel = nb.fit(trainingData)
#==============================
# select example rows to display.
predictions = nbModel.transform(testData)
predictions.show(5)
# compute accuracy on the test set
evaluator = MulticlassClassificationEvaluator(labelCol="label",␣
 ↪predictionCol="prediction", metricName="accuracy")
accuracy = evaluator.evaluate(predictions)
print("Test set accuracy = " + str(accuracy))

modelNames.append("Naive Bayes")
accuracies.append(accuracy)
```

```
+-----+--------------------+----+---------+--------+---------+-------------+----
---------------+---------------+------------+------+-----+------------+---------
---+--------------+--------------+------+--------------------+-----------+------
----+
|label|            features| age|workclass|  fnlwgt|education|education_num|
marital_status|     occupation|relationship|  race|
sex|capital_gain|capital_loss|hours_per_week|native_country|income|
rawPrediction|probability|prediction|
+-----+--------------------+----+---------+--------+---------+-------------+----
---------------+---------------+------------+------+-----+------------+---------
---+--------------+--------------+------+--------------------+-----------+------
----+
|  0.0|(100,[0,8,23,29,4…|32.0|  Private|130304.0|  HS-grad|          9.0|
Married-civ-spouse| Prof-specialty|     Husband| White| Male|         0.0|
1485.0|          48.0| United-States| <=50K|[-13217.142541473…|  [1.0,0.0]|
0.0|
|  0.0|(100,[0,8,23,29,4…|29.0|  Private| 40295.0|  HS-grad|          9.0|
Married-civ-spouse| Prof-specialty|     Husband| White| Male|         0.0|
0.0|          40.0| United-States| <=50K|[-843.18372583750…|  [1.0,0.0]|
0.0|
|  0.0|(100,[0,8,23,29,4…|31.0|  Private| 62374.0|  HS-grad|          9.0|
Married-civ-spouse| Prof-specialty|     Husband| White| Male|         0.0|
0.0|          50.0| United-States| <=50K|[-979.03449430897…|  [1.0,0.0]|
0.0|
|  0.0|(100,[0,8,23,29,4…|37.0|  Private|282951.0|  HS-grad|          9.0|
Married-civ-spouse| Prof-specialty|     Husband| White| Male|         0.0|
0.0|          40.0| United-States| <=50K|[-1282.3632603125…|  [1.0,0.0]|
0.0|
|  0.0|(100,[0,8,23,29,4…|50.0|  Private| 81548.0|  HS-grad|          9.0|
Married-civ-spouse| Prof-specialty|     Husband| White| Male|         0.0|
0.0|          40.0| United-States| <=50K|[-1085.8675897020…|  [1.0,0.0]|
0.0|
+-----+--------------------+----+---------+--------+---------+-------------+----
---------------+---------------+------------+------+-----+------------+---------
---+--------------+--------------+------+--------------------+-----------+------
----+
only showing top 5 rows

Test set accuracy = 0.7822667694680031
```

[27]:

```python
#Decision Tree
from pyspark.ml.classification import DecisionTreeClassifier
dt = DecisionTreeClassifier(labelCol = "label", featuresCol = "features")
dtModel = dt.fit(trainingData)
```

```
predictions = dtModel.transform(testData)
# compute accuracy on the test set
evaluator = MulticlassClassificationEvaluator(labelCol="label",␣
 ↪predictionCol="prediction", metricName="accuracy")
accuracy = evaluator.evaluate(predictions)
print("Test set accuracy = " + str(accuracy))

modelNames.append("Decision Tree")
accuracies.append(accuracy)
```

```
Test set accuracy = 0.8385505011565151
```

[28]:

```
#Gradient Boosting Trees
from pyspark.ml.classification import GBTClassifier
gbt = GBTClassifier(labelCol = "label", featuresCol = "features")
gbtModel = gbt.fit(trainingData)
predictions = gbtModel.transform(testData)
# compute accuracy on the test set
evaluator = MulticlassClassificationEvaluator(labelCol="label",␣
 ↪predictionCol="prediction", metricName="accuracy")
accuracy = evaluator.evaluate(predictions)
print("Test set accuracy = " + str(accuracy))

modelNames.append("Gradient Boosting Trees")
accuracies.append(accuracy)
```

```
[Stage 767:>                                                    (0 + 1) / 1]

Test set accuracy = 0.8585967617579029
```

[29]:

```
# Multi-layer Perceptron

from pyspark.ml.classification import MultilayerPerceptronClassifier
from pyspark.ml.evaluation import MulticlassClassificationEvaluator

layers = [100, 32, 2]

# create the trainer and set its parameters
trainer = MultilayerPerceptronClassifier(maxIter=100, layers=layers,␣
 ↪blockSize=1, seed=100)

# train the model
mlpModel = trainer.fit(trainingData)
```

```
# compute accuracy on the test set
predictions = mlpModel.transform(testData)

evaluator = MulticlassClassificationEvaluator(labelCol="label",␣
 ↪predictionCol="prediction", metricName="accuracy")
accuracy = evaluator.evaluate(predictions)
print("Test set accuracy = " + str(accuracy))

modelNames.append("Multilayer Perceptron Classifier")
accuracies.append(accuracy)
```

```
[Stage 953:>                                                          (0 + 1) / 1]

Test set accuracy = 0.7941403238242097
```

[31]:

```
# Linear Support Vector Machine
from pyspark.ml.classification import LinearSVC
lsvc = LinearSVC(labelCol = "label", featuresCol = "features")
lsvcModel = lsvc.fit(trainingData)
predictions = lsvcModel.transform(testData)
# compute accuracy on the test set
evaluator = MulticlassClassificationEvaluator(labelCol="label",␣
 ↪predictionCol="prediction", metricName="accuracy")
accuracy = evaluator.evaluate(predictions)
print("Test set accuracy = " + str(accuracy))

modelNames.append("Linear Support Vector Machine")
accuracies.append(accuracy)
```

```
[Stage 1467:>                                                         (0 + 1) / 1]

Test set accuracy = 0.845952197378566
```

[32]:

```
# One-vs-Rest
from pyspark.ml.classification import OneVsRest
ovr = OneVsRest(classifier = lr)
ovrModel = ovr.fit(trainingData)
predictions = ovrModel.transform(testData)
# compute accuracy on the test set
evaluator = MulticlassClassificationEvaluator(labelCol="label",␣
 ↪predictionCol="prediction", metricName="accuracy")
accuracy = evaluator.evaluate(predictions)
print("Test set accuracy = " + str(accuracy))
```

```
modelNames.append("One-vs-Rest")
accuracies.append(accuracy)
```

```
[Stage 1498:>                                                       (0 + 1) / 1]

Test set accuracy = 0.8425597532767926
```

4. Comparison and analysis

[33]:

```
# Rank models according to Test set accuracy
#=====your code here==========

zip_data = dict(zip(modelNames, accuracies))
res = sorted(zip_data.items(), key = lambda x: x[1], reverse=True)

for item in res:
    print("Method: %s; Accuracy: %f" % (item[0], item[1]))

#================================
```

```
Method: Gradient Boosting Trees; Accuracy: 0.858597
Method: Linear Support Vector Machine; Accuracy: 0.845952
Method: Linear Regression; Accuracy: 0.842560
Method: One-vs-Rest; Accuracy: 0.842560
Method: Decision Tree; Accuracy: 0.838551
Method: Random Forest; Accuracy: 0.823285
Method: Multilayer Perceptron Classifier; Accuracy: 0.794140
Method: Naive Bayes; Accuracy: 0.782267
```

your analysis

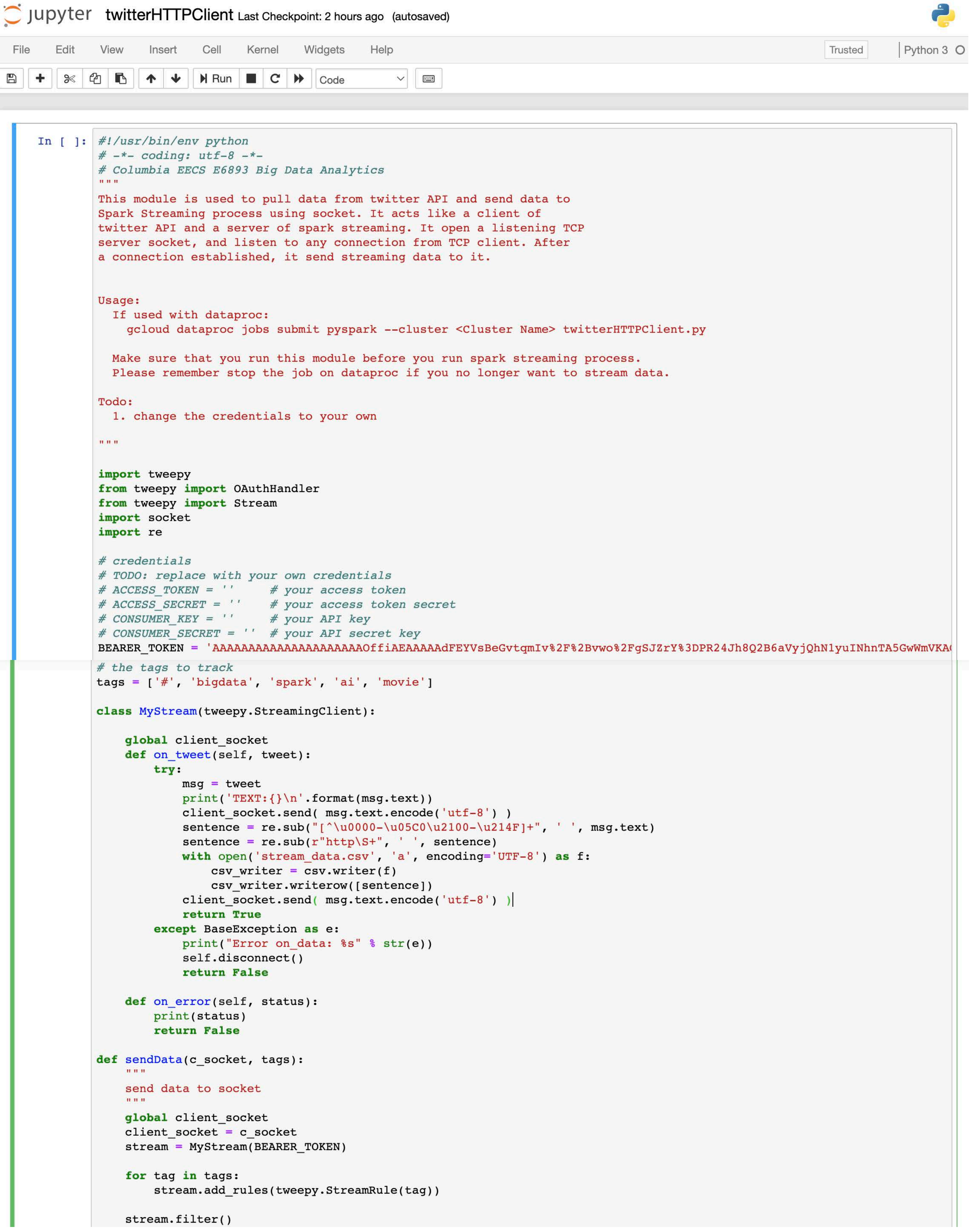
Gradient Boosting Tree reaches the highest accuracy. The Naive Beyes has the lowest prediction accuracy.

for Multi-layer Perception Model, if a deeper and wider Perception model is used, the accuracy would increase.

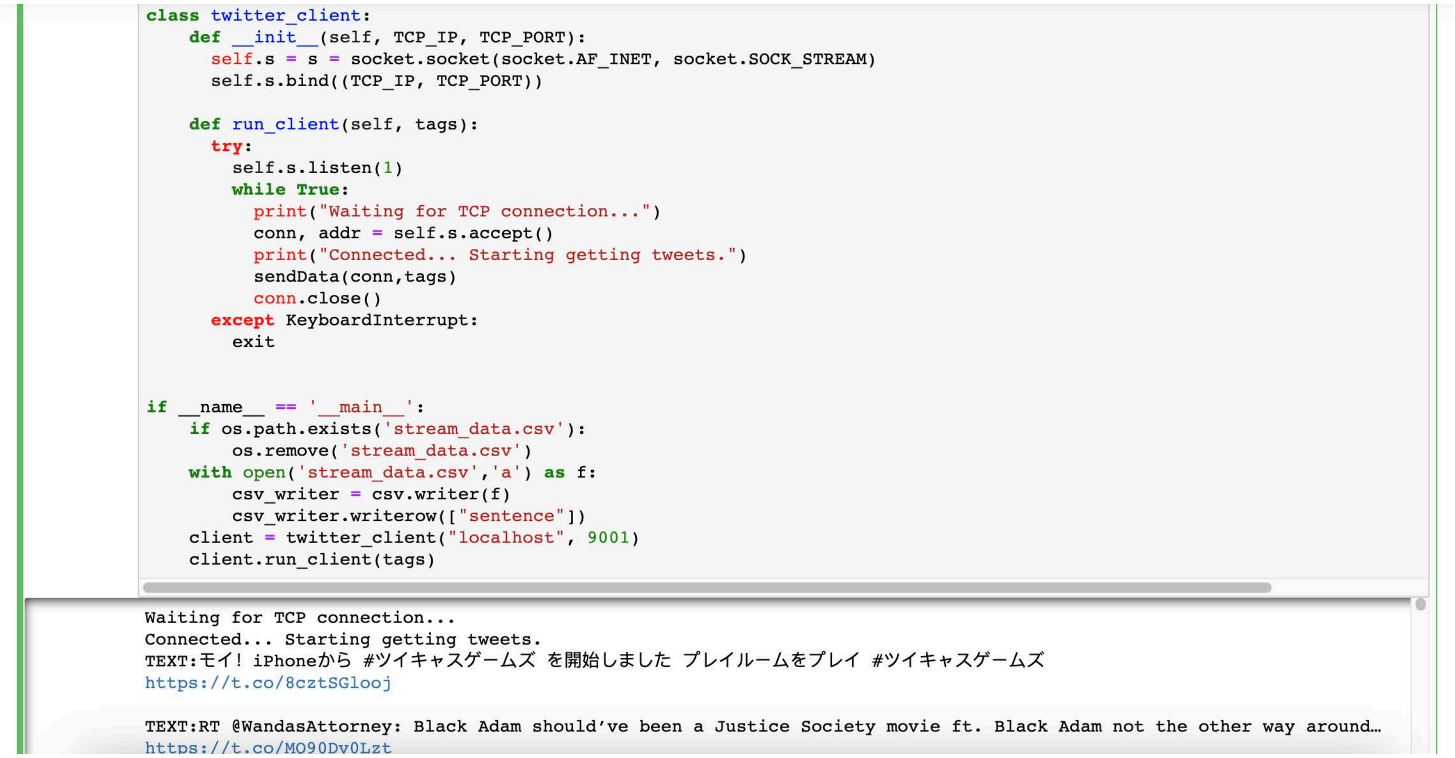
Part II

(Task1-3) A report includes:

a. Screenshot of your code to do all the tasks.

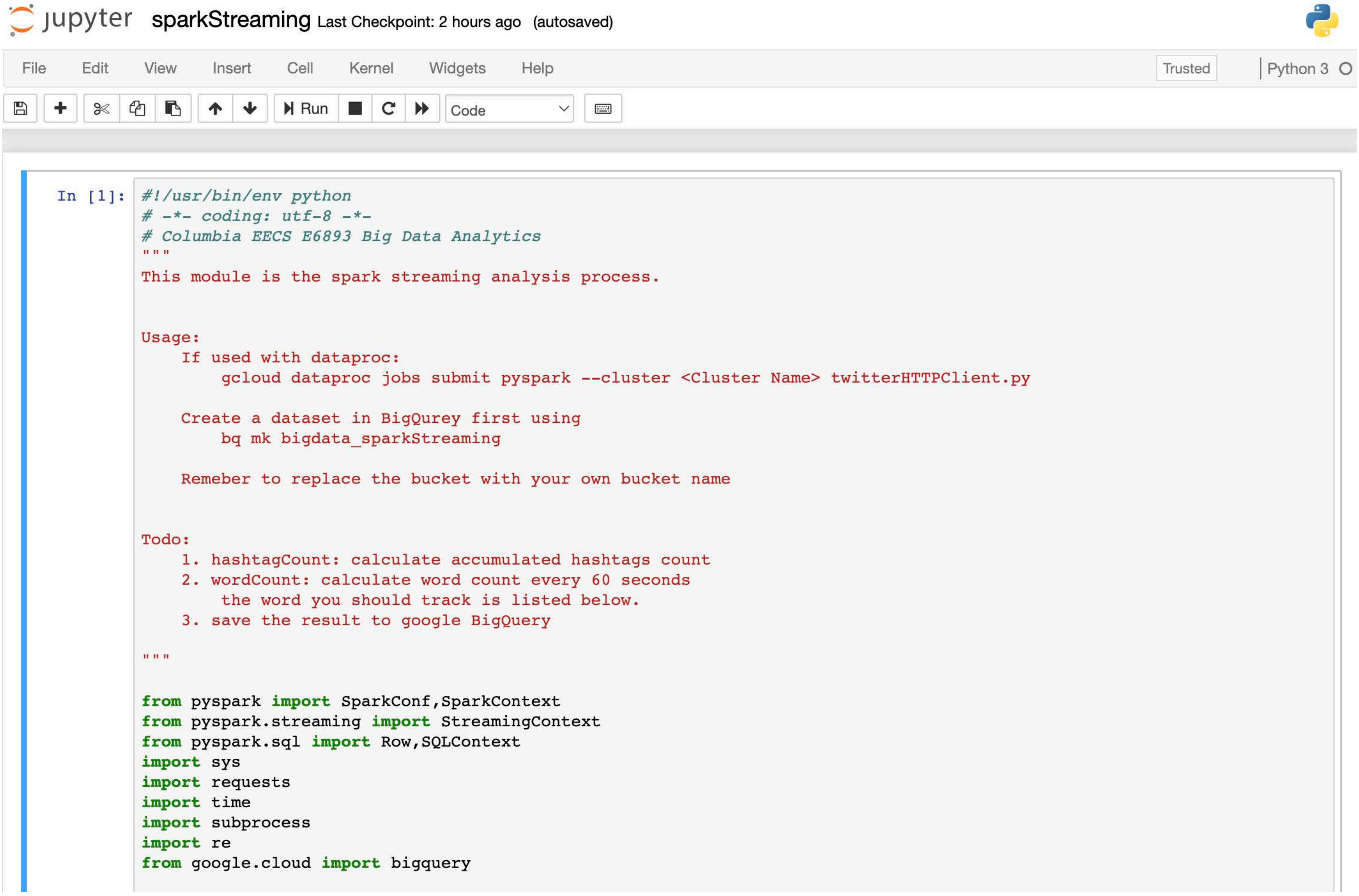


```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
# Columbia EECS E6893 Big Data Analytics
"""
This module is used to pull data from twitter API and send data to
Spark Streaming process using socket. It acts like a client of
twitter API and a server of spark streaming. It open a listening TCP
server socket, and listen to any connection from TCP client. After
a connection established, it send streaming data to it.


Usage:
  If used with dataproc:
    gcloud dataproc jobs submit pyspark --cluster <Cluster Name> twitterHTTPClient.py

  Make sure that you run this module before you run spark streaming process.
  Please remember stop the job on dataproc if you no longer want to stream data.

Todo:
  1. change the credentials to your own

"""

import tweepy
from tweepy import OAuthHandler
from tweepy import Stream
import socket
import re

# credentials
# TODO: replace with your own credentials
# ACCESS_TOKEN = ''      # your access token
# ACCESS_SECRET = ''     # your access token secret
# CONSUMER_KEY = ''      # your API key
# CONSUMER_SECRET = ''   # your API secret key
BEARER_TOKEN = 'AAAAAAAAAAAAAAAAAAAAAOffiAEAAAAAdFEYVsBeGvtqmIv%2F%2Bvwo%2FgSJZrY%3DPR24Jh8Q2B6aVyjQhN1yuINhnTA5GwWmVKA
# the tags to track
tags = ['#', 'bigdata', 'spark', 'ai', 'movie']

class MyStream(tweepy.StreamingClient):

    global client_socket
    def on_tweet(self, tweet):
        try:
            msg = tweet
            print('TEXT:{}\n'.format(msg.text))
            client_socket.send( msg.text.encode('utf-8') )
            sentence = re.sub("[^\u0000-\u05C0\u2100-\u214F]+", ' ', msg.text)
            sentence = re.sub(r"http\S+", ' ', sentence)
            with open('stream_data.csv', 'a', encoding='UTF-8') as f:
                csv_writer = csv.writer(f)
                csv_writer.writerow([sentence])
            client_socket.send( msg.text.encode('utf-8') )
            return True
        except BaseException as e:
            print("Error on_data: %s" % str(e))
            self.disconnect()
            return False

    def on_error(self, status):
        print(status)
        return False

def sendData(c_socket, tags):
    """
    send data to socket
    """
    global client_socket
    client_socket = c_socket
    stream = MyStream(BEARER_TOKEN)

    for tag in tags:
        stream.add_rules(tweepy.StreamRule(tag))

    stream.filter()
```



```
class twitter_client:
    def __init__(self, TCP_IP, TCP_PORT):
      self.s = s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
      self.s.bind((TCP_IP, TCP_PORT))

    def run_client(self, tags):
      try:
        self.s.listen(1)
        while True:
          print("Waiting for TCP connection...")
          conn, addr = self.s.accept()
          print("Connected... Starting getting tweets.")
          sendData(conn,tags)
          conn.close()
      except KeyboardInterrupt:
        exit


if __name__ == '__main__':
    if os.path.exists('stream_data.csv'):
        os.remove('stream_data.csv')
    with open('stream_data.csv','a') as f:
        csv_writer = csv.writer(f)
        csv_writer.writerow(["sentence"])
    client = twitter_client("localhost", 9001)
    client.run_client(tags)
```

```
Waiting for TCP connection...
Connected... Starting getting tweets.
TEXT:モイ! iPhoneから #ツイキャスゲームズ を開始しました プレイルームをプレイ #ツイキャスゲームズ
https://t.co/8cztSGlooj

TEXT:RT @WandasAttorney: Black Adam should’ve been a Justice Society movie ft. Black Adam not the other way around…
https://t.co/MO90Dv0Lzt
```



```
In [1]: #!/usr/bin/env python
        # -*- coding: utf-8 -*-
        # Columbia EECS E6893 Big Data Analytics
        """
        This module is the spark streaming analysis process.


        Usage:
            If used with dataproc:
                gcloud dataproc jobs submit pyspark --cluster <Cluster Name> twitterHTTPClient.py

            Create a dataset in BigQurey first using
                bq mk bigdata_sparkStreaming

            Remeber to replace the bucket with your own bucket name


        Todo:
            1. hashtagCount: calculate accumulated hashtags count
            2. wordCount: calculate word count every 60 seconds
                the word you should track is listed below.
            3. save the result to google BigQuery

        """

        from pyspark import SparkConf,SparkContext
        from pyspark.streaming import StreamingContext
        from pyspark.sql import Row,SQLContext
        import sys
        import requests
        import time
        import subprocess
        import re
        from google.cloud import bigquery
```

```
# global variables
bucket = "twitter_streaming_bucket"    # TODO : replace with your own bucket name
output_directory_hashtags = 'gs://{}/hadoop/tmp/bigquery/pyspark_output/hashtagsCount'.format(bucket)
output_directory_wordcount = 'gs://{}/hadoop/tmp/bigquery/pyspark_output/wordcount'.format(bucket)

# output table and columns name
output_dataset = 'twitter_data'                      #the name of your dataset in BigQuery
output_table_hashtags = 'hashtags'
columns_name_hashtags = ['hashtags', 'count']
output_table_wordcount = 'wordcount'
columns_name_wordcount = ['word', 'count', 'time']

# parameter
IP = 'localhost'    # ip port
PORT = 9001       # port

STREAMTIME = 600          # time that the streaming process runs 600

WORD = ['data', 'spark', 'ai', 'movie', 'good']     #the words you should filter and do word count

# Helper functions
def saveToStorage(rdd, output_directory, columns_name, mode):
    """
    Save each RDD in this DStream to google storage
    Args:
        rdd: input rdd
        output_directory: output directory in google storage
        columns_name: columns name of dataframe
        mode: mode = "overwirte", overwirte the file
              mode = "append", append data to the end of file
    """
    if not rdd.isEmpty():
        (rdd.toDF( columns_name ) \
        .write.save(output_directory, format="json", mode=mode))
```

```
def saveToBigQuery(sc, output_dataset, output_table, directory):
    """
    Put temp streaming json files in google storage to google BigQuery
    and clean the output files in google storage
    """
    files = directory + '/part-*'
    subprocess.check_call(
        'bq load --source_format NEWLINE_DELIMITED_JSON '
        '--replace '
        '--autodetect '
        '{dataset}.{table} {files}'.format(
            dataset=output_dataset, table=output_table, files=files
        ).split())
    output_path = sc._jvm.org.apache.hadoop.fs.Path(directory)
    output_path.getFileSystem(sc._jsc.hadoopConfiguration()).delete(
        output_path, True)


def hashtagCount(words):
    """
    Calculate the accumulated hashtags count sum from the beginning of the stream
    and sort it by descending order of the count.
    Ignore case sensitivity when counting the hashtags:
        "#Ab" and "#ab" is considered to be a same hashtag
    You have to:
    1. Filter out the word that is hashtags.
       Hashtag usually start with "#" and followed by a serious of alphanumeric
    2. map (hashtag) to (hashtag, 1)
    3. sum the count of current DStream state and previous state
    4. transform unordered DStream to a ordered Dstream
    Hints:
        you may use regular expression to filter the words
        You can take a look at updateStateByKey and transform transformations
    Args:
        dstream(DStream): stream of real time tweets
    Returns:
        DStream Object with inner structure (hashtag, count)
    """
    # TODO: insert your code here
    def updateFunc(new_values, last_sum):
        return sum(new_values) + (last_sum or 0)

    hashtag = words.map(lambda x: x.lower()).filter(
        lambda x: len(x) > 2 and x[0] == "#").map(
        lambda x: (x, 1))
    hashtag_cnt = hashtag.reduceByKey(lambda cnt1, cnt2: cnt1 + cnt2)
    hashtag_cnt_total = hashtag_cnt.updateStateByKey(updateFunc).transform(
        lambda rdd: rdd.sortBy(lambda x: x[1], ascending=False))
    return hashtag_cnt_total
    pass
```

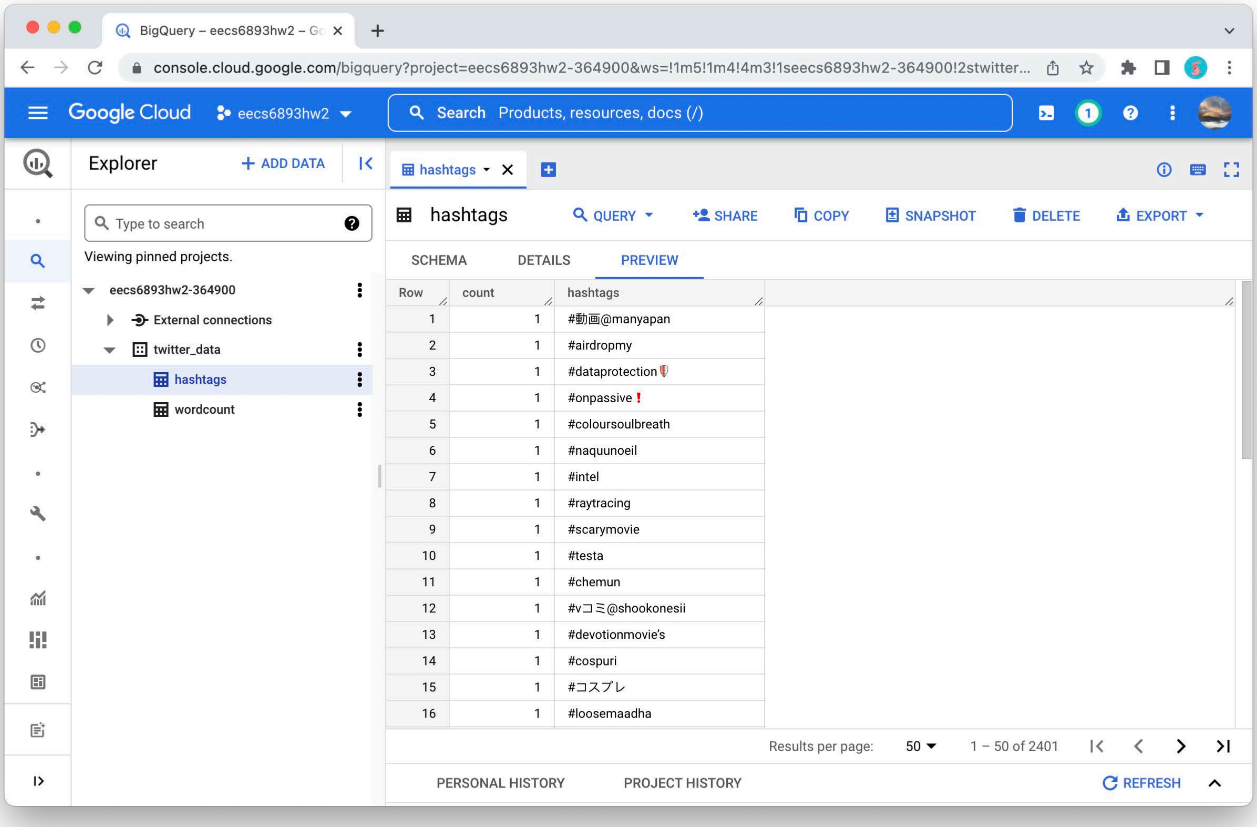
```python
def wordCount(words):
    """
    Calculte the count of 5 sepcial words for every 60 seconds (window no overlap)
    You can choose your own words.
    Your should:
    1. filter the words
    2. count the word during a special window size
    3. add a time related mark to the output of each window, ex: a datetime type
    Hints:
        You can take a look at reduceByKeyAndWindow transformation
        Dstream is a serious of rdd, each RDD in a DStream contains data from a certain interval
        You may want to take a look of transform transformation of DStream when trying to add a time
    Args:
        dstream(DStream): stream of real time tweets
    Returns:
        DStream Object with inner structure (word, (count, time))
    """

    # TODO: insert your code here
    word_cnt = words.map(lambda x: x.lower()).filter(lambda x: x in WORD).map(
        lambda x: (x, 1)).reduceByKeyAndWindow(lambda x, y: x + y,
                                               lambda x, y: x - y, 60, 60)
    word_cnt_total = word_cnt.transform(
        lambda time, rdd: rdd.map(
            lambda x: (x[0], x[1], time.strftime("%Y-%m-%d %H:%M:%S"))))
    return word_cnt_total
    pass


if __name__ == '__main__':
    # Spark settings
    conf = SparkConf()
    conf.setMaster('local[2]')
    conf.setAppName("TwitterStreamApp")
```

```python
    # create spark context with the above configuration
    sc = SparkContext(conf=conf)
    sc.setLogLevel("ERROR")

    # create sql context, used for saving rdd
    sql_context = SQLContext(sc)

    # create the Streaming Context from the above spark context with batch interval size 5 seconds
    ssc = StreamingContext(sc, 5)
    # setting a checkpoint to allow RDD recovery
    ssc.checkpoint("~/checkpoint_TwitterApp")

    # read data from port 9001
    dataStream = ssc.socketTextStream(IP, PORT)
    dataStream.pprint()

    words = dataStream.flatMap(lambda line: line.split(" "))

    # calculate the accumulated hashtags count sum from the beginning of the stream
    topTags = hashtagCount(words)
    topTags.pprint()

    # Calculte the word count during each time period 6s
    wordCount = wordCount(words)
    wordCount.pprint()
```

```python
    # TODO: insert your code here
    topTags.foreachRDD(lambda rdd: saveToStorage(rdd, output_directory_hashtags,
                                                 columns_name_hashtags,
                                                 mode="overwrite"))

    wordCount.foreachRDD(
        lambda rdd: saveToStorage(rdd, output_directory_wordcount,
                                  columns_name_wordcount, mode="append"))
    # start streaming process, wait for 600s and then stop.
    ssc.start()
    time.sleep(STREAMTIME)
    ssc.stop(stopSparkContext=False, stopGraceFully=True)

    # put the temp result in google storage to google BigQuery
    saveToBigQuery(sc, output_dataset, output_table_hashtags, output_directory_hashtags)
    saveToBigQuery(sc, output_dataset, output_table_wordcount, output_directory_wordcount)
```

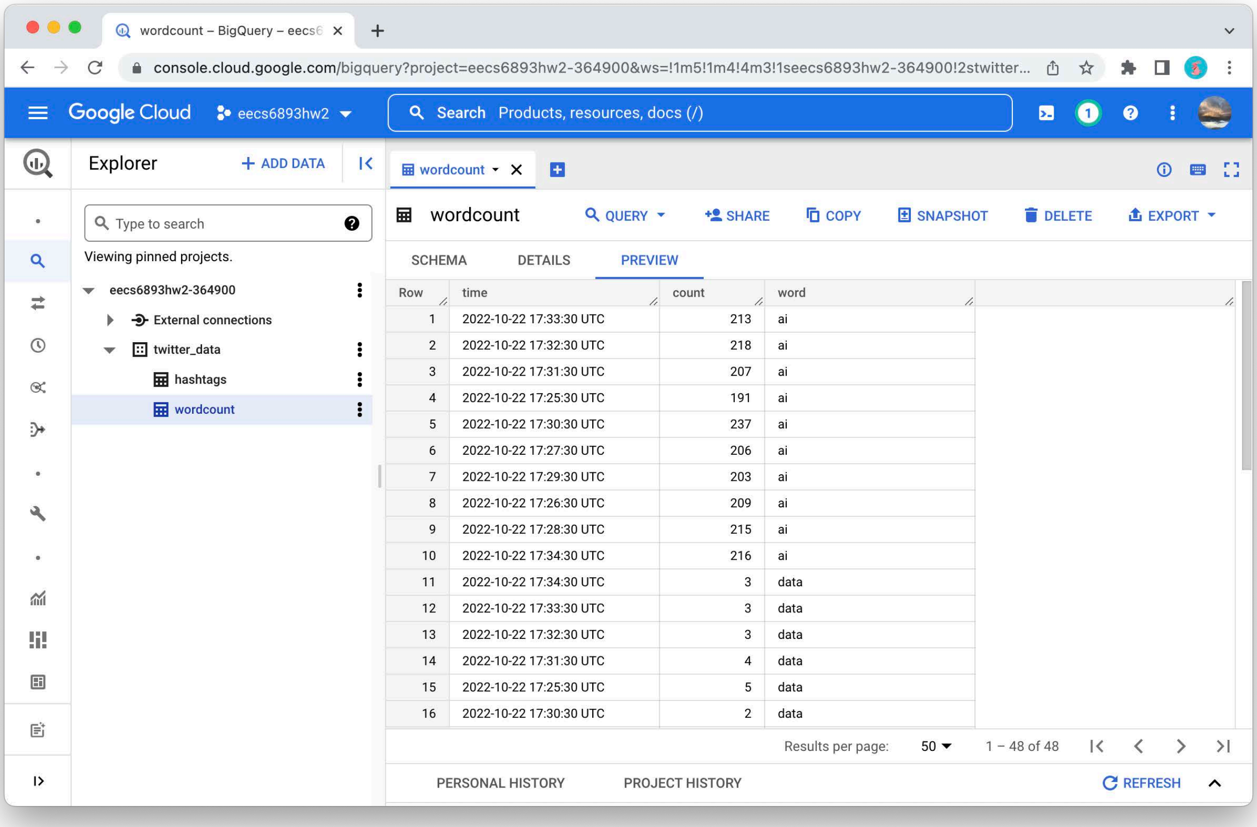
```
-------------------------------------------
Time: 2022-10-22 17:24:45
-------------------------------------------
RT @MegaFamily_Fans: Cute Pic Of #RamCharan Garu Along With #Upasana Garu In #RRR Movie Promotions ,Japan ❤️

Clicked By @ssrajamouli garu…@JXN1L_ That should be me@jeanmarcjimenez Salut t’ai toujour dispo ?j’en ai marre jv pt
un câbleRT @hairyween: my little movie star 🫶 https://t.co/GADnzU1bfu@gauty2166 Em biên hoà nè@haineworId ça m’a dit
« c’est toi t’es bizare à regarder qui s’abonnent » j’ai laissé tomber mdrrI am participating in the IGU tokens AirDr
op, which is worth $100,000!
@iguverse 🦎IguVerse GameFi app redefines the whole concept of NFT using AI / ML technologies. Unique user-generated
NFTs will become the new standard NFT 2.0
https://t.co/6D3JD7Bowl
#IguVerse #IGU #AirdropRT @Sosaaaaaaaw: Il est tard personne verras . Un soir j’ai voulu récupérer mon ex avec les pa
roles de Taken de Kalash elle a rigolé et rac…RT @Solution17Green: 1-10👉🏽
```

b. Screenshot of the preview of your data stored in BigQuery. You have to include two tables: hashtags and wordcount.



Row	count	hashtags
1	1	#動画@manyapan
2	1	#airdropmy
3	1	#dataprotection🛡
4	1	#onpassive❗
5	1	#coloursoulbreath
6	1	#naquunoeil
7	1	#intel
8	1	#raytracing
9	1	#scarymovie
10	1	#testa
11	1	#chemun
12	1	#vコミ@shookonesii
13	1	#devotionmovie’s
14	1	#cospuri
15	1	#コスプレ
16	1	#loosemaadha

Results per page: 50 1 – 50 of 2401



Row	time	count	word
1	2022-10-22 17:33:30 UTC	213	ai
2	2022-10-22 17:32:30 UTC	218	ai
3	2022-10-22 17:31:30 UTC	207	ai
4	2022-10-22 17:25:30 UTC	191	ai
5	2022-10-22 17:30:30 UTC	237	ai
6	2022-10-22 17:27:30 UTC	206	ai
7	2022-10-22 17:29:30 UTC	203	ai
8	2022-10-22 17:26:30 UTC	209	ai
9	2022-10-22 17:28:30 UTC	215	ai
10	2022-10-22 17:34:30 UTC	216	ai
11	2022-10-22 17:34:30 UTC	3	data
12	2022-10-22 17:33:30 UTC	3	data
13	2022-10-22 17:32:30 UTC	3	data
14	2022-10-22 17:31:30 UTC	4	data
15	2022-10-22 17:25:30 UTC	5	data
16	2022-10-22 17:30:30 UTC	2	data

Results per page: 50 1 – 48 of 48

Task 4:
Save the ipynb file as pdf format, containing your code and result.

In [2]:

```
from pyspark import SparkConf, SparkContext, SQLContext
from pyspark.sql import SparkSession
from pyspark.ml.feature import Word2Vec,CountVectorizer
from pyspark.ml.clustering import LDA, LDAModel
from pyspark.sql.functions import col, udf
from pyspark.sql.types import IntegerType,ArrayType,StringType
import pylab as pl
```

In [3]:

```
def to_word(termIndices):
  words = []
  for termID in termIndices:
    words.append(vocab_broadcast.value[termID])
  return words
```

In [7]:

```
#Load your document dataframe here
#================your code here==================

spark = SparkSession.builder \
    .appName('CSV_Handler').getOrCreate()

spark_df = spark.read.options(header=True, inferSchema=True) \
    .csv('gs://twitter_streaming_bucket/stream_data.csv')

spark_df = spark_df.dropna(subset=['sentence'])

#================================================
spark_df.show(5)
```

```
+--------------------+
|            sentence|
+--------------------+
|   iPhone  #
|                   "
|RT @WandasAttorne...|
|     RT @T_Az38: AI |
|     RT @T_Az38: AI |
+--------------------+
only showing top 5 rows
```

In [4]:

```
#CountVectorizer
#================your code here==================

from pyspark.ml.feature import HashingTF, IDF, Tokenizer

tokenizer = Tokenizer(inputCol="sentence", outputCol="words")
wordsData = tokenizer.transform(spark_df)

wordsData.show(5)

cv = CountVectorizer(inputCol="words", outputCol="features", vocabSize=100)
cvModel = cv.fit(wordsData)

cvResult = cvModel.transform(wordsData)
cvResult.show(5, truncate=False)

#==================================================
```

```
+--------------------+--------------------+
|            sentence|               words|
+--------------------+--------------------+
|   iPhone  #      # |[, iphone, , #, ,...|
|                   "|                [, "]|
|RT @WandasAttorne...|[rt, @wandasattor...|
|     RT @T_Az38: AI |  [rt, @t_az38:, ai]|
|     RT @T_Az38: AI |  [rt, @t_az38:, ai]|
+--------------------+--------------------+
only showing top 5 rows

+---------------------------------------------------------------------------------------------------
-------------+--------------------------------------------------------------------------------------
-------------------------------------------+-----------------------------------------+
|sentence
|words
|features                                 |
+---------------------------------------------------------------------------------------------------
-------------+--------------------------------------------------------------------------------------
-------------------------------------------+-----------------------------------------+
| iPhone  #      #
|[, iphone, , #, , , , , , #]
|(100,[0,7],[7.0,2.0])                    |
| "
|[, "]
|(100,[0,2],[1.0,1.0])                    |
|RT @WandasAttorney: Black Adam should ve been a Justice Society movie ft. Black Adam not the other
way around   |[rt, @wandasattorney:, black, adam, should, ve, been, a, justice, society, movie, ft.,
black, adam, not, the, other, way, around]|(100,[1,4,5,6,57],[1.0,1.0,1.0,1.0,1.0])|
|RT @T_Az38: AI
|[rt, @t_az38:, ai]
|(100,[1,3],[1.0,1.0])                    |
|RT @T_Az38: AI
|[rt, @t_az38:, ai]
|(100,[1,3],[1.0,1.0])                    |
+---------------------------------------------------------------------------------------------------
-------------+--------------------------------------------------------------------------------------
-------------------------------------------+-----------------------------------------+
only showing top 5 rows
```

In [5]:

```
#train LDA model, cluster the documents into 10 topics
#================your code here==================

lda = LDA(k = 10, seed = 1, optimizer = "em")
lda.setMaxIter(100)
ldaModel = lda.fit(cvResult)

#==================================================
```

In [6]:
```
transformed = ldaModel.transform(cvResult).select("topicDistribution")
#show the weight of every topic Distribution
transformed.show(5, truncate=False)
```

```
+---------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------
-+
|topicDistribution
|
+---------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------
-+
|[0.08695708608228897,0.08695675244901402,0.12324203388850045,0.1326341745492554,0.0869567566740017
4,0.13488126970778308,0.08746744017308353,0.0869628761072495,0.08695913908263181,0.0869824712861914
8] |
|[0.09677501473212528,0.09677442940493965,0.11029027233693207,0.10398842590378818,0.0967743884340890
9,0.1042594207363663,0.10075042490494877,0.09679009181753573,0.09678212772917134,0.0968154040001035
8]|
|[0.10705924599334717,0.09951851418933283,0.09255830961327359,0.0923081903894185,0.1134210442683797
3,0.09230889160006081,0.10205975842679695,0.09392077326024492,0.11169235001705781,0.0951529222420877
3]|
|[0.09677840089246245,0.09677897430877841,0.09705597531812218,0.0967747939258664,0.0967748001436392
6,0.09677583766485486,0.10687079743882232,0.10108213888407892,0.09768981855081718,0.1134184628725578
8]|
|[0.09677840089246245,0.09677897430877841,0.09705597531812218,0.0967747939258664,0.0967748001436392
6,0.09677583766485486,0.10687079743882232,0.10108213888407892,0.09768981855081718,0.1134184628725578
8]|
+---------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------
-+
only showing top 5 rows
```

In [7]:
```
#The higher ll is, the lower lp is, the better model is.
ll = ldaModel.logLikelihood(cvResult)
lp = ldaModel.logPerplexity(cvResult)
print("ll: ", ll)
print("lp: ", lp)
```

```
ll:  -219892.38051280688
lp:  3.6965400348452895
```

In [8]:
```
wordNumbers = 10
vocabArray = cvModel.vocabulary

sc = SparkContext.getOrCreate()

topicIndices = ldaModel.describeTopics(maxTermsPerTopic = wordNumbers)
vocab_broadcast = sc.broadcast(vocabArray)
udf_to_word = udf(to_word, ArrayType(StringType()))

topics = topicIndices.withColumn("words", udf_to_word(topicIndices.termIndices))
topics.show(5, truncate=False)
```

```
+-----+--------------------------------------+------------------------------------------------------
----------------------------------------------------------------------------------------------------
-----------------------------------------------------------------+----------------------------------
--------------+
|topic|termIndices                           |termWeights
|words                                           |
+-----+--------------------------------------+------------------------------------------------------
----------------------------------------------------------------------------------------------------
-----------------------------------------------------------------+----------------------------------
--------------+
|0    |[5, 11, 17, 19, 4, 37, 38, 42, 58, 56]|[0.26020547667015287, 0.15567354379579496, 0.106328723
0976588, 0.09723443326064725, 0.04532390207588795, 0.04302923498744867, 0.0429265393193716, 0.035904
65705430056, 0.03296036998575142, 0.027746298445421595]          |[a, is, for, that, the, all, as, b
e, he, an]    |
|1    |[4, 12, 13, 9, 16, 47, 53, 30, 66, 36]|[0.1761869011061337, 0.12228981005847112, 0.1188518239
2665622, 0.11155003402909794, 0.0989451361923473, 0.03443100671876502, 0.030585873003439217, 0.02857
7304070445907, 0.02839525114321954, 0.027102502399314454]        |[the, of, in, to, you, are, have,
s, we, your]  |
|2    |[2, 7, 0, 65, 1, 52, 3, 83, 24, 27]   |[0.47034245760186144, 0.3077904643919394, 0.1669452321
7406076, 0.033026471513932454, 0.012945941786984634, 0.0026526930351439422, 9.752665983070322E-4, 5.
139341465683023E-4, 4.358655787114076E-4, 3.889489668255988E-4]  |[", #, , #ai, rt, 1, ai, 3, 2, -]
|
|3    |[0, 54, 90, 2, 83, 1, 3, 52, 30, 24]  |[0.9656023452716982, 0.03096444575397968, 7.0751380990
14053E-4, 1.2443231319797816E-4, 1.1266824591515614E-4, 6.543553189548248E-5, 5.945329772842025E-5,
5.690732140991388E-5, 5.493300964359554E-5, 5.008245551133074E-5]|[, ., ?, ", 3, rt, ai, 1, s, 2]
|
|4    |[10, 4, 15, 18, 21, 9, 28, 33, 34, 35]|[0.14748629907224978, 0.12687480365968315, 0.103975217
42460969, 0.08246446538299902, 0.07171607908986861, 0.0683807387784247, 0.05379815931899003, 0.04374
194450773947, 0.0426309218981958, 0.040359945432020594]          |[and, the, it, this, was, to, my,
so, but, like]|
+-----+--------------------------------------+------------------------------------------------------
----------------------------------------------------------------------------------------------------
-----------------------------------------------------------------+----------------------------------
--------------+
only showing top 5 rows
```

In [9]:

```
# Output topics. Each is a distribution over words (matching word count vectors)
print("Learned topics (as distributions over vocab of " + str(ldaModel.vocabSize())+ " words):")
topics = ldaModel.topicsMatrix()
print(topics)
```

```
Learned topics (as distributions over vocab of 100 words):
DenseMatrix([[4.44025656e-01, 3.38664462e-01, 9.37252313e+02, 6.29260387e+03,
               3.39404730e-01, 6.48613679e+03, 9.21858314e+01, 1.53735391e+00,
               9.02863222e-01, 5.25888432e+00],
              [1.66685708e+00, 1.99447506e+00, 7.26802061e+01, 4.26428005e-01,
               5.35555200e-01, 7.40506718e-01, 2.46618364e+03, 4.31182788e+02,
               2.60024868e+02, 7.76564677e+02],
              [5.65331394e-01, 3.54350114e-01, 2.64056392e+03, 8.10896183e-01,
               3.27200611e-01, 7.65152514e+00, 8.92237684e+02, 3.92480873e+00,
               2.33686135e+00, 9.22742563e+00],
              [3.51578754e-01, 2.75492801e-01, 5.47527391e+00, 3.87443189e-01,
               2.68974569e-01, 5.01502347e-01, 1.72050083e+01, 4.52451874e+02,
               7.61680788e-01, 2.12032117e+03],
              [2.72496692e+02, 1.13373992e+03, 3.84032310e-01, 2.28626432e-01,
               8.08515319e+02, 2.29134979e-01, 5.58541252e-01, 4.52149442e-01,
               7.99723651e+01, 4.23215633e-01],
              [1.56440925e+03, 1.82507619e+01, 4.30049325e-01, 2.22193959e-01,
               2.01376764e+00, 2.22408993e-01, 1.08715828e+00, 4.37689998e+00,
               3.38175853e+02, 1.08116603e+01],
              [7.26507874e+01, 8.89442129e+00, 5.95122325e-01, 2.29947289e-01,
               1.10175737e+00, 2.31519292e-01, 5.59667421e+00, 5.14890053e-01,
               1.78749672e+03, 6.88161230e-01],
              [1.94517906e-01, 1.63015890e-01, 1.72797582e+03, 2.16388392e-01,
               1.63644506e-01, 2.21427466e-01, 2.97525932e-01, 2.83387490e-01,
               2.29993917e-01, 2.54277841e-01],
              [2.01329467e+00, 1.76519563e+01, 2.73828162e-01, 1.86319042e-01,
               1.22174905e+02, 1.87099257e-01, 3.72727508e-01, 3.82728068e-01,
               1.25616809e+03, 3.58905473e+00],
              [2.92597095e+01, 7.17810044e+02, 3.86622937e-01, 2.26376730e-01,
               4.35759294e+02, 2.25248783e-01, 5.65918617e-01, 5.02360350e-01,
               2.56991706e+01, 5.65255238e-01],
              [3.26694886e+00, 8.16141426e+00, 2.52738230e-01, 1.69510818e-01,
               9.39862988e+02, 1.69368553e-01, 3.25570582e-01, 3.17237139e-01,
               3.18880800e+00, 2.85415208e-01],
              [9.35941605e+02, 2.19517388e+00, 2.48521680e-01, 1.64624990e-01,
               9.07407344e-01, 1.63093604e-01, 3.09201699e-01, 2.81749427e-01,
               2.51420330e+00, 2.74419327e-01],
              [3.31558322e+01, 7.86919113e+02, 3.18085363e-01, 1.92272331e-01,
               8.73092676e+01, 1.91948328e-01, 4.53115449e-01, 3.72688001e-01,
               2.97170119e+01, 3.70665303e-01],
              [1.05605152e+01, 7.64796117e+02, 3.60842422e-01, 2.03501978e-01,
               1.27524380e+01, 2.04724575e-01, 5.78249665e-01, 4.38215050e-01,
               6.36862278e+01, 4.19167961e-01],
              [1.10837579e-01, 8.86090616e-02, 1.18897047e-01, 8.86680868e-02,
               8.94908943e-02, 8.88231571e-02, 1.37483862e-01, 7.40920393e+02,
               1.16540994e-01, 2.40256449e-01],
              [4.08520835e-01, 4.07973329e-01, 1.87331608e-01, 1.34351241e-01,
               6.62586689e+02, 1.33503525e-01, 2.18798647e-01, 2.26829866e-01,
               4.77952177e-01, 2.18049857e-01],
              [5.88599959e+00, 6.36699156e+02, 2.50561408e-01, 1.65235650e-01,
               9.48314562e-01, 1.66215321e-01, 3.10270580e-01, 2.89884461e-01,
               1.00640560e+00, 2.77956784e-01],
              [6.39270317e+02, 1.35470835e+00, 2.73697108e-01, 1.71888823e-01,
               4.26636820e-01, 1.73033887e-01, 3.44038809e-01, 3.06674433e-01,
               1.39238786e+00, 2.86616675e-01],
              [1.54998794e+01, 1.81538263e+01, 2.94692473e-01, 1.84387363e-01,
               5.25508467e+02, 1.84120031e-01, 3.97686200e-01, 3.75317507e-01,
               4.90365445e+01, 3.65079693e-01],
              [5.84593562e+02, 3.43737218e-01, 1.77436972e-01, 1.22952250e-01,
               6.53154609e-01, 1.22471803e-01, 2.04020412e-01, 2.01600904e-01,
               3.86379080e-01, 1.94684585e-01],
              [4.65879856e+01, 6.61041996e+01, 4.56126246e-01, 2.35122095e-01,
               2.81645056e+00, 2.36590242e-01, 9.71620088e-01, 4.32277457e+00,
               3.51712183e+02, 5.56947794e-01],
              [3.61371829e-01, 2.67288938e-01, 1.68631663e-01, 1.25316214e-01,
               4.57013898e+02, 1.25766634e-01, 1.94276164e-01, 2.02161792e-01,
               3.56466745e-01, 1.84822256e-01],
              [1.17519892e-01, 9.23798950e-02, 1.23851374e-01, 9.25007167e-02,
               9.34231803e-02, 9.24659731e-02, 1.41002817e-01, 4.43744784e+02,
               1.21064223e-01, 3.81008396e-01],
              [5.19909570e-01, 3.76746505e-01, 2.50911208e-01, 1.64481220e-01,
               5.68728238e+00, 1.64800557e-01, 3.34137037e-01, 4.09642593e+02,
               5.23527951e+00, 6.23859206e-01],
              [3.33573749e-01, 2.60429481e-01, 2.44700622e+00, 3.26375609e-01,
               2.57881913e-01, 3.66473214e-01, 4.04608230e+02, 8.43602317e-01,
               5.54825458e-01, 1.00160232e+00],
              [3.72996489e-01, 6.76154699e-01, 2.62995636e-01, 2.00379211e-01,
               4.80971750e-01, 2.00905646e-01, 3.86381270e-01, 5.06614732e-01,
               1.24362621e+01, 3.84476338e+02],
              [4.08061042e+00, 4.22629976e+00, 2.56585443e-01, 1.72170522e-01,
               2.73511264e+00, 1.71330754e-01, 3.35680350e-01, 3.30219359e-01,
               3.76382694e+02, 3.09297173e-01],
              [3.76438592e-01, 2.97951601e-01, 2.18361024e+00, 2.77774892e-01,
               2.72297623e-01, 2.91910874e-01, 3.57698400e+02, 6.10512103e-01,
               1.26741553e+00, 7.23688111e-01],
              [4.47264255e-01, 8.53771870e-01, 2.37133002e-01, 1.58447452e-01,
               3.42831159e+02, 1.58189719e-01, 2.95572150e-01, 2.94905759e-01,
               1.45537298e+00, 2.68184233e-01],
              [1.44634488e-01, 1.03391061e-01, 1.42921563e-01, 1.07455979e-01,
               1.03739335e-01, 1.06307210e-01, 1.58398352e-01, 3.07547969e-01,
               1.47073491e-01, 3.24678531e+02],
              [2.28563224e+00, 1.83891256e+02, 5.40071287e-01, 3.57985532e-01,
               1.00579909e+01, 3.70973766e-01, 6.95450653e-01, 1.00413556e+00,
               1.15015362e+02, 6.78114215e+00],
              [1.43196983e-01, 1.17381218e-01, 1.61120976e-01, 1.39820727e-01,
               1.15923170e-01, 1.37608509e-01, 2.02881930e-01, 8.66661748e-01,
               2.18775058e-01, 2.97896630e+02],
              [1.00277812e-01, 8.33860263e-02, 1.07847085e-01, 8.20617248e-02,
               8.24790654e-02, 8.19932430e-02, 1.23838467e-01, 2.83087885e+02,
               1.10212997e-01, 1.40018588e-01],
              [8.40729228e-01, 8.86213953e-01, 2.38591373e-01, 1.61431508e-01,
               2.78747483e+02, 1.61310889e-01, 2.97598817e-01, 3.34897776e-01,
               1.98170991e+00, 3.50033291e-01],
              [3.89994887e-01, 4.13858273e-01, 1.78220896e-01, 1.27739001e-01,
               2.71667442e+02, 1.28512712e-01, 2.11641605e-01, 2.26484781e-01,
               4.36626391e-01, 2.19479099e-01],
              [6.86191563e+00, 2.41710816e+00, 2.44250592e-01, 1.63339913e-01,
               2.57195544e+02, 1.64283292e-01, 3.14281392e-01, 2.90037054e-01,
               1.04985118e+00, 2.99388777e-01],
              [8.50080210e+01, 1.74401098e+02, 3.45282909e-01, 1.97140152e-01,
               5.56831218e-01, 1.98247297e-01, 4.26921012e-01, 3.64522101e-01,
               4.14103310e+00, 3.60903102e-01],
              [2.58700677e+02, 4.46150858e-01, 2.03342446e-01, 1.34867308e-01,
               3.14416321e-01, 1.35425757e-01, 2.61685133e-01, 2.27943077e-01,
               3.62093932e-01, 2.13397690e-01],
              [2.58083250e+02, 3.05912946e-01, 2.11094552e-01, 1.45371632e-01,
               7.84132284e-01, 1.46510296e-01, 2.47315738e-01, 3.24445732e-01,
               3.99829906e-01, 3.52137076e-01],
              [7.04355045e-02, 6.10362574e-02, 7.75856326e-02, 6.21724196e-02,
               6.14058224e-02, 6.17629840e-02, 8.78515931e-02, 2.59351158e+02,
               7.67532511e-02, 8.98383810e-02],
              [2.21750006e-01, 1.91464730e-01, 2.12538937e-01, 1.67430841e-01,
               2.13308521e-01, 1.66618744e-01, 2.28487496e-01, 5.52710257e-01,
               5.25804305e-01, 2.38519886e+02],
              [8.99844858e-02, 7.44170989e-02, 9.69533429e-02, 7.64694279e-02,
               7.40865102e-02, 7.59104099e-02, 1.11130649e-01, 2.38191062e+02,
               9.54316519e-02, 1.14554754e-01],
              [2.15866239e+02, 1.84082414e+01, 2.73184170e-01, 1.81174115e-01,
               1.09162832e+00, 1.76862454e-01, 3.42582965e-01, 3.41998281e-01,
               1.01495083e+00, 3.03138946e-01],
              [1.23808680e-01, 9.20148110e-02, 1.19964610e-01, 9.67109927e-02,
               9.20596075e-02, 9.73732258e-02, 1.32813768e-01, 2.55415859e-01,
               1.21353134e-01, 2.34868485e+02],
              [1.67435513e-01, 1.38192520e-01, 1.84956464e-01, 1.75765525e-01,
               1.29188093e-01, 1.72997783e-01, 2.31698552e-01, 2.24189700e+02,
               2.81017677e-01, 8.32904834e+00],
              [1.40510691e-01, 1.07879164e-01, 1.47186032e-01, 1.06023028e-01,
               1.09266088e-01, 1.06230634e-01, 1.63519702e-01, 5.33186628e-01,
               1.41038030e-01, 2.31445160e+02],
              [6.32741485e+00, 1.22805110e+01, 4.93841829e-01, 2.34093693e-01,
               6.24979000e-01, 2.35237288e-01, 8.78214064e-01, 4.73458640e-01,
               2.07909400e+02, 5.42849711e-01],
              [1.14868687e+00, 2.21559076e+02, 2.31572834e-01, 1.56534136e-01,
               4.23330477e+00, 1.55865877e-01, 3.01176320e-01, 2.71414884e-01,
               6.66906289e-01, 2.75462323e-01],
              [2.72506575e-01, 2.14412788e-01, 9.29801695e-01, 3.13531978e-01,
               2.12148035e-01, 3.22925114e-01, 2.16284161e+02, 1.67327233e+00,
               3.47273693e-01, 4.29966850e-01],
              [1.71321970e-01, 1.45889948e-01, 2.61150675e-01, 1.82123815e-01,
               1.46861108e-01, 1.83829218e-01, 2.15214913e+02, 2.60700873e-01,
               2.02561915e-01, 2.30647402e-01],
              [4.01505216e-01, 4.35982150e-01, 4.28452241e-01, 3.20276010e-01,
               3.15953203e-01, 3.59528658e-01, 1.06592715e+00, 3.59432219e-01,
               2.07970460e+02, 3.42482666e-01],
              [3.22539206e-01, 3.86349132e-01, 2.19233091e-01, 1.48262555e-01,
               4.00532858e-01, 1.49116967e-01, 2.66731813e-01, 4.36397948e-01,
               3.43863615e-01, 2.08326973e+02],
              [4.28439439e-01, 3.38418541e-01, 1.48925648e+01, 3.70851658e-01,
               2.83069485e-01, 4.29703841e-01, 1.70742549e+02, 2.30581284e+00,
               8.02804826e-01, 1.74057856e+01],
              [3.83866833e+00, 1.96816137e+02, 2.48549410e-01, 1.56630965e-01,
               2.46840502e+00, 1.57250446e-01, 2.96724757e-01, 2.77207464e-01,
               3.44634223e+00, 2.94083892e-01],
              [2.84167042e-01, 2.32251082e-01, 2.96577944e-01, 2.01788026e+02,
               2.44562052e-01, 4.91989265e-01, 3.55746015e-01, 5.95016878e-01,
               3.30049765e-01, 3.81613860e-01],
              [4.62128120e+00, 1.74310899e+02, 3.05226930e-01, 1.87746753e-01,
               1.39611816e+01, 1.87958049e-01, 3.94885696e-01, 3.73005922e-01,
               7.31210141e+00, 3.45713546e-01],
              [1.66816496e+02, 3.14882609e+01, 2.83515597e-01, 1.81886269e-01,
               4.56764694e-01, 1.80913456e-01, 4.00030648e-01, 3.46680860e-01,
               1.52370608e+00, 3.21745530e-01],
              [1.00228429e+00, 8.73222755e-01, 2.15171443e-01, 1.51833960e-01,
               1.97289192e+02, 1.51853682e-01, 2.65231845e-01, 2.57579677e-01,
               5.29335207e-01, 2.64294709e-01],
              [1.98164575e+02, 2.49694595e-01, 1.61149202e-01, 1.21579753e-01,
               3.28563405e-01, 1.22401239e-01, 1.94300562e-01, 1.83372837e-01,
               3.01226405e-01, 1.73137434e-01],
              [7.90381673e-02, 6.73479092e-02, 8.87254651e-02, 6.92418791e-02,
               6.77122649e-02, 6.89117373e-02, 1.03382534e-01, 1.99268541e+02,
               8.63110136e-02, 1.00787969e-01],
              [8.36313783e-02, 7.06945609e-02, 9.14005621e-02, 7.16029267e-02,
               7.09971658e-02, 7.14000647e-02, 1.06423958e-01, 1.97214271e+02,
               9.06179478e-02, 1.28960039e-01],
              [2.01035542e+01, 1.72742693e+02, 2.89680404e-01, 1.83636228e-01,
               1.19651410e+00, 1.81383770e-01, 3.45628734e-01, 3.32661176e-01,
               1.28317959e+00, 3.41069049e-01],
              [4.42945292e-01, 4.63724365e-01, 1.88548289e-01, 1.32838925e-01,
               1.90509437e+02, 1.32463123e-01, 2.30190338e-01, 2.20678668e-01,
               4.64196390e-01, 2.14977701e-01],
              [7.41566218e-02, 6.25294249e-02, 8.00112727e-02, 6.48322717e-02,
               6.28215197e-02, 6.35197422e-02, 9.17087985e-02, 1.91328191e+02,
               7.94268578e-02, 9.28029096e-02],
              [9.23847151e-02, 7.52992233e-02, 9.91339804e-02, 7.60336708e-02,
               7.54893415e-02, 7.54161810e-02, 1.11996791e-01, 1.90182321e+02,
               9.88830646e-02, 1.13042240e-01],
              [2.48141023e-01, 2.11786880e-01, 1.85414920e+02, 2.62012248e-01,
               2.03385962e-01, 2.71939725e-01, 4.62022308e-01, 3.40072801e-01,
               2.86857143e-01, 2.98862280e-01],
              [1.06616892e+00, 1.82719769e+02, 2.29481293e-01, 1.53529008e-01,
               4.15640395e-01, 1.54633483e-01, 3.01648780e-01, 2.57564048e-01,
               4.37407777e-01, 2.64157202e-01],
              [2.52885581e+01, 1.56051219e+02, 2.87411775e-01, 1.84687896e-01,
               8.24252480e-01, 1.84289420e-01, 3.86343548e-01, 3.65465771e-01,
               2.03699302e+00, 3.90778977e-01],
              [1.33098803e+02, 2.62450241e+01, 3.36718167e-01, 2.08685383e-01,
               1.13130411e+01, 2.10260944e-01, 4.33886400e-01, 3.66545472e-01,
               1.24395700e+01, 3.47465847e-01],
              [4.55909960e+00, 1.73961150e+02, 2.62390501e-01, 1.69094411e-01,
               9.06304282e-01, 1.69209300e-01, 3.68948068e-01, 3.24540383e-01,
               2.94853024e+00, 3.30733540e-01],
              [2.72097893e+00, 1.67221904e+02, 3.04428864e-01, 1.96684592e-01,
               9.36069118e-01, 1.98652664e-01, 4.43361542e-01, 3.42088825e-01,
               4.26529274e+00, 3.70539079e-01],
              [9.44056135e-02, 8.15561931e-02, 1.09578989e-01, 8.24445154e-02,
               8.24069502e-02, 8.25642025e-02, 1.30951261e-01, 1.74120637e+02,
               1.04427119e-01, 1.11028217e-01],
              [3.03472453e+00, 3.02908316e+00, 2.49787667e-01, 1.63818241e-01,
               1.63405149e+02, 1.63992933e-01, 2.97386729e-01, 3.04572584e-01,
               1.06980462e+00, 2.81680458e-01],
              [2.93879397e-01, 2.30952471e-01, 4.73890669e-01, 2.79076900e-01,
               2.35326161e-01, 2.83897704e-01, 1.67484762e+02, 4.05789060e-01,
               8.58795186e-01, 4.53630574e-01],
              [1.32581063e+00, 2.85428205e+01, 3.06699091e-01, 1.82637544e-01,
               7.66128849e-01, 1.81997715e-01, 4.07426347e-01, 4.34127235e-01,
               1.37496287e+02, 3.56065562e-01],
              [1.08534414e+00, 1.62092032e+02, 2.52147646e-01, 1.53684017e-01,
               9.41039682e-01, 1.53527811e-01, 2.72848128e-01, 2.55781829e-01,
               5.32402914e-01, 2.61191418e-01],
              [5.18959224e-01, 1.19417480e+00, 2.18022146e-01, 1.48175446e-01,
               1.62363418e+02, 1.47941173e-01, 2.59678270e-01, 2.44799265e-01,
               6.36974294e-01, 2.67857060e-01],
              [4.03541438e-01, 2.89724320e-01, 2.84606790e-01, 1.89144114e-01,
               3.91603566e-01, 1.91348740e-01, 3.21266984e-01, 6.02895167e-01,
               4.38401061e-01, 1.59887468e+02],
              [3.29172664e-01, 3.50016758e-01, 2.07137656e-01, 1.40085324e-01,
               3.09040477e-01, 1.39664914e-01, 2.60183547e-01, 2.80205756e-01,
               1.59758907e+02, 2.25585694e-01],
              [7.41009671e-02, 6.30057531e-02, 8.05758931e-02, 6.31936444e-02,
               6.31356984e-02, 6.30794562e-02, 9.03601927e-02, 1.60327827e+02,
               8.03942095e-02, 9.43268628e-02],
              [1.70839246e-01, 1.48751522e-01, 2.47338971e-01, 1.88397222e-01,
               1.48923491e-01, 1.89538774e-01, 1.57209208e+02, 2.55187355e-01,
               2.01085765e-01, 2.40729417e-01],
              [8.44373660e-02, 7.21791489e-02, 9.53547737e-02, 7.37620234e-02,
               7.25377228e-02, 7.31054288e-02, 1.14597641e-01, 1.55214897e+02,
               9.30704554e-02, 1.06058682e-01],
              [8.08685753e-02, 6.89194206e-02, 9.17755710e-02, 7.05977134e-02,
               6.99506748e-02, 6.99552247e-02, 1.02070284e-01, 1.53252008e+02,
               8.81241951e-02, 1.05730792e-01],
              [4.26699961e-01, 3.88216710e-01, 2.88529335e+00, 7.34232517e-01,
               3.20964776e-01, 1.70750945e+00, 1.32712729e+02, 4.78761690e+00,
               1.80959652e+00, 3.22714074e+00],
              [1.04985787e+00, 8.18981649e+00, 2.75737617e-01, 1.76746784e-01,
               1.35837535e+02, 1.76766231e-01, 3.62807639e-01, 3.17172921e-01,
               1.30887461e+00, 3.04684506e-01],
              [4.33577437e-01, 3.82063099e-01, 1.88002568e-01, 1.33840112e-01,
               1.35610205e+02, 1.34715978e-01, 2.32344084e-01, 2.36902875e-01,
               4.33460818e-01, 2.14887661e-01],
              [1.59617391e-01, 1.38255339e-01, 1.69495053e-01, 1.20073592e-01,
               1.31611048e-01, 1.20300741e-01, 1.98970839e-01, 2.72855079e-01,
               1.84332257e-01, 1.34504489e+02],
              [1.03178445e+00, 1.27906318e+02, 2.70200736e-01, 1.77309260e-01,
               1.15432476e+00, 1.76627250e-01, 3.40842331e-01, 3.04178454e-01,
               3.34561177e+00, 2.92802545e-01],
              [1.27379346e-01, 1.05523062e-01, 1.38695504e-01, 1.13860221e-01,
               1.06114455e-01, 1.13371144e-01, 1.77934656e-01, 1.32616741e+02,
               1.62857130e-01, 3.37523123e-01],
              [7.76066630e-01, 5.17820297e+00, 3.25909585e-01, 2.01192110e-01,
               5.51841275e-01, 2.02412587e-01, 4.19571212e-01, 4.20204188e-01,
               1.23571171e+02, 3.53428723e-01],
              [1.93784154e-01, 1.69602968e-01, 2.20900063e-01, 4.61070146e+00,
               1.54430965e-01, 5.99955725e-01, 2.45154135e-01, 1.22274413e+02,
               2.25867182e-01, 3.05190278e-01],
              [1.63428007e-01, 1.25028447e-01, 1.71756468e-01, 1.23572730e-01,
               1.31229406e-01, 1.23464387e-01, 2.02928943e-01, 6.92457672e-01,
               1.70573618e-01, 1.26095560e+02],
              [1.24470981e+02, 2.99154757e-01, 2.71563702e-01, 1.70374577e-01,
               2.22333660e-01, 1.73595979e-01, 3.01681158e-01, 4.13702919e-01,
               4.04257241e-01, 2.72355088e-01],
              [1.49625119e-01, 1.07828028e-01, 1.50247952e-01, 1.07891638e-01,
               1.09002453e-01, 1.08279716e-01, 1.64375956e-01, 3.53937057e-01,
               1.45631036e-01, 1.25603181e+02],
              [6.57269057e+00, 1.13039676e+00, 3.22472425e-01, 1.97086810e-01,
               1.04586799e+02, 1.95827261e-01, 4.03741757e-01, 3.72725597e-01,
               1.18714495e+01, 3.46810111e-01],
              [1.24965171e+02, 1.32682866e-01, 1.01234393e-01, 7.31955001e-02,
               1.79443225e-01, 7.34689416e-02, 1.06830690e-01, 1.03616237e-01,
               1.65038790e-01, 9.93183744e-02],
              [1.07119981e-01, 8.39715410e-02, 1.10599550e-01, 8.41025306e-02,
               8.48700933e-02, 8.40728805e-02, 1.28094280e-01, 1.22921977e+02,
               1.10976177e-01, 2.84216466e-01],
              [1.16824583e+02, 9.68386405e-01, 2.48098825e-01, 1.57357698e-01,
               8.50954958e-01, 1.55360520e-01, 2.97026578e-01, 3.00329372e-01,
               1.94693784e+00, 2.50964906e-01],
              [3.02690723e-01, 1.83780718e-01, 5.08586178e-01, 2.61717065e-01,
               1.86805153e-01, 2.68824876e-01, 1.16027129e+02, 4.72895721e-01,
               3.33346098e-01, 4.54224521e-01],
              [1.16517603e-01, 1.06831454e-01, 1.43136508e-01, 1.29374362e-01,
               1.00007769e-01, 1.30574157e-01, 1.73311294e-01, 2.19529362e-01,
               1.78037908e-01, 1.17702680e+02]])
```