print ("\nAssignment 5 - 09/26/2019")

print("Name: Carolina Carvalho Manhaes Leite")

print("NetID: leite2")

print("\n------------\n")

# IMPORTING LIBRARIES #

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn import preprocessing

from sklearn import linear\_model

from sklearn.model\_selection import train\_test\_split

from sklearn.metrics import mean\_squared\_error

from sklearn.metrics import r2\_score

from sklearn.decomposition import PCA

from sklearn.svm import SVR

from time import process\_time

# SETTINGS #

plt.clf()

# IMPORTING THE DATA #

# Import csv file

df = pd.read\_csv('/Users/carolinacmleite/Documents/01 - Documents/04 - Academic/03 - Master/02 - Financial Engineering (UIUC)/01 - Fall 2019/IE598 - Machine Learning in Fin Lab/03 - Assignments/IE598\_F19\_HW5\_Data\_09292019.csv',header=0)

# Display its shape (#rows and #columns)

print('The original dataset has ' + str(df.shape[0]) + ' rows and ' + str(df.shape[1]) + ' columns.\n')

print("------------\n")

# DS of Target Variable

print ("Descriptive Statistics of Target Variable (Adj\_Close)\n")

for x in range(-1,0):

lastvar = df[df.columns[x]]

print (lastvar.describe())

print(" ")

#plt.boxplot(lastvar)

#plt.show()

#plt.clf()

#print("--\n")

print("------------\n")

# The count of the target is less than the total number of rows.

# It means we have some missing values in the target value. While we could choose an input method to infer these values but,

# to make it simple, I'll drop them.

df = df.dropna(subset=['Adj\_Close'])

print('The dataset without the target missing values has ' + str(df.shape[0]) + ' rows and ' + str(df.shape[1]) + ' columns.')

# Drop the 1st column

df = df.iloc[:, 1:]

print('The dataset without the first column has ' + str(df.shape[0]) + ' rows and ' + str(df.shape[1]) + ' columns.')

print("\n------------\n")

# Drop the target (Adj\_Close) from the features array

X = df.drop('Adj\_Close', axis=1).values

print('The features array has ' + str(X.shape[0]) + ' rows and ' + str(X.shape[1]) + ' columns.')

# Extract the target (Adj\_Close) and move it to its own array

y = df['Adj\_Close'].values

y = y.reshape(-1, 1)

print('The target array has ' + str(y.shape[0]) + ' rows.')

print("\n------------\n")

# DESCRIPTIVE STATISTICS #

# DS of Feature Variables

#print ("Descriptive Statistics of Original Variables\n")

#for x in range(0, 30):

#lastvar = df[df.columns[x]]

#print (lastvar.describe())

#print(" ")

#plt.boxplot(lastvar)

#plt.show()

#plt.clf()

#print("--\n")

#print("------------\n")

# CORRELATION MATRIX #

# Compute Pearson correlation

corr = df.corr()

# Create heat map with correlations calculated above

print ("Heat Map of Features + Target Correlation Matrix\n")

sns.heatmap(corr)

plt.show()

plt.clf()

print ("\nCorrelations with the target variable\n")

print(np.transpose(corr[30:]))

print("\n------------\n")

X\_axis\_corr = np.arange(1, 31)

y\_axis\_corr = -1\*np.transpose(corr[30:])

y\_axis\_corr = y\_axis\_corr[1:31]

axes = plt.gca()

axes.set\_ylim([0.5,1])

plt.scatter(X\_axis\_corr, y\_axis\_corr, alpha=0.5)

plt.title('Scatter plot: Correlation of features with target variable')

plt.xlabel('x')

plt.ylabel('y')

plt.show()

plt.clf()

print(" ")

X\_max\_corr = df['SVENF15'].values

X\_min\_corr = df['SVENF30'].values

plt.scatter(y, X\_max\_corr, alpha=0.5)

plt.title('Scatter plot: feature with maximum correlation x target')

plt.xlabel('Adj\_Close (Target)')

plt.ylabel('SVENF15')

plt.show()

plt.clf()

print(" ")

plt.scatter(y, X\_min\_corr, alpha=0.5)

plt.title('Scatter plot: feature with minimum correlation x target')

plt.xlabel('Adj\_Close (Target)')

plt.ylabel('SVENF30')

plt.show()

plt.clf()

print("\n------------\n")

# SPLITTING TRAIN AND TEST SAMPLES + STANDARDIZATION #

X\_train,X\_test,y\_train,y\_test = train\_test\_split(X, y, test\_size = 0.15, random\_state = 42)

sc\_X = preprocessing.StandardScaler().fit(X\_train)

sc\_y = preprocessing.StandardScaler().fit(y\_train)

X\_train = sc\_X.transform(X\_train)

X\_test = sc\_X.transform(X\_test)

y\_train = sc\_y.fit\_transform(y\_train)

y\_test = sc\_y.fit\_transform(y\_test)

# PCA #

# Setting the object with the PCA (with all principal components)

pca = PCA()

X\_train\_PCA = pca.fit\_transform(X\_train)

X\_test\_PCA = pca.transform(X\_test)

# Printing table and charts with the explained variance (individual and cumulative)

variance\_pca\_full = np.array(pca.explained\_variance\_ratio\_)

print("The explained variance for each of the components/features is (in order):\n")

print(variance\_pca\_full)

print("\nGraphically:\n")

y\_full = np.arange(len(variance\_pca\_full))

plt.bar(y\_full, variance\_pca\_full)

plt.xlabel('Number of Components')

plt.ylabel('Explained Variance')

plt.show()

print(" ")

print("The cumulative explained variance is (in order):\n")

print(np.cumsum(pca.explained\_variance\_ratio\_))

print("\nGraphically:\n")

plt.plot(np.cumsum(pca.explained\_variance\_ratio\_))

plt.xlabel('Number of Components')

plt.ylabel('Cumulative Explained Variance')

plt.show()

print("\n------------\n")

# Setting the object with the PCA (with 3 components)

pca3 = PCA(n\_components=3)

X\_train\_PCA3 = pca3.fit\_transform(X\_train)

X\_test\_PCA3 = pca3.transform(X\_test)

# Printing table and charts with the explained variance (individual and cumulative)

variance\_pca\_3 = np.array(pca3.explained\_variance\_ratio\_)

print("The cumulative explained variance for those 3 components is (in order):\n")

print(np.cumsum(pca3.explained\_variance\_ratio\_))

sum\_variance = np.cumsum(pca3.explained\_variance\_ratio\_)

print("\nIt means that, together, they explain %.3f of all variance." % sum\_variance[2])

print("\n------------\n")

# 1st MODEL: LINEAR REGRESSION WITH ALL ATTRIBUTES #

print ("1st Model: LINEAR REGRESSION with all attributes")

# Setting the object with the model we're going to use

linreg\_all = linear\_model.SGDRegressor(loss = 'squared\_loss', penalty=None, random\_state=42)

# Setting the start time

start = process\_time()

# Fitting the model

linreg\_all.fit(X\_train, y\_train.ravel())

# Setting the end time

end = process\_time()

# Calculating processing time

proc\_time = end - start

# Using the model to predict values of y for the training and test sets

y\_train\_pred\_all = linreg\_all.predict(X\_train)

y\_test\_pred\_all = linreg\_all.predict(X\_test)

# Metrics: mean squared error

print ("Metrics:\n")

print('MSE train: %.3f\nMSE test: %.3f' % (mean\_squared\_error(y\_train,y\_train\_pred\_all),mean\_squared\_error(y\_test,y\_test\_pred\_all)))

# Metrics: R2

print('R2 train: %.3f\nR2 test: %.3f' % (r2\_score(y\_train, y\_train\_pred\_all),r2\_score(y\_test, y\_test\_pred\_all)))

# Processing time

print ('\nThe processing time was: %.3f' % proc\_time)

print("\n------------\n")

# 2nd MODEL: LINEAR REGRESSION WITH 3 PRINCIPAL COMPONENTS #

print ("2nd Model: LINEAR REGRESSION with 3 principal components")

# Setting the object with the model we're going to use

linreg\_3 = linear\_model.SGDRegressor(loss = 'squared\_loss', penalty=None, random\_state=42)

# Setting the start time

start = process\_time()

# Fitting the model

linreg\_3.fit(X\_train\_PCA3, y\_train.ravel())

# Setting the end time

end = process\_time()

# Calculating processing time

proc\_time = end - start

# Using the model to predict values of y for the training and test sets

y\_train\_pred\_3 = linreg\_3.predict(X\_train\_PCA3)

y\_test\_pred\_3 = linreg\_3.predict(X\_test\_PCA3)

# Metrics: mean squared error

print ("Metrics:\n")

print('MSE train: %.3f\nMSE test: %.3f' % (mean\_squared\_error(y\_train,y\_train\_pred\_3),mean\_squared\_error(y\_test,y\_test\_pred\_3)))

# Metrics: R2

print('R2 train: %.3f\nR2 test: %.3f' % (r2\_score(y\_train, y\_train\_pred\_3),r2\_score(y\_test, y\_test\_pred\_3)))

# Processing time

print ('\nThe processing time was: %.3f' % proc\_time)

print("\n------------\n")

# 3rd MODEL: SVR WITH WITH ALL ATTRIBUTES #

print ("3rd Model: SVR with all attributes")

# Setting the object with the model we're going to use

svm\_all = SVR(kernel = 'linear', gamma='scale', C=1.0)

# Setting the start time

start = process\_time()

# Fitting the model

svm\_all.fit(X\_train, y\_train.ravel())

# Setting the end time

end = process\_time()

# Calculating processing time

proc\_time = end - start

# Using the model to predict values of y for the training and test sets

y\_train\_pred\_all = svm\_all.predict(X\_train)

y\_test\_pred\_all = svm\_all.predict(X\_test)

# Metrics: mean squared error

print ("Metrics:\n")

print('MSE train: %.3f\nMSE test: %.3f' % (mean\_squared\_error(y\_train,y\_train\_pred\_all),mean\_squared\_error(y\_test,y\_test\_pred\_all)))

# Metrics: R2

print('R2 train: %.3f\nR2 test: %.3f' % (r2\_score(y\_train, y\_train\_pred\_all),r2\_score(y\_test, y\_test\_pred\_all)))

# Processing time

print ('\nThe processing time was: %.3f' % proc\_time)

print("\n------------\n")

# 4th MODEL: SVR WITH WITH 3 PRINCIPAL COMPONENTS #

print ("4th Model: SVR with 3 principal components")

# Setting the object with the model we're going to use

svm\_3 = SVR(kernel = 'linear', gamma='scale', C=1.0)

# Setting the start time

start = process\_time()

# Fitting the model

svm\_3.fit(X\_train\_PCA3, y\_train.ravel())

# Setting the end time

end = process\_time()

# Calculating processing time

proc\_time = end - start

# Using the model to predict values of y for the training and test sets

y\_train\_pred\_3 = svm\_3.predict(X\_train\_PCA3)

y\_test\_pred\_3 = svm\_3.predict(X\_test\_PCA3)

# Metrics: mean squared error

print ("Metrics:\n")

print('MSE train: %.3f\nMSE test: %.3f' % (mean\_squared\_error(y\_train,y\_train\_pred\_3),mean\_squared\_error(y\_test,y\_test\_pred\_3)))

# Metrics: R2

print('R2 train: %.3f\nR2 test: %.3f' % (r2\_score(y\_train, y\_train\_pred\_3),r2\_score(y\_test, y\_test\_pred\_3)))

# Processing time

print ('\nThe processing time was: %.3f' % proc\_time)

print("\n------------\n")

print("My name is Carolina Carvalho Manhaes Leite")

print("My NetID is: leite2")

print("I hereby certify that I have read the University policy on Academic Integrity and that I am not in violation.")