Big Data Analytics Assignment 2

Question 1

```
# QUESTION 1
pandas_df = pd.read_csv("stockdata.csv")
```

In this question, the stock data CSV file is loaded into the data frame using the pandas read_csv() function. This function reads the data and puts it into rows and columns with each row corresponding to a record of a stock on a specific date and each column storing a specific field such as close price and name.

Question 2

```
# QUESTION 2
print("\n\n\nQUESTION 2\n")


# 2a

#gets all unique names from the dataframe and sorts
set_of_all_unique_names = pandas_df['Name'].unique()
sorted_set_of_all_unique_names = sorted(set_of_all_unique_names)

# 2b

# outputs number of values in the sorted set of all unique names
print("number of names: ", len(sorted_set_of_all_unique_names))


# 2c

# outputs first and last 5 names in sorted set of all unique names
print("\nfirst 5 names: ",sorted_set_of_all_unique_names[:5])
print("\nlast 5 names: ",sorted_set_of_all_unique_names[-5:])
```

In 2a, the set of all names is gathered by selecting “Name” from the data frame of stock data and then using the unique() function to remove duplicate names. It is then sorted into alphabetical order using the sorted() function. Then in 2b, the number of names is output by using the len() function on the sorted set of all of the unique names which returns the length of the set. In 2c, outputs the first and last five stock names from the set by getting a subset of the set of sorted names using [:5] and [-5:].

```
QUESTION 2

number of names:  500

first 5 names:  ['A', 'AAPL', 'ABBV', 'ABNB', 'ABT']

last 5 names:  ['XYL', 'YUM', 'ZBH', 'ZBRA', 'ZTS']
```

Question 3

```python
# QUESTION 3
print("\n\n\nQUESTION 3\n")

# 3a

# initialise the arrays to store the stocks with either a succicient or insuffiecient amount of data
suffiecient_data_stocks = []
insuffiecient_data_stocks = []

#loops through all the names of stocks in the set of all unique names
for name in set_of_all_unique_names:
    # get all records for current name and gets min and max dates
    individual_stock_data = pandas_df[pandas_df['Name'] == name]
    min_date = individual_stock_data['date'].min()
    max_date = individual_stock_data['date'].max()

    # if they have only traded between the two dates they will be added to insufficient_data_stocks
    # to be output in 3b and the others will be stored in sufficient_data stocks
    if min_date <= '2019-11-01' and max_date >= '2022-10-31' :
        suffiecient_data_stocks.append(name)
    else:
        insuffiecient_data_stocks.append(name)


# 3b

print("removed names: ",insuffiecient_data_stocks)


# 3c

#outputs number of stocks that have sufficient amount of data
print("\nnames left: ",len(suffiecient_data_stocks))
```

In 3a, a for loop is used to loop through the stock names. Then, the data frame is searched for the stock with the matching name and all of its records are then stored in individual_stock_data where the min() and max() functions are applied on the date column to get the minimum and maximum trading dates for that stock. An if statement then checks whether the minimum date is before 1/11/2019 and that the maximum date is after 31/10/2022 these stocks are then added to sufficient_data_stocks, any stocks that don't meet this requirement are added to insufficient_data_stocks and are then output in 3b. In 3c, the len() function is used on sufficient_data stocks to output the number of stocks left over.

```
QUESTION 3

removed names:  ['ABNB', 'CARR', 'CEG', 'GEHC', 'GEV', 'KVUE', 'OTIS', 'PLTR', 'SOLV', 'VLTO']

names left:  490
```

Question 4

```python
# QUESTION 4

print("\n\n\nQUESTION 4\n")


# 4a

# filters dataframe to only contain records with names that appear in suffiecient_data_stocks from q3
suffiecient_data_df = pandas_df[pandas_df['Name'].isin(suffiecient_data_stocks)]
# groups by date and stores count for each of them (the number of records for each date)
record_counts_per_date = suffiecient_data_df.groupby('date').size()
# finds common dates by finding where the count of each date equals the count of suffient_data_stocks
common_dates = record_counts_per_date[record_counts_per_date == len(suffiecient_data_stocks)].index.tolist()


# 4b

# goes through common dates and adds dates that are between 2019-11-01 and 2022-10-31
filtered_common_dates = []
for date in common_dates:
    if '2019-11-01' <= date <= '2022-10-31':
        filtered_common_dates.append(date)

# 4c

#outputs the number of records in the filtered common dates
print("Dates left: ", len(filtered_common_dates))


# 4d

# outputs the first and last 5 dates in filtered common dates
print("\nFirst 5 dates: ", filtered_common_dates[:5])
print("\nLast 5 dates: ", filtered_common_dates[-5:])
```

Question 4a starts with creating a new data frame storing all of the stocks that were identified to have a sufficient amount of data in question 3 by using the isin()

function. The filtered data frame is then grouped by "Date" with groupby(), and the size() function is used to count the number of records per date. The date is then added to common_dates if the number of records for that date is equal to the number of stocks that have a sufficient amount of data, as this means that there is a record for every valid stock for that date. In 4b a for loop that loops through the dates in common_dates, then an if statement is used to filter out dates that aren't before 1/11/2019 and aren't after 31/10/2022, these dates are then added to filtered_common_dates. In 4c, the len() function is used on filtered_common_dates to output the number of dates that are left after filtering them. In 4d, the first and last five common dates are output by getting a subset of the filtered common dates using [:5] and [-5:] to get the first five and last five elements.

```
QUESTION 4

Dates left:  755

First 5 dates:  ['2019-11-01', '2019-11-04', '2019-11-05', '2019-11-06', '2019-11-07']

Last 5 dates:  ['2022-10-25', '2022-10-26', '2022-10-27', '2022-10-28', '2022-10-31']
```

Question 5

```
# QUESTION 5
print("\n\n\nQUESTION 5\n")


# 5a

# filters the dataframe to only include rows where name is in suffiecient_data_stocks and date is in filtered_common_dates
sufficient_data_stocks_on_valid_days = pandas_df[(pandas_df['Name'].isin(suffiecient_data_stocks)) & (pandas_df['date'].isin(filtered_common_dates))]
# creates dataframe where each row is the date and each column is the stock name and stores the closing price for each stock name on that date
closing_price_df = (sufficient_data_stocks_on_valid_days.groupby(['date', 'Name'])['close'].first().unstack())


#5b
print("closing price data frame:\n\n",closing_price_df)
```

In 5a, the Data frame is filtered using the isin() function to include only rows where the stock names are in the sufficient_data_stocks list and the dates are in the filtered_common_dates list. The filtered data frame is then turned into a new one using the groupby() and unstack() functions, creating a Data frame called closing_price_df, which stores the closing prices for each stock on each date. In 5b, the closing_price_df Data frame is printed.

```
QUESTION 5

closing price data frame:

 Name              A     AAPL    ABBV     ABT  ...     YUM      ZBH    ZBRA     ZTS
date                                            ...
2019-11-01    76.32   62.290   79.13   82.61  ...   99.19  134.126  237.06  125.26
2019-11-04    76.73   63.845   82.14   82.22  ...   98.19  132.097  239.19  124.31
2019-11-05    75.47   64.080   81.47   80.90  ...   96.94  135.068  238.66  119.81
2019-11-06    74.94   63.842   80.35   82.38  ...   97.40  137.641  238.18  120.04
2019-11-07    75.68   64.527   81.08   82.91  ...   98.62  137.019  239.36  120.14
...             ...      ...     ...     ...  ...     ...      ...     ...     ...
2022-10-25   132.30  149.360  147.29   97.66  ...  108.01  109.110  266.57  149.26
2022-10-26   134.00  148.040  150.70   97.88  ...  111.42  111.360  271.17  151.13
2022-10-27   135.33  144.130  152.06   96.77  ...  112.88  111.000  274.89  149.52
2022-10-28   135.02  147.820  142.34   97.20  ...  114.10  111.870  271.72  150.52
2022-10-31   137.35  151.920  144.08   98.15  ...  116.40  112.350  278.84  149.84

[755 rows x 490 columns]
```

Question 6

```python
# 6a

# create a new datafram to store returns using rows and columns from closing_price_df
stock_returns_df = pd.DataFrame(index=closing_price_df.index[1:], columns=closing_price_df.columns)

#loops through each stock
for stock in closing_price_df.columns:
    # list to store returns for current stock
    stock_returns_list = []

    # loop starts at second row because no returns for first date
    for i in range(1, len(closing_price_df)):
        # gets old and new price
        current_close_price = closing_price_df.at[closing_price_df.index[i], stock]
        previous_close_price = closing_price_df.at[closing_price_df.index[i - 1], stock]
        # calculates return using formula and adds to returns list
        stock_returns_list.append((current_close_price - previous_close_price) / previous_close_price )

    # adds the stocks returns to the datafrae
    stock_returns_df[stock] = stock_returns_list



# 6b
print("stock returns data frame:\n\n", stock_returns_df)
```

In 6a, a new data frame, stock_returns_df, is created to store the daily returns for each stock, and it has the same structure as closing_price_df, with rows representing dates and columns representing stock names. A for loop is used to loop through each stock, and then a second for loop is used to loop through each date in the data frame (starting from the second date as the first day will have no return) and calculate the return for that day using the current and previous close price which is then added to a list of returns for that stock. After each date has

looped, the list of returns is added to the stock returns data frame for the stock. In 6b, the stock_returns_df is printed.

```
QUESTION 6

stock returns data frame:

Name                A      AAPL      ABBV  ...       ZBH      ZBRA       ZTS
date                                       ...
2019-11-04   0.005372  0.024964  0.038039  ... -0.015128  0.008985 -0.007584
2019-11-05  -0.016421  0.003681 -0.008157  ...  0.022491 -0.002216 -0.036200
2019-11-06  -0.007023 -0.003714 -0.013747  ...  0.019050 -0.002011  0.001920
2019-11-07   0.009875  0.010730  0.009085  ... -0.004519  0.004954  0.000833
2019-11-08  -0.000396 -0.004866  0.012457  ...  0.000073 -0.008982 -0.031879
...               ...       ...       ...  ...       ...       ...       ...
2022-10-25   0.019182  0.023014 -0.009482  ...  0.015638  0.029188  0.006338
2022-10-26   0.012850 -0.008838  0.023152  ...  0.020621  0.017256  0.012528
2022-10-27   0.009925 -0.026412  0.009025  ... -0.003233  0.013718 -0.010653
2022-10-28  -0.002291  0.025602 -0.063922  ...  0.007838 -0.011532  0.006688
2022-10-31   0.017257  0.027736  0.012224  ...  0.004291  0.026203 -0.004518

[754 rows x 490 columns]
```

Question 7

```
# QUESTION 7
print("\n\n\nQUESTION 7\n")


# 7a

# initialises pca and fits it to the stock_returns_df
pca = PCA()
pca.fit(stock_returns_df)

# 7b

# outputs top 5 components from pca
print("first 5 components from pca:\n\n", pca.components_[:5])
```

In 7a, principal component analysis is applied to stock_returns_df, using the fit() function from the PCA class to calculate the principal components. In 7b, the first five PCA components are outputted using the print() function and using [:5] to get the first five components.

```
QUESTION 7

first 5 components from pca:

[[-0.03053307 -0.03228756 -0.02491468 ... -0.04591686 -0.04311538 -0.0321947 ]
 [-0.05702523 -0.06911258 -0.01543889 ... -0.00071617 -0.04360575 -0.05302061]
 [-0.00428185  0.00599959 -0.02495018 ... -0.00225642 -0.00013562 -0.00782042]
 [-0.0256462  -0.03204411 -0.00718046 ...  0.02355439 -0.02285197  0.00351133]
 [ 0.01751229  0.02690375 -0.05168101 ... -0.02504652  0.049497   -0.04116101]]
```

Question 8

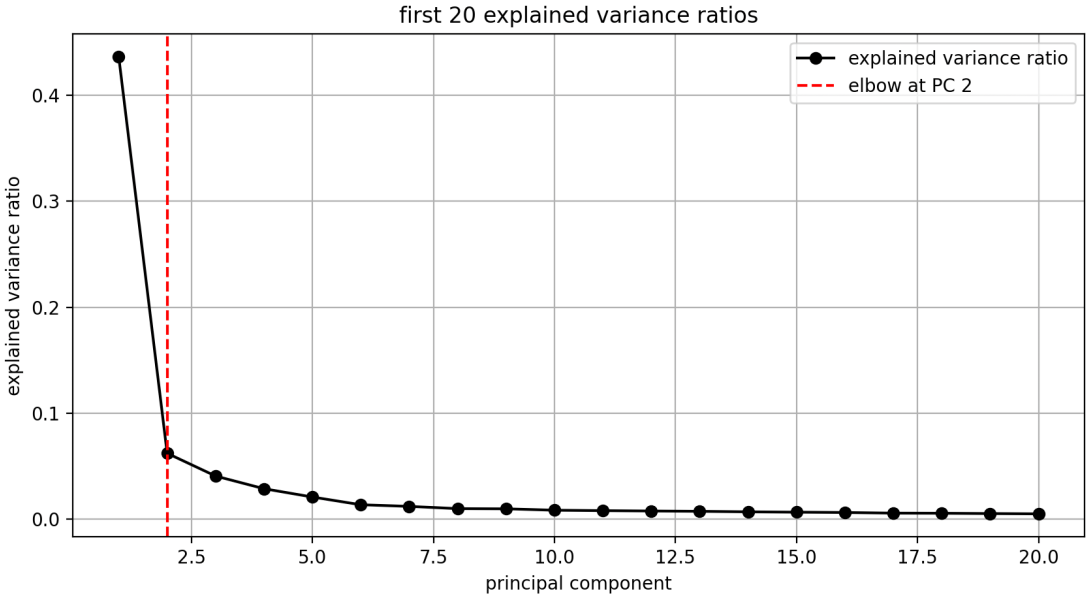
```python
# QUESTION 8
print("\n\n\nQUESTION 8\n")


# 8a

pca_explained_variance_ratios = pca.explained_variance_ratio_

# 8b

# outputs first ratio as percentage
first_pc_variance_ratio_percentage = pca_explained_variance_ratios[0] * 100
print("1st principal component variance ratio percentage: ", first_pc_variance_ratio_percentage, "%")


# 8c

#creates the graphs size
plt.figure(figsize=(10, 5))
#adds the datapoints and line to the graph
plt.plot(range(1, 21), pca_explained_variance_ratios[:20], marker='o', color ='black', label='explained variance ratio')
#adds label to graph, x axis and y axis
plt.title('first 20 explained variance ratios')
plt.xlabel('principal component')
plt.ylabel('explained variance ratio')
plt.grid(True)
```

```python
# 8d

# calculates first and second derivatives of explained variance ratios to find elbow
first_pca_derivative = np.diff(pca_explained_variance_ratios)
second_pca_derivative = np.diff(first_pca_derivative)

# gets the index of the elbow and the elbow point(the plus 1s are to account for
# the shortening of the array from the np.diff and converting from zero based to
# 1 based for the point)
pca_elbow_point = np.argmax(second_pca_derivative) + 2

# anottates the elbow on the graph
plt.axvline(x=pca_elbow_point, color= 'red', linestyle='--', label= f'elbow at PC {pca_elbow_point}')
plt.legend()
plt.show()
```

In 8a, the explained variance ratios are extracted using the explained_variance_ratio_ function of PCA. In 8b, the percentage of variance explained by the first principal component is extracted by accessing the element at index 0 in pca_explained_variance_ratios and multiplying it by 100 to convert

it into a percentage, and then it is printed. In 8c, matplotlib.pyplot is used to plot the first 20 explained variance ratios on a graph. The range() function is used to specify that only the first 20 ratios should be included. And pca_explained_variance_ratios[:20] specifies the y-values as the first 20 explained variance ratios. The graph is annotated using a label for the data being plotted, as well as a title for the graph and x and y-axis labels, as well as adding a grid to the graph. In 8d, the diff() function from NumPy is used twice to calculate the second derivative of the explained variance ratios, which is used to find the elbow point. The np.argmax() function is then used to find the index of the largest second derivative. The + 2 accounts for the shortening of the array caused by the diff() function and adjusting the pca_elbow_point from zero-based indexing to one-based indexing. Then, the elbow is annotated on the graph using a vertical line.

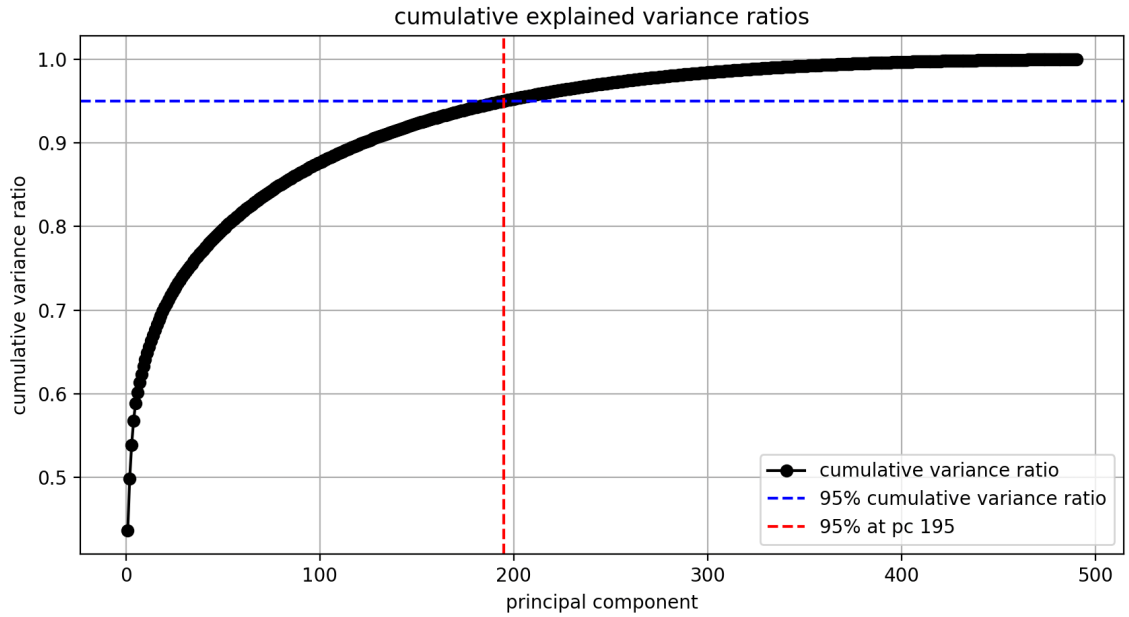
```
QUESTION 8

1st principal component variance ratio percentage:  43.65285667678825 %
2024-11-26 17:53:20.927 python[4937:246118] +[IMKClient subclass]: chose IMKClient_Legacy
2024-11-26 17:53:20.927 python[4937:246118] +[IMKInputSession subclass]: chose IMKInputSession_Legacy
```



Question 9

```
# QUESTION 9


# 9a

cumulative_pca_variance_ratio = np.cumsum(pca_explained_variance_ratios)


# 9b

#creates the graphs size
plt.figure(figsize=(10,5))
#adds the datapoints and line to the graph
plt.plot(range(1, len(cumulative_pca_variance_ratio) + 1), cumulative_pca_variance_ratio, marker='o', color ='black', label='cumulative variance ratio')
#adds label to graph, x axis and y axis
plt.title('cumulative explained variance ratios')
plt.xlabel('principal component')
plt.ylabel('cumulative variance ratio')
plt.grid(True)


# 9c

#gets number of components to explain 95% variance
num_pc_95 = np.argmax(cumulative_pca_variance_ratio >= 0.95) + 1

# adds x and y lines at 95% variance on graph
plt.axhline(y=0.95, color='blue', linestyle='--', label='95% cumulative variance ratio')
plt.axvline(x=num_pc_95, color='red', linestyle='--', label=f'95% at pc {num_pc_95}')
plt.legend()
plt.show()
```

In 9a, the np.cumsum() function is used to calculate the cumulative explained variance ratios using the explained_variance_ratio_ function of pca. This function computes the cumulative sum of the explained variance ratios, where each value represents the total variance explained by all principal components up to that point. In 9b, the cumulative variance ratios are plotted using the principal components on the x-axis and the cumulative variance ratio for the y-axis. They are then labelled, and a grid is added. In 9c, the np.argmax() function is used to find the index of the first cumulative variance ratio that meets or exceeds 95%. The +1 adjusts from zero-based to one-based indexing. Then, the graph is annotated on the graph using 2 x and y lines.



Question 10

```
# QUESTION 10
print("\n\n\nQUESTION 10\n")


# 10a
normalised_stock_returns = (stock_returns_df - stock_returns_df.mean()) / stock_returns_df.std()


# 10b

# initialises pca_normalised and fits it to the normalised_stock_returns_df
pca_normalised = PCA()
pca_normalised.fit(normalised_stock_returns)

# outputs first 5 components from pca normalise
print("first 5 components from pca normalised:\n", pca_normalised.components_[:5])
```

```
# 10c

explained_variance_ratios_normalised = pca_normalised.explained_variance_ratio_

# outputs first normalised ratio as percentage
first_pc_variance_normalised = explained_variance_ratios_normalised[0] * 100
print("\n\n1st normalised principal component variance ratio percentage: ",first_pc_variance_normalised,"%")

#creates the graphs size
plt.figure(figsize=(10, 5))
#adds the datapoints and line to the graph
plt.plot(range(1, 21), explained_variance_ratios_normalised[:20], marker='o', color ='black', label='explained variance ratio')
#adds label to graph, x axis and y axis
plt.title('first 20 normalised explained variance ratios')
plt.xlabel('principal component')
plt.ylabel('explained variance ratio')
plt.grid(True)

# calculates first and second derivatives of explained variance ratios to find elbow
first_derivative_normalised = np.diff(explained_variance_ratios_normalised)
second_derivative_normalised = np.diff(first_derivative_normalised)
# gets the index of the elbow and the elbow point(the plus 1s are to account for
# the shortening of the array from the np.diff and converting from zero based to
# 1 based for the point)
elbow_index_normalised = np.argmax(second_derivative_normalised) + 1
elbow_point_normalised = elbow_index_normalised + 1

# anottates the elbow on the graph
plt.axvline(x=elbow_point_normalised, color='red', linestyle='--', label=f'elbow at PC {elbow_point_normalised}')
plt.legend()
plt.show()
```

```
# 10d

cumulative_pca_variance_ratios_normalised = np.cumsum(explained_variance_ratios_normalised)

#creates the graphs size
plt.figure(figsize=(10, 5))
#adds the datapoints and line to the graph
plt.plot(range(1, len(cumulative_pca_variance_ratios_normalised) + 1),cumulative_pca_variance_ratios_normalised, marker='o', color ='black',label='cumulative variance ratio')
#adds label to graph, x axis and y axis
plt.title('normalised cumulative variance ratios')
plt.xlabel('principal component')
plt.ylabel('cumulative variance ratio')
plt.grid(True)

#gets number of components to explain 95% variance
num_pc_95_normalised = np.argmax(cumulative_pca_variance_ratios_normalised >= 0.95) + 1

# adds x and y lines at 95% variance on graph
plt.axhline(y=0.95, color='blue', linestyle='--', label='95% cumulative variance ratio')
plt.axvline(x=num_pc_95_normalised, color='red', linestyle='--', label=f'95% at pc {num_pc_95_normalised}')
plt.legend()
plt.show()
```

In 10a, the returns Data frame is normalised to have zero mean and unit variance for each stock, which is done by subtracting the mean and dividing by the standard deviation. In 10b, PCA is applied to the normalised returns Data frame using the fit() method to recalculate the principal components for the normalised data. In questions 10c and 10d, the same operations as in questions 8 and 9 are performed but with normalised data instead.

The normalisation of the stock returns has a significant impact on the PCA results. In the unnormalised data frame, the first principal component explains 43.65% of the variance, but after it is normalised, it decreases by 2.8% to 42.42%. This shows a more even distribution of variance across the components. As well as this, the number of components needed to explain 95%of the variance increases from 195 in the unnormalised data frame by 21 to 216 in the normalised data

frame. This, once again, shows how the normalisation spreads the variance more evenly.

```
QUESTION 10

first 5 components from pca normalised:
 [[-0.04721487 -0.04291645 -0.03819902 ... -0.05032292 -0.04976909 -0.04953298]
 [-0.05893595 -0.06277619 -0.02346537 ...  0.0185683  -0.01900993 -0.05791755]
 [ 0.0477138   0.05759062 -0.01619209 ...  0.00444402  0.04034065  0.03497978]
 [ 0.04287662  0.04973237 -0.04098237 ... -0.03946162  0.06441288 -0.04746672]
 [ 0.00211802  0.00489516 -0.06471687 ... -0.00530118  0.01146898 -0.00565032]]


1st normalised principal component variance ratio percentage:  42.41766357639556 %
```