Dataset link: <https://data.cityofnewyork.us/Social-Services/311-Service-Requests-from-2010-to-Present/erm2-nwe9/data>

Code:

import csv

import logging

from datetime import datetime

from mrjob.job import MRJob

from mrjob.util import log\_to\_stream

import calendar

# Complaint status

OPEN = 'Open'

CLOSED = 'Closed'

ASSIGNED = 'Assigned'

# Complaint values

STREET\_CONDITION = 'Street Condition'

ILLEGAL\_PARKING = 'Illegal Parking'

NOISE\_COMPLAINT = 'Noise Complaint'

NOISE = 'noise'

# CSV Fields

COMPLAINT\_TYPE = 'Complaint Type'

STATUS = 'Status'

CREATED\_DATE = 'Created Date'

# Key identifiers

QUARTER = "Quarter"

BOROUGH = "Borough"

TOTAL\_NO\_OF = "Total number of"

MONTH = "Month"

# Variable array

STATUSES = [OPEN, ASSIGNED, CLOSED]

COMPLAINT\_TYPES = [STREET\_CONDITION, ILLEGAL\_PARKING, NOISE\_COMPLAINT]

YEAR\_TO\_FILTER = 2017

log = logging.getLogger(\_\_name\_\_)

class MRWordFrequencyCount(MRJob):

def set\_up\_logging(cls, quiet=False, verbose=False, stream=None):

log\_to\_stream(name='mrjob', debug=verbose, stream=stream)

log\_to\_stream(name='\_\_main\_\_', debug=verbose, stream=stream)

def mapper(self, \_, line):

cols = 'Unique Key,Created Date,Closed Date,Agency,Agency Name,Complaint Type,Descriptor,Location Type,' \

'Incident Zip,Incident Address,Street Name,Cross Street 1,Cross Street 2,Intersection Street 1,' \

'Intersection Street 2,Address Type,City,Landmark,Facility Type,Status,Due Date,Resolution ' \

'Description,Resolution Action Updated Date,Community Board,BBL,Borough'.split(',')

status = None

complaint\_type = None

borough = None

quarter = 0

year = None

month = None

try:

for row in csv.reader([line]):

dict\_row = dict(zip(cols, row))

complaint\_type = dict\_row[COMPLAINT\_TYPE]

status = dict\_row[STATUS]

date\_obj = datetime.strptime(dict\_row[CREATED\_DATE], '%m/%d/%Y %H:%M:%S %p')

year = date\_obj.year

quarter = self.get\_quarter\_of\_date(date\_obj)

borough = dict\_row[BOROUGH]

month = date\_obj.month

except Exception:

log.info("Skipping the line " + line)

else:

# Total complaints

for complaint\_var in COMPLAINT\_TYPES:

total\_complaints = 0

for status\_var in STATUSES:

count = 0

# If status matches and complaint matches set the count to 1

if status\_var == status:

if complaint\_type.lower() == complaint\_var.lower():

count = 1

elif complaint\_var == NOISE\_COMPLAINT and complaint\_type.lower().startswith(NOISE):

count = 1

total\_complaints += count

if count > 0:

yield "{} {} {} complaints".format(TOTAL\_NO\_OF, status\_var, complaint\_var), count

# Monthly calculations

for month\_var in range(1, 13):

m\_count = 0

if month\_var == month and count == 1 and year == YEAR\_TO\_FILTER:

m\_count = total\_complaints

if m\_count > 0:

yield "{} wise [{}] {} complaints".format(MONTH, calendar.month\_name[month\_var], complaint\_var), m\_count

# Quarter wise calculations

for quarter\_var in range(1, 5):

q\_count = 0

if quarter\_var == quarter and count == 1 and year == YEAR\_TO\_FILTER:

q\_count = 1

if q\_count > 0:

yield "{} {} {} {} complaints".format(QUARTER, str(quarter\_var), status\_var, complaint\_var), q\_count

# Borough with illegal parking

for quarter\_var in range(1, 5):

q\_count = 0

if quarter\_var == quarter and complaint\_type.lower() == ILLEGAL\_PARKING.lower() and year == YEAR\_TO\_FILTER:

q\_count = 1

if q\_count > 0:

yield "{} with {} for the quarter [{}]".format(BOROUGH, ILLEGAL\_PARKING, str(quarter\_var)), (borough, q\_count)

def reducer(self, key, values):

if key.startswith(BOROUGH):

result\_dict = dict()

for item in list(values):

borough = item[0]

count = item[1]

result\_dict[borough] = result\_dict.get(borough, 0) + count

yield key, [(k, v) for (k, v) in result\_dict.items() if v == max(result\_dict.values())]

elif key.startswith(QUARTER):

yield key, int(sum(values) // 3)

else:

yield key, sum(values)

def get\_quarter\_of\_date(self, date\_obj):

"""

Assign created date to a variable

Convert string to date

Extract month from date

(month-1)/3 +1

"""

month = date\_obj.month

quarter = ((month - 1) // 3) + 1

return quarter

if \_\_name\_\_ == '\_\_main\_\_':

MRWordFrequencyCount.run()

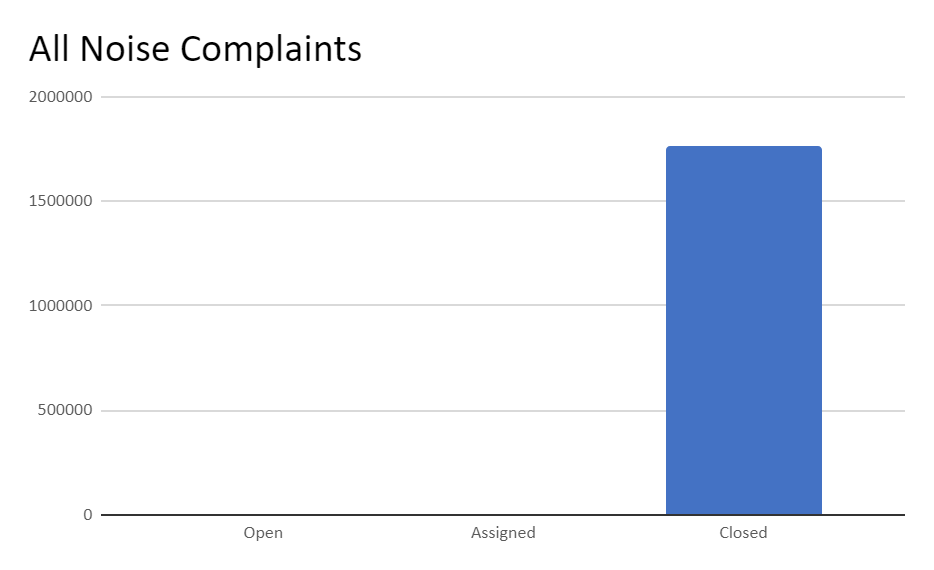
a)What is the number of open, assigned and closed issues for the following 3 complaint types:

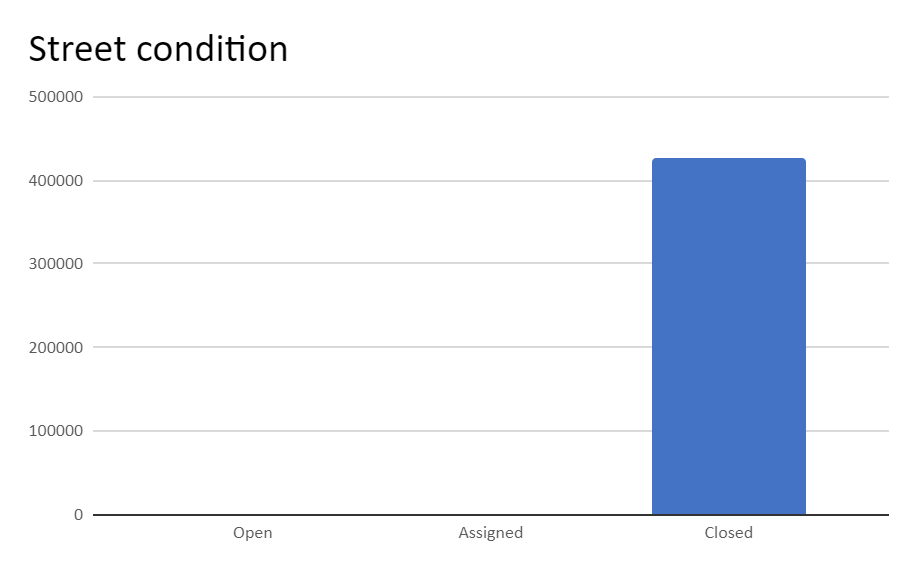
1.All noise complaints (See the column “Complaint Type”)

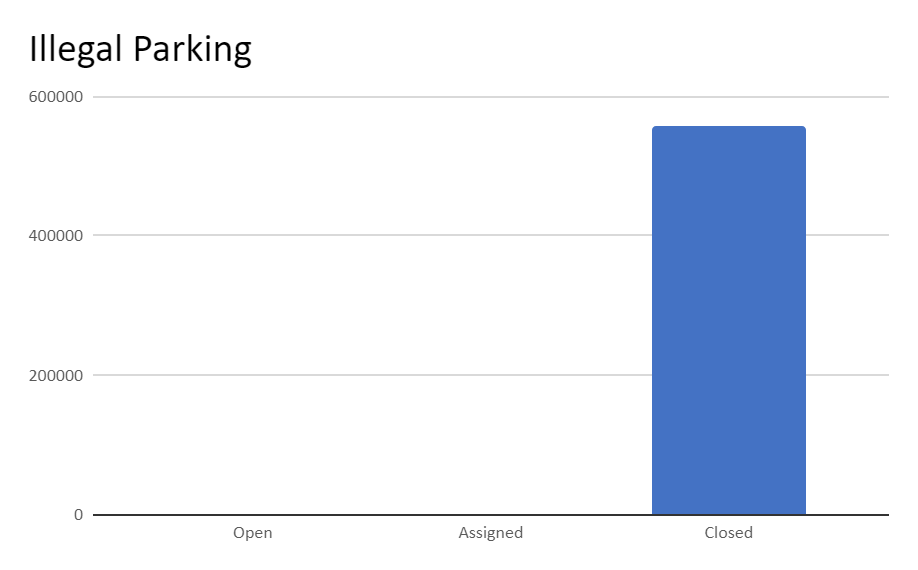
2. Street Condition

3. Illegal Parking

|  |  |  |  |
| --- | --- | --- | --- |
| Status | All noise complaints | Street condition | Illegal Parking |
| Open | 6311 | 1294 | 2073 |
| Assigned | 2053 | 366 | 835 |
| Closed | 1761077 | 425617 | 558302 |



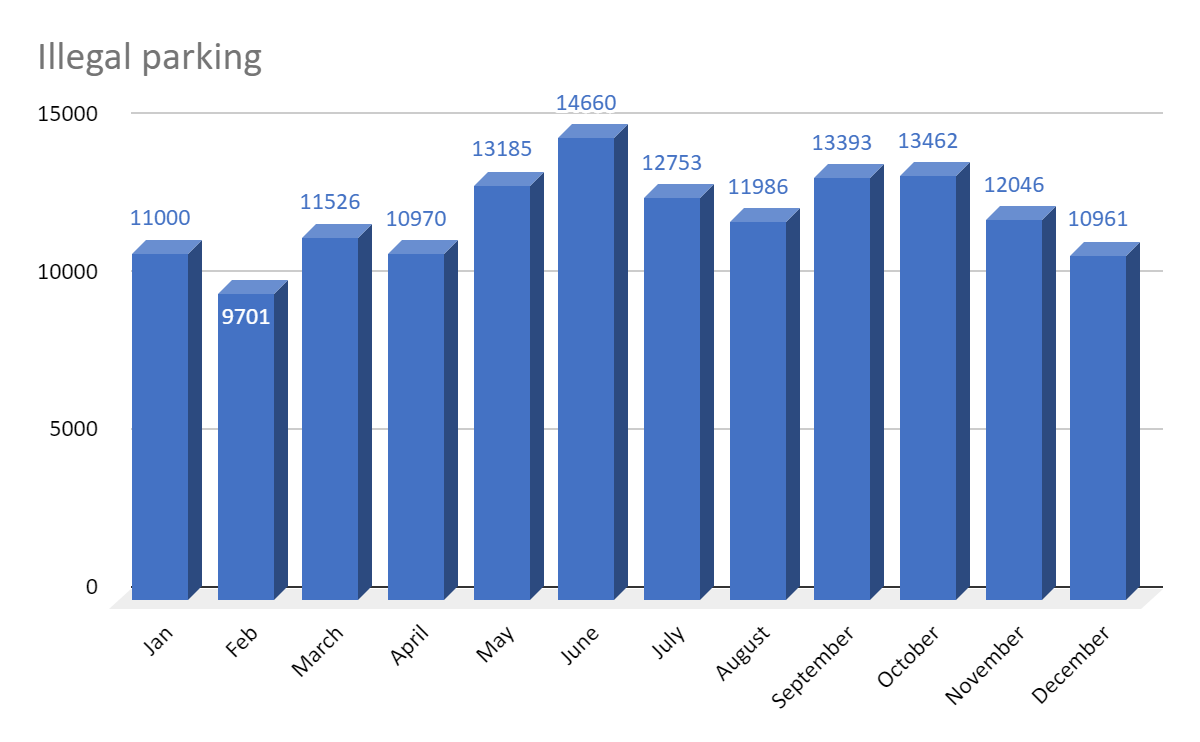


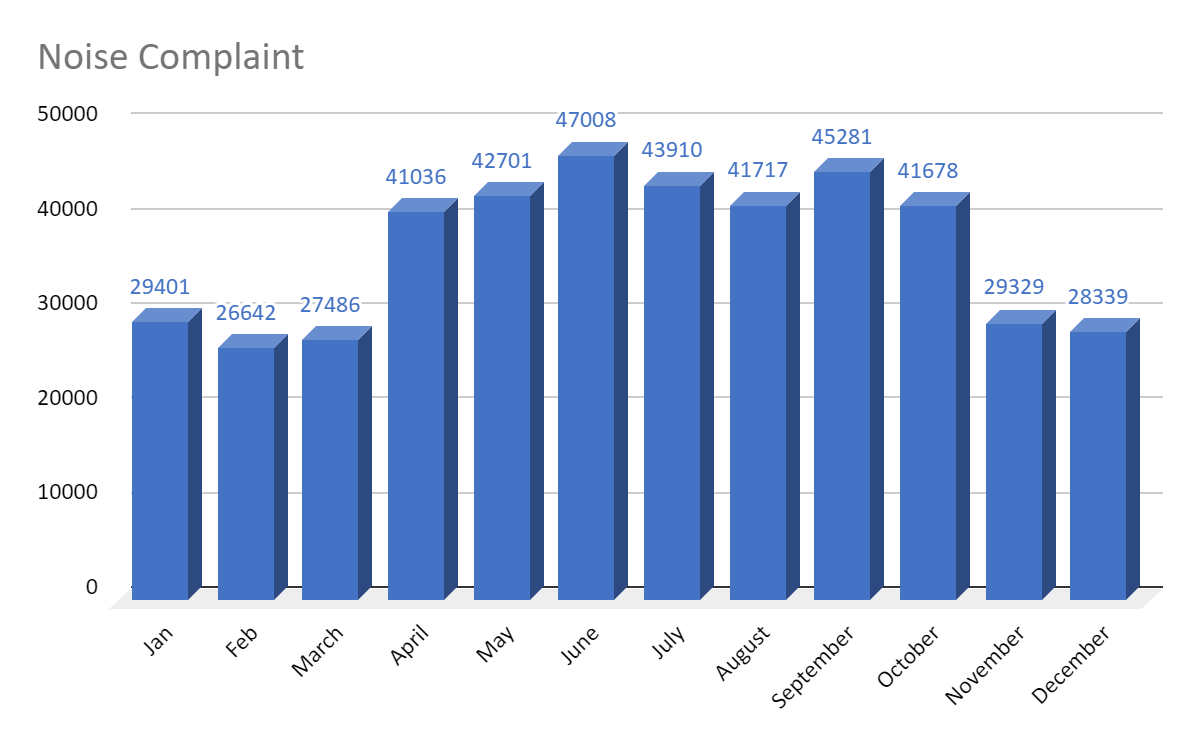


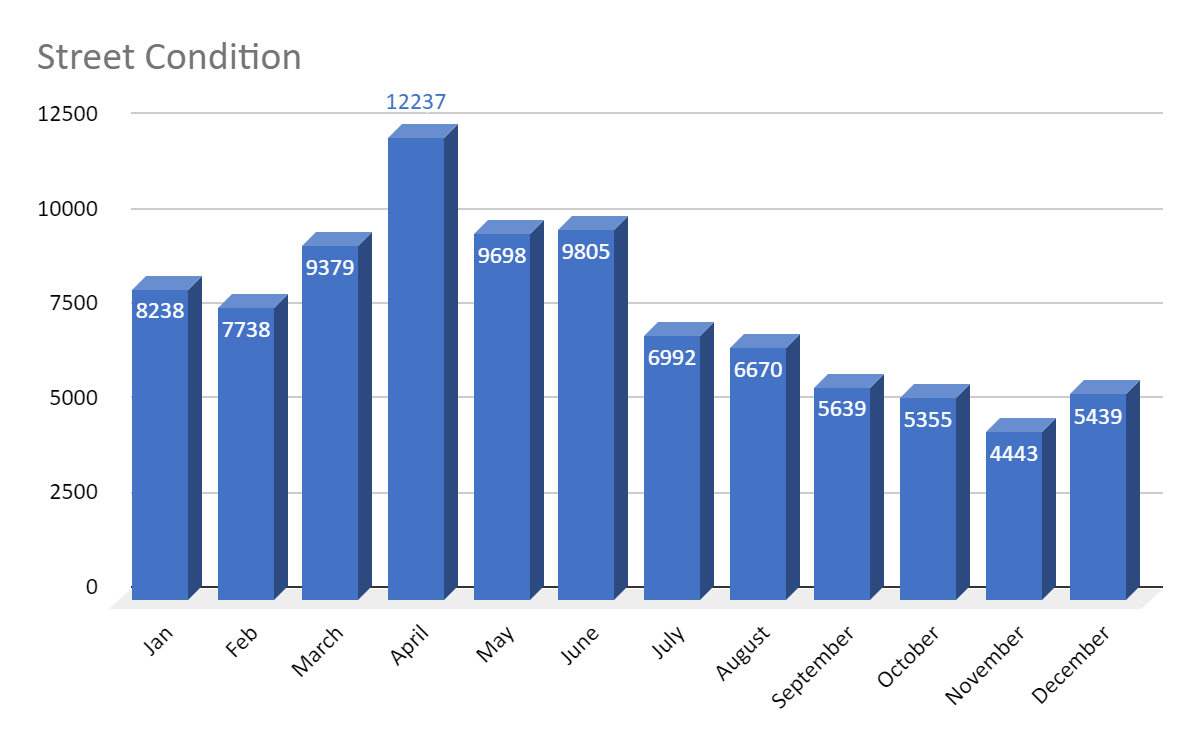
b) For the year 2017, for each quarter (Jan-Mar, April-June, July-Sept, Oct-Dec)

1)What is the average number of the above complaints per month?

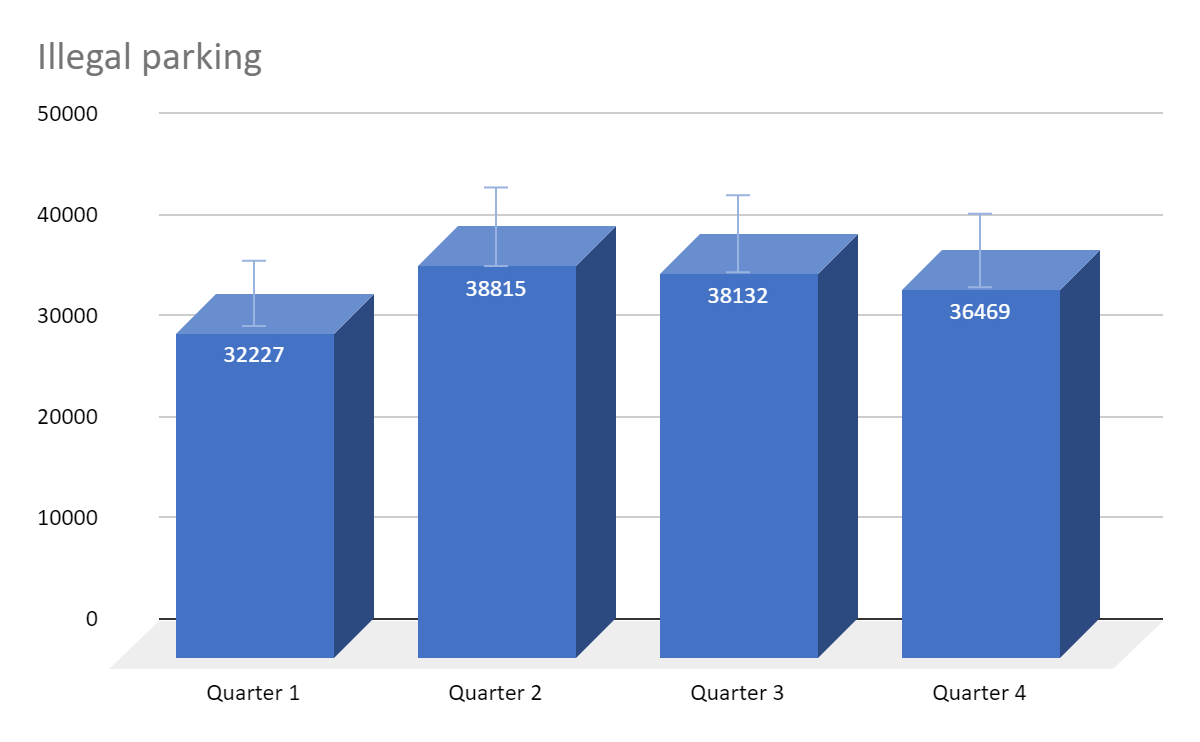
|  |  |  |  |
| --- | --- | --- | --- |
| Month | Illegal Parking | Noise Complaints | Street Condition |
| Jan | 11000 | 29401 | 8238 |
| Feb | 9701 | 26642 | 7738 |
| March | 11526 | 27486 | 9379 |
| April | 10970 | 41036 | 12237 |
| May | 13185 | 42701 | 9698 |
| June | 14660 | 47008 | 9805 |
| July | 12753 | 43910 | 6992 |
| August | 11986 | 41717 | 6670 |
| September | 13393 | 45281 | 5639 |
| October | 13462 | 41678 | 5355 |
| November | 12046 | 29329 | 4443 |
| December | 10961 | 28339 | 5439 |

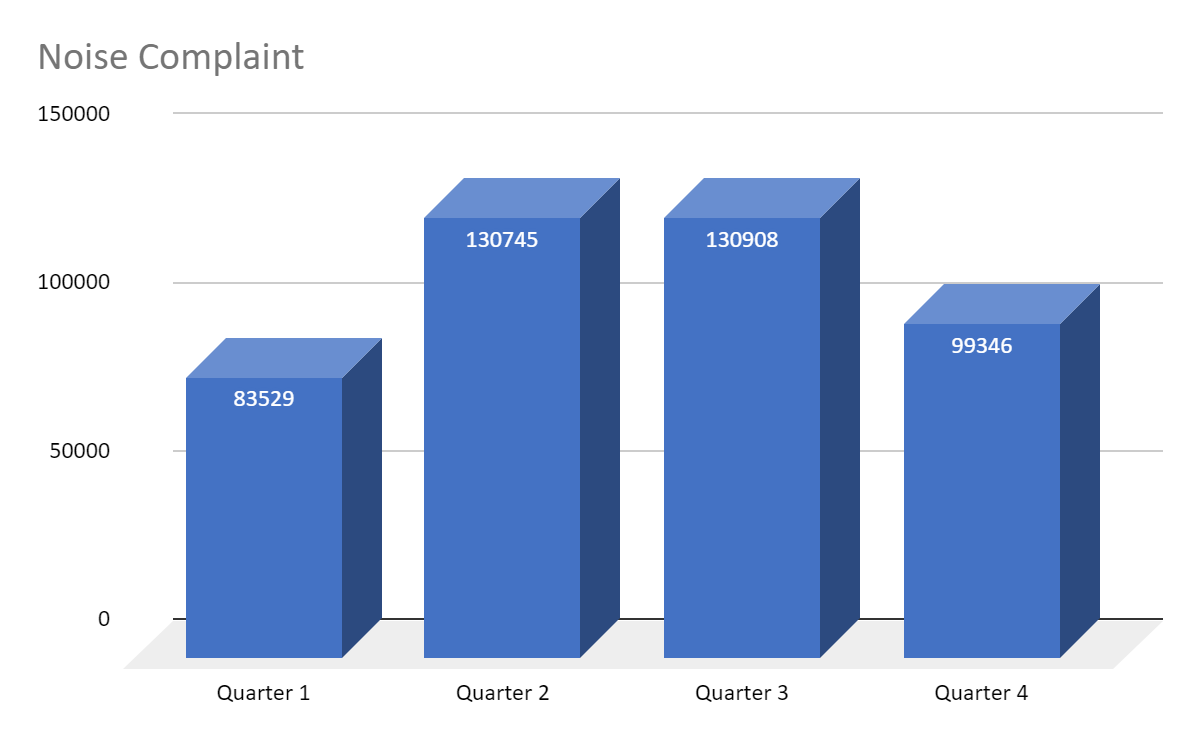


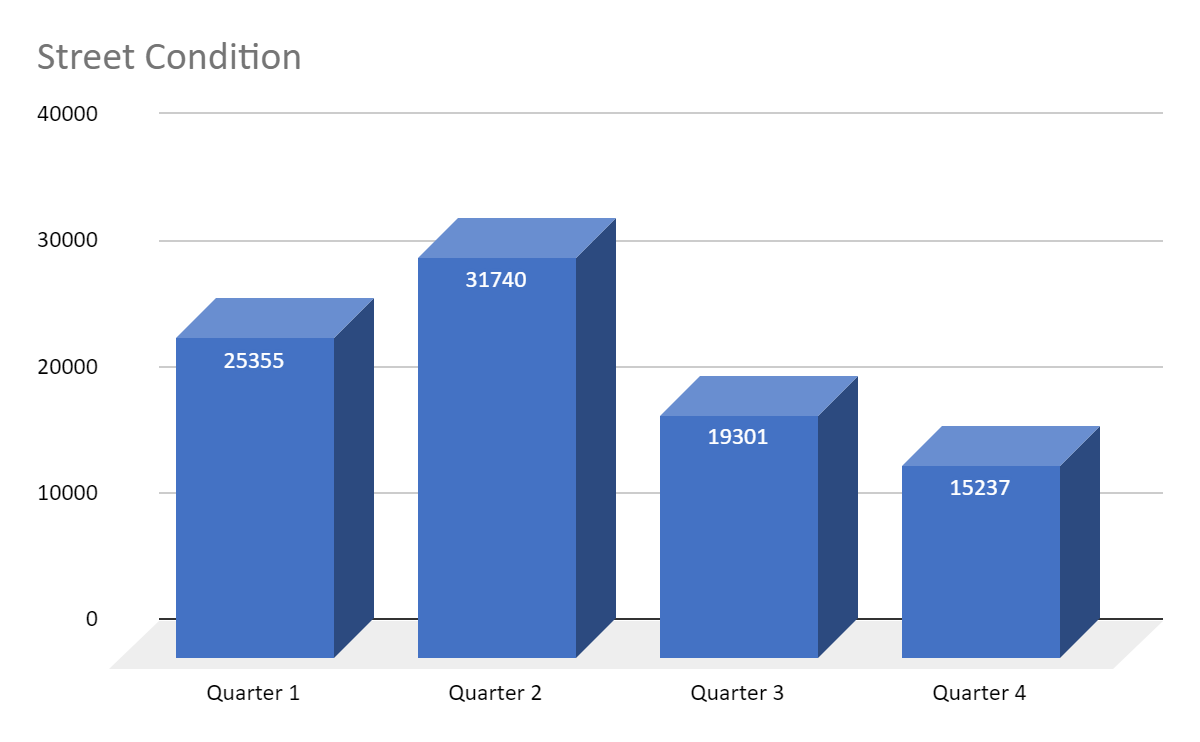




|  |  |  |  |
| --- | --- | --- | --- |
| Quarters | **Illegal Parking** | **All Noise complaints** | **Street Condition** |
| Quarter 1 | 32227 | 83529 | 25355 |
| Quarter 2 | 38815 | 130745 | 31740 |
| Quarter 3 | 38132 | 130908 | 19301 |
| Quarter 4 | 36469 | 99346 | 15237 |

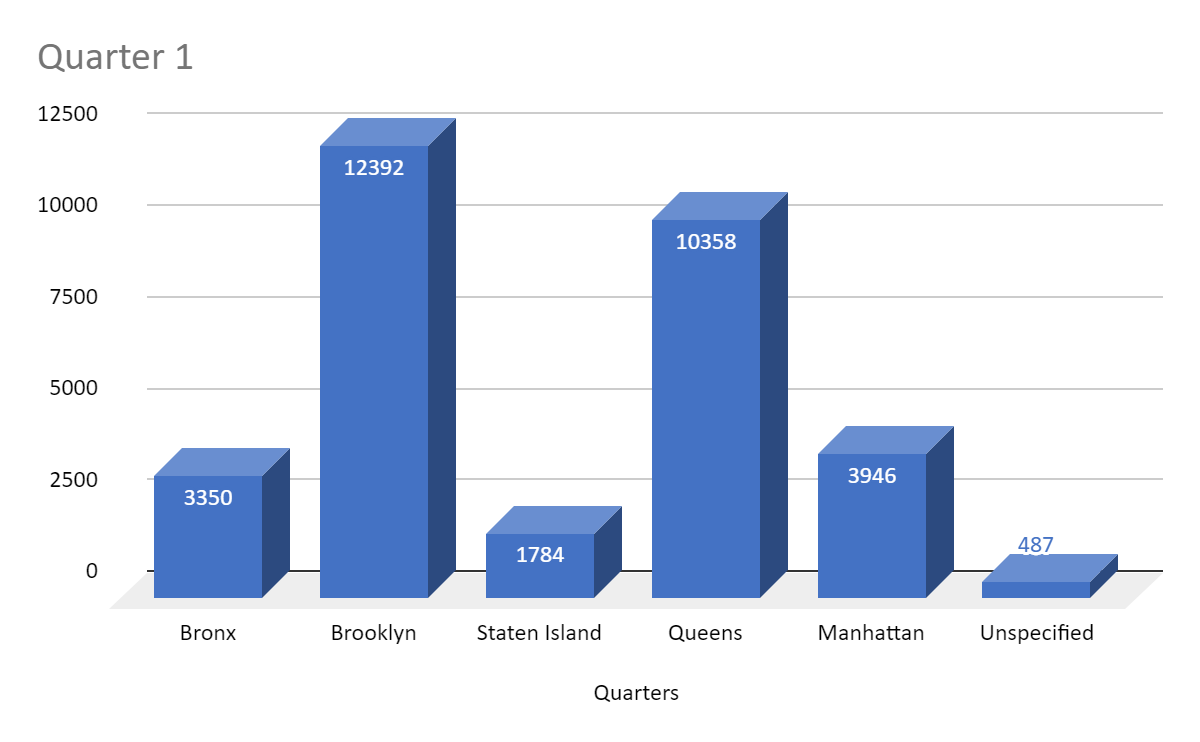


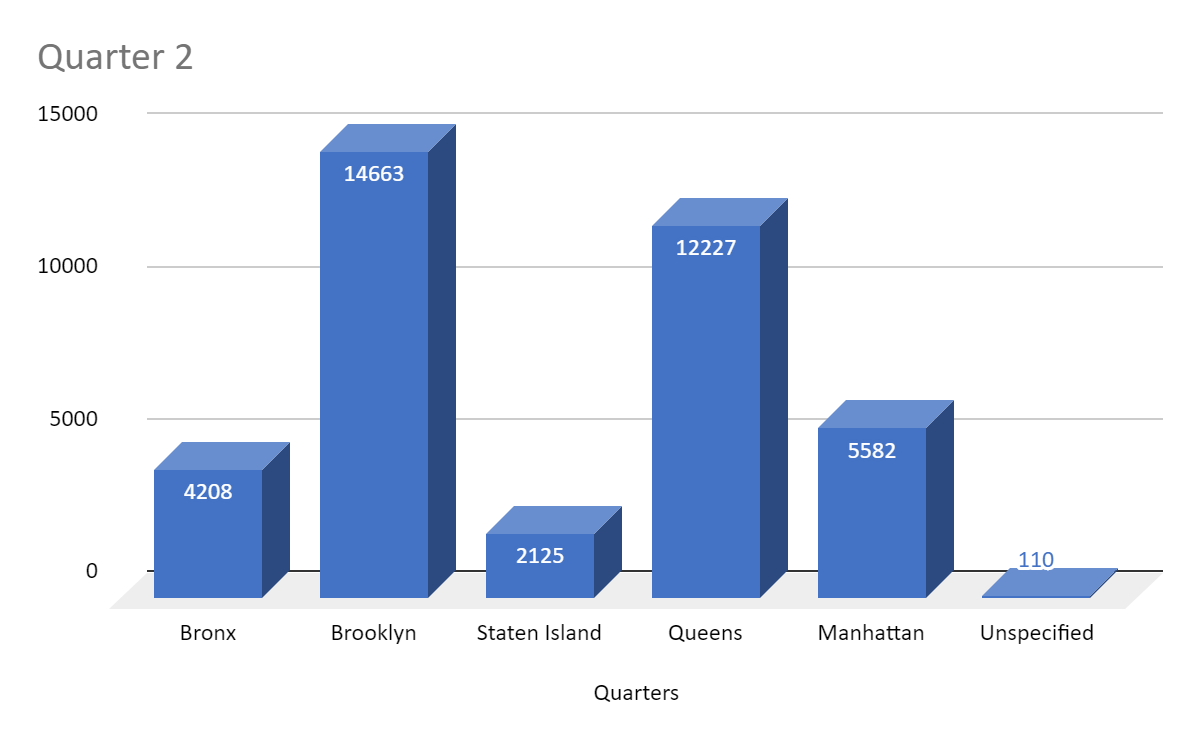


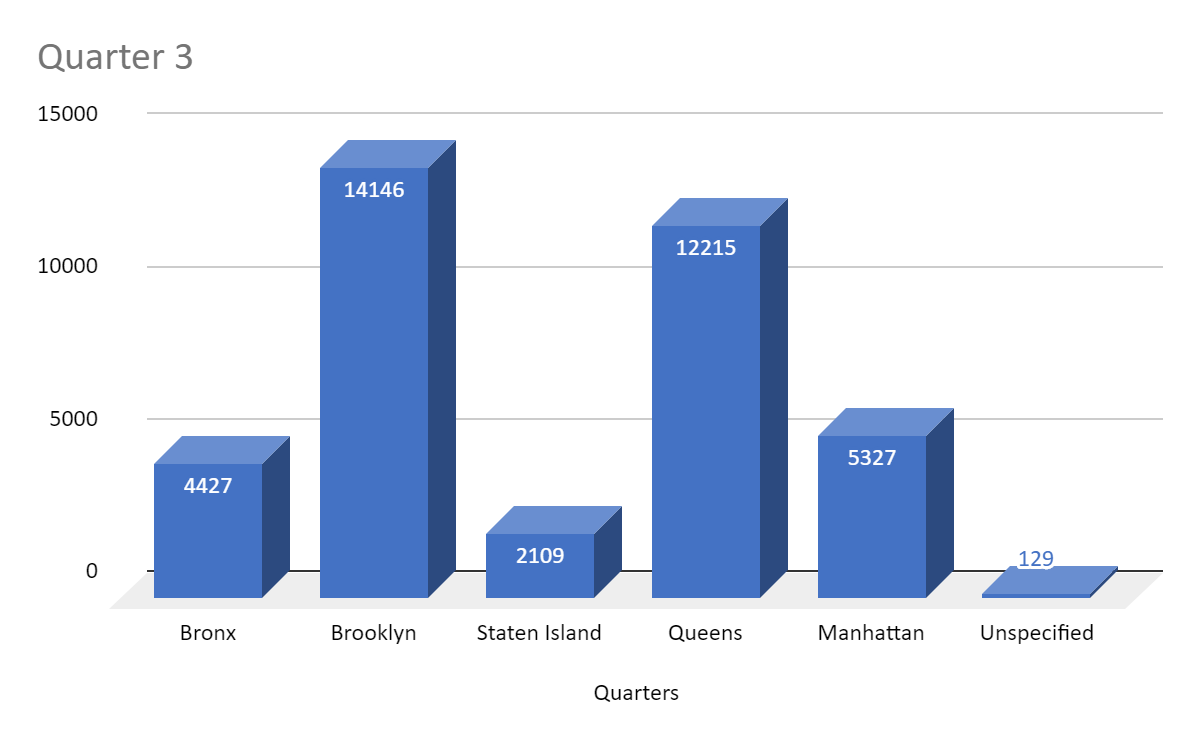


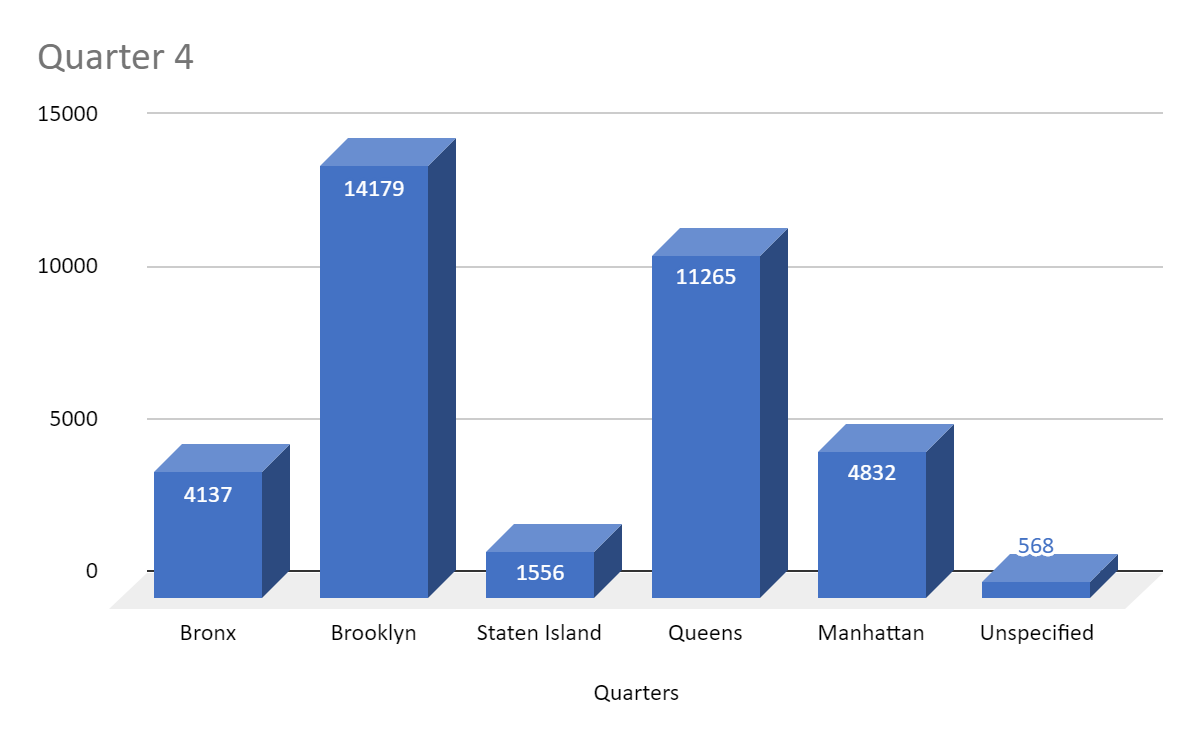
2.Which borough had the highest number of ”Illegal Parking” in each quarter?

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Quarters | **Bronx** | **Brooklyn** | **Staten Island** | **Queens** | **Manhattan** | **Unspecified** |
| **Quarter 1** | 3350 | 12392 | 1784 | 10358 | 3946 | 487 |
| **Quarter 2** | 4208 | 14663 | 2125 | 12227 | 5582 | 110 |
| **Quarter 3** | 4427 | 14146 | 2109 | 12215 | 5327 | 129 |
| **Quarter 4** | 4137 | 14179 | 1556 | 11265 | 4832 | 568 |









As you can observe, Brooklyn had the highest number of “Illegal Parking” in all Quarters.

3. Specific question: describe the order in which map and reduce tasks are executed in Hadoop and the duration of map and reduce phases for the different job execution steps.

Reduce job usually splits the input data-set into independent chunks which are processed by the map tasks in a completely parallel manner. A Map task is executed first.. The framework sorts the outputs of the maps, which are then input to the reduce tasks. Typically both the input and the output of the job are stored in a file-system. The framework takes care of scheduling tasks, monitoring them and re-executes the failed tasks.

The MapReduce framework operates exclusively on <key, value> pairs, that is, the framework views the input to the job as a set of <key, value> pairs and produces a set of <key, value> pairs as the output of the job, conceivably of different types.

The key and value classes have to be serializable by the framework and hence need to implement the Writable interface. Additionally, the key classes have to implement the WritableComparable interface to facilitate sorting by the framework.

**Below was the output from the code:**

----------------------------------------------------------------------------------------------------------------------------

PS D:\MBS Stuff\MBS Fall 2019\Intro to cloud\Week 4\Project> python complaint\_analytics.py s3://pgarias-bucket-cloud/311\_Se

rvice\_Requests\_from\_2015\_to\_Present.csv -r emr --output-dir=s3://aparna-hw2-us-east-2/wc\_out\_t9

Using configs in C:\Users\aparn\.mrjob.conf

Using s3://mrjob-2debcf6abc4e5185/tmp/ as our temp dir on S3

Creating temp directory C:\Users\aparn\AppData\Local\Temp\complaint\_analytics.aparn.20191030.015330.355265

writing master bootstrap script to C:\Users\aparn\AppData\Local\Temp\complaint\_analytics.aparn.20191030.015330.355265\b.sh

uploading working dir files to s3://mrjob-2debcf6abc4e5185/tmp/complaint\_analytics.aparn.20191030.015330.355265/files/wd...

Copying other local files to s3://mrjob-2debcf6abc4e5185/tmp/complaint\_analytics.aparn.20191030.015330.355265/files/

Created new cluster j-39RTMQGP20DDZ

Added EMR tags to cluster j-39RTMQGP20DDZ: \_\_mrjob\_label=complaint\_analytics, \_\_mrjob\_owner=aparn, \_\_mrjob\_version=0.6.10

Waiting for Step 1 of 1 (s-2AHSVE67GXPC) to complete...

PENDING (cluster is STARTING)

PENDING (cluster is STARTING)

PENDING (cluster is STARTING)

PENDING (cluster is STARTING)

PENDING (cluster is STARTING: Configuring cluster software)

PENDING (cluster is BOOTSTRAPPING: Running bootstrap actions)

PENDING (cluster is BOOTSTRAPPING: Running bootstrap actions)

PENDING (cluster is BOOTSTRAPPING: Running bootstrap actions)

PENDING (cluster is BOOTSTRAPPING: Running bootstrap actions)

master node is ec2-18-222-113-8.us-east-2.compute.amazonaws.com

Connect to resource manager at: http://localhost:40659/cluster

RUNNING for 0:00:14

RUNNING for 0:00:46

5.1% complete

RUNNING for 0:01:16

12.2% complete

RUNNING for 0:01:46

20.2% complete

RUNNING for 0:02:17

29.8% complete

RUNNING for 0:02:47

36.2% complete

RUNNING for 0:03:18

42.1% complete

RUNNING for 0:03:48

48.1% complete

RUNNING for 0:04:18

57.7% complete

RUNNING for 0:04:48

COMPLETED

Attempting to fetch counters from logs...

Looking for step log in /mnt/var/log/hadoop/steps/s-2AHSVE67GXPC on ec2-18-222-113-8.us-east-2.compute.amazonaws.com...

Parsing step log: ssh://ec2-18-222-113-8.us-east-2.compute.amazonaws.com/mnt/var/log/hadoop/steps/s-2AHSVE67GXPC/syslog.2

019-10-30-01

Parsing step log: ssh://ec2-18-222-113-8.us-east-2.compute.amazonaws.com/mnt/var/log/hadoop/steps/s-2AHSVE67GXPC/syslog

Counters: 56

File Input Format Counters

Bytes Read=5924711248

File Output Format Counters

Bytes Written=3424

File System Counters

FILE: Number of bytes read=11651379

FILE: Number of bytes written=40963179

FILE: Number of large read operations=0

FILE: Number of read operations=0

FILE: Number of write operations=0

HDFS: Number of bytes read=10947

HDFS: Number of bytes written=0

HDFS: Number of large read operations=0

HDFS: Number of read operations=89

HDFS: Number of write operations=0

S3: Number of bytes read=5924711248

S3: Number of bytes written=3424

S3: Number of large read operations=0

S3: Number of read operations=0

S3: Number of write operations=0

Job Counters

Data-local map tasks=89

Killed map tasks=2

Killed reduce tasks=1

Launched map tasks=89

Launched reduce tasks=13

Total megabyte-milliseconds taken by all map tasks=6225624576

Total megabyte-milliseconds taken by all reduce tasks=3310915584

Total time spent by all map tasks (ms)=4053141

Total time spent by all maps in occupied slots (ms)=194550768

Total time spent by all reduce tasks (ms)=1077772

Total time spent by all reduces in occupied slots (ms)=103466112

Total vcore-milliseconds taken by all map tasks=4053141

Total vcore-milliseconds taken by all reduce tasks=1077772

Map-Reduce Framework

CPU time spent (ms)=1409760

Combine input records=0

Combine output records=0

Failed Shuffles=0

GC time elapsed (ms)=70932

Input split bytes=10947

Map input records=10420595

Map output bytes=226727227

Map output materialized bytes=11696085

Map output records=4267658

Merged Map outputs=1157

Physical memory (bytes) snapshot=62279307264

Reduce input groups=61

Reduce input records=4267658

Reduce output records=61

Reduce shuffle bytes=11696085

Shuffled Maps =1157

Spilled Records=8535316

Total committed heap usage (bytes)=56546033664

Virtual memory (bytes) snapshot=354817527808

Shuffle Errors

BAD\_ID=0

CONNECTION=0

IO\_ERROR=0

WRONG\_LENGTH=0

WRONG\_MAP=0

WRONG\_REDUCE=0

job output is in s3://aparna-hw2-us-east-2/wc\_out\_t9/

Removing s3 temp directory s3://mrjob-2debcf6abc4e5185/tmp/complaint\_analytics.aparn.20191030.015330.355265/...

Removing temp directory C:\Users\aparn\AppData\Local\Temp\complaint\_analytics.aparn.20191030.015330.355265...

Removing log files in s3://mrjob-2debcf6abc4e5185/tmp/logs/j-39RTMQGP20DDZ/...

Killing our SSH tunnel (pid 15388)

Terminating cluster: j-39RTMQGP20DDZ

PS D:\MBS Stuff\MBS Fall 2019\Intro to cloud\Week 4\Project> aws s3 cp s3://aparna-hw2-us-east-2/wc\_out\_t9 testfolder9 --re

cursive

download: s3://aparna-hw2-us-east-2/wc\_out\_t9/\_SUCCESS to testfolder9\\_SUCCESS

download: s3://aparna-hw2-us-east-2/wc\_out\_t9/part-00000 to testfolder9\part-00000

download: s3://aparna-hw2-us-east-2/wc\_out\_t9/part-00008 to testfolder9\part-00008

download: s3://aparna-hw2-us-east-2/wc\_out\_t9/part-00002 to testfolder9\part-00002

download: s3://aparna-hw2-us-east-2/wc\_out\_t9/part-00001 to testfolder9\part-00001

download: s3://aparna-hw2-us-east-2/wc\_out\_t9/part-00006 to testfolder9\part-00006

download: s3://aparna-hw2-us-east-2/wc\_out\_t9/part-00003 to testfolder9\part-00003

download: s3://aparna-hw2-us-east-2/wc\_out\_t9/part-00004 to testfolder9\part-00004

download: s3://aparna-hw2-us-east-2/wc\_out\_t9/part-00007 to testfolder9\part-00007

download: s3://aparna-hw2-us-east-2/wc\_out\_t9/part-00010 to testfolder9\part-00010

download: s3://aparna-hw2-us-east-2/wc\_out\_t9/part-00012 to testfolder9\part-00012

download: s3://aparna-hw2-us-east-2/wc\_out\_t9/part-00005 to testfolder9\part-00005

download: s3://aparna-hw2-us-east-2/wc\_out\_t9/part-00009 to testfolder9\part-00009

download: s3://aparna-hw2-us-east-2/wc\_out\_t9/part-00011 to testfolder9\part-00011

----------------------------------------------------------------------------------------------------------------------------

For the above program, below are the time values for map and reduce tasks:

Total time spent by all map tasks (ms) = **4053141**

Total time spent by all reduce tasks (ms) = **1077772**

CPU time spent (ms) for Map reduce framework = **1409760**

**Code**:

sc.install\_pypi\_package("pandas")

# Load the dataset

#file\_path = "s3://an674-hw3-bucket/311\_Service\_Requests\_from\_2015\_to\_Present\_head\_1000.csv"

file\_path = "s3://pgarias-bucket-cloud/311\_Service\_Requests\_from\_2015\_to\_Present.csv"

df=spark.read.csv(file\_path,inferSchema = True,header=True)

# Print the initial count

print('Data count: ', df.count())

# Filter on 'Closed' status alone

df=df.filter((df.Status=='Closed'))

print('Data count for \'Closed status\': ', df.count())

# Drop columns that are not required

keep\_columns = ['Created Date', 'Closed Date', 'Agency','Complaint Type', 'Location Type',

'Facility Type', 'Borough']

all\_columns = df.columns

drop\_columns = [c for c in all\_columns if c not in keep\_columns]

df = df.drop(\*drop\_columns)

print('Relevant columns: ', df.columns)

# Drop N/A

print('Count before dropna: ', df.count())

df = df.dropna()

print('Count after dropna: ', df.count())

# Remove Unspecified data

df = df.filter(df['Borough'] != 'Unspecified')

print('Count after removing unspecified data: ', df.count())

# Compute the duration in seconds and extract month of created date

from pyspark.sql.functions import month, to\_timestamp

date\_format = 'MM/dd/yyyy hh:mm:ss aa'

duration\_column = 'Duration'

created\_month\_column = 'Created Date\_month'

df = df.withColumn(created\_month\_column,month(to\_timestamp('Created Date', date\_format)))

df = df.withColumn(duration\_column,(

(to\_timestamp('Closed Date', date\_format).cast('int') -to\_timestamp('Created Date', date\_format).cast('int')

)/3600).cast('int'))

print('Computed the duration')

# Remove the data whose duration is 0 ['Created Date'] == ['Closed Date']

print('Count before timestamp filtering: ', df.count())

df = df.filter(df[duration\_column] > 0)

print('Count after timestamp filtering: ', df.count())

df = df.drop('Created Date', 'Closed Date')

# Normalize the complaint types

from pyspark.sql.functions import when, col

df = df.withColumn('Complaint Type',

when(col('Complaint Type').startswith('Noise'),'Noise')

.when(col('Complaint Type').startswith('Advocate'),'Advocate')

.when(col('Complaint Type').startswith('Bus Stop'),'Bus Stop Shleter Complaint/Placement')

.when(col('Complaint Type').startswith('Dead'),'Dead/Dying')

.when(col('Complaint Type').startswith('Damaged'),'Dead/Dying')

.when(col('Complaint Type').startswith('Derelict'),'Derelict Vehicle')

.when(col('Complaint Type').startswith('DOF Parking'),'DOF Parking')

.when(col('Complaint Type').startswith('DOF Property'),'DOF Property')

.when(col('Complaint Type').startswith('Ferry'),'Ferry complaint/inquiry/permit')

.when(col('Complaint Type').startswith('Highway Sign'),'Highway Sign-Damaged/dangling/missing')

.when(col('Complaint Type').startswith('For Hire'),'For Hire Vehicle Complaint/Report')

.when(col('Complaint Type').startswith('Street Sign'),'Street Sign-Damaged/Dangling/Missing')

.when(col('Complaint Type').startswith('Sweeping'),'Sweeping-Missed/Inadequate')

.otherwise(col('Complaint Type')))

# Get the maximum number of categories in a dataframe

categorical\_columns = list(df.columns)

categorical\_columns.remove(duration\_column)

label\_column = duration\_column

max\_categories = 0

for col in categorical\_columns:

item=df.select(col).distinct().count()

if item > max\_categories:

max\_categories = item

print('Category count for ', col, ' is: ', item)

print('Max categories: ', max\_categories)

# Prepare the data

from pyspark.ml import Pipeline

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

from pyspark.sql.functions import col

categoricalCols = categorical\_columns

indexers = [ StringIndexer(inputCol=c, outputCol="{0}\_indexed".format(c))

for c in categoricalCols ]

# default setting: dropLast=True

encoders = [ OneHotEncoder(inputCol=indexer.getOutputCol(),

outputCol="{0}\_encoded".format(indexer.getOutputCol()))

for indexer in indexers ]

assembler = VectorAssembler(inputCols=[encoder.getOutputCol() for encoder in encoders]

, outputCol="features")

pipeline = Pipeline(stages=indexers + encoders + [assembler])

model=pipeline.fit(df)

data = model.transform(df)

sc.install\_pypi\_package("scipy")

def remove\_outlier(column, data):

df\_outlier = data.select(column)

from scipy import stats

import numpy as np

df\_outlier\_pandas = df\_outlier.toPandas()

z = np.abs(stats.zscore(df\_outlier\_pandas))

df\_outlier\_pandas = df\_outlier\_pandas[((z < 3) & (z > -3)).all(axis=1)]

df\_outlier = spark.createDataFrame(df\_outlier\_pandas)

data = data.join(df\_outlier, data[column] == df\_outlier[column],"left\_semi")

return data

# Prepare for the outliers using the z score

'''

Category count for Agency is: 15

Category count for Complaint Type is: 130

Category count for Location Type is: 129

Category count for Facility Type is: 4

Category count for Borough is: 6

Category count for Created Date\_month is: 12

'''

print('Count before removing the outliers: ', data.count())

outlier\_columns = ['Complaint Type\_indexed', 'Location Type\_indexed']

print('Outlier columns: ', outlier\_columns)

for col in outlier\_columns:

data = remove\_outlier(col, data)

print('Count after removing the outliers: ', data.count())

# Prediction part begins

from pyspark.ml import Pipeline

from pyspark.ml.regression import LinearRegression

from pyspark.ml.feature import VectorIndexer

from pyspark.ml.evaluation import RegressionEvaluator

featureIndexer = VectorIndexer(inputCol="features", \

outputCol="indexedFeatures",\

maxCategories=max\_categories).fit(data)

data = featureIndexer.transform(data)

#Dividing the dataset into a trainingData and testData with 30% of the data used as testData

(trainingData, testData) = data.randomSplit([0.7, 0.3])

from pyspark.ml.regression import RandomForestRegressor

# Define LinearRegression algorithm

prediction\_column = "Prediction\_Duration"

rf = RandomForestRegressor(featuresCol="indexedFeatures", labelCol=duration\_column,

predictionCol=prediction\_column, seed=42)

# featuresCol="indexedFeatures",numTrees=2, maxDepth=2, seed=42

# Chain indexer and tree in a Pipeline

from pyspark.ml import Pipeline

pipeline = Pipeline(stages=[featureIndexer, rf])

model = pipeline.fit(trainingData)

predictions = model.transform(testData)

# Select example rows to display.

predictions.select("features", duration\_column, prediction\_column).show(50)

# Select (prediction, true label) and compute test error

evaluator = RegressionEvaluator(

labelCol=duration\_column, predictionCol=prediction\_column, metricName="rmse")

rmse = evaluator.evaluate(predictions)

print("Root Mean Squared Error (RMSE) on test data = %g" % rmse)

import pandas as pd

def ExtractFeatureImp(featureImp, dataset, featuresCol):

list\_extract = []

for i in dataset.schema[featuresCol].metadata["ml\_attr"]["attrs"]:

list\_extract = list\_extract + dataset.schema[featuresCol].metadata["ml\_attr"]["attrs"][i]

varlist = pd.DataFrame(list\_extract)

varlist['score'] = varlist['idx'].apply(lambda x: featureImp[x])

return(varlist.sort\_values('score', ascending = False))

ExtractFeatureImp(model.stages[-1].featureImportances, predictions, "features").head(10)

**Data Cleaning results:**

**Data count: 10420594**

**Data count for 'Closed status': 9871711**

**Relevant columns: ['Created Date', 'Closed Date', 'Agency', 'Complaint Type', 'Location Type', 'Facility Type', 'Borough']**

**Count before dropna: 9871711**

**Count after dropna: 7775683**

**Count after removing unspecified data: 7642465**

**Computed the duration**

**Count before timestamp filtering: 7642465**

**Count after timestamp filtering: 6927643**

**Category count for Agency is: 14**

**Category count for Complaint Type is: 126**

**Category count for Location Type is: 124**

**Category count for Facility Type is: 4**

**Category count for Borough is: 5**

**Category count for Created Date\_month is: 12**

**Max categories: 126**

**Count before removing the outliers: 6927643**

**Outlier columns: ['Complaint Type\_indexed', 'Location Type\_indexed']**

**Count after removing the outliers: 6642369**

The data initially had about 10,420,594 rows. The rows having status as “Closed” were filtered from the entire set. There were a total of 9,871,711 rows with “Closed” status. Agency, Complaint Type, Location Type, Facility Type and Borough were the columns chosen as relevant to the problem statement. Among these, those rows that had null values for any of the columns were removed, and those that had the same values for Created and Closed dates were removed. Apart from this, the timestamp duration was calculated from the created and closed dates. The number of rows were reduced to 6,927,643 rows after timestamp filtering. The final count after removing the outliers is 6,642,369.

For this dataset, except duration, all the others were categorical columns and these were converted to indexed columns using String Indexer. Further, since every attribute had multiclass variables, one hot encoder was used to convert them to numerical data, and this was transformed to feature vector to be used in an algorithm.

**Random Forest Regressor:**

The data was then split as 70% training and the remaining 30% as test data. We first tried with Decision trees, and then moved to Random Forest Regressor since it controls overfitting and works with multiple trees.

Outliers were removed by using the Z-score method. Rows containing z-scores <-3 and >3 are excluded from the dataset and the model was trained using the feature indexes and labeled data. This was then tested in the rest of the 30% data . The prediction results are as follows:

**Prediction Results**

+--------------------+--------+-------------------+

| features|Duration|Prediction\_Duration|

+--------------------+--------+-------------------+

|(279,[10,37,146,2...| 270| 306.3634986260086|

|(279,[4,33,146,26...| 53| 903.5002243069682|

|(279,[4,33,146,26...| 215| 903.5002243069682|

|(279,[4,33,146,26...| 302| 903.5002243069682|

|(279,[4,33,146,26...| 649| 903.5002243069682|

|(279,[4,33,146,26...| 1074| 943.6757831026404|

|(279,[4,33,146,26...| 3142| 943.6757831026404|

|(279,[4,33,146,26...| 28| 943.6757831026404|

|(279,[4,33,146,26...| 53| 943.6757831026404|

|(279,[4,33,146,26...| 597| 943.6757831026404|

|(279,[4,33,146,26...| 1126| 943.6757831026404|

|(279,[4,33,146,26...| 7104| 943.6757831026404|

|(279,[4,33,146,26...| 13818| 943.6757831026404|

|(279,[4,33,146,26...| 27| 943.6757831026404|

|(279,[4,33,146,26...| 1991| 943.6757831026404|

|(279,[4,33,146,26...| 21| 958.6604033472786|

|(279,[4,33,146,26...| 30| 958.6604033472786|

|(279,[4,33,146,26...| 125| 958.6604033472786|

|(279,[4,33,146,26...| 141| 958.6604033472786|

|(279,[4,33,146,26...| 141| 958.6604033472786|

|(279,[4,33,146,26...| 353| 958.6604033472786|

|(279,[4,33,146,26...| 457| 958.6604033472786|

|(279,[4,33,146,26...| 1096| 958.6604033472786|

|(279,[4,33,146,26...| 6531| 958.6604033472786|

|(279,[4,33,146,26...| 28| 932.473765589448|

|(279,[4,33,146,26...| 165| 932.473765589448|

|(279,[4,33,146,26...| 206| 932.473765589448|

|(279,[4,33,146,26...| 546| 932.473765589448|

|(279,[4,33,146,26...| 884| 932.473765589448|

|(279,[4,33,146,26...| 1170| 932.473765589448|

|(279,[4,33,146,26...| 2045| 932.473765589448|

|(279,[4,33,146,26...| 5634| 932.473765589448|

|(279,[4,33,146,26...| 14044| 932.473765589448|

|(279,[4,33,146,26...| 13| 969.6514540140518|

|(279,[4,33,146,26...| 49| 969.6514540140518|

|(279,[4,33,146,26...| 194| 969.6514540140518|

|(279,[4,33,146,26...| 734| 969.6514540140518|

|(279,[4,33,146,26...| 959| 969.6514540140518|

|(279,[4,33,146,26...| 1002| 969.6514540140518|

|(279,[4,33,146,26...| 1076| 969.6514540140518|

|(279,[4,33,146,26...| 3| 943.6757831026404|

|(279,[4,33,146,26...| 197| 943.6757831026404|

|(279,[4,33,146,26...| 251| 943.6757831026404|

|(279,[4,33,146,26...| 380| 943.6757831026404|

|(279,[4,33,146,26...| 385| 943.6757831026404|

|(279,[4,33,146,26...| 476| 943.6757831026404|

|(279,[4,33,146,26...| 593| 943.6757831026404|

|(279,[4,33,146,26...| 670| 943.6757831026404|

|(279,[4,33,146,26...| 673| 943.6757831026404|

|(279,[4,33,146,26...| 839| 943.6757831026404|

+--------------------+--------+-------------------+

|  |  |  |
| --- | --- | --- |
| **features** | **Duration** | **Prediction\_Duration** |
| **(279,[10,37,146,2...** | **270** | **306.3634986** |
| **(279,[4,33,146,26...** | **53** | **903.5002243** |
| **(279,[4,33,146,26...** | **215** | **903.5002243** |
| **(279,[4,33,146,26...** | **302** | **903.5002243** |
| **(279,[4,33,146,26...** | **649** | **903.5002243** |
| **(279,[4,33,146,26...** | **1074** | **943.6757831** |
| **(279,[4,33,146,26...** | **3142** | **943.6757831** |
| **(279,[4,33,146,26...** | **28** | **943.6757831** |
| **(279,[4,33,146,26...** | **53** | **943.6757831** |
| **(279,[4,33,146,26...** | **597** | **943.6757831** |
| **(279,[4,33,146,26...** | **1126** | **943.6757831** |
| **(279,[4,33,146,26...** | **7104** | **943.6757831** |
| **(279,[4,33,146,26...** | **13818** | **943.6757831** |
| **(279,[4,33,146,26...** | **27** | **943.6757831** |
| **(279,[4,33,146,26...** | **1991** | **943.6757831** |
| **(279,[4,33,146,26...** | **21** | **958.6604033** |
| **(279,[4,33,146,26...** | **30** | **958.6604033** |
| **(279,[4,33,146,26...** | **125** | **958.6604033** |
| **(279,[4,33,146,26...** | **141** | **958.6604033** |
| **(279,[4,33,146,26...** | **141** | **958.6604033** |
| **(279,[4,33,146,26...** | **353** | **958.6604033** |
| **(279,[4,33,146,26...** | **457** | **958.6604033** |
| **(279,[4,33,146,26...** | **1096** | **958.6604033** |
| **(279,[4,33,146,26...** | **6531** | **958.6604033** |
| **(279,[4,33,146,26...** | **28** | **932.4737656** |
| **(279,[4,33,146,26...** | **165** | **932.4737656** |
| **(279,[4,33,146,26...** | **206** | **932.4737656** |
| **(279,[4,33,146,26...** | **546** | **932.4737656** |
| **(279,[4,33,146,26...** | **884** | **932.4737656** |
| **(279,[4,33,146,26...** | **1170** | **932.4737656** |
| **(279,[4,33,146,26...** | **2045** | **932.4737656** |
| **(279,[4,33,146,26...** | **5634** | **932.4737656** |
| **(279,[4,33,146,26...** | **14044** | **932.4737656** |
| **(279,[4,33,146,26...** | **13** | **969.651454** |
| **(279,[4,33,146,26...** | **49** | **969.651454** |
| **(279,[4,33,146,26...** | **194** | **969.651454** |
| **(279,[4,33,146,26...** | **734** | **969.651454** |
| **(279,[4,33,146,26...** | **959** | **969.651454** |
| **(279,[4,33,146,26...** | **1002** | **969.651454** |
| **(279,[4,33,146,26...** | **1076** | **969.651454** |
| **(279,[4,33,146,26...** | **3** | **943.6757831** |
| **(279,[4,33,146,26...** | **197** | **943.6757831** |
| **(279,[4,33,146,26...** | **251** | **943.6757831** |
| **(279,[4,33,146,26...** | **380** | **943.6757831** |
| **(279,[4,33,146,26...** | **385** | **943.6757831** |
| **(279,[4,33,146,26...** | **476** | **943.6757831** |
| **(279,[4,33,146,26...** | **593** | **943.6757831** |
| **(279,[4,33,146,26...** | **670** | **943.6757831** |
| **(279,[4,33,146,26...** | **673** | **943.6757831** |
| **(279,[4,33,146,26...** | **839** | **943.6757831** |
| **(279,[4,33,146,26...** | **839** | **943.6757831** |

**Root Mean square:**

Root Mean Squared Error (RMSE) on test data = 1024.67

**Feature Importance:**

idx name score

50 50 Complaint Type\_indexed\_encoded\_New Tree Request 0.356957

42 42 Complaint Type\_indexed\_encoded\_Overgrown Tree/... 0.179886

4 4 Agency\_indexed\_encoded\_DPR 0.170705

142 142 Location Type\_indexed\_encoded\_Street 0.035688

261 261 Facility Type\_indexed\_encoded\_N/A 0.035434

1 1 Agency\_indexed\_encoded\_NYPD 0.033546

33 33 Complaint Type\_indexed\_encoded\_Dead/Dying 0.022142

146 146 Location Type\_indexed\_encoded\_Park 0.020534

48 48 Complaint Type\_indexed\_encoded\_Root/Sewer/Side... 0.018931

264 264 Borough\_indexed\_encoded\_BROOKLYN 0.018122

From the results, it is seen that Complaint Type with value “**Complaint Type-New Trees Request**” was the feature with highest importance.