Exploring Bitcoin transactions

Part 1 – Transaction Analysis

In order to analyze the Bitcoin dataset (compressed dataset with 212576 blocks), I have used Pandas, a software library in Python and Jupyter Notebook that is used to execute the code live in an interactive interface. I have also attached the Python code of each answer as Q1,Q2 and so on. Code segments are given with each answer with comments to explain the same.

1) What is the number of transactions and addresses in the dataset?

Answer:
(dataset to produce answer: bh.csv, addresses.csv)
Number of transactions - **10000055**
Number of addresses - **8385065**

Code Segments:

```
data = pd.read_csv(file_path, sep='\t', header=None)                    #Read bh.csv file
data.columns = ['blockID', 'hash', 'timestamps', 'n_Txns']              #Name the columns
df = pd.DataFrame(data)                                                 #Put the read data in a data frame object
df['n_Txns'].sum()                                                      #Get the sum of the transactions
```

```
data = pd.read_csv(file_path, sep='\t', header=None)                    #Read addresses.csv file
data.columns = ['addrID', 'addr']                                       #Name the columns
df = pd.DataFrame(data)                                                 #Put the read data in a data frame object
print(df['addrID'].count())    #This print gives the number of address rows which is total no of addresses
```

```
In [8]: df = pd.DataFrame(data)

In [10]: print(df['n_Txns'].sum())

10000055
```

Screenshot:

```
In [15]: df2 = pd.DataFrame(data2)

In [17]: print(df2['addrID'].count())
         8385065
```

==========

2) What is the Bitcoin address that is holding the greatest amount of bitcoins? How much is that exactly? Note that the address here must be a valid Bitcoin address string. To answer this, you need to calculate the balance of each address. The balance here is the total amount of bitcoins in the UTXOs of an address.

Answer:

(dataset to produce answer: txin.csv, addresses.csv)
Greatest amount of Bitcoins: **11111100000000.0 satoshis (111111.00000000 bitcoins)**
Address: **1933phfhK3ZgFQNLGSDXvqCn32k2buXY8a (**addrID**: 1083442)**

Code Segment:

```
dataSum_in = pd.DataFrame(datain.groupby(['addrID'])['sum'].sum())  #Dataframe for txin.data
dataSum_out = pd.DataFrame(dataout.groupby(['addrID'])['sum'].sum())          #Dataframe for txout.dat
ans = pd.merge(dataSum_in, dataSum_out,how='right',on = 'addrID')          #Do a Right join
ans=ans.fillna(0)
ans['balance'] = ans['sum_y'] - ans['sum_x']              #Find the balance of each address
print(ans['balance'].max())                               #This max value prints the greatest sum
ans.loc[ans['balance'].idxmax()]                          #This will give the addrID
```

After getting the **addrID(1083442),** we can find the address from the address from the addresses.csv file using the command **print(df.loc(df[‘addrID’] == 1083442)**

Screenshot:

```
In [31]: ans['balance'] = ans['sum_y'] - ans['sum_x']

In [32]: print(ans)
```

```
                  sum_x         sum_y       balance
addrID
-1         2.304004e+08  261650240850  2.614198e+11
 9         1.770000e+10   19500000000  1.800000e+09
 78        5.000000e+09    5000000000  0.000000e+00
 171       1.000000e+09    1000000000  0.000000e+00
 185       1.000000e+08     100000000  0.000000e+00
...                 ...           ...           ...
 8385059   0.000000e+00       2400000  2.400000e+06
 8385060   0.000000e+00       1000000  1.000000e+06
 8385062   0.000000e+00     515134000  5.151340e+08
 8385063   0.000000e+00       1045420  1.045420e+06
 8385064   0.000000e+00       1038480  1.038480e+06

[8385061 rows x 3 columns]
```

```
In [33]: print(ans['balance'].max())
```

```
11111100000000.0
```

```
In [42]: ans.loc[ans['balance'].idxmax()]
```

```
Out[42]: sum_x      0.000000e+00
         sum_y      1.111110e+13
         balance    1.111110e+13
         Name: 1083442, dtype: float64
```

==========

3) What is the average balance per address?

Answer:

Average Balance per address: **125990615.08928594**

Code Segment:

This is same as the previous question code segment, we add the following additionally,
print(ans['balance'].sum()/8385061) **#Divide the balance sum by the total addresses**

Screenshot:

```
In [35]: print(ans['balance'].sum()/8385061)
```

```
125990615.08928594
```

==========

4) What is the average number of input and output transactions per address? What is the average number of transactions per address (including both inputs and outputs)? An output transaction of an address is the transaction that is originated from that address. Likewise, an input transaction of an address is the transaction that sends bitcoins to that address.

Answer:

(dataset to produce answer: txout.csv,txin.csv)
Average Input Transactions per address: **2.65343925924659**
Average Output Transactions per address: **2.774792693815823**
Average number of transactions per address (both input and output): **5.1800886123547585**

Code Segments:

```
df = pd.DataFrame(dfout.drop_duplicates('addrID'))          #Dataframe for txout.csv file
transOutCount = dfout['txID'].count()                       #gets count of txIDs
addrCnt= df['addrID'].count()                               #gets addressCount
transOutCount / addrCnt      #This here gives the average 'output' transactions per address

tcount = dfin['txID'].count()                               #Gets count of txIDs in the txin.csv file
addr_in = pd.DataFrame(dfin.drop_duplicates('addrID'))  #Dataframe for txin.csv
addrCount = addr_in['addrID'].count()                       #Count of address IDs
tcount / addrCount                #This here gives the average 'output' transactions per address

tabCombined = pd.concat([dfout, dfin])      #Combine dataframes of txin and txout
addresses = pd.DataFrame(tabCombined.drop_duplicates('addrID')) #drop duplicate
addressIDs
totalAddr = addresses['addrID'].count()   #Gets count of AddrIDs
AvgPerAddr =  (tcount + transOutCount) / totalAddr  #This here prints avg transactions per
addresses (both input and output)
print(AvgPerAddr)
```

```
In [14]: transOutCount = dfout['txID'].count()

In [15]: addrCnt= df['addrID'].count()

In [16]: transOutCount / addrCnt
Out[16]: 2.774792693815823
```

Screenshot:

```
In [25]: tabCombined = pd.concat([dfout, dfin])

In [26]: addresses = pd.DataFrame(tabCombined.drop_duplicates('addrID'))

In [27]: totalAddr = addresses['addrID'].count()

In [28]: AvgPerAddr =  (tcount + transOutCount) / totalAddr

In [29]: print(AvgPerAddr)
         5.1800886123547585

In [20]: tcount = dfin['txID'].count()

In [21]: addr_in = pd.DataFrame(dfin.drop_duplicates('addrID'))

In [22]: addrCount = addr_in['addrID'].count()

In [23]: tcount / addrCount
Out[23]: 2.65343925924659
```

==========

5) What is the transaction that has the greatest number of inputs? How many inputs exactly? Show the hash of that transaction. If there are multiple transactions that have the same greatest number of inputs, show all of them.

Answers:

(dataset to produce answer: tx.csv, txh.csv)
Transaction with greatest inputs: **7553001**
Highest Inputs: **1312**
Hash of the transaction:
9621b3c67f9bddd3de65fafc488087b8f2b40b638e3a06209a904c66c0b32982

Code Segments:

```
df = pd.DataFrame(data)          #Data frame for data here which is the tx.csv dataset
df.loc[df['n_input'].idxmax]     #This here gives the details of highest input including txID
df2 = pd.DataFrame(data2)        #Data frame for data here which is the txh.csv dataset
df2.loc[df['txid'] == 7553001]   #This gives the hash of txID with highes input
```

```
In [6]: df.loc[df['n_input'].idxmax]
Out[6]: txid        7553001
        blockid      201341
        n_input        1312
        n_output          3
        Name: 7553001, dtype: int64

In [7]: file_path2 = 'D:/Ash/UFL/Sem2/Blockchain Optimization and Applications/HW2/compressed/data/txh.csv'

In [8]: data2 = pd.read_csv(file_path2, sep='\t', header=None)

In [9]: data2.columns = ['txid', 'hash']

In [11]: df2 = pd.DataFrame(data2)

In [12]: df2.loc[df['txid'] == 7553001]
```

Out[12]:

	txid	hash
7553001	7553001	9621b3c67f9bddd3de65fafc488087b8f2b40b638e3a06...

Screenshot:

==========

6) What is the average transaction value? Transaction value is the sum of all outputs' value.

Answer:

(dataset to produce answer: txout.csv)

Average Transaction value: **12315588064.03543**

Code Segments:

```
df = pd.DataFrame(data)                 #Data frame for data from txout.csv
txids = df.drop_duplicates('txID')      #Drop duplicate txIDs
txidCount = txids['txID'].count()       #Get the count of txID now
sum = df['sum'].sum()                   #Sum of sums
avgValue = sum / txidCount              #This will give Average transaction value
```

Screenshot:

```
In [16]: txids = df.drop_duplicates('txID')

In [17]: txidCount = txids['txID'].count()

In [18]: sum = df['sum'].sum()

In [19]: avgValue = sum / txidCount

In [20]: print(avgValue)
         12315588064.03543
```

==========

7) How many coinbase transactions are there in the dataset

Answer: 212576 (dataset to produce answer: tx.csv)

Code Segments:

```
df = pd.DataFrame(data)    #Data Frame for tx.csv dataset
df.loc[df['n_input'] == 0]    #Get information for coinbase transactions that has n_input = 0
```

Screenshot:

```
In [6]: df.loc[df['n_input'] == 0]
```

Out[6]:

	txId	blockId	n_input	n_output
0	0	0	0	1
1	1	1	0	1
2	2	2	0	1
3	3	3	0	1
4	4	4	0	1
...	...	...	...	...
9999111	9999111	212571	0	1
9999371	9999371	212572	0	1
9999461	9999461	212573	0	1
9999694	9999694	212574	0	1
9999783	9999783	212575	0	1

212576 rows × 4 columns

==========

8) What is the average number of transactions per block?

Answers: **47.04225782778865** (dataset to produce answer: bh.csv)

Code Segments:

```
df = pd.DataFrame(data)                 #Data Frame for bh.csv dataset
print(df['n_Txns'].mean())              #This here prints the average transactions per block
```

Screenshot:

```
In [7]: print(df['n_Txns'].mean())
        47.04225782778865
```

==========

Part 2: Address de-anonymization

In joint control we assume that all input addresses of a transaction are controlled by the same user. The addr_sccs.dat file generated from the Joint Control heuristic in the dataset contains a total of 4225481 users (476884 on removing duplicates).
Serial control assumes that the output address of a transaction with only a single output is controlled by the same user owning the input addresses.

- How many users are there in the dataset?

  After applying serial control on top of the joint control the number of users are around **4318422 users.**

- Answer questions 2, 3, and 4 in part 1 by replacing "address" with "user". Note that each user is identified by the addresses that are owned by him/her. Thus, in answering

question 2 (i.e., the user who is holding the greatest amount of bitcoins), you need to list all the user's addresses.

- The highest balance in satoshis: **6010245309814.** The user ID`s addresses are:

6851763(addrID) **17SzMdJbw6wR8b4HzmFCn1kDhEbexbLPXK(addr)**

6851857 (addrID) **1FxVdS6c1HSYDsMEHch8GkfegnuuPPrkoJ (addr)**

6849838(addrID) **1AY4dizXEGJA8mQKJFBcuEAResGt4uRZTV (addr)**

6850395(addrID) **1NnqM24fFeAGf7NWxmhhFkQAciPqeWo3L (addr)**

- Average input transactions per user**: 4.670352503761791**
  Average output transactions per user**: 5.387802766844**
  Average number of input and output transactions per user**: 10.058155270605791**

- **Average balance per user can be calculated as in part A Q3, except replacing addresses by number of users = 25501698.0293**

- Hash of the transaction sending greatest amount of bitcoins:
  70d46f768b73e50440e41977eb13ab25826137a8d34486958c7d55c5931c6081

  (TxID: 922967)

Instructions to Run:

Install Anaconda. This will have the Jupyter notebook command interface. Launch Anaconda Navigator and launch Jupyter notebook. Upload the .ipynb file to get the code and execute it.

To run the .py files, in cmd run 'python <filename> .py' . Remember that the file location should be changed inside the code pertaining to the location the dataset is contained.