BIG DATA PROCESSING

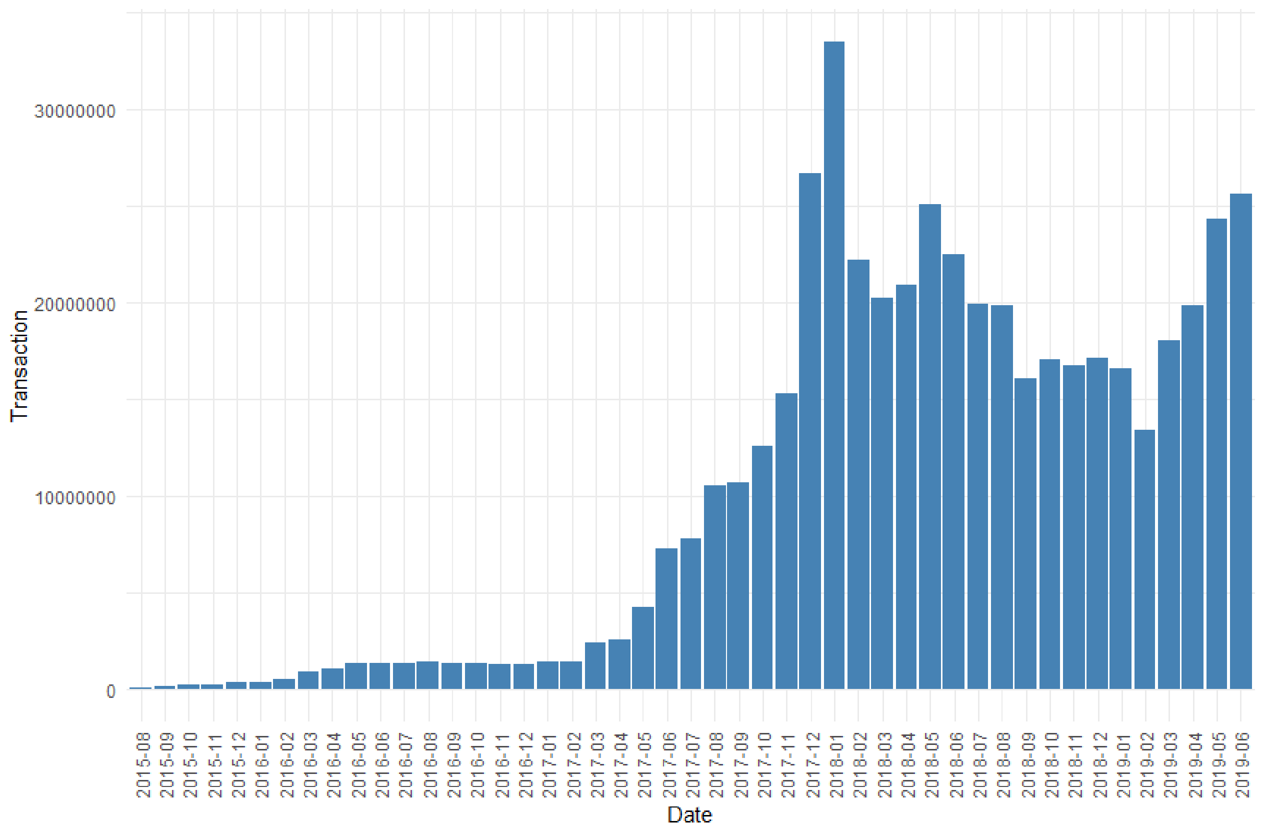
COURSEWORK: ETHEREUM ANALYSIS

ANALYSIS OF ETHEREUM TRANSACTIONS AND SMART CONTRACTS

Muhammad Hamza — 150741163

PART A: TIME ANALYSIS

Job id:
http://andromeda.student.eecs.qmul.ac.uk:8088/proxy/application_1575381276332_2623/)



The bar chat above is plotted using the ggplot2 library. The bar chart shows the number of Ethereum transactions recorded for each month from August 2015 till June 2019. The bar chart shows little to no positive correlation from Ethereum's inception till February 2017, except for slight increase in May of 2016. Then, there is a great increase from May of 2017 until January 2018, which is also the highest peak of the bar chat. The result then varies, showing an overall negative correlation from the highest peak till February 2019, and then a high positive correlation till June of 2019. This frequent change in correlation proves the high volatility of Ethereum.

```
from mrjob.job import MRJob
import time
import re

class partA(MRJob):

    def mapper(self, _, line):
        fields = line.split(',')
        try:
            if (len(fields) == 7):
                time_epoch = int(fields[6])
                date = time.strftime("%Y-%m",time.gmtime(time_epoch))
                yield (date, 1)
        except:
            pass

    def combiner(self, date, counts):
        yield (date, sum(counts))

    def reducer(self, date, counts):
        yield (date, sum(counts))

if __name__ == '__main__':
    partA.run()
```

This implementation makes use of the MapReduce program. There are three functions: mapper, combiner, and reducer. The mapper function checks if the line correctly formatted, to prevent malformed lines. Therefore, the try and except construct is used to catch any potential exception. If the lines are correctly formatted, then the data, in this case time, which is the 6th element of the array, *fields*, is stored onto the variable *time_epoch* and is emitted with value of 1.

The combiner and reducer function are very similar as they run on each partition. However, the combiner function acts as a preliminary reducer. Even though the combiner function is optional, it is very useful for improving efficiency by reducing the number of items emitted.

PART B: TOP TEN MOST POPULAR SERVICES

(Job id: application_1575381276332_2899

[Retrieved using the command yarn top since it's a spark job]

Appending to the end of the URL for format:

http://andromeda.student.eecs.qmul.ac.uk:8088/proxy/application_1575381276332_2899)

Output:

```
0xaa1a6e3e6ef20068f7f8d8c835d2d22fd5116444: 84155100809965865822726776
0xfa52274dd61e1643d2205169732f29114bc240b3: 45787484483189352986478805
0x7727e5113d1d161373623e5f49fd568b4f543a9e: 45620624001350712557268573
0x209c4784ab1e8183cf58ca33cb740efbf3fc18ef: 43170356092262468919298969
0x6fc82a5fe25a5cdb58bc74600a40a69c065263f8: 27068921582019542499882877
0xbfc39b6f805a9e40e77291aff27aee3c96915bdd: 21104195138093660050000000
0xe94b04a0fed112f3664e45adb2b8915693dd5ff3: 15562398956802112254719409
0xbb9bc244d798123fde783fcc1c72d3bb8c189413: 11983608729202893846818681
0xabbb6bebfa05aa13e908eaa492bd7a8343760477: 11706457177940895521770404
0x341e790174e3a4d35b65fdc067b6b5634a61caea: 8379000751917755624057500
```

```python
import pyspark

def transactions_clean(line):
    try:
        fields = line.split(',')
        if len(fields) != 7:
            return False
        int(fields[3])
        return True
    except:
        pass


def contracts_clean(line):
    try:
        fields = line.split(',')
        if len(fields) != 5:
            return False
        return True
    except:
        pass

sc = pyspark.SparkContext()
```

```
transactions = sc.textFile('/data/ethereum/transactions')
contracts = sc.textFile('/data/ethereum/contracts')

transaction_filter = transactions.filter(transactions_clean)
transaction_mapper = transaction_filter.map(lambda n: (str(n.split(',')[2]),
long(n.split(',')[3]) ))

contract_filter  = contracts.filter(contracts_clean)
contract_mapper = contract_filter.map(lambda a: (a.split(',')[0],1))

transaction_reducer = transaction_mapper.reduceByKey(lambda a,b: a + b)

join = transaction_reducer.join(contract_mapper).map(lambda a: (a[0], a[1][0]))

top_ten = join.takeOrdered(10,key = lambda a: -a[1])

for top in top_ten:
    print("{}: {}".format(top[0], top[1]))
```

This implementation makes use of the Spark program. The two functions take one element corresponding to a line and return True or False. A return of False is only possible if the line that is passed is malformed. Moreover, the sc object is used to create the first RDD.

Furthermore, these two functions are used to filter the lines and then stored into the variables, *transaction_filter* and *contract_filter*. Both of which are passed onto the transformation function called *map*. The values selected are then stored and then joined using the *join* operation returning all possible pairs from the defined variables. Afterward, the *takeOrdered* function is used containing -a[1] which specifies that two values should be sorted in descending order by the value element 1.

PART C: DATA EXPLORATION

MACHINE LEARNING

(Job id: application_1575381276332_3792

[Retrieved using the command yarn top since it's a spark job]

Appending to the end of the URL for format:

http://andromeda.student.eecs.qmul.ac.uk:8088/proxy/application_1575381276332_3792)

Output:

```
99.42756168581363
0.8019673312769673
+--------------------+------------------+
|            features|        prediction|
+--------------------+------------------+
|[7.236285862E7,20...|43.776371402585596|
|[7.253323941E7,23...|    42.96464040356|
|[7.258752112E7,24...| 42.73392969641361|
|[7.269010159E7,35...|43.487801465652296|
|[7.288930112E7,45...|43.304505489765575|
|[7.293093378E7,50...| 43.77676276353077|
|[7.304158503E7,54...|43.487081117673824|
|[7.314580987E7,52...| 42.46252539544548|
|[7.322562222E7,59...| 42.86343889164641|
|[7.332857941E7,50...| 40.87608891122443|
|[7.335447362E7,56...|41.462842837630205|
|[7.348974597E7,72...| 42.76142009778198|
|[7.374902191E7,61...| 39.30007324033056|
|[7.392953862E7,65...|38.583145780587415|
|[7.400951769E7,66...|38.154082157628636|
|[7.421694972E7,60...| 35.85738197222554|
|[7.448653706E7,71...|35.418928190707675|
|[7.456945722E7,69...| 34.58704858473561|
|[7.459717362E7,68...|  34.1547193225889|
|[7.467860253E7,77...| 34.91205557780313|
+--------------------+------------------+
only showing top 20 rows
```

```
from pyspark import SparkConf, SparkContext
from pyspark.sql import SQLContext
from pyspark.ml.regression import LinearRegression
from pyspark.ml.linalg import Vectors
from pyspark.ml.feature import VectorAssembler


hdfs = "hdfs://andromeda.student.eecs.qmul.ac.uk"
eth_path = hdfs + "/user/mh326/data/eth.csv"

sc = SparkContext()
sql_context = SQLContext(sc)
df = sql_context.read.format('csv') \
            .options(header='true', inferschema='true') \
            .load(eth_path)

assembler = VectorAssembler(inputCols=['SplyCur','TxTfrCnt'],
                  outputCol='features')


output = assembler.transform(df)

final_data = output.select('features', 'PriceUSD').filter(df['PriceUSD'].isNotNull())

train_data,test_data = final_data.randomSplit([0.7,0.3])

lr = LinearRegression(labelCol='PriceUSD', maxIter=50,
             regParam=0.3, elasticNetParam=0.5)

lr_model = lr.fit(train_data)

test_results = lr_model.evaluate(test_data)

print(test_results.rootMeanSquaredError)

print(test_results.r2)

un_data = test_data.select('features')

predictions = lr_model.transform(un_data)

predictions.show()
```

This implementation makes use of the Spark program, also utilising the mllib build that is used to train the price forecasting model based on the dataset from Coin Metrics. (Coin Metrics, 2019)

Firstly, the dataset is read and stored onto a variable. The *VectorAssembler* contains the inputCol and outputCol which is a single feature; it uses these two arguments and then stores them to assembler variable as a single vector. Then, to transform the data, the transform method is used on the assembler object by also passing the original data from the variable *df*. To select the final data, the '*features'* and the *'PriceUSD'* column are selected. Then a train-test split is performed of 0.7:0.3.

The *LinearRegression* and *fit* method are used on the train_data, and then the *evaluate* method is used to retrieve the test_results.

Lastly, the r-squared value is a statistical measurement of how close the data are to the fitted regression line—also known as the coefficient of determination. A coefficient of 0.802 (rounded up to 3 decimal places) indicates that the model explains all variability of the data indicating that the forecast model is a good fit for the stated period. Moreover, it also indicated that the data is more surrounding the mean value. However, it isn't 1 or 100% which means that the forecasting value will not remain accurate indefinitely—which can be explained due to the volatile nature of Ethereum in general.

Sources:

Coin Metrics. (2019). Data downloads - Coin Metrics. [online] Available at: https://coinmetrics.io/data-downloads/ [Accessed 5 Dec. 2019].