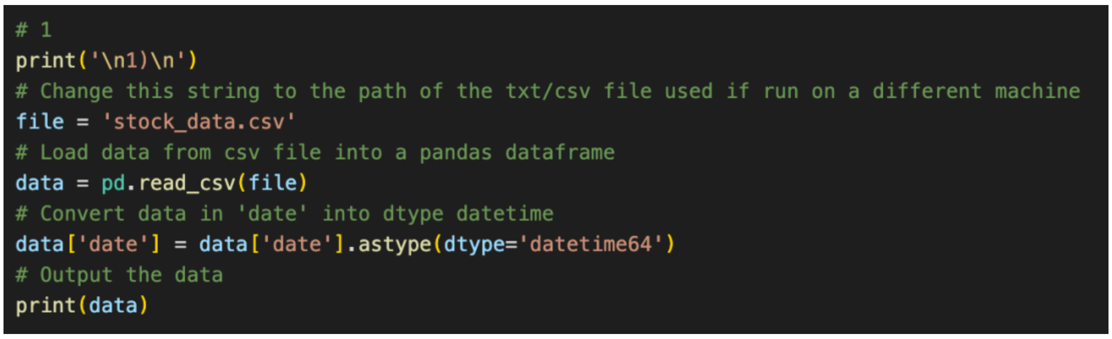
Noushin Islam Student No. 201508438

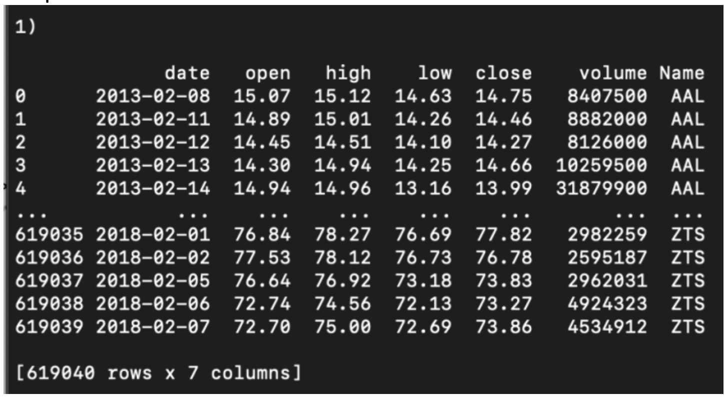
Assignment 2 – COMP336
Report

Task 1

Brief:
Unzip stock_data.zip, which contains historical market data for stocks in the S&P 500 index. The S&P 500 is a stock market index tracking the performance of 500 large companies listed on stock exchanges in the United States. Load the data into a pandas dataframe.

Source Code:


```
# 1
print('\n1)\n')
# Change this string to the path of the txt/csv file used if run on a different machine
file = 'stock_data.csv'
# Load data from csv file into a pandas dataframe
data = pd.read_csv(file)
# Convert data in 'date' into dtype datetime
data['date'] = data['date'].astype(dtype='datetime64')
# Output the data
print(data)
```

Output:


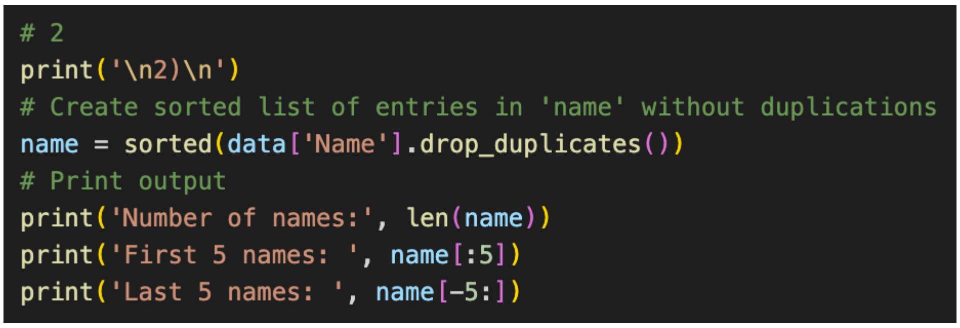
```
1)

             date   open   high    low  close    volume Name
0      2013-02-08  15.07  15.12  14.63  14.75   8407500  AAL
1      2013-02-11  14.89  15.01  14.26  14.46   8882000  AAL
2      2013-02-12  14.45  14.51  14.10  14.27   8126000  AAL
3      2013-02-13  14.30  14.94  14.25  14.66  10259500  AAL
4      2013-02-14  14.94  14.96  13.16  13.99  31879900  AAL
...           ...    ...    ...    ...    ...       ...  ...
619035 2018-02-01  76.84  78.27  76.69  77.82   2982259  ZTS
619036 2018-02-02  77.53  78.12  76.73  76.78   2595187  ZTS
619037 2018-02-05  76.64  76.92  73.18  73.83   2962031  ZTS
619038 2018-02-06  72.74  74.56  72.13  73.27   4924323  ZTS
619039 2018-02-07  72.70  75.00  72.69  73.86   4534912  ZTS

[619040 rows x 7 columns]
```

Description:
To load the data into a data frame I use the pd.read_csv() function. To ensure compatibility for tasks later I convert data in 'date' into dtype datetime.

Task 2

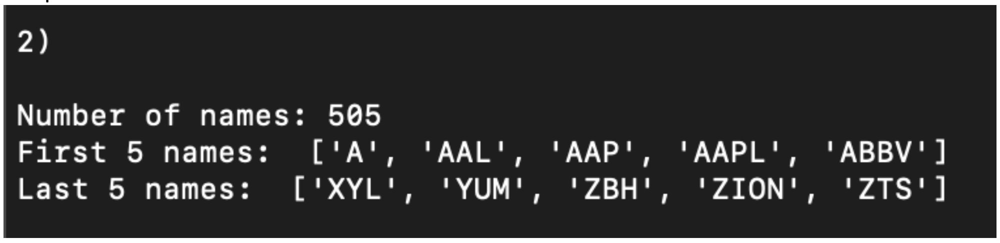
Brief:
Identify the set of all names in the data, and sort these names in alphabetical order. How many are there? List the first and last 5 names.

Source Code:



```python
# 2
print('\n2)\n')
# Create sorted list of entries in 'name' without duplications
name = sorted(data['Name'].drop_duplicates())
# Print output
print('Number of names:', len(name))
print('First 5 names: ', name[:5])
print('Last 5 names: ', name[-5:])
```

Output:



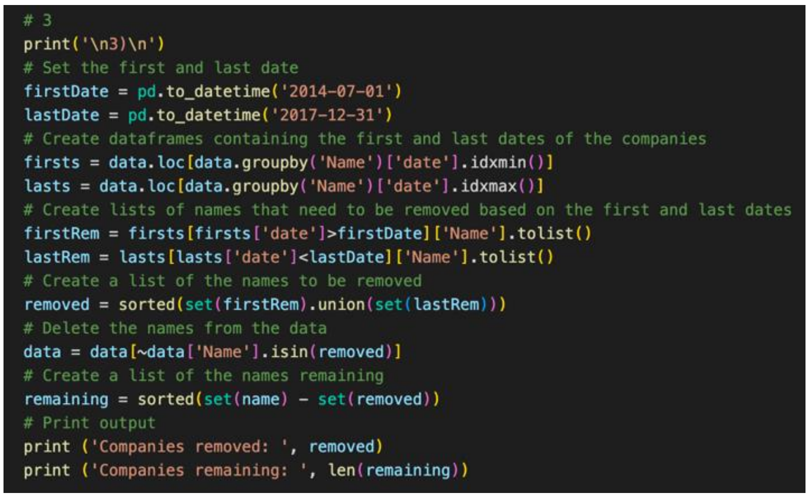
```
2)

Number of names: 505
First 5 names:  ['A', 'AAL', 'AAP', 'AAPL', 'ABBV']
Last 5 names:  ['XYL', 'YUM', 'ZBH', 'ZION', 'ZTS']
```

Description:
To identify all unique names in the data frame I use the sorted() function with the .drop_duplicates() function.

Task 3

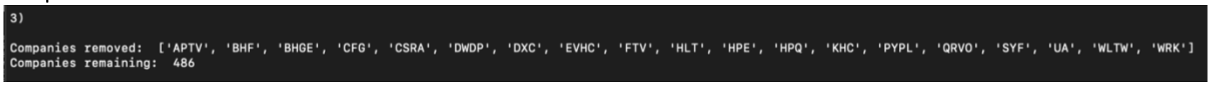
Brief:
Remove all names for which the first date is after 1st Jul 2014 or the last date is before 31st Dec 2017. Which names were removed? How many are left?

Source Code:



```
# 3
print('\n3)\n')
# Set the first and last date
firstDate = pd.to_datetime('2014-07-01')
lastDate = pd.to_datetime('2017-12-31')
# Create dataframes containing the first and last dates of the companies
firsts = data.loc[data.groupby('Name')['date'].idxmin()]
lasts = data.loc[data.groupby('Name')['date'].idxmax()]
# Create lists of names that need to be removed based on the first and last dates
firstRem = firsts[firsts['date']>firstDate]['Name'].tolist()
lastRem = lasts[lasts['date']<lastDate]['Name'].tolist()
# Create a list of the names to be removed
removed = sorted(set(firstRem).union(set(lastRem)))
# Delete the names from the data
data = data[~data['Name'].isin(removed)]
# Create a list of the names remaining
remaining = sorted(set(name) - set(removed))
# Print output
print ('Companies removed: ', removed)
print ('Companies remaining: ', len(remaining))
```

Output:



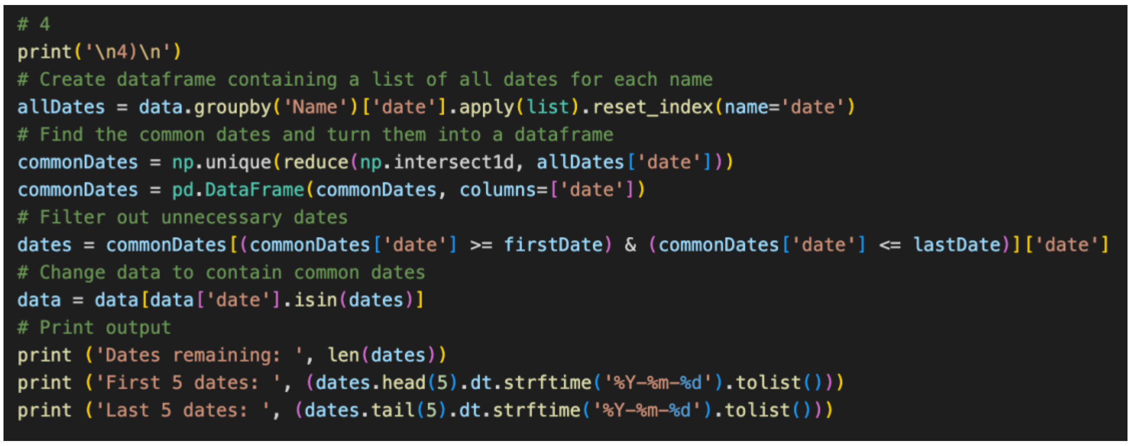
```
3)

Companies removed:  ['APTV', 'BHF', 'BHGE', 'CFG', 'CSRA', 'DWDP', 'DXC', 'EVHC', 'FTV', 'HLT', 'HPE', 'HPQ', 'KHC', 'PYPL', 'QRVO', 'SYF', 'UA', 'WLTW', 'WRK']
Companies remaining:  486
```

Description:
To set the first and last dates I use the pd.to_datetime() function. After which I use the .loc[] function to create two new data frames with all the first and last dates of each company. Using these I compare the first and last dates to filter out the companies that need to be removed. The latter is added to a list and then removed from the data.

Task 4

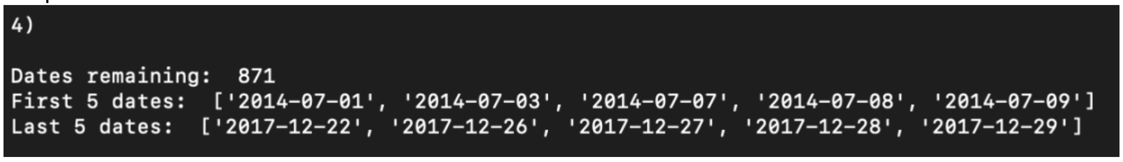
Brief:
Identify the set of dates that are common to all the remaining names. Remove all the dates that are before 1st Jul 2014 or after 31st Dec 2017. How many dates are left? What are the first and last 5 dates?

Source Code:



```
# 4
print('\n4)\n')
# Create dataframe containing a list of all dates for each name
allDates = data.groupby('Name')['date'].apply(list).reset_index(name='date')
# Find the common dates and turn them into a dataframe
commonDates = np.unique(reduce(np.intersect1d, allDates['date']))
commonDates = pd.DataFrame(commonDates, columns=['date'])
# Filter out unnecessary dates
dates = commonDates[(commonDates['date'] >= firstDate) & (commonDates['date'] <= lastDate)]['date']
# Change data to contain common dates
data = data[data['date'].isin(dates)]
# Print output
print ('Dates remaining: ', len(dates))
print ('First 5 dates: ', (dates.head(5).dt.strftime('%Y-%m-%d').tolist()))
print ('Last 5 dates: ', (dates.tail(5).dt.strftime('%Y-%m-%d').tolist()))
```

Output:



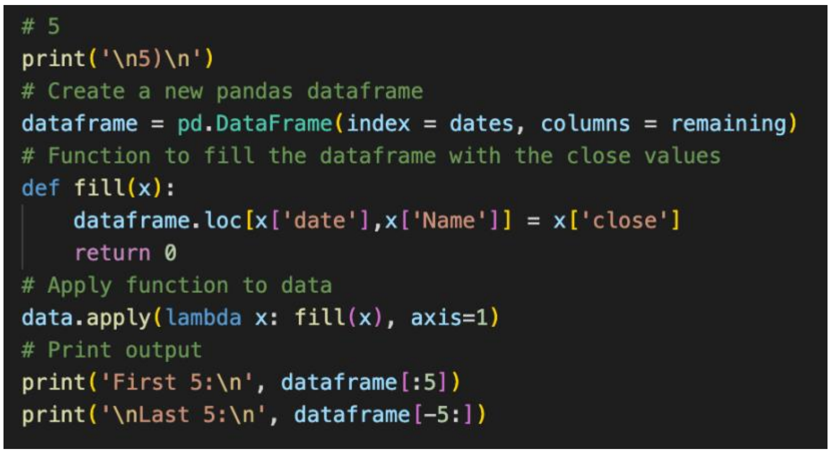
```
4)

Dates remaining:  871
First 5 dates:  ['2014-07-01', '2014-07-03', '2014-07-07', '2014-07-08', '2014-07-09']
Last 5 dates:  ['2017-12-22', '2017-12-26', '2017-12-27', '2017-12-28', '2017-12-29']
```

Description:
I start by creating a data frame containing all dates for each company, on which I use the np.unique() function to find all the common dates. Then I removed the unnecessary dates and applied it to the data.

Task 5

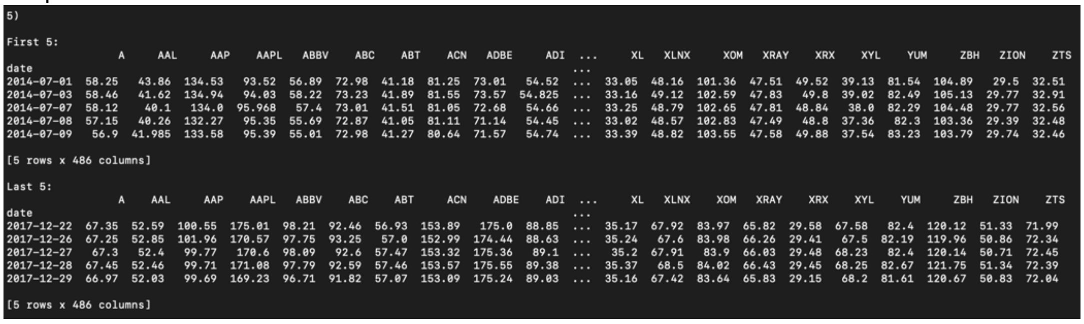
Brief:
Build a new pandas dataframe which has a column for each of the names from task 3 and a row for each of the dates from task 4. The dataframe should contain the "close" values for each corresponding name and date. Print the first 5 and last 5 rows of this dataframe.

Source Code:



```
# 5
print('\n5)\n')
# Create a new pandas dataframe
dataframe = pd.DataFrame(index = dates, columns = remaining)
# Function to fill the dataframe with the close values
def fill(x):
    dataframe.loc[x['date'],x['Name']] = x['close']
    return 0
# Apply function to data
data.apply(lambda x: fill(x), axis=1)
# Print output
print('First 5:\n', dataframe[:5])
print('\nLast 5:\n', dataframe[-5:])
```

Output:

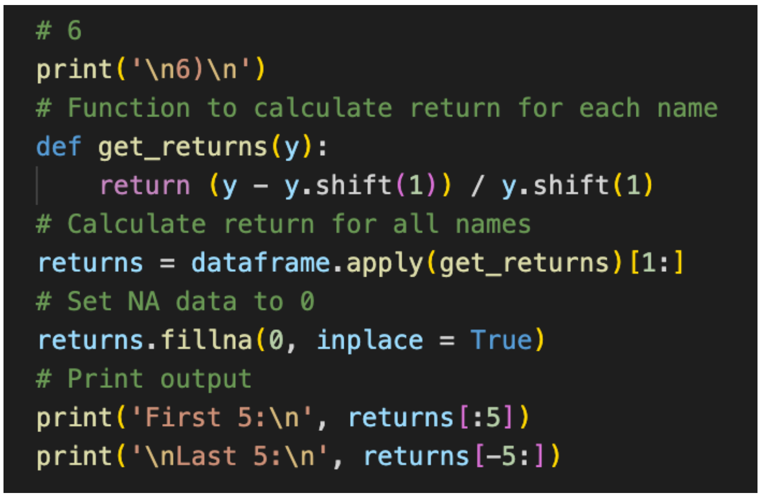


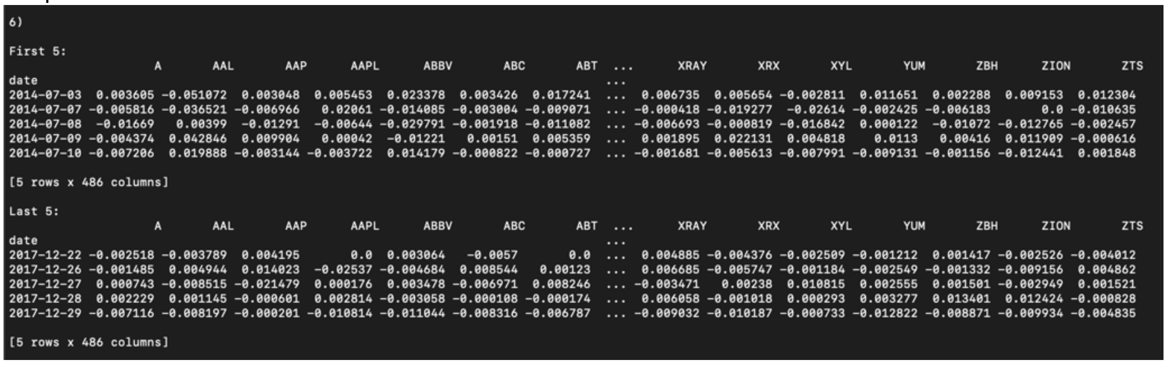
```
5)

First 5:
                A     AAL     AAP    AAPL   ABBV    ABC    ABT    ACN   ADBE     ADI  ...     XL   XLNX     XOM   XRAY    XRX    XYL    YUM     ZBH   ZION    ZTS
date                                                                                  ...                                                                      
2014-07-01  58.25   43.86  134.53   93.52  56.89  72.98  41.18  81.25  73.01   54.52  ...  33.05  48.16  101.36  47.51  49.52  39.13  81.54  104.89   29.5  32.51
2014-07-03  58.46   41.62  134.94   94.03  58.22  73.23  41.89  81.55  73.57  54.825  ...  33.16  49.12  102.59  47.83   49.8  39.02  82.49  105.13  29.77  32.91
2014-07-07  58.12    40.1   134.0  95.968   57.4  73.01  41.51  81.05  72.68   54.66  ...  33.25  48.79  102.65  47.81  48.84   38.0  82.29  104.48  29.77  32.56
2014-07-08  57.15   40.26  132.27   95.35  55.69  72.87  41.05  81.11  71.14   54.45  ...  33.02  48.57  102.83  47.49   48.8  37.36   82.3  103.36  29.39  32.48
2014-07-09   56.9  41.985  133.58   95.39  55.01  72.98  41.27  80.64  71.57   54.74  ...  33.39  48.82  103.55  47.58  49.88  37.54  83.23  103.79  29.74  32.46

[5 rows x 486 columns]

Last 5:
                A    AAL     AAP    AAPL   ABBV    ABC    ABT     ACN    ADBE    ADI  ...     XL   XLNX    XOM   XRAY    XRX    XYL    YUM     ZBH   ZION    ZTS
date                                                                                  ...                                                                     
2017-12-22  67.35  52.59  100.55  175.01  98.21  92.46  56.93  153.89   175.0  88.85  ...  35.17  67.92  83.97  65.82  29.58  67.58   82.4  120.12  51.33  71.99
2017-12-26  67.25  52.85  101.96  170.57  97.75  93.25   57.0  152.99  174.44  88.63  ...  35.24   67.6  83.98  66.26  29.41   67.5  82.19  119.96  50.86  72.34
2017-12-27   67.3   52.4   99.77   170.6  98.09   92.6  57.47  153.32  175.36   89.1  ...   35.2  67.91   83.9  66.03  29.48  68.23   82.4  120.14  50.71  72.45
2017-12-28  67.45  52.46   99.71  171.08  97.79  92.59  57.46  153.57  175.55  89.38  ...  35.37   68.5  84.02  66.43  29.45  68.25  82.67  121.75  51.34  72.39
2017-12-29  66.97  52.03   99.69  169.23  96.71  91.82  57.07  153.09  175.24  89.03  ...  35.16  67.42  83.64  65.83  29.15   68.2  81.61  120.67  50.83  72.04

[5 rows x 486 columns]
```

Description:
I create a new data frame using the dates and names to label the rows and columns respectively. Then I use a function to fill the data frame with the 'close' values for each corresponding name and date.

Task 6

Brief:
Create another dataframe containing returns calculated as:
return(name, date) = (close(name, date) - close(name, previous date)) / close(name, previous date)
Note that this dataframe should have one less row than the dataframe from task 5, because you can't calculate returns for the first date (there's no previous date). Print the first 5 and last 5 rows of this dataframe.

Source Code:

```
# 6
print('\n6)\n')
# Function to calculate return for each name
def get_returns(y):
    return (y - y.shift(1)) / y.shift(1)
# Calculate return for all names
returns = dataframe.apply(get_returns)[1:]
# Set NA data to 0
returns.fillna(0, inplace = True)
# Print output
print('First 5:\n', returns[:5])
print('\nLast 5:\n', returns[-5:])
```

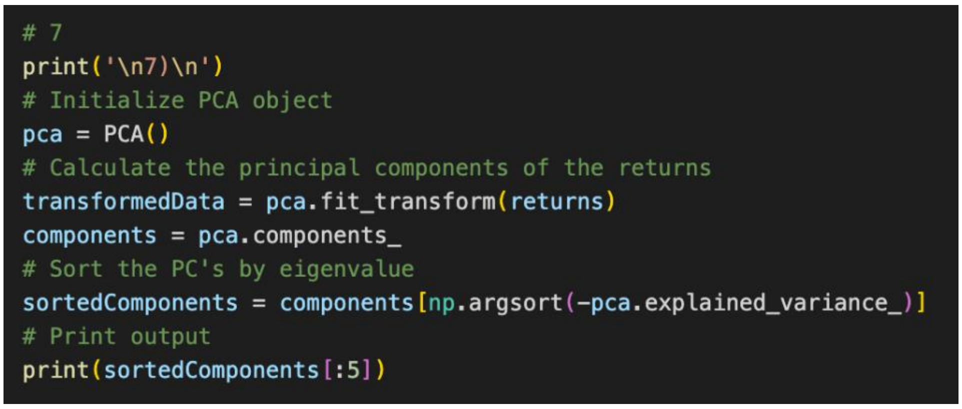
Output:

```
6)

First 5:
                    A       AAL       AAP      AAPL      ABBV       ABC       ABT  ...      XRAY       XRX       XYL       YUM       ZBH      ZION       ZTS
date                                                                               ...
2014-07-03  0.003605 -0.051072  0.003048  0.005453  0.023378  0.003426  0.017241  ...  0.006735  0.005654 -0.002811  0.011651  0.002288  0.009153  0.012304
2014-07-07 -0.005816 -0.036521 -0.006966   0.02061 -0.014085 -0.003004 -0.009071  ... -0.000418 -0.019277  -0.02614 -0.002425 -0.006183       0.0 -0.010635
2014-07-08  -0.01669   0.00399  -0.01291  -0.00644 -0.029791 -0.001918 -0.011082  ... -0.006693 -0.000819 -0.016842  0.000122  -0.01072 -0.012765 -0.002457
2014-07-09 -0.004374  0.042846  0.009904   0.00042  -0.01221   0.00151  0.005359  ...  0.001895  0.022131  0.004818    0.0113   0.00416  0.011909 -0.000616
2014-07-10 -0.007206  0.019888 -0.003144 -0.003722  0.014179 -0.000822 -0.000727  ... -0.001681 -0.005613 -0.007991 -0.009131 -0.001156 -0.012441  0.001848

[5 rows x 486 columns]

Last 5:
                    A       AAL       AAP      AAPL      ABBV       ABC       ABT  ...      XRAY       XRX       XYL       YUM       ZBH      ZION       ZTS
date                                                                               ...
2017-12-22 -0.002518 -0.003789  0.004195       0.0  0.003064   -0.0057       0.0  ...  0.004885 -0.004376 -0.002509 -0.001212  0.001417 -0.002526 -0.004012
2017-12-26 -0.001485  0.004944  0.014023  -0.02537 -0.004684  0.008544   0.00123  ...  0.006685 -0.005747 -0.001184 -0.002549 -0.001332 -0.009156  0.004862
2017-12-27  0.000743 -0.008515 -0.021479  0.000176  0.003478 -0.006971  0.008246  ... -0.003471   0.00238  0.010815  0.002555  0.001501 -0.002949  0.001521
2017-12-28  0.002229  0.001145 -0.000601  0.002814 -0.003058 -0.000108 -0.000174  ...  0.006058 -0.001018  0.000293  0.003277  0.013401  0.012424 -0.000828
2017-12-29 -0.007116 -0.008197 -0.000201 -0.010814 -0.011044 -0.008316 -0.006787  ... -0.009032 -0.010187 -0.000733 -0.012822 -0.008871 -0.009934 -0.004835

[5 rows x 486 columns]
```

Description:
To get the returns I use a function that I named get_returns(). The first row is removed as there is no return to be calculated.

Task 7

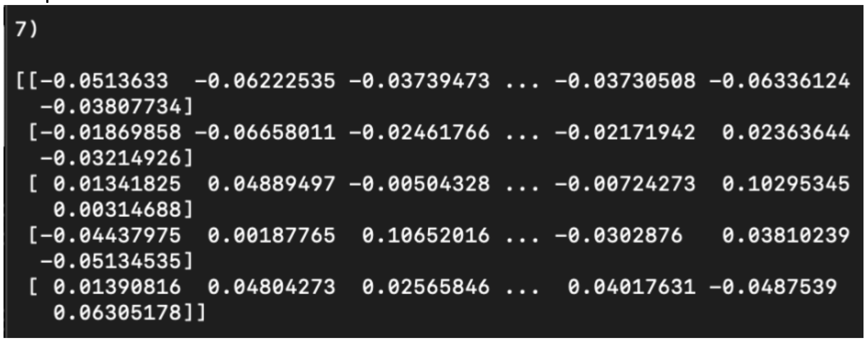
Brief:
Use the class sklearn.decomposition.PCA to calculate the principal components of the returns from task 6. Print the top five PCs when ranked according to their eigenvalue (the bigger the better).

Source Code:



```
# 7
print('\n7)\n')
# Initialize PCA object
pca = PCA()
# Calculate the principal components of the returns
transformedData = pca.fit_transform(returns)
components = pca.components_
# Sort the PC's by eigenvalue
sortedComponents = components[np.argsort(-pca.explained_variance_)]
# Print output
print(sortedComponents[:5])
```

Output:



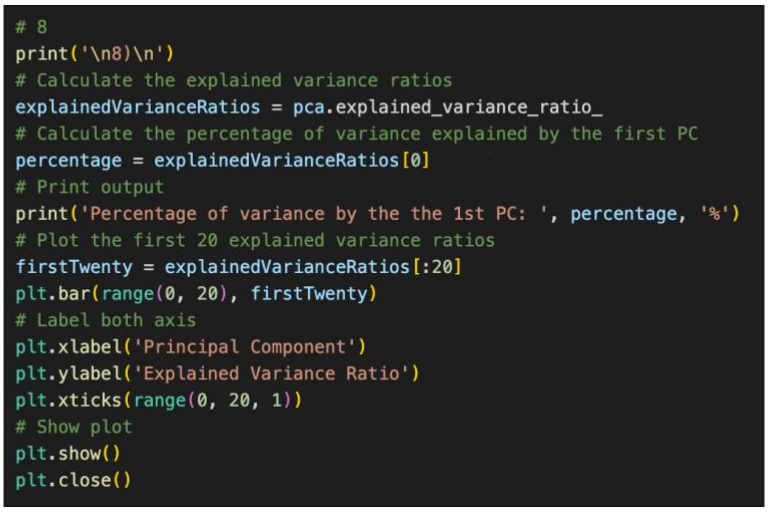
```
7)

[[-0.0513633  -0.06222535 -0.03739473 ... -0.03730508 -0.06336124
  -0.03807734]
 [-0.01869858 -0.06658011 -0.02461766 ... -0.02171942  0.02363644
  -0.03214926]
 [ 0.01341825  0.04889497 -0.00504328 ... -0.00724273  0.10295345
   0.00314688]
 [-0.04437975  0.00187765  0.10652016 ... -0.0302876   0.03810239
  -0.05134535]
 [ 0.01390816  0.04804273  0.02565846 ...  0.04017631 -0.0487539
   0.06305178]]
```

Description:
To calculate the principal components of the returns I use the pc.fit_transform and pca.components_ functions. Once I have the components, I calculate the eigenvalues by using pca.explained_variance_ and sort the PC’s based on that.

Task 8

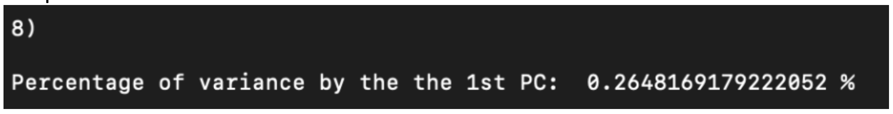
Brief:
For the principal components calculated in task 7, extract the explained variance ratios (you can get these from the PCA object). What percentage of variance is explained by the first principal component? Plot the first 20 explained variance ratios.

Source Code:

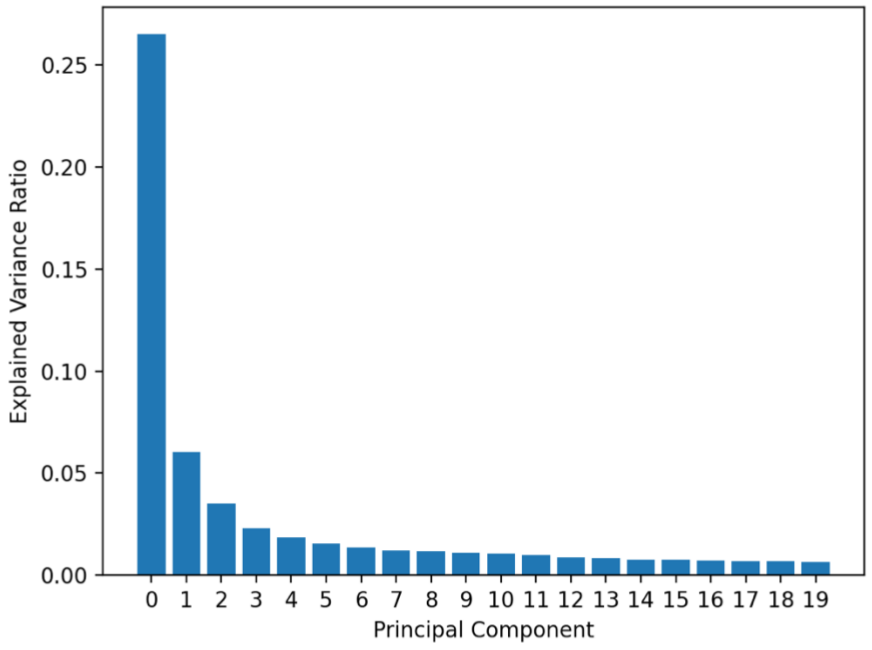


```python
# 8
print('\n8)\n')
# Calculate the explained variance ratios
explainedVarianceRatios = pca.explained_variance_ratio_
# Calculate the percentage of variance explained by the first PC
percentage = explainedVarianceRatios[0]
# Print output
print('Percentage of variance by the the 1st PC: ', percentage, '%')
# Plot the first 20 explained variance ratios
firstTwenty = explainedVarianceRatios[:20]
plt.bar(range(0, 20), firstTwenty)
# Label both axis
plt.xlabel('Principal Component')
plt.ylabel('Explained Variance Ratio')
plt.xticks(range(0, 20, 1))
# Show plot
plt.show()
plt.close()
```

Output:



```
8)

Percentage of variance by the the 1st PC:  0.2648169179222052 %
```

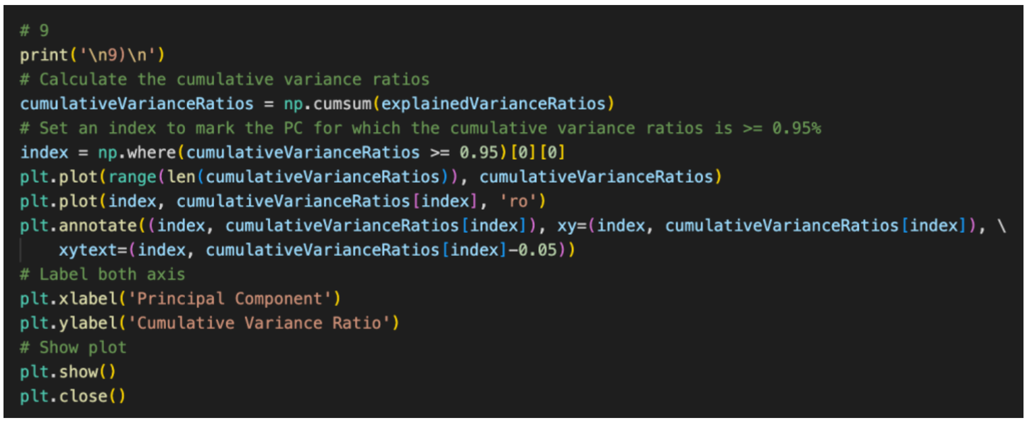


Description:
To calculate the explained variance ratios, I use the pca.explained_variance_ratio_ function. I also get the percentage of variance by the 1st PC. After which I plot the first 20 explained variance ratios.

Task 9

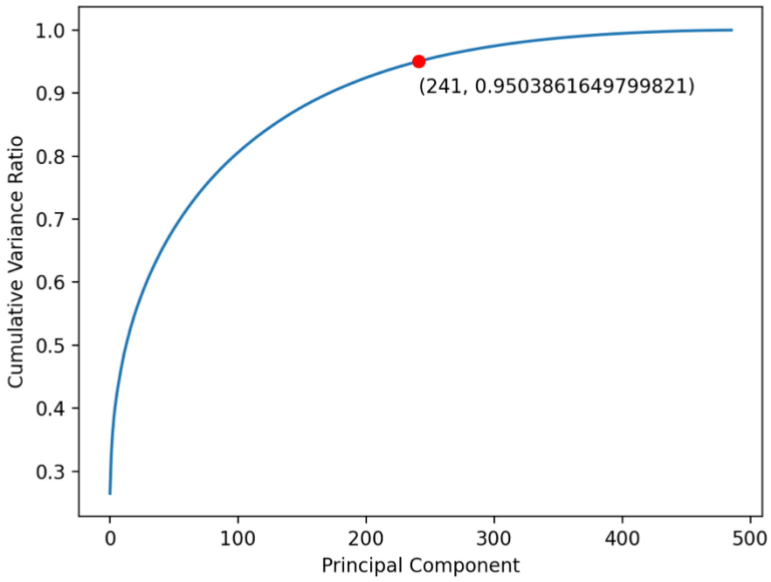
Brief:
Calculate the cumulative variance ratios using numpy.cumsum on the list of explained variance ratios from task 8. Plot all these cumulative variance ratios (x-axis = principal component, y-axis = cumulative variance ratio). Mark on your plot the principal component for which the cumulative variance ratio is greater than or equal to 95%.

Source Code:



```
# 9
print('\n9)\n')
# Calculate the cumulative variance ratios
cumulativeVarianceRatios = np.cumsum(explainedVarianceRatios)
# Set an index to mark the PC for which the cumulative variance ratios is >= 0.95%
index = np.where(cumulativeVarianceRatios >= 0.95)[0][0]
plt.plot(range(len(cumulativeVarianceRatios)), cumulativeVarianceRatios)
plt.plot(index, cumulativeVarianceRatios[index], 'ro')
plt.annotate((index, cumulativeVarianceRatios[index]), xy=(index, cumulativeVarianceRatios[index]), \
    xytext=(index, cumulativeVarianceRatios[index]-0.05))
# Label both axis
plt.xlabel('Principal Component')
plt.ylabel('Cumulative Variance Ratio')
# Show plot
plt.show()
plt.close()
```

Output:

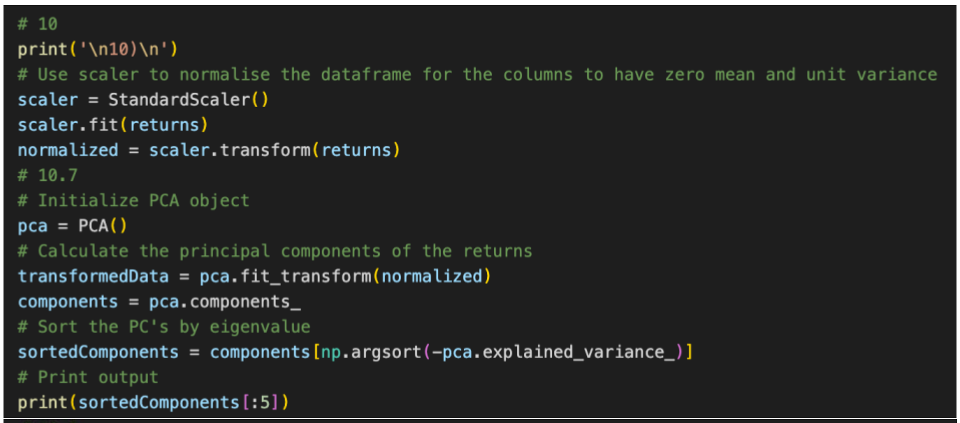


Description:
To calculate the cumulative variance ratios, I use the np.cumsum() function with the explained variance ratios calculated in task 8. Then I set an index for the point where the cumulative variance ratio is >= 0.95%.

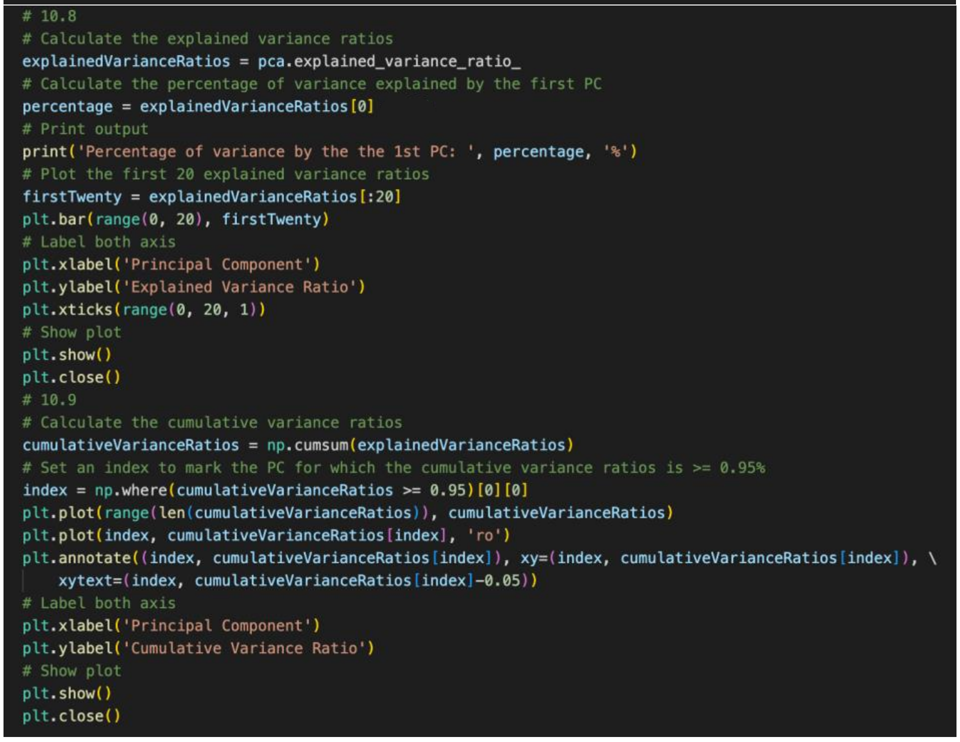
Task 10

Brief:
Normalise your dataframe from task 6 so that the columns have zero mean and unit variance.
Repeat tasks 7-9 for this new dataframe.

Source Code:

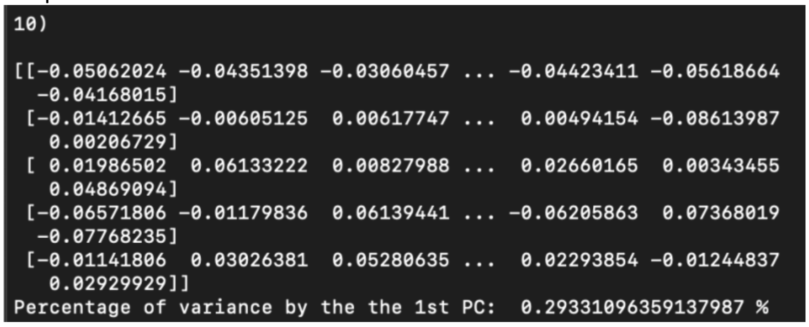


```
# 10
print('\n10)\n')
# Use scaler to normalise the dataframe for the columns to have zero mean and unit variance
scaler = StandardScaler()
scaler.fit(returns)
normalized = scaler.transform(returns)
# 10.7
# Initialize PCA object
pca = PCA()
# Calculate the principal components of the returns
transformedData = pca.fit_transform(normalized)
components = pca.components_
# Sort the PC's by eigenvalue
sortedComponents = components[np.argsort(-pca.explained_variance_)]
# Print output
print(sortedComponents[:5])
```

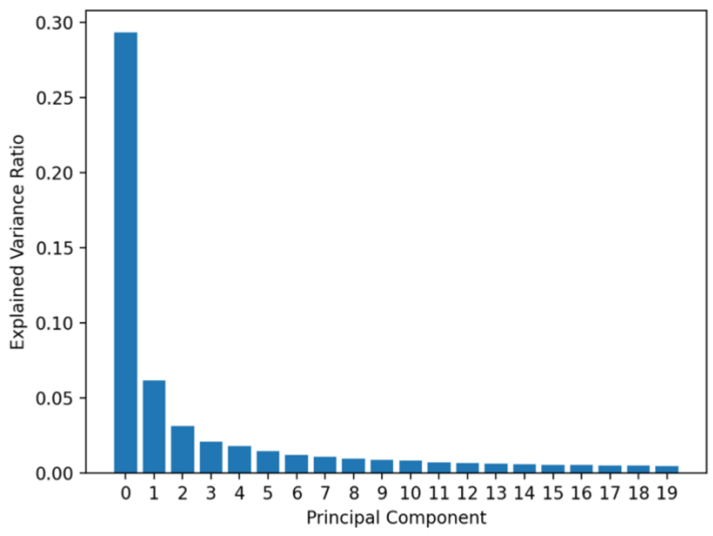


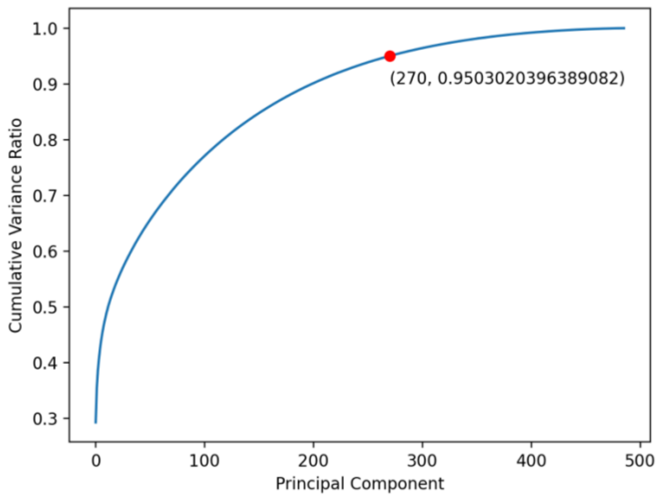
```
# 10.8
# Calculate the explained variance ratios
explainedVarianceRatios = pca.explained_variance_ratio_
# Calculate the percentage of variance explained by the first PC
percentage = explainedVarianceRatios[0]
# Print output
print('Percentage of variance by the the 1st PC: ', percentage, '%')
# Plot the first 20 explained variance ratios
firstTwenty = explainedVarianceRatios[:20]
plt.bar(range(0, 20), firstTwenty)
# Label both axis
plt.xlabel('Principal Component')
plt.ylabel('Explained Variance Ratio')
plt.xticks(range(0, 20, 1))
# Show plot
plt.show()
plt.close()
# 10.9
# Calculate the cumulative variance ratios
cumulativeVarianceRatios = np.cumsum(explainedVarianceRatios)
# Set an index to mark the PC for which the cumulative variance ratios is >= 0.95%
index = np.where(cumulativeVarianceRatios >= 0.95)[0][0]
plt.plot(range(len(cumulativeVarianceRatios)), cumulativeVarianceRatios)
plt.plot(index, cumulativeVarianceRatios[index], 'ro')
plt.annotate((index, cumulativeVarianceRatios[index]), xy=(index, cumulativeVarianceRatios[index]), \
    xytext=(index, cumulativeVarianceRatios[index]-0.05))
# Label both axis
plt.xlabel('Principal Component')
plt.ylabel('Cumulative Variance Ratio')
# Show plot
plt.show()
plt.close()
```

Output:



```
10)

[[-0.05062024 -0.04351398 -0.03060457 ... -0.04423411 -0.05618664
  -0.04168015]
 [-0.01412665 -0.00605125  0.00617747 ...  0.00494154 -0.08613987
   0.00206729]
 [ 0.01986502  0.06133222  0.00827988 ...  0.02660165  0.00343455
   0.04869094]
 [-0.06571806 -0.01179836  0.06139441 ... -0.06205863  0.07368019
  -0.07768235]
 [-0.01141806  0.03026381  0.05280635 ...  0.02293854 -0.01244837
   0.02929929]]
Percentage of variance by the the 1st PC:  0.29331096359137987 %
```





Description:
After using StandardScaler() to normalize the data I repeat the steps taken in task 7-9 on the normalized data.