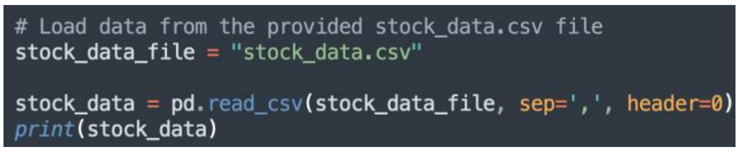
MIKOLAJ KLINIEWSKI – PANDAS - REPORT

Question 1:

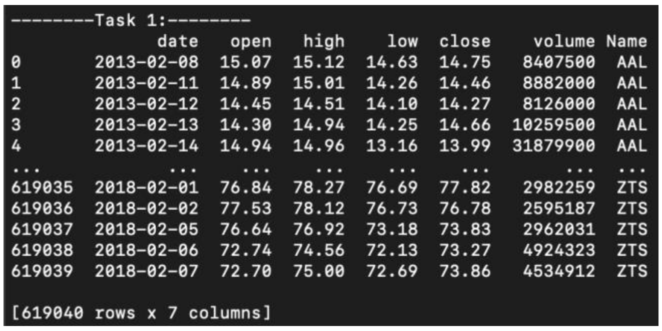
“Load the data into a pandas data frame.”

Code:



```
# Load data from the provided stock_data.csv file
stock_data_file = "stock_data.csv"

stock_data = pd.read_csv(stock_data_file, sep=',', header=0)
print(stock_data)
```

Output:



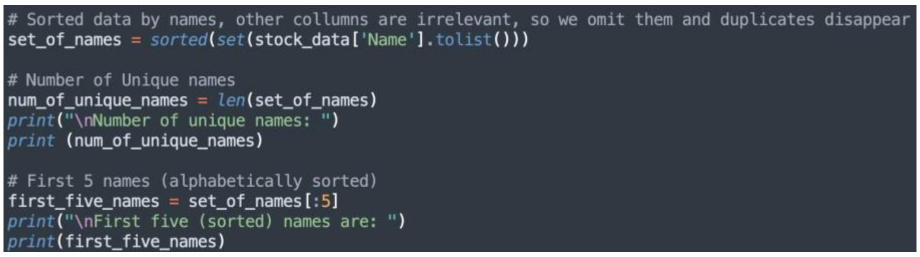
```
--------Task 1:--------
              date   open   high    low  close    volume Name
0       2013-02-08  15.07  15.12  14.63  14.75   8407500  AAL
1       2013-02-11  14.89  15.01  14.26  14.46   8882000  AAL
2       2013-02-12  14.45  14.51  14.10  14.27   8126000  AAL
3       2013-02-13  14.30  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]
```

Question 2:

“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.”

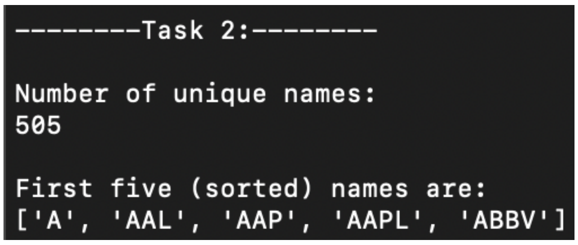
I use set() to eliminate the duplicates. The data from the set is unordered so I must sort it.

Code:



```
# Sorted data by names, other collumns are irrelevant, so we omit them and duplicates disappear
set_of_names = sorted(set(stock_data['Name'].tolist()))

# Number of Unique names
num_of_unique_names = len(set_of_names)
print("\nNumber of unique names: ")
print (num_of_unique_names)

# First 5 names (alphabetically sorted)
first_five_names = set_of_names[:5]
print("\nFirst five (sorted) names are: ")
print(first_five_names)
```

Output:



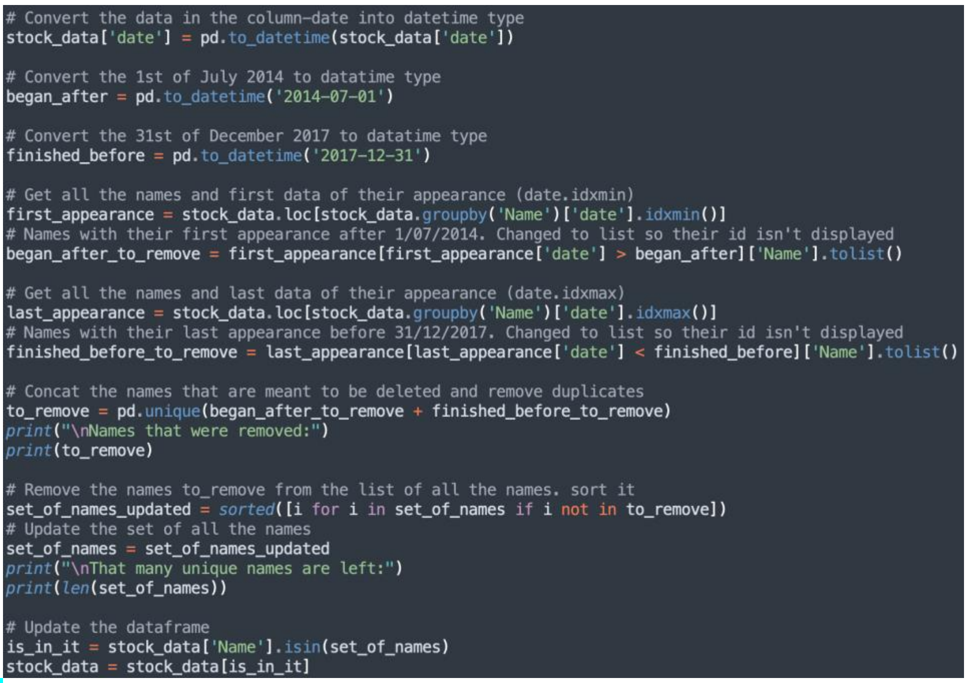
```
--------Task 2:--------

Number of unique names:
505

First five (sorted) names are:
['A', 'AAL', 'AAP', 'AAPL', 'ABBV']
```

Question 3:

“Filter out 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?”

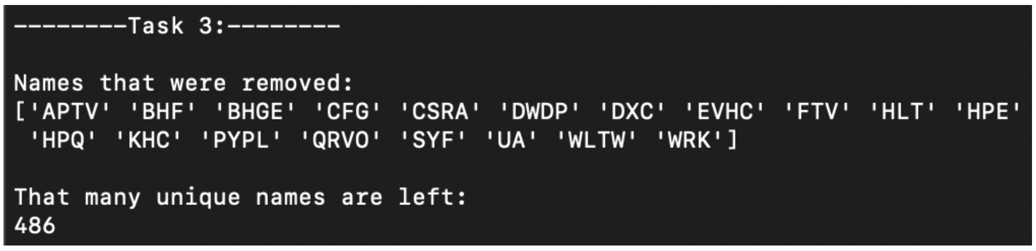
I find the names that began after 2014-07-01, and the names that finished before 2017-12-31, sum them, remove duplicates, and print them out. Update the stock_data

Code:



```
# Convert the data in the column-date into datetime type
stock_data['date'] = pd.to_datetime(stock_data['date'])

# Convert the 1st of July 2014 to datatime type
began_after = pd.to_datetime('2014-07-01')

# Convert the 31st of December 2017 to datatime type
finished_before = pd.to_datetime('2017-12-31')

# Get all the names and first data of their appearance (date.idxmin)
first_appearance = stock_data.loc[stock_data.groupby('Name')['date'].idxmin()]
# Names with their first appearance after 1/07/2014. Changed to list so their id isn't displayed
began_after_to_remove = first_appearance[first_appearance['date'] > began_after]['Name'].tolist()

# Get all the names and last data of their appearance (date.idxmax)
last_appearance = stock_data.loc[stock_data.groupby('Name')['date'].idxmax()]
# Names with their last appearance before 31/12/2017. Changed to list so their id isn't displayed
finished_before_to_remove = last_appearance[last_appearance['date'] < finished_before]['Name'].tolist()

# Concat the names that are meant to be deleted and remove duplicates
to_remove = pd.unique(began_after_to_remove + finished_before_to_remove)
print("\nNames that were removed:")
print(to_remove)

# Remove the names to_remove from the list of all the names. sort it
set_of_names_updated = sorted([i for i in set_of_names if i not in to_remove])
# Update the set of all the names
set_of_names = set_of_names_updated
print("\nThat many unique names are left:")
print(len(set_of_names))

# Update the dataframe
is_in_it = stock_data['Name'].isin(set_of_names)
stock_data = stock_data[is_in_it]
```

Output:



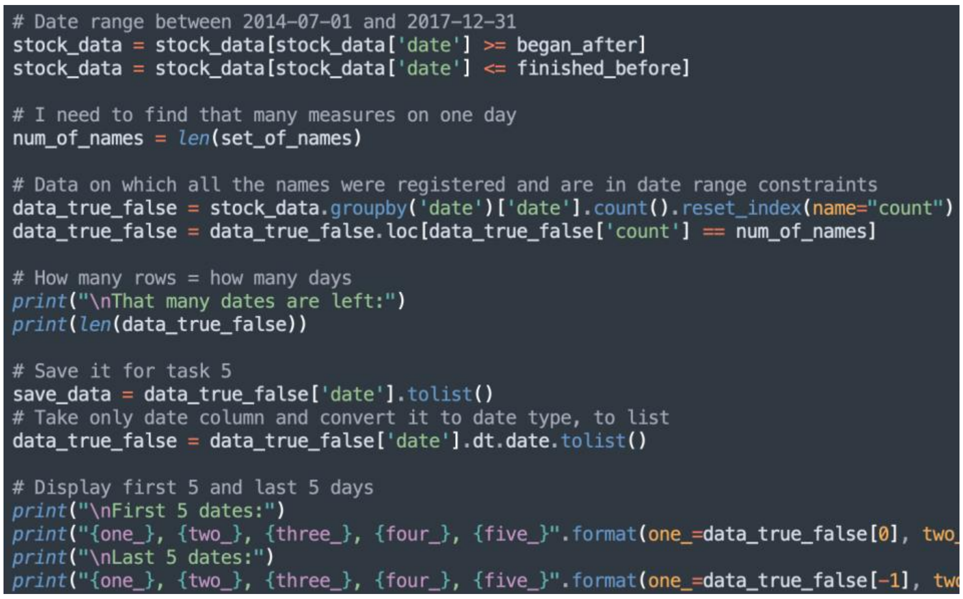
```
--------Task 3:--------

Names that were removed:
['APTV' 'BHF' 'BHGE' 'CFG' 'CSRA' 'DWDP' 'DXC' 'EVHC' 'FTV' 'HLT' 'HPE'
 'HPQ' 'KHC' 'PYPL' 'QRVO' 'SYF' 'UA' 'WLTW' 'WRK']

That many unique names are left:
486
```

Question 4:

"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?"

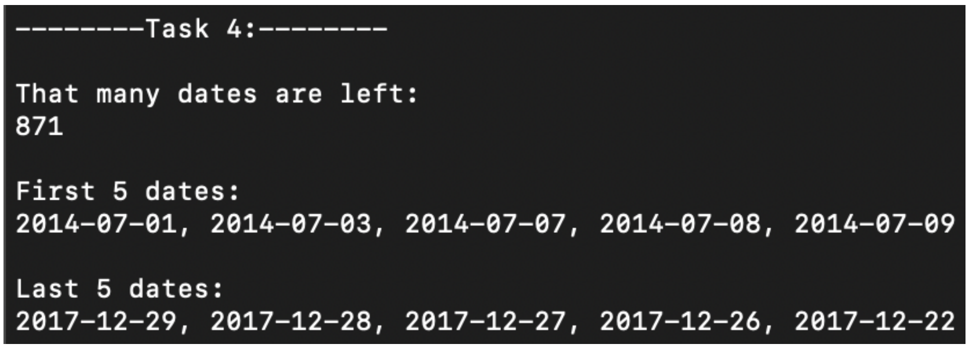
Find the dates for which len(set_of_names) points were registered. I used 'count' for dates

Code:



```python
# Date range between 2014-07-01 and 2017-12-31
stock_data = stock_data[stock_data['date'] >= began_after]
stock_data = stock_data[stock_data['date'] <= finished_before]

# I need to find that many measures on one day
num_of_names = len(set_of_names)

# Data on which all the names were registered and are in date range constraints
data_true_false = stock_data.groupby('date')['date'].count().reset_index(name="count")
data_true_false = data_true_false.loc[data_true_false['count'] == num_of_names]

# How many rows = how many days
print("\nThat many dates are left:")
print(len(data_true_false))

# Save it for task 5
save_data = data_true_false['date'].tolist()
# Take only date column and convert it to date type, to list
data_true_false = data_true_false['date'].dt.date.tolist()

# Display first 5 and last 5 days
print("\nFirst 5 dates:")
print("{one_}, {two_}, {three_}, {four_}, {five_}".format(one_=data_true_false[0], two_
print("\nLast 5 dates:")
print("{one_}, {two_}, {three_}, {four_}, {five_}".format(one_=data_true_false[-1], tw
```

Output:



```
--------Task 4:--------

That many dates are left:
871

First 5 dates:
2014-07-01, 2014-07-03, 2014-07-07, 2014-07-08, 2014-07-09

Last 5 dates:
2017-12-29, 2017-12-28, 2017-12-27, 2017-12-26, 2017-12-22
```

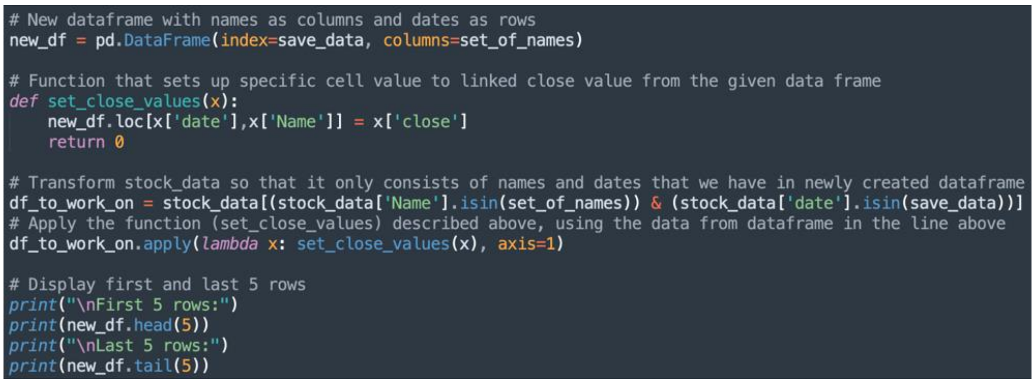
Question 5:

"Build a new pandas data frame which has a column for each of the names from step (3) and a row for each of the dates from step (4). The data frame should contain the "close" values for each corresponding name and date. Print the first 5 and last 5 rows of this data frame."

I included an alternative solution commented out in the code. It was too slow though.

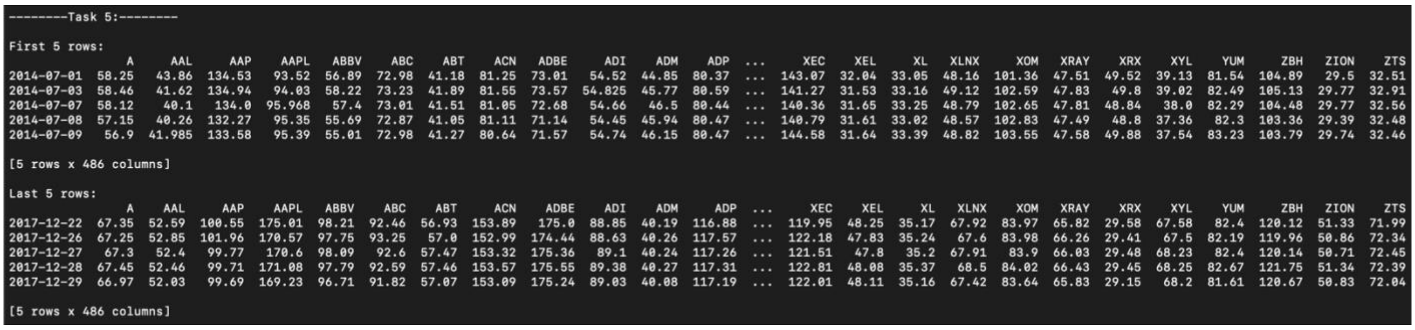
I build a new data frame. I create a function that will fill the values. I reduce the stock_data data frame so that it only contains the needed dates and names. I use the function.

Code:



```python
# New dataframe with names as columns and dates as rows
new_df = pd.DataFrame(index=save_data, columns=set_of_names)

# Function that sets up specific cell value to linked close value from the given data frame
def set_close_values(x):
    new_df.loc[x['date'],x['Name']] = x['close']
    return 0

# Transform stock_data so that it only consists of names and dates that we have in newly created dataframe
df_to_work_on = stock_data[(stock_data['Name'].isin(set_of_names)) & (stock_data['date'].isin(save_data))]
# Apply the function (set_close_values) described above, using the data from dataframe in the line above
df_to_work_on.apply(lambda x: set_close_values(x), axis=1)

# Display first and last 5 rows
print("\nFirst 5 rows:")
print(new_df.head(5))
print("\nLast 5 rows:")
print(new_df.tail(5))
```

Output:



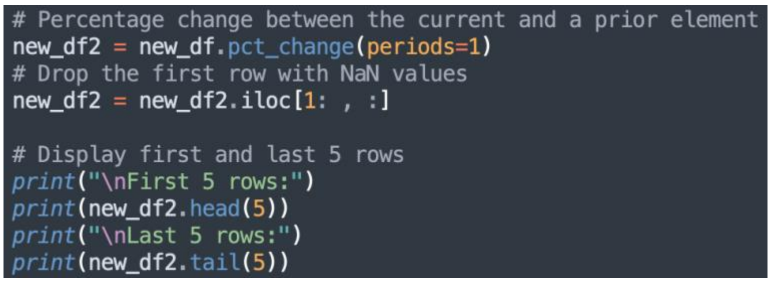
```
--------Task 5:--------

First 5 rows:
                A     AAL     AAP    AAPL   ABBV    ABC    ABT    ACN   ADBE     ADI    ADM    ADP  ...     XEC    XEL     XL   XLNX     XOM   XRAY    XRX    XYL    YUM     ZBH   ZION    ZTS
2014-07-01  58.25   43.86  134.53   93.52  56.89  72.98  41.18  81.25  73.01   54.52  44.85  80.37  ...  143.07  32.04  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  45.77  80.59  ...  141.27  31.53  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   46.5  80.44  ...  140.36  31.65  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  45.94  80.47  ...  140.79  31.61  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  46.15  80.47  ...  144.58  31.64  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 rows:
                A    AAL     AAP    AAPL   ABBV    ABC    ABT     ACN    ADBE    ADI    ADM     ADP  ...     XEC    XEL     XL   XLNX    XOM   XRAY    XRX    XYL    YUM     ZBH   ZION    ZTS
2017-12-22  67.35  52.59  100.55  175.01  98.21  92.46  56.93  153.89   175.0  88.85  40.19  116.88  ...  119.95  48.25  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  40.26  117.57  ...  122.18  47.83  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  40.24  117.26  ...  121.51   47.8   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  40.27  117.31  ...  122.81  48.08  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  40.08  117.19  ...  122.01  48.11  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]
```

Question 6:

"Create another data frame containing returns. Print the first 5 and last 5 rows."

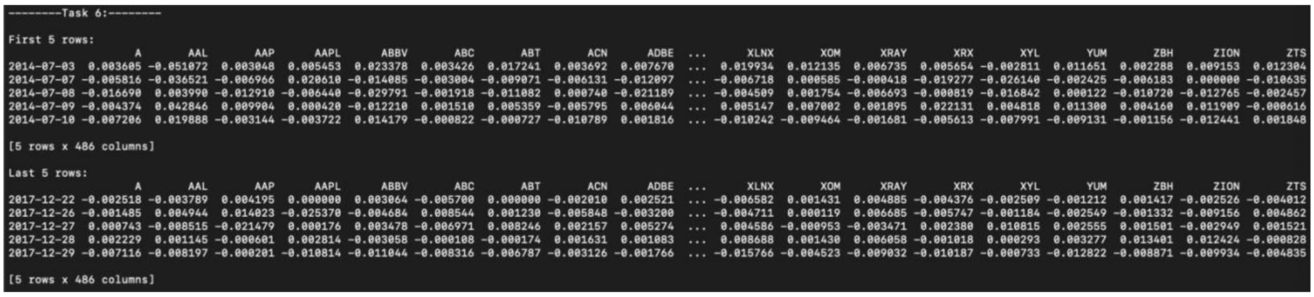
I use pct_change to calculate return values. I remove the first row that contains NaN values.

Code:



```
# Percentage change between the current and a prior element
new_df2 = new_df.pct_change(periods=1)
# Drop the first row with NaN values
new_df2 = new_df2.iloc[1: , :]

# Display first and last 5 rows
print("\nFirst 5 rows:")
print(new_df2.head(5))
print("\nLast 5 rows:")
print(new_df2.tail(5))
```

Output:



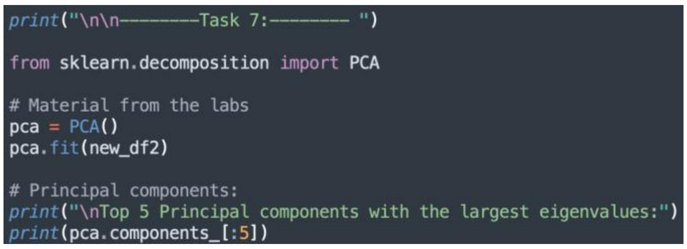
```
--------Task 6:--------

First 5 rows:
                   A       AAL       AAP      AAPL      ABBV       ABC       ABT       ACN      ADBE  ...      XLNX       XOM      XRAY       XRX       XYL       YUM       ZBH      ZION       ZTS
2014-07-03  0.003605 -0.051072  0.003048  0.005453  0.023378  0.003426  0.017241  0.003692  0.007670  ...  0.019934  0.012135  0.006735  0.005654 -0.002811  0.011651  0.002288  0.009153  0.012304
2014-07-07 -0.005816 -0.036521 -0.006966  0.020610 -0.014085 -0.003004 -0.009071 -0.006131 -0.012097  ... -0.006718  0.000585 -0.000418 -0.019277 -0.026140 -0.002425 -0.006183  0.000000 -0.010635
2014-07-08 -0.016690  0.003990 -0.012910 -0.006440 -0.029791 -0.001918 -0.011082  0.000740 -0.021189  ... -0.004509  0.001754 -0.006693 -0.000819 -0.016842  0.000122 -0.010720 -0.012765 -0.002457
2014-07-09 -0.004374  0.042846  0.009904  0.000420 -0.012210  0.001510  0.005359 -0.005795  0.006044  ...  0.005147  0.007002  0.001895  0.022131  0.004818  0.011300  0.004160  0.011909 -0.000616
2014-07-10 -0.007206  0.019888 -0.003144 -0.003722  0.014179 -0.000822 -0.000727 -0.010789  0.001816  ... -0.010242 -0.009464 -0.001681 -0.005613 -0.007991 -0.009131 -0.001156 -0.012441  0.001848

[5 rows x 486 columns]

Last 5 rows:
                   A       AAL       AAP      AAPL      ABBV       ABC       ABT       ACN      ADBE  ...      XLNX       XOM      XRAY       XRX       XYL       YUM       ZBH      ZION       ZTS
2017-12-22 -0.002518 -0.003789  0.004195  0.000000  0.003064 -0.005700  0.000000 -0.002010  0.002521  ... -0.006582  0.001431  0.004885 -0.004376 -0.002509 -0.001212  0.001417 -0.002526 -0.004012
2017-12-26 -0.001485  0.004944  0.014023 -0.025370 -0.004684  0.008544  0.001230 -0.005848 -0.003200  ... -0.004711  0.000119  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  0.002157  0.005274  ...  0.004586 -0.000953 -0.003471  0.002380  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  0.001631  0.001083  ...  0.008688  0.001430  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 -0.003126 -0.001766  ... -0.015766 -0.004523 -0.009032 -0.010187 -0.000733 -0.012822 -0.008871 -0.009934 -0.004835

[5 rows x 486 columns]
```

Question 7:

"Use the class sklearn.decomposition.PCA to calculate the principal components of the returns from step (6). Print the top five PCs with the largest eigenvalue."

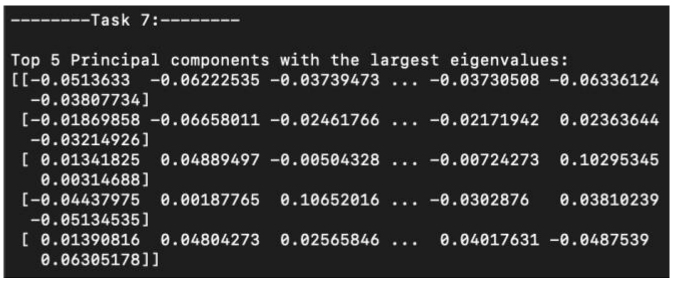
I assume that I didn't have to print out the eigenvalues. If I was meant to print the eigenvalues, then the code to do that is commented out further in the code.
The principal component with the largest eigenvalue is the first principle etc.

Code:



```
print("\n\n--------Task 7:-------- ")

from sklearn.decomposition import PCA

# Material from the labs
pca = PCA()
pca.fit(new_df2)

# Principal components:
print("\nTop 5 Principal components with the largest eigenvalues:")
print(pca.components_[:5])
```

Output:



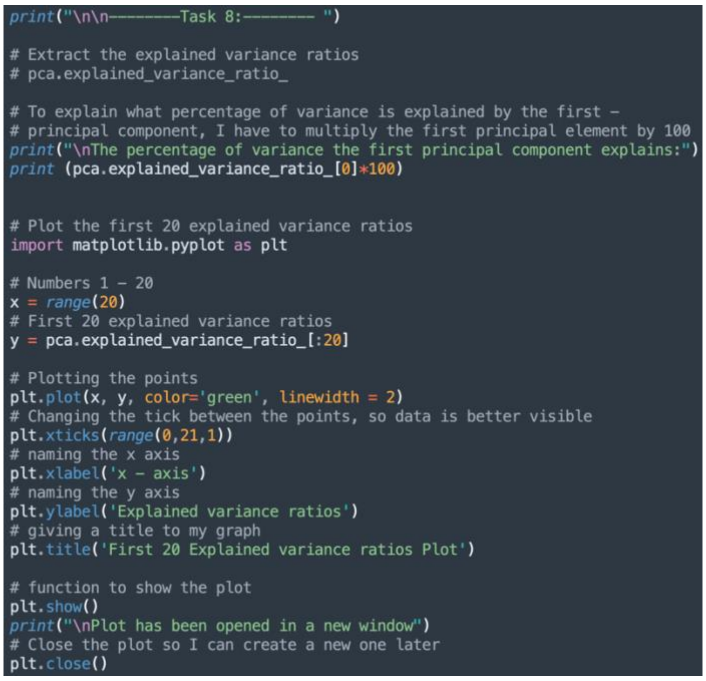
```
--------Task 7:--------

Top 5 Principal components with the largest eigenvalues:
[[-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]]
```

Question 8:

"For the principal components calculated in step (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."

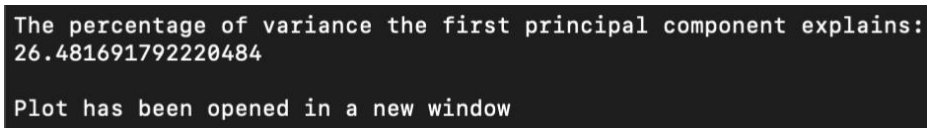
I multiply the explained variance ratio by 100 to get %. (1 = 100%, 0.9 = 90%)
I organise the data using basic operations. I close the plot to open a new one later.

Code:

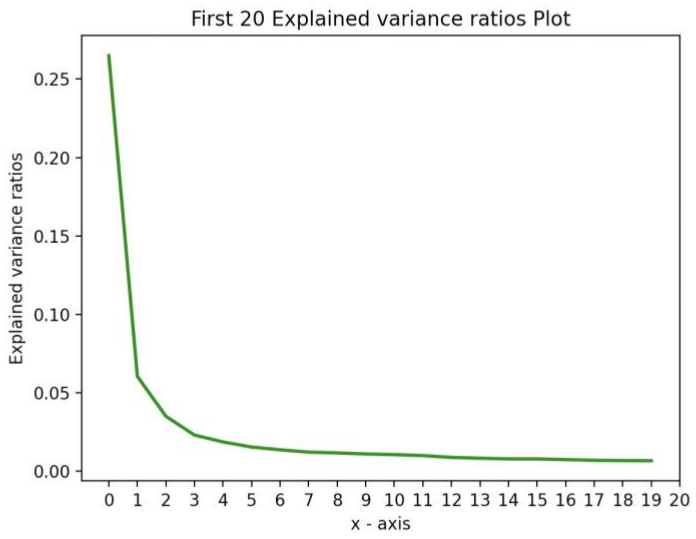


```
print("\n\n--------Task 8:-------- ")

# Extract the explained variance ratios
# pca.explained_variance_ratio_

# To explain what percentage of variance is explained by the first -
# principal component, I have to multiply the first principal element by 100
print("\nThe percentage of variance the first principal component explains:")
print (pca.explained_variance_ratio_[0]*100)


# Plot the first 20 explained variance ratios
import matplotlib.pyplot as plt

# Numbers 1 - 20
x = range(20)
# First 20 explained variance ratios
y = pca.explained_variance_ratio_[:20]

# Plotting the points
plt.plot(x, y, color='green', linewidth = 2)
# Changing the tick between the points, so data is better visible
plt.xticks(range(0,21,1))
# naming the x axis
plt.xlabel('x - axis')
# naming the y axis
plt.ylabel('Explained variance ratios')
# giving a title to my graph
plt.title('First 20 Explained variance ratios Plot')

# function to show the plot
plt.show()
print("\nPlot has been opened in a new window")
# Close the plot so I can create a new one later
plt.close()
```

Output:



```
The percentage of variance the first principal component explains:
26.481691792220484

Plot has been opened in a new window
```



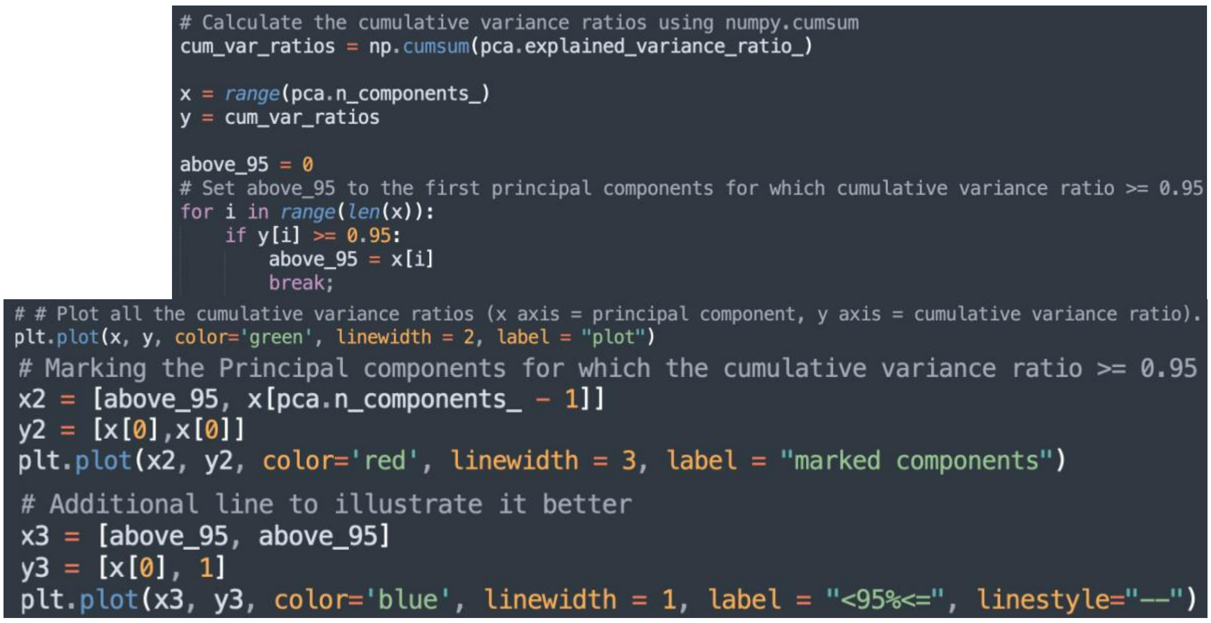
Question 9:

"Calculate the cumulative variance ratios on the list of explained variance ratios from step (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%.”

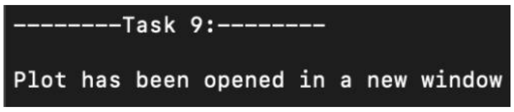
I find the first principal component for which cumulative variance ratio >= 95%. I perform basic operations creating the plot and marking mentioned points. There is a commented-out solution that marks all the points >=95% specifically.

Code:

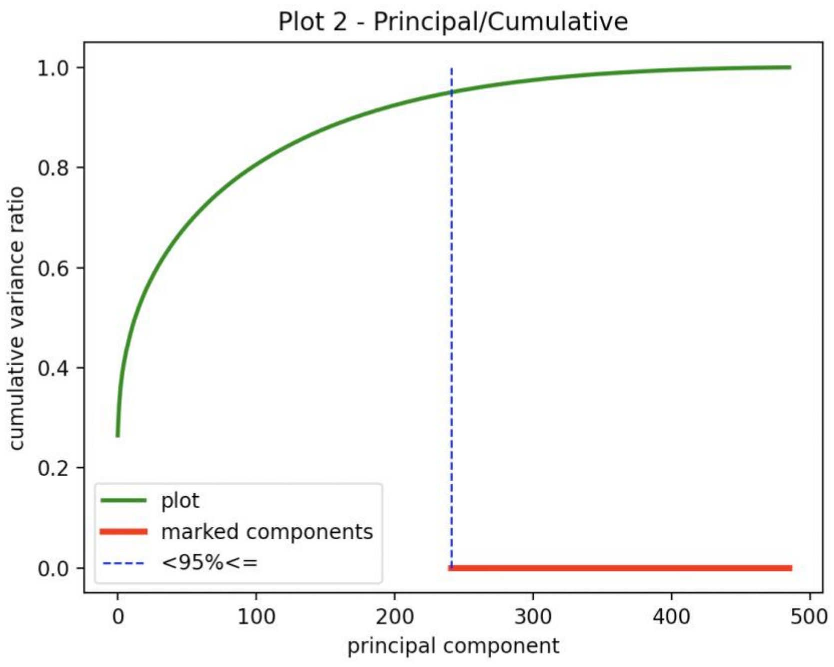


```
# Calculate the cumulative variance ratios using numpy.cumsum
cum_var_ratios = np.cumsum(pca.explained_variance_ratio_)

x = range(pca.n_components_)
y = cum_var_ratios

above_95 = 0
# Set above_95 to the first principal components for which cumulative variance ratio >= 0.95
for i in range(len(x)):
    if y[i] >= 0.95:
        above_95 = x[i]
        break;
# # Plot all the cumulative variance ratios (x axis = principal component, y axis = cumulative variance ratio).
plt.plot(x, y, color='green', linewidth = 2, label = "plot")
# Marking the Principal components for which the cumulative variance ratio >= 0.95
x2 = [above_95, x[pca.n_components_ - 1]]
y2 = [x[0],x[0]]
plt.plot(x2, y2, color='red', linewidth = 3, label = "marked components")
# Additional line to illustrate it better
x3 = [above_95, above_95]
y3 = [x[0], 1]
plt.plot(x3, y3, color='blue', linewidth = 1, label = "<95%<=", linestyle="--")
```

Output:



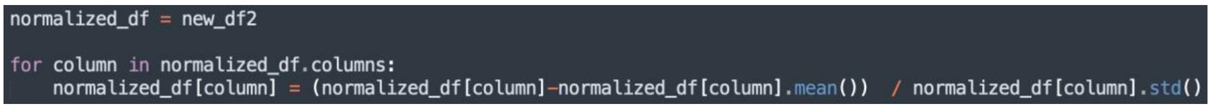
```
--------Task 9:--------

Plot has been opened in a new window
```



Question 10:

“Normalise your data frame from step **(6)** so that the columns have zero mean and unit variance. Repeat steps **(7)** - **(9)** for this new data frame.”

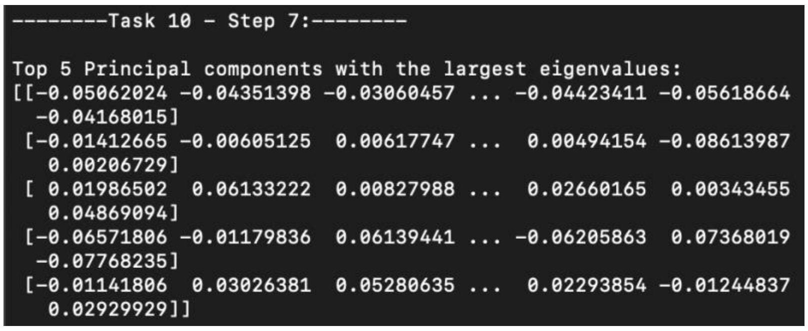
Code:



```
normalized_df = new_df2

for column in normalized_df.columns:
    normalized_df[column] = (normalized_df[column]-normalized_df[column].mean())  / normalized_df[column].std()
```

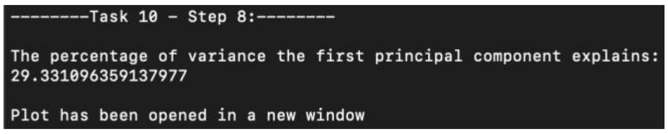
Everything I will do in the following steps is a copy of steps (7) – (9).
The only thing that changed is the line pca.fit(normalized_df)

Step 10.7 output:



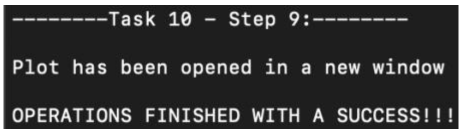
```
--------Task 10 - Step 7:--------

Top 5 Principal components with the largest eigenvalues:
[[-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]]
```

Step 10.8 output:

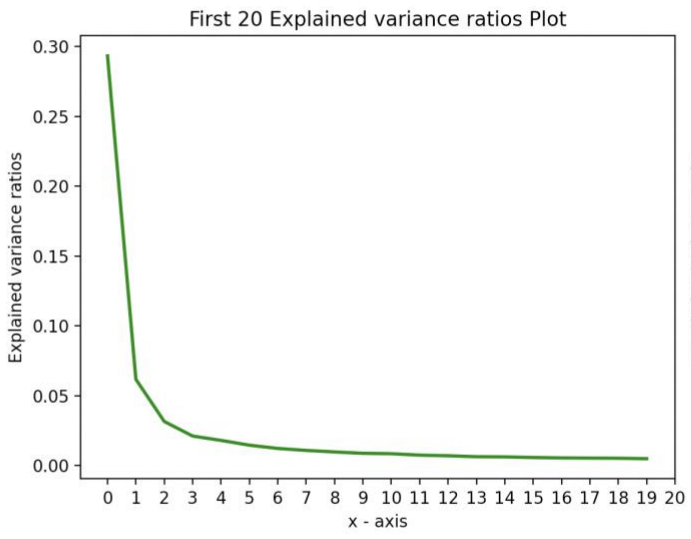


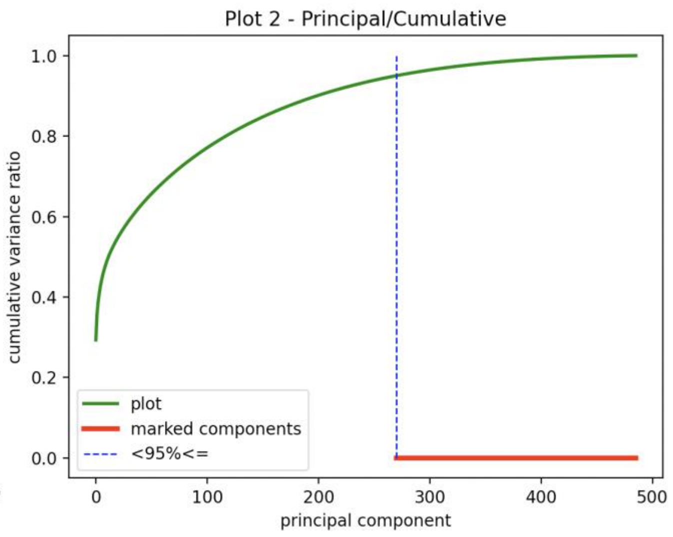
```
--------Task 10 - Step 8:--------

The percentage of variance the first principal component explains:
29.331096359137977

Plot has been opened in a new window
```

Step 10.9 output:



```
--------Task 10 - Step 9:--------

Plot has been opened in a new window

OPERATIONS FINISHED WITH A SUCCESS!!!
```


First 20 Explained variance ratios Plot
Explained variance ratios
x - axis


Plot 2 - Principal/Cumulative
cumulative variance ratio
principal component
plot
marked components
<95%<=