Financial Data Science
Assignment 3: 8th March, 2023

Author: Gavin Connolly
Student Number: 18308483

Module: Financial Data Science

Academic Year - 2022/23
Submitted: 8th March, 2023

Q1:

Read trade data for the first $<1,000 + yourclassnumber \times 100>$ crypto Punks into a DB.

```
import requests
from bs4 import BeautifulSoup
import sqlite3 as lite
import pandas as pd
import time
from numpy.random import rand
import datetime as dt

my_student_number = 18308483
class_list = pd.read_csv('ClassList1.csv', sep=',')
class_number = class_list.loc[class_list['Student ID'] == my_student_number, 'Class Number'].item()

niters = 1000 + class_number*100

BaseStr = 'https://www.larvalabs.com/cryptopunks/details/'

con = lite.connect('CryptoPunk.db')
with con:
    cur=con.cursor()
    cur.execute('''DROP TABLE IF EXISTS PunkTrades''')
    cur.execute('''CREATE TABLE PunkTrades(TDate date, PunkID INT, TType TEXT, TFrom TEXT, TTo TEXT, TAmt INT)''')

    for punk in range(niters):

        PunkNo = str(punk)
        print('Processing Punk No.:' + PunkNo)
        time.sleep(2+0.5*rand())
        page = requests.get(BaseStr + PunkNo) # Getting page HTML through request
        soup = BeautifulSoup(page.content, 'html.parser')
        table = soup.find('table', attrs={'class':'table'})
        rows = table.find_all('tr')

        col_order = [4, 0, 1, 2, 3]

        for row in rows:
            cols = row.find_all('td')

            if not cols: # Screen out empty rows
                continue

            cols = [ele.text.strip() for ele in cols]
            cols = [cols[i] for i in col_order]
            cols.insert(1, PunkNo) # Add PunkID variable to our list

            # Some standardization of data
            cols[0] = dt.datetime.strptime(cols[0], '%b %d, %Y').strftime('%Y-%m-%d') # Convert to usable date format
            cols[-1] = cols[-1].split('Xi')[0].replace('<','') # Extract ETH price of transaction

            try:
                cur.execute('''INSERT OR IGNORE INTO PunkTrades
                    (TDate, PunkID, TType, TFrom, TTo, TAmt) VALUES(?,?,?,?,?,?)''', cols)

            except:
                print('DB error: row was not inserted', cols)
```

Q2:

Report the punk with the highest price.

Punk ID	Price
561	500 Ξ

The CryptoPunk with the highest price was punk number 561, with a price of 500 Ξ. In order to calculate this, we only consider the completed transaction and disregard any bids/offers which were not accepted by the seller.

```
sqliteConnection = lite.connect('CryptoPunk.db')
cursor = sqliteConnection.cursor()

query_price = """
            SELECT PunkID, max(TAmt) MaxPrice
            FROM PunkTrades
            WHERE TAmt <> '' AND TType == 'Sold'
            """

df_price_query = pd.read_sql_query(query_price, sqliteConnection)
print(df_price_query.to_latex(index=False))
```

Q3:

Report which punk was traded the most.

Punk ID	Number Of Trades
441	25

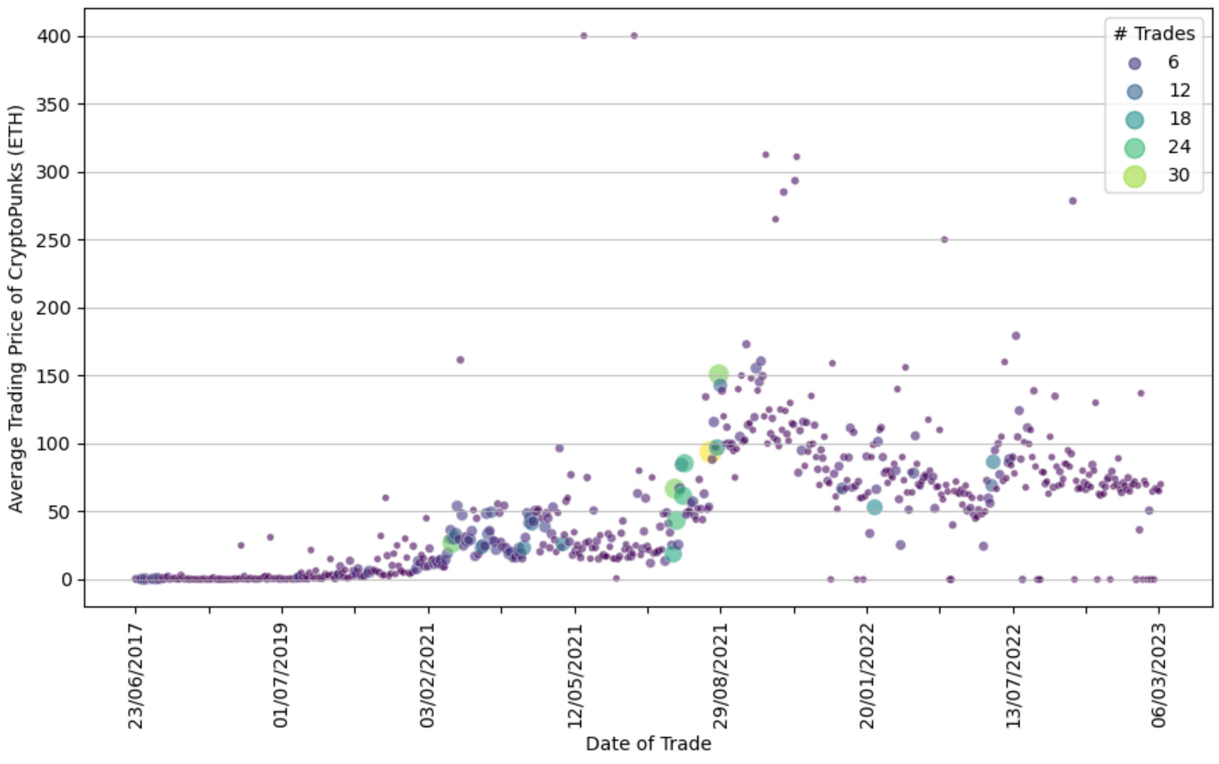
The CryptoPunk which was traded most often was punk number 441, which was traded a total of 25 times. Here, we count any transaction which results in the ownership of the punk being changed as being a 'trade' of the punk. Namely, we include the selling, transfer or initial claim of a CryptoPunk in our calculation of the number of trades.

```
query_volume = """
            SELECT PunkID, COUNT(TType) NumberOfTrades
            FROM PunkTrades
            WHERE TType == 'Sold' OR TType =='Transfer' OR TType == 'Claimed'
            GROUP BY PunkID
            ORDER BY NumberOfTrades DESC
            LIMIT 1
            """

df_volume_query = pd.read_sql_query(query_volume, sqliteConnection)
print(df_volume_query.to_latex(index=False))
```

Q4:

Plot the average price by day in the DB.



Bubble Chart of Average Trade Price of CryptoPunks, Coloured & Sized by Daily Number of Trades

Looking at bubble chart of the average price of punks by day, we see that prices stayed relatively low up until around August of 2021, where they rose rapidly, before falling slightly and then stagnating in the following months.
The other interesting observation is that there are a few outliers, in which the price of punks was extremely high (namely two days where the average price exceeded 400 Ξ). It is perhaps unsurprising that these extreme values all seem to coincide with days where there was an exceptionally low volume of punk trades, leading to a few large trades inflating the average.
.

```python
import matplotlib.pyplot as plt
import seaborn as sns

query_avg_price = """
            SELECT TDate, COUNT(TAmt) NumTrades, AVG(TAmt) AveragePrice
            FROM PunkTrades
            WHERE TType == 'Sold'
            GROUP BY TDate
            ORDER BY TDate ASC
            """
df_avg_price_query = pd.read_sql_query(query_avg_price, sqliteConnection)
df_avg_price_query['TDate'] = pd.to_datetime(df_avg_price_query['TDate'], format="%Y-%m-%d")
df_avg_price_query['TDate'] = df_avg_price_query['TDate'].dt.strftime("%d/%m/%Y")
df_avg_price_query.set_index("TDate", inplace=True)

ax = sns.scatterplot(data=df_avg_price_query, x="TDate", y="AveragePrice", size="NumTrades",
                palette='viridis', hue="NumTrades", legend=True, sizes=(15, 150), alpha=0.6)

plt.xticks(rotation='vertical')
plt.xlabel("Date of Trade")
plt.ylabel("Average Trading Price of CryptoPunks")
plt.grid(axis='y', alpha=0.75)
plt.subplots_adjust(bottom=0.2)

leg = plt.legend(title='# Trades')
for lh in leg.legendHandles:
    lh.set_alpha(0.6)
[label.set_visible(False) for (i,label) in enumerate(ax.xaxis.get_ticklabels()) if i % 90 != 0]
[tick.set_visible(False) for (i,tick) in enumerate(ax.xaxis.get_ticklines()) if i % 90 != 0]
plt.show()
```

Q5:

Which owner has the most valuable portfolio of Crypto Punks?

Owner	Portfolio Value
0x2be665	617 Ξ

User 0x2be665, has the highest portfolio value with a total of 617 Ξ.
Before we can calculate the portfolio value of each user, there are a number of problems we must address and assumptions we must make. The first assumption we make is that the current value of each CryptoPunk is just the most recent selling price. This obviously does not account for changes in the price post the sale of the punk and may lead to innaccurate and out-of-date valuations being applied. We could instead look at the most recent bid/offer prices, but as there is no agreement between buyer/seller made in these instances, this would also not likely lead to a fair valuation of the CryptoPunk's market price.
The second assumption we make is that each user ID belongs to a unique individual (there are likely users with various punks split over multiple accounts which would lead to an undervaluation of the value of their CryptoPunks).
The main problem which we encounter is that the person who the CryptoPunk was last sold to may not be the person who currently owns the punk. We must therefore take transfers of CryptoPunks into account, and in the case of a transfer, we assume the current value of the punk to be the most recent selling price.
Each of these are dealt with using a series of CTEs in our query.

```
query_portfolio = """
                ;WITH PunkTrades_Ordered AS (
                    SELECT PunkID, TType, TTo, TAmt, ROW_NUMBER() OVER(PARTITION BY PunkID ORDER BY TDate DESC) RowNum
                    FROM PunkTrades
                    WHERE TType == 'Sold' OR TType == 'Transfer'
                    GROUP BY PunkID, TDate, TTo
                    ORDER BY TDate DESC
                ),
                MostRecentSalePrice AS (
                    SELECT PunkID, min(RowNum) RecentIndex, TAmt SalePrice
                    FROM PunkTrades_Ordered
                    WHERE TType == 'Sold'
                    GROUP BY PunkID
                    ORDER BY RowNum
                )
                SELECT A.TTo Owner, SUM(B.SalePrice) PortfolioValue
                FROM PunkTrades_Ordered A
                LEFT JOIN MostRecentSalePrice B
                ON A.PunkID == B.PunkID
                WHERE RowNum == 1
                GROUP BY Owner
                ORDER BY PortfolioValue DESC
                LIMIT 1
            """
df_portfolio_query = pd.read_sql_query(query_portfolio, sqliteConnection)
print(df_portfolio_query.to_latex(index=False))
```