Big Data Processing

ECS 765P

Coursework

Ethereum Analysis

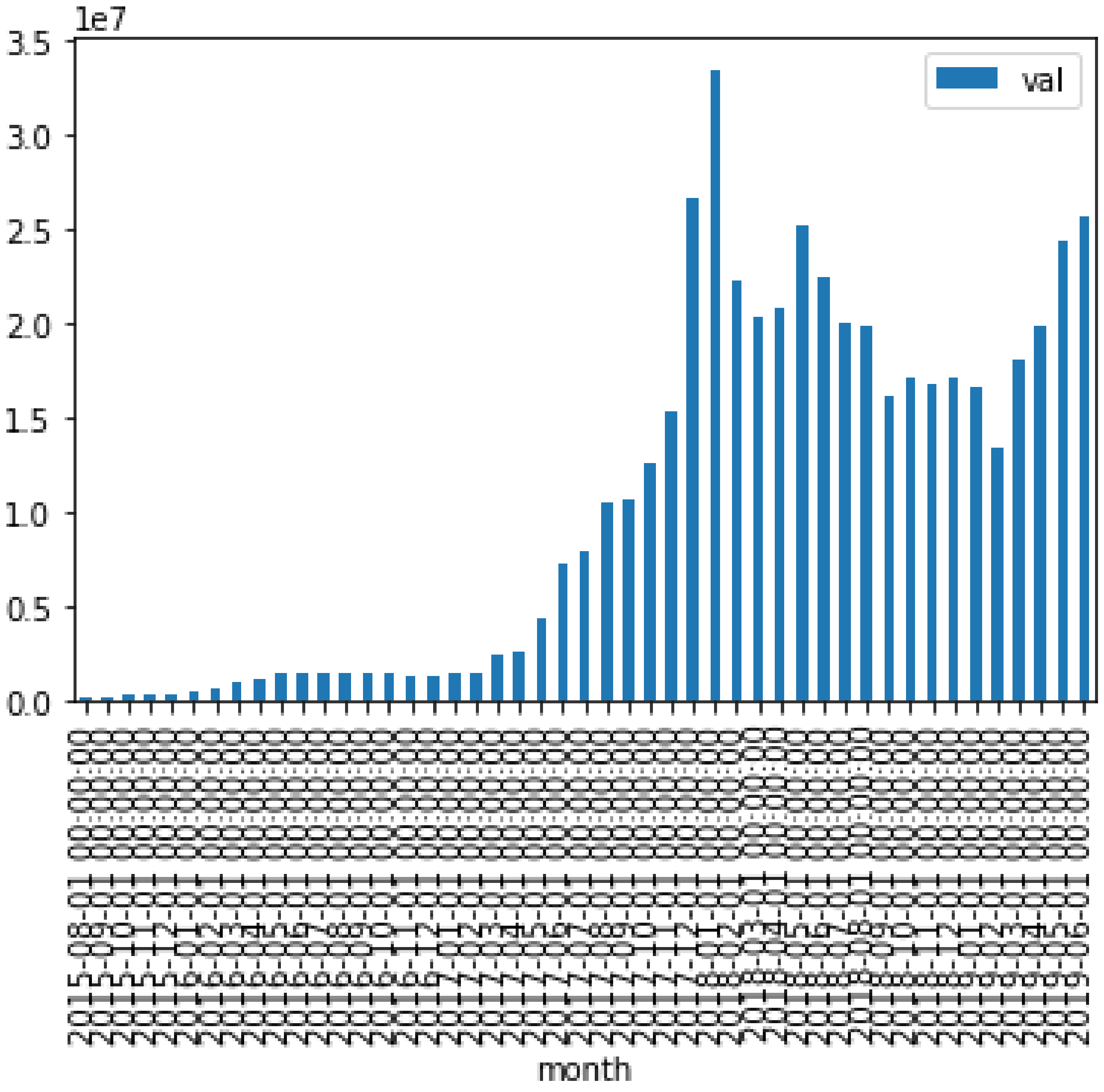
Submitted By:
Sparsh Shubham
MSc Big Data Science, December 2020
EC20063

Table of Content

Part A

1) Create a bar plot showing the number of transactions occurring every month between the start and end of the dataset.

Job ID:
http://andromeda.student.eecs.qmul.ac.uk:8088/proxy/application_1607539937312_6634



Key:
Val - Number of transactions in that month
Y-axis - scientific notation of the number of transactions

<u>Method</u>
Mapper:

1. Reads the file line by line and splits at ‘,’. Then timestamp is converted to time.
2. It yields, key (month and year) and value (1)

Reducer:

1. Aggregates the count of transactions and yields it.

Plotting:

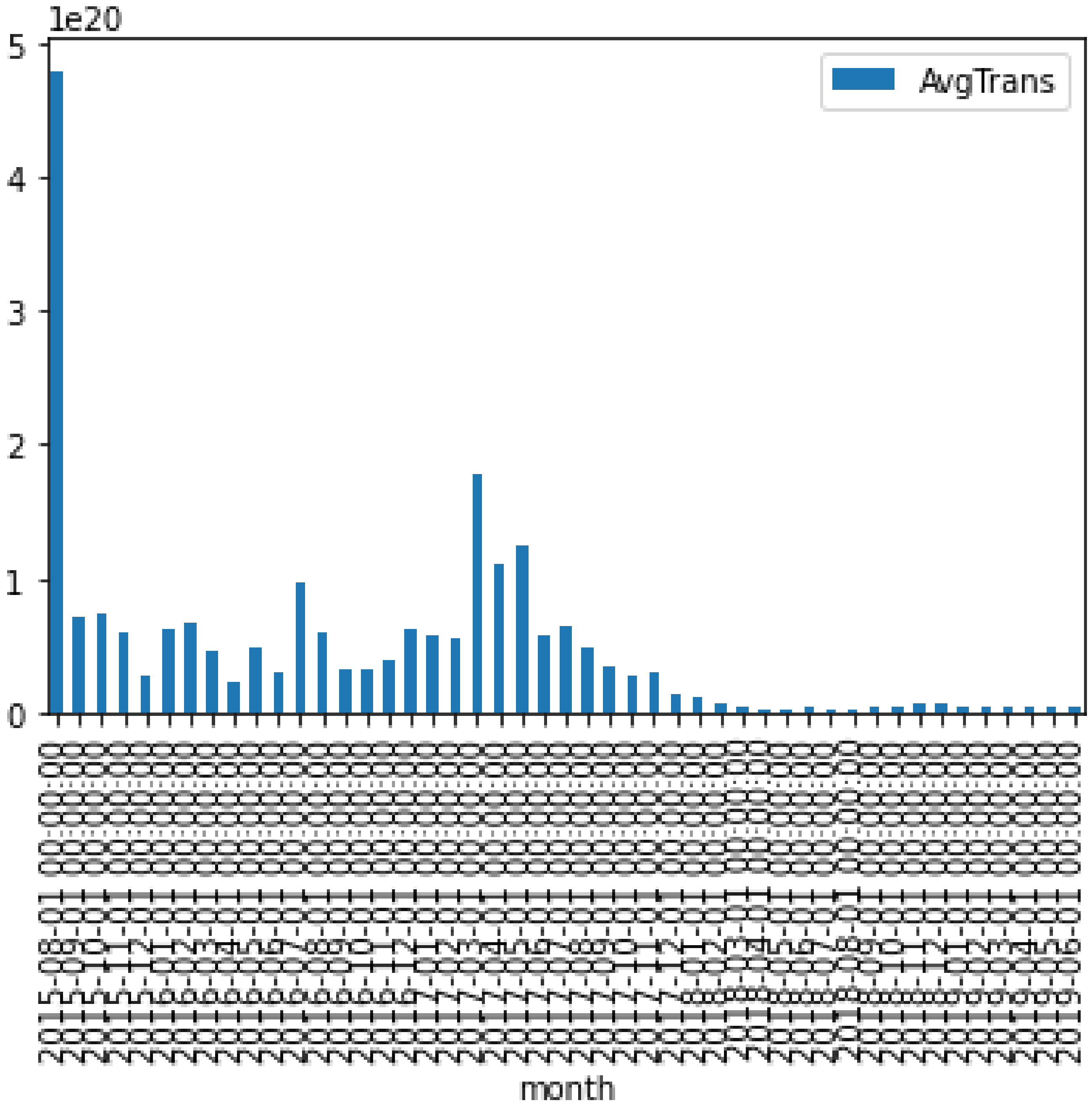
1. The output is loaded as a dataframe, cleaned and plotted into a bar plot.

Sample Output:
```
[1, 2017]   1409664
[1, 2019]   16569597
[10, 2015]   205045
[10, 2017]   12570063
[11, 2016]   1301586
[11, 2018]   16713911
[12, 2015]   347092
[12, 2017]   26687692
[2, 2016]   520040
[2, 2018]   22231978
[3, 2017]   2426471
[3, 2019]   18029582
```

Observation: The number of transactions follows the rise and fall of the cryptocurrency bubble of 2018.

2) Create a bar plot showing the average value of a transaction in each month between the start and end of the dataset.

Job ID:
http://andromeda.student.eecs.qmul.ac.uk:8088/proxy/application_1607539937312_9271



The average value of a transaction

Key:
Y-axis: in scientific notation
Method

Mapper:

1. Reads the file line by line and splits at ‘,’. Then timestamp is converted to time.
2. It yields, key (month and year) and value (1, value of the transaction)

Reducer:

1. Aggregates the count of transactions and the total value of transactions in the month
2. Then divides the total by count and yields it.

Plotting:

1. The output is loaded as a dataframe, cleaned and plotted into a bar plot.

Sample Output:

```
[1, 2017]     5.620285956535426e+19
[10, 2019]    4.4548138889025403e+18
[10, 2015]    7.416931809333608e+19
[10, 2017]    2.6736910424747225e+19
```

Observation: The extremely high values of transactions early on can be attributed to the lesser number to total transactions in the beginning, as popularity increased so did the total number of transaction and hence the average value of each transaction decreased.

Part B

1) Initial Aggregation

Job ID:
http://andromeda.student.eecs.qmul.ac.uk:8088/proxy/application_1607539937312_7070

<u>Method</u>

Mapper:

1. Reads the file line by line and splits at ‘,’.
2. It yields, key (to_address) and value (Value)

Combiner/Reducer:

1. Aggregates the total value of transactions yields it.

Result:
The resultant file is a 3GB+ txt file too large to open at once. This is then moved to the cluster using:
Hadoop fs -copyFromLocal Aggregate.txt input/

2) Joining transactions/contracts and filtering

Job ID:
http://andromeda.student.eecs.qmul.ac.uk:8088/proxy/application_1607539937312_8605

Next, I performed a replication join of aggregate.txt and data/ethereum/contracts to get only smart contracts that have a record in “contracts”.

Mapper:

1. Read input files line by line, if split along ‘\t’ and has 2 fields, then it is from aggregate.txt. We yield key(address) and values(total transaction value)
2. If split along ‘,’ and has 5 fields, it is from contracts. We yield (address) and values(block number).

Reducer:

1. Receives key-value pairs, and has empty bkl_number and value initialized.

2. Only when a key(address) receives values from both Aggregate.txt and contracts, does the reducer yields (address, value). (hence only smart contracts make it through)

Output:
Contains a list of the smart contracts along with their aggregate values.

3) Top Ten

Job ID:
http://andromeda.student.eecs.qmul.ac.uk:8088/proxy/application_1607539937312_9143

Mapper:

1. Read input files line by line, if split along '\t' and has 2 fields. We yield key(None) and values(address, total transaction value)

Reducer:

1. Using sort function all the pairs are sorted in descending order. Next, we iterate through the sorted values 10 times yielding them and then stop.

Output:

```
Top 10 Smart Contracts
0xaa1a6e3e6ef20068f7f8d8c835d2d22fd5116444    8.415510080996593e+25
0x3f5ce5fbfe3e9af3971dd833d26ba9b5c936f0be    5.834331302252918e+25
0x32be343b94f860124dc4fee278fdcbd38c102d88    5.432072061064933e+25
0xfa52274dd61e1643d2205169732f29114bc240b3    4.578748448318936e+25
0x7727e5113d1d161373623e5f49fd568b4f543a9e    4.56206240013507e+25
0x209c4784ab1e8183cf58ca33cb740efbf3fc18ef    4.317035609226244e+25
0x876eabf441b2ee5b5b0554fd502a8e0600950cfa    4.015767887861935e+25
0x6fc82a5fe25a5cdb58bc74600a40a69c065263f8    2.706892158201953e+25
0x2910543af39aba0cd09dbb2d50200b3e800a63d2    2.6386022236908247e+25
0xcafb10ee663f465f9d10588ac44ed20ed608c11e    2.3078610109547e+25
```

Part C

1) Top Ten Most Active Miners

Job ID:
http://andromeda.student.eecs.qmul.ac.uk:8088/proxy/application_1607539937312_6796

Aggregating

Mapper:

1. Reads the file line by line and splits at ‘,’ and the number of fields is equal to 9.
2. It yields, key (miner) and value (size)

Combiner/Reducer:

3. Aggregates the size for each miner and yields it.

Top 10

Mapper:

1. Read input files line by line, if split along ‘\t’ and has 2 fields. We yield key(None) and values(miner, size)

Reducer:

1. Using sort function all the pairs are sorted in descending order. Next, we iterate through the sorted values 10 times yielding them and then stop.

Output:

```
Top 10 Miners
0xea674fdde714fd979de3edf0f56aa9716b898ec8        23989401188
0x829bd824b016326a401d083b33d092293333a830        15010222714
0x5a0b54d5dc17e0aadc383d2db43b0a0d3e029c4c 1      3978859941
0x52bc44d5378309ee2abf1539bf71de1b7d7be3b5        10998145387
0xb2930b35844a230f00e51431acae96fe543a0347        7842595276
0x2a65aca4d5fc5b5c859090a6c34d164135398226        3628875680
0x4bb96091ee9d802ed039c4d1a5f6216f90f81b01        1221833144
0xf3b9d2c81f2b24b0fa0acaaa865b7d9ced5fc2fb        1152472379
0x1e9939daaad6924ad004c2560e90804164900341        1080301927
0x61c808d82a3ac53231750dadc13c777b59310bd9        692942577
```

Part D: Data exploration

1) Price Forecasting

Introduction

I have chosen an ethereum dataset, preprocessed the data (removing Null) using pandas and then loaded it pyspark. The entire process of data splitting, model training and validation has been done using Spark mllib.

Dataset

Sourced from: https://datahub.io/cryptocurrency/ethereum

Fields:

1. **txVolume(USD)** - on-chain transaction volume.
2. **txCount** - refers to the number of transactions happening on the public blockchain a day.
3. **marketcap(USD)** - This is the unit price multiplied by the number of units in circulation. There has been quite a bit of controversy over this indicator.
4. **Price** - not much to say about this one. We get it from CoinMarketCap, with all the caveats that entail.
5. **exchangevolume(USD)** the dollar value of the volume at exchanges
6. **generatedCoins** - this refers to the number of new coins that have been brought into existence on that day.
7. **Fees** - Fees in our data are based on the native currency
8. **averaDifficulty** - the average measure of how difficult it is to find a hash below a given target.

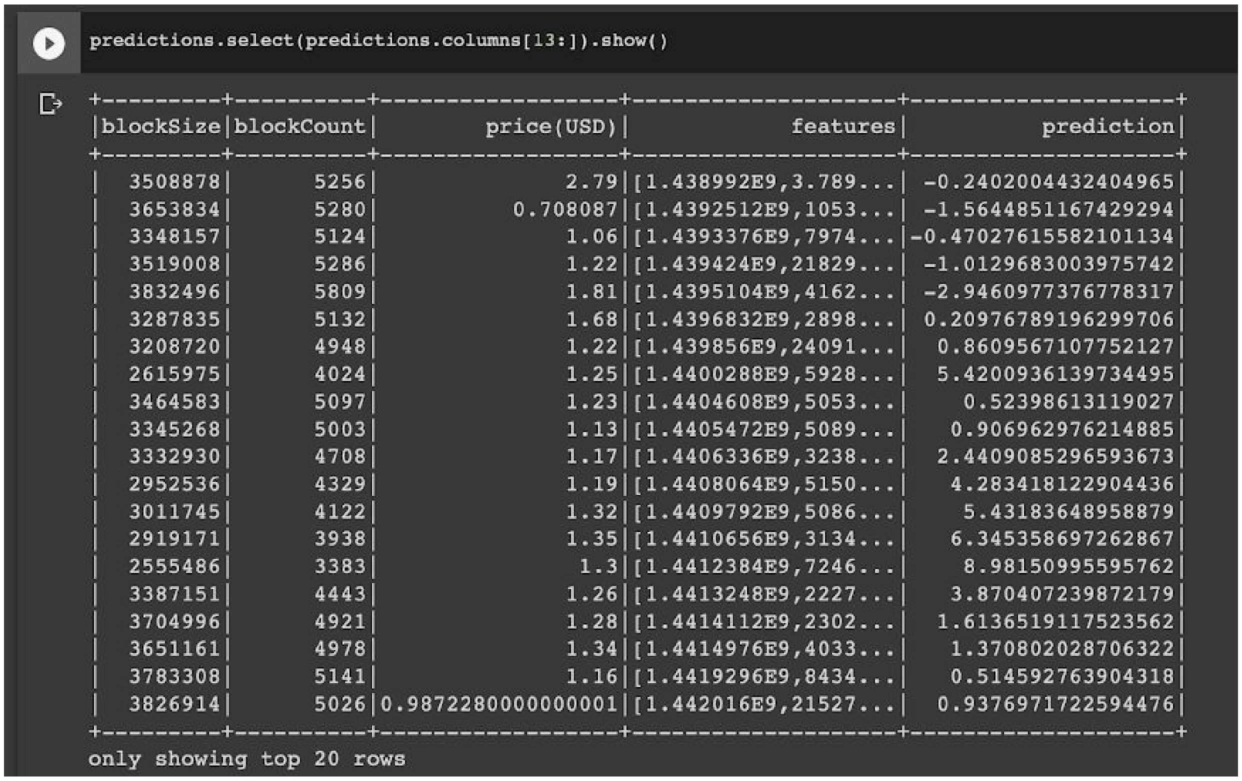
Code:

```
#importing necessary libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import time
import datetime

#installing pyspark and setting up a session
```

```
!pip install pyspark
from pyspark.sql import SparkSession
spark = SparkSession.builder.getOrCreate()

#Read data, file1.csv has been preprocessed
data = spark.read.csv("file1.csv",header=True, inferSchema=True)

#Final Column is the one to be predicted
feature_columns = data.columns[:-1] # here we omit the final column
from pyspark.ml.feature import VectorAssembler
assembler = VectorAssembler(inputCols=feature_columns,outputCol="features")


#Converting string date to timestamp to int, as str & timestamp can’t be
trasformed
from pyspark.sql.functions import col, unix_timestamp, to_date,to_timestamp
data = data.withColumn('date', to_timestamp(unix_timestamp(col('date'),
'yyyy-MM-dd').cast("timestamp")))
data = data.withColumn('date', (unix_timestamp(col('date'),
'yyyy-MM-dd').cast("int")))

data_2 = assembler.transform(data)

#Data split
train, test = data_2.randomSplit([0.7, 0.3])

#Loading Spark’s ML library
from pyspark.ml.regression import LinearRegression
algo = LinearRegression(featuresCol="features", labelCol="price(USD)")
#Training
model = algo.fit(train)

#Evaluation
evaluation_summary = model.evaluate(test)
evaluation_summary.meanAbsoluteError
evaluation_summary.rootMeanSquaredError
Evaluation_summary.r2

#Output
2.0224342814547174
```

2.5593805373398384
0.9998763658117484

#Predictions:
predictions = model.transform(test)
predictions.select(predictions.columns[13:]).show()



```
predictions.select(predictions.columns[13:]).show()
```

```
+---------+----------+------------------+--------------------+--------------------+
|blockSize|blockCount|        price(USD)|            features|          prediction|
+---------+----------+------------------+--------------------+--------------------+
|  3508878|      5256|              2.79|[1.438992E9,3.789...| -0.2402004432404965|
|  3653834|      5280|          0.708087|[1.4392512E9,1053...| -1.5644851167429294|
|  3348157|      5124|              1.06|[1.4393376E9,7974...|-0.47027615582101134|
|  3519008|      5286|              1.22|[1.439424E9,21829...| -1.0129683003975742|
|  3832496|      5809|              1.81|[1.4395104E9,4162...| -2.9460977376778317|
|  3287835|      5132|              1.68|[1.4396832E9,2898...| 0.20976789196299706|
|  3208720|      4948|              1.22|[1.439856E9,24091...|  0.8609567107752127|
|  2615975|      4024|              1.25|[1.4400288E9,5928...|  5.4200936139734495|
|  3464583|      5097|              1.23|[1.4404608E9,5053...|    0.52398613119027|
|  3345268|      5003|              1.13|[1.4405472E9,5089...|   0.906962976214885|
|  3332930|      4708|              1.17|[1.4406336E9,3238...|  2.4409085296593673|
|  2952536|      4329|              1.19|[1.4408064E9,5150...|   4.283418122904436|
|  3011745|      4122|              1.32|[1.4409792E9,5086...|    5.43183648958879|
|  2919171|      3938|              1.35|[1.4410656E9,3134...|   6.345358697262867|
|  2555486|      3383|               1.3|[1.4412384E9,7246...|    8.98150995595762|
|  3387151|      4443|              1.26|[1.4413248E9,2227...|   3.870407239872179|
|  3704996|      4921|              1.28|[1.4414112E9,2302...|  1.6136519117523562|
|  3651161|      4978|              1.34|[1.4414976E9,4033...|   1.370802028706322|
|  3783308|      5141|              1.16|[1.4419296E9,8434...|   0.514592763904318|
|  3826914|      5026|0.9872280000000001|[1.442016E9,21527...|  0.9376971722594476|
+---------+----------+------------------+--------------------+--------------------+
only showing top 20 rows
```

<u>Observations:</u>

The simple linear regression model was able to make a close call for dates within its dataset, but generated meaningless data for extremities, eg: -0.2402 for a date at the very beginning of the dataset(March 2015)

2) Fork the Chain

Job ID:
http://andromeda.student.eecs.qmul.ac.uk:8088/proxy/application_1607539937312_9492

Method:
We will observe the forks on December 15, 2017, and December 14, 2017. As these 2 forks very close together, their disruption should be more visible.

Using MapReduce I derived the daily price and number of transaction for the month of December 2017.

Mapper:

1. Reads the file line by line and splits at ‘,’. Then timestamp is converted to time.
2. It yields, key (day_of_month) and value (1, value of the GasPrice), if month == 12 and year == 2017

Reducer:

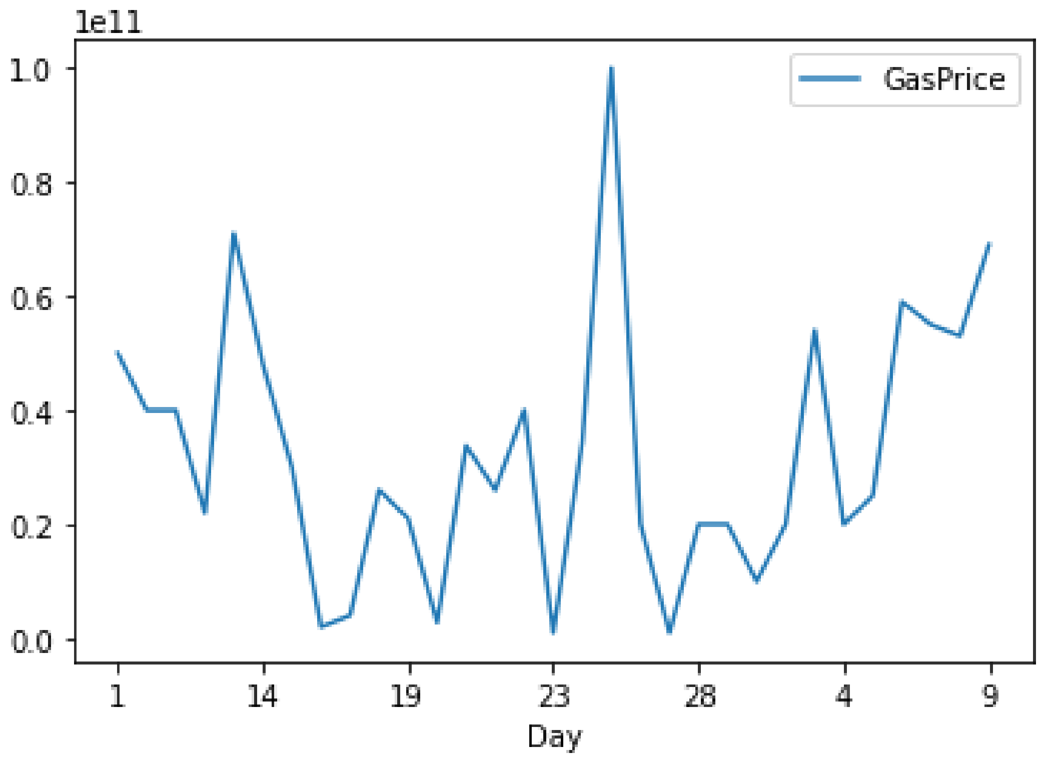
1. Aggregates the count of transactions and the value of Gas in the month
2. Then yields both values.

Plotting:

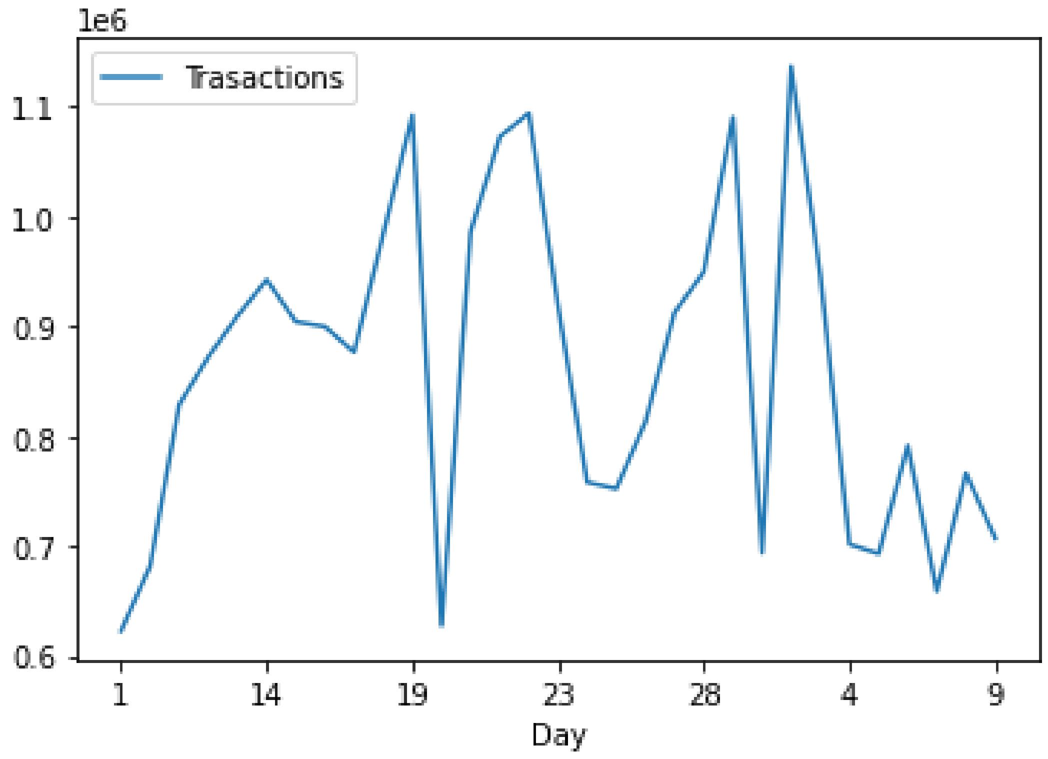
1. The output is loaded as a dataframe, cleaned and plotted into a bar plot.

Sample Output:
```
1    [622720, 50000000000.0]
10   [681677, 40000000000.0]
12   [872340, 22000000000.0]
14   [942559, 48100070000.0]
16   [899857, 2000000000.0]
18   [984021, 26000000000.0]
```

PLOTS for Analysis:



1e11
GasPrice
1.0
0.8
0.6
0.4
0.2
0.0
1
14
19
23
28
4
9
Day



1e6
Trasactions
1.1
1.0
0.9
0.8
0.7
0.6
1
14
19
23
28
4
9
Day

Observations:

Gas price drastically drops on the days of forking, while number drastically after a few days. But both these falls are short-lived as evident that, the price and number of transactions shot back up again.

3) Gas Guzzlers

Method:
Using MapReduce to get average Gas Price and average Gas Limit for each month in the dataset and then plotting the result to observe patterns

Job ID:
http://andromeda.student.eecs.qmul.ac.uk:8088/proxy/application_1607539937312_8206 - Gas Price
http://andromeda.student.eecs.qmul.ac.uk:8088/proxy/application_1607539937312_8468 - Gas Limit

Mapper:

1. Reads the file line by line and splits at ‘,’. Then timestamp is converted to time.
2. It yields, key (month and year) and value (1, value of the GasLimit/GasPrice)

Reducer:

1. Aggregates the count of transactions and the total value of GasLimit/GasPrice in the month
2. Then divides the total by count and yields it.

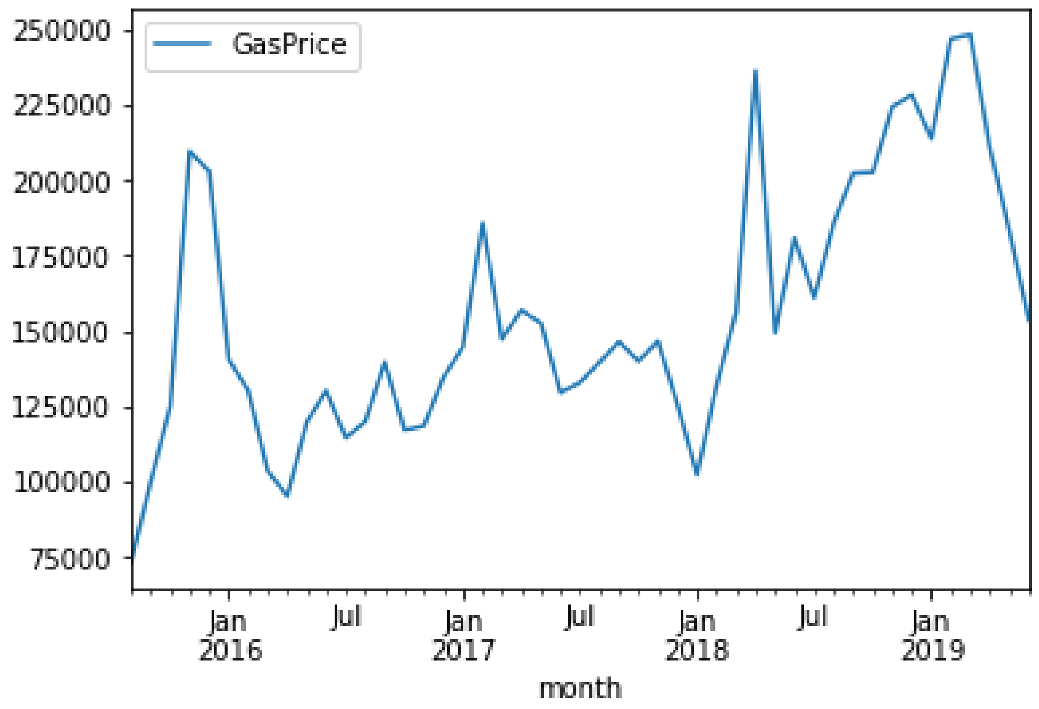
Plotting:

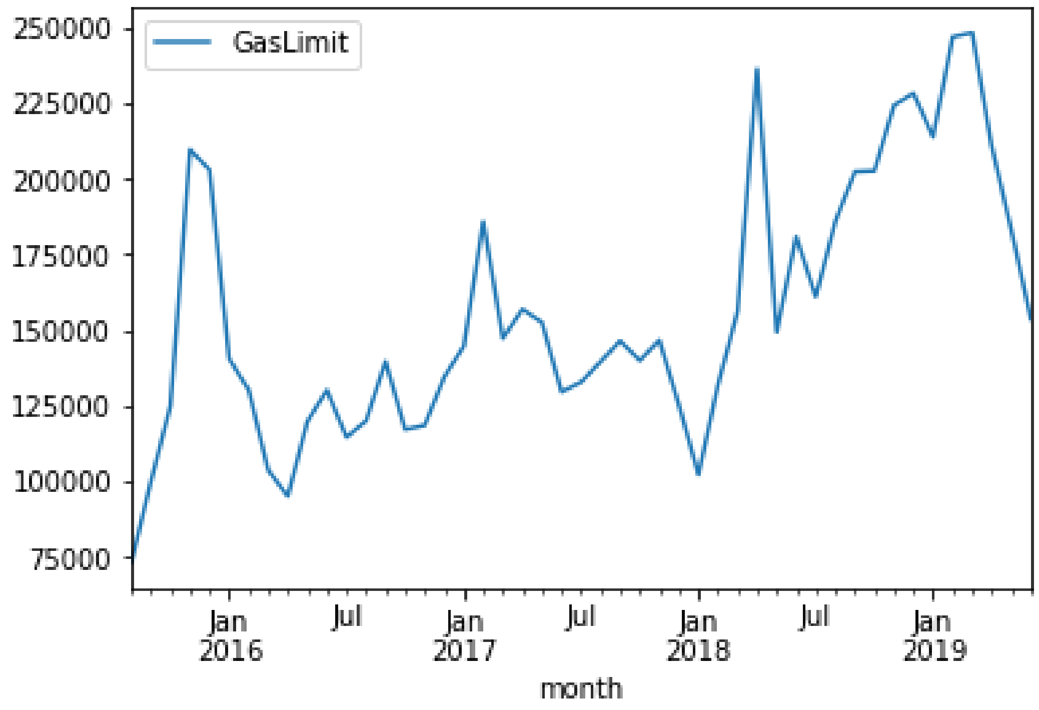
1. The output is loaded as a dataframe, cleaned and plotted into a bar plot.

Sample Output:

```
Gas Limit:
[1, 2017]    144585.8323444452
[1, 2019]    213715.9605048934
[10, 2015]    124898.76345065859
[10, 2017]    139870.05913735816
[11, 2016]    118474.20698056064
[11, 2018]    224208.95181702235
[12, 2015]    202883.632316504
[12, 2017]    125269.51047898784

Gas Price:
[1, 2017]    22507570807.719795
[1, 2019]    14611816445.785261
[10, 2015]    53898497955.07804
[10, 2017]    17509171844.770645
[11, 2016]    24634294365.279037
[11, 2018]    16034859008.681646
[12, 2015]    55899526672.35498
```

PLOTS for Analysis:





<u>Observation:</u>

As visible both Gas Limit and Gas price follow a generally upward trend with time. Both for identical graphs as well. Contracts are requiring more Gas with time.

The large peaks are caused just before the forking of ethereum (March 12, 2018, January 19, 2018, January 01, 2018, December 15, 2017, December 14, 2017)