In [ ]:
```
#*****************************************************************************
#         Project 4 :Assignment on Project
#
#*****************************************************************************
# Loading the Data from the CSV file

# importing the packages and Loading the Data
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
% matplotlib inline

data = pd.read_csv("C:/Users/Maximus/pythonsessions/session31/data_stocks.csv")

print(data.head(2))
```

In [ ]:
```
# Analysing  the Stock Data , Checking for Null values , variation , missing values

print( " Analysing the Data and knowing about the data")
print('-'*80)

print(data.info())

print('-'*80)

print(data.describe())
print('-'*80)

print(data.shape)
print('-'*80)

print(data.isnull().sum())
print('-'*80)
print( " We observe there are no null values \n",'-'*80)
```

In [ ]:
```
# As there are many columns , to get appropriate features , we do the feature selection first
# and removing the unwanted Features

# Since Date and SP500 columns have high std as compared to others as seen above, we can remove the columns
# Date is in epoch format

data.drop(['DATE','SP500'],axis=1,inplace=True)

```

problem 1 : To Show all stocks are apparently similar in performance

Solution : So here we do Factor Analysis to find the similair stocks

In [ ]:
```
# Assumption: As the Data Size huge a Sampling of 1000 records is considered

# We use Factor analysis here to get similiar columns


from sklearn.decomposition import FactorAnalysis
from sklearn.model_selection import cross_val_score

# numFac number of Factors
n_components = np.arange(2,200)
fa_scores = []

# to find the number of factors
for numFac in n_components:
    # define the Constructor for Factor Analysis
    fa = FactorAnalysis(random_state=101)
    fa.n_components = numFac
    acc_score = np.mean(cross_val_score(fa,data.sample(1000), cv=5))
    fa_scores.append((numFac,acc_score))


```

In [ ]:
```
# splitting the tupples into separate values of components and scores

n_compon,fa_score = zip(*fa_scores)

bstscore = fa_score[np.argmax(fa_score)]
bst_component = n_compon[np.argmax(fa_score)]
print(" The best Value of the score for the components is {} at components  {}\n ".format(bstscore,bst_component))
print('-'*80)

```

In [ ]:
```
# Visualising the scores and component value by  plotting the graph for fa_Scores

# ncompon = n_compon[25:]
# fa_score1 = fa_scores[25:]

title  = " N_components Vs Cv Score "
plt.figure()
plt.plot(n_compon[25:], fa_score[25:], 'r', label='FA scores')
plt.axvline(bst_component, color='r',
            label='FactorAnalysis CV: %d' %bst_component,
            linestyle='--')
plt.xlabel('nb of components')
plt.ylabel('CV scores')
plt.legend(loc='lower right')
plt.title(title)

plt.show()

```

In [ ]:
```
# To find the factors at the obtained best components value achieved
# Approach 1


factor = FactorAnalysis(n_components=bst_component , random_state=101).fit(data.sample(1000))
#print(factor.components_)

print(''' Following is the DataFrame showing the relationship of columns Stock,
          Those with positive values are similiar \n''','-'*80)
fact_data = pd.DataFrame(factor.components_,columns=data.columns)
print(fact_data.head(5))

```

In [ ]:
```
# Another approach to show in each  Factor the stocks that are similiar
from factor_analyzer  import FactorAnalyzer
fa = FactorAnalyzer()

fa.analyze(data, bst_component, rotation=None)

print(" By FactorAnalyzer approach we can see each Factor comprises which stocks\n",'-'*80)
print(" Factor1 has AAL,AAPL,ADBE,ADI,ADPADSK... similiar, those having positive values\n",'-'*80)
print(fa.loadings)

```

Problem 2 : How many Unique patterns that exist in the historical stock data set, based on

fluctuations in price.

Solution 2: Here we do PCA to reduce the dimensions and then KMean Clustering to get Unique patterns in Data

In [ ]:
```
from sklearn.decomposition import PCA

for ncomponents in range(2,19):
    pca = PCA(n_components=ncomponents) # two components
    pca.fit(data) # run PCA, putting in raw version for fun
    variance = pca.explained_variance_ratio_ #calculate variance ratios
    var=np.cumsum(np.round(pca.explained_variance_ratio_, decimals=3)*100)
    print(var) #cumulative sum of variance explained with [n] features

print(ncomponents)
print('-'*80)
print("\n The component at which we get maximum variance about 99percent is : ", ncomponents)
```

In [ ]:
```
# now for the reduced number of componenets ,apply the clustering
pca= PCA(n_components=ncomponents)

pca.fit(data)

df1 = pca.transform(data)


print('-'*80,"\n The Stocks Data with the PCA done and reduced Dimensionality\n",'-'*80)
print(df1.shape)
df2 = pd.DataFrame(df1)
print(df2.head())

```

In [ ]:
```
# using KMeans clustering to know the number of clusters /Unique patterns
# Assumption : The Sample size of 10000  taken of data represents the complete Data

from sklearn.metrics import silhouette_score

from sklearn.cluster import KMeans

# Finding the Appropriate number of clusters , when the error is least
cluster_range = range( 2, 30 )
cluster_errors = []

X = df2.sample(10000)
for num_cluster in cluster_range:
    cluster = KMeans(n_clusters=num_cluster, random_state=100)
    cluster.fit(X)
    error = round(cluster.inertia_,2)
    cluster_errors.append(error)

clusters_df = pd.DataFrame( { "num_clusters":cluster_range, "cluster_errors": cluster_errors } )

```

In [ ]:
```
print(" Following is Elbow method for getting number of clusters with least errors \n",'-'*80)
plt.figure(figsize=(12,6))
plt.plot( clusters_df.num_clusters, clusters_df.cluster_errors, marker = "o" )
plt.ylabel('Custer errors')
plt.legend(loc='lower right')
plt.title('Elbow Method for number of Clusters Vs Error')

plt.show()
```

In [ ]:

```
# From the Graph of errors we observe that 6 clusters uniquely define the Data

print('-'*80)
print( ''' Based on the Fluctuation in the price, from the error graph we observe\n
          there are about 6 unique patterns in the Data  ''')
print('-'*80)
X = df2.sample(10000)
print(X.head(2))

cluster = KMeans(n_clusters=6, random_state=100)
cluster.fit_predict(X)

plt.scatter(X[0],X[1] ,label='True Position')
```

Problem 3:Identify which all stocks are moving together and which all stocks are different from each other.

In [ ]:

```
# Assumption : 1. Checking for first 10 stocks from the total stocks ,
#              same logic to be applied to all the 500 stocks
#              2. the threshold for checking if the stocks move together is considered 0.7


# getting the names of all the stocks from the Data Columns
import statsmodels.formula.api as sm

lst=[[]]
difflst =[[]]

# list of Stock Names
stock = []
stockNames = data.columns
for col in stockNames:
    stock.append(str.split(col,'.')[1])

# for ols apply ols in loop to find the slope and intercept

#outer loop for the Stock being compared
for i in range(15):
    smstcklst = []
    diffstcklst = []
    smstcklst.append(stock[i])
    diffstcklst.append(stock[i])
    for j in range(15):
        if i == j:
            continue
       # print(i,j)

        X = data[data.columns[i]]
        Y = data[data.columns[j]]
        lm = sm.ols(formula='X ~ Y',data=data)
        lm = lm.fit()
        #print(lm.params.Y)
        if (lm.params.Y > 0.7):
            smstcklst.append(stock[j])
        else:
            diffstcklst.append(stock[j])
    if(len(smstcklst) != 1):
        lst.append(smstcklst)
```

```
    if(len(diffstcklst) != 1):
        difflst.append(diffstcklst)

print(" The Stocks which move together are \n",'-'*80)
for i in range(1,len(lst)):
    print(lst[i])
print(" The stocks which are differnt from each other are \n",'-'*80)
for i in range(1,len(difflst)):
    print(difflst[i])

```