**Using Machine Learning to Predict Stock Returns in the Energy Sector**

**Group Members: Aaron Zelmanov, Gloria Stach, Carlo De los Santos, Charles Yu, Maurits Giese**

**Executive Summary**

In the era of big data, machine learning and artificial intelligence provide ideal avenues through which to predict stock markets prices and trends. The energy sector continues to be an industry of particular interest, because of its interconnectedness with current events and far-reaching effects on the global economy due to rapid and frequent commodity price fluctuations. Subsequently, utilizing historical data to predict future stock returns in the energy sector may be useful in foreseeing industry trends.

We conducted comprehensive evaluations on frequently used machine learning models (linear regression, lasso regression, gradient boosting regression, random forests, neural networks) and concluded that our proposed portfolio of 20 stocks outperforms an out-of-sample benchmark. In our evaluation of these predictive models, four different datasets were used: the previously used signals dataset from class, a signals dataset with smoothed return data without NA values, a signals dataset with smoothed return data with NA values, and a signals dataset with currency volatility data. For all models, we employed rolling and expanding windows, tested many valid windows, and determined that an expanding window produced the best results.

| **Company (Ticker)** | **Projected Return** |
| --- | --- |
| PPL Corp. (PPL) | 5.2% |
| Adams Resources & Energy Inc. (AE) | 0.4% |
| Matrix Service Co. (MTRX) | 5.5% |
| NOV Inc. (NOV) | 4.2% |
| Helix Energy Solutions Group Inc. (HLX) | 3.2% |
| Southern Company (SO) | 3.9% |
| Entergy Corp. (ETR) | 7.5% |
| Pinnacle West Capital Corp. (PNW) | 1.6% |
| PG&E Corp. (PCG) | 5.4% |
| Avista Corp. (AVA) | 2% |
| NiSource Inc. (NI) | 10.1% |
| Spire Inc. (SR) | 8.4% |
| Atmos Energy Corp. (ATO) | 6.4% |
| Northwest Natural Holding Co. (NWN) | 6.1% |
| Southwest Gas Holdings Inc. (SWX) | 8.9% |
| Southwestern Energy Co. (SWN) | 14.4% |
| Coterra Energy Inc. (CTRA) | 8.8% |
| Williams Co. (WMB) | 8.5% |
| SJW Group (SJW) | 7.7% |
| Marathon Oil Corp. (MRO) | 14.6% |

**Data Preparation**

In addition to the data we were provided in the class, we added exchange rate data, relative to the strength of the U.S. Dollar, from five countries: Canada, Russia, Saudi Arabia, Iraq, and the United Arab Emirates. During our preliminary research, we discovered that political instability in oil-producing nations plays a critical role in stock price movements. The previously listed countries were chosen because they constitute the countries that produce the most oil annually apart from the United States.

We also decided to remove outliers from the stock returns. Not doing so would mean inputting noisy data into our models which would make it harder for the models to learn a certain trend on our data. We passed the returns of each stock through a Tukey smoother function and used these returns in all our models. Overall, we saw better performance on our validation and test sets.

Finally, before training our models, we made sure to remove any identifier columns, dates, returns, and market value lag as these might feed unnatural data and/or grant bias to our models.

**Methodology**

Model evaluation is one of the most vital parts of data science. It allows us to understand the accuracy and performance of the model and makes it simple and easy to present to other people. There are three main metrics used for model evaluation: Adjusted R-squared, Mean Square Error (RME)/Root Mean Square Error (RMSE), and Mean Absolute Error (MAE). Each statistic has its own benefits and limitations, but all give an approximation of how close a prediction is to the real value.

R-Squared measures how much variability can be explained in the model. It is also calculated by taking the sum of the square of prediction error and dividing it by the total sum of the square. The R-Squared value ranges from 0 to 1, and larger values indicate a better fit between the prediction and actual value. However, it does not take into consideration the overfitting problem, which is why Adjusted R-Squared is introduced. Adjusted R-Squared penalizes additional independent variables added to the model and adjusts to prevent these overfitting issues.[[1]](#footnote-0)

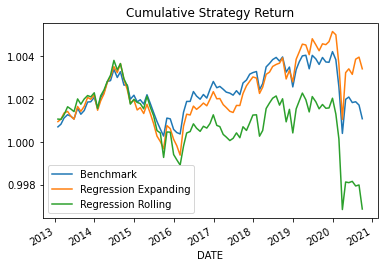
Mean Square Error (MSE)/Root Mean Square Error (RMSE) is an absolute measure of the fit, contrasting how R-Squared is a relative measure of the fit. RMSE is calculated by taking the square root of MSE, which is calculated by the sum of the square of prediction error and then divided by the number of data points.

Mean Absolute Error (MAE) is similar to MSE, but instead of the sum of squares, MAE takes the sum of the absolute value of error. An important consideration between the two metrics is that MSE penalizes prediction errors more while MAE treats all of them the same.[[2]](#footnote-1)

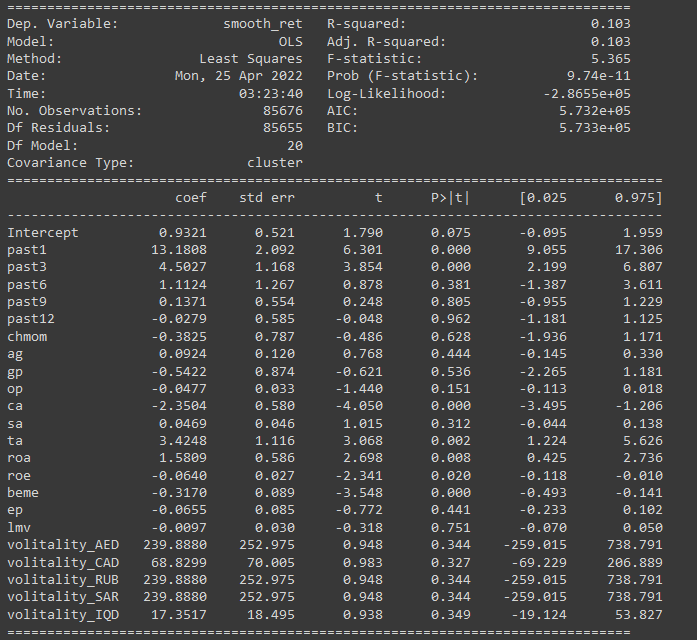
We considered all three metrics and concluded that Adjusted R-Squared, specifically out-of-sample Adjusted R-Squared would produce the best results. Adjusted R-Squared is the only metric that considers the overfitting problem, while the other statistics are generally better suited to compare performance between different models. Subsequently, Adjusted R-Squared would provide better insights, especially when our dataset contains more than 500 different stocks.

1. Linear Regression

The linear regression model was the simplest model in our analysis of predicted stock returns in the energy sector. A linear regression “attempts to model the relationship between two variables by fitting a linear equation to observed data.” One variable, stock returns, in our case, is considered the explanatory variable, and other variables are considered the dependent variables, such as return on equity, return on assets, the book to market value, etc.[[3]](#footnote-2) The main objective of linear regression is to understand the relationship between the variables of interest.

Tuning parameters for the linear regression model involved choosing the appropriate training, validation, and testing sets, and whether to use a rolling or expanding window. We determined that the linear regression produced the highest out-of-sample validation R-Squared with a training set before 2016, a validation set from 2017 to 2019, and a test set of 2020. The expanding window with the large training set was suitable for this application because of the inclusion of currency volatility data. In order to capture the total variation in this data, which accounts for movements in global politics, a larger training set was necessary.

The model summary (below) shows that the most statistically significant predictors are ‘past1’, ‘past3’, ‘ca’, ‘ta’, and ‘beme’. The variables ‘ca’ and ‘beme’ seem to be negatively correlated with returns while the others are positively correlated. Contrary to our initial assumptions, none of the currency volatilities appear to be significant in the linear regression model.



1. Lasso Penalized Regression

In many situations, the imported data has a large number of variables that are superior to the number of samples, which may lead to overfitting. An alternative approach to resolve this issue

is Lasso Penalized Regression. It allows for the creation of a linear regression model that is penalized for having too many variables by adding a constraint in the equation. Lasso Penalized Regression is also used to select relevant features by setting coefficients to zero. It is an extension of linear regression in the manner that a regularization parameter multiplied by the summation of the absolute value of weights gets added to the loss function (ordinary least squares) of linear regression.

The main objective of Lasso Regression is to shrink the coefficients to avoid overfitting and find the optimal variables that best predict stock returns in the energy sector. The best approach to predict the stock returns is maximizing R-squared in the out-of-sample datasets. Out-of-sample testing refers to using “new” data which is not found in the dataset used to build the model.

In terms of tuning hyperparameters, the portfolio dataset is split into training, validation, and testing sets. For the training set, we subset the dataset that is prior to 2016. The validation set is a subset from 2017 to 2019. The test set is in the year 2020. The goal of using the validation set is to evaluate the model from the training model. It helps us to tune parameters and find the optimal model before confirming results with the test set. An important financial metric is called alpha, which measures the returns on investment. After trying different values for initialized alpha and the objective, we found that the best R-squared value is approximately 0.3%. This result implies that the model fails to explain the variations of energy stock returns around its mean.

After finding the R-squared value, we plot the benchmark return value with the Lasso Regression return value, which shows that the benchmark return outperformed lasso regression returns. This again proves that Lasso Regression is not the best to predict the best stock returns.



1. Random Forests

Essentially, a random forest works similar to a tree regression except instead of using the result of one tree it aggregates the result of multiple trees. Tree models in general are used as opposed to linear regression when the data appears to be non-linear. Essentially, in the tree, every node is an if-else statement that determines the final value predicted. Every node gets split dependent on different variable criteria, such as if book to market ratio is greater than 0.4, go left, else go right. This keeps going for the depth of the tree and after following the path a prediction is given. A forest works by taking the results of multiple trees generated with limited pools of random predictors with small tree depth and finding the average prediction from all of them. To create the next best path in the tree, the algorithm will take the if statement that adds the most information gain.

For random forests, we looked at the best information gain being which would give the highest validation r squared. To tune the hyperparameters we looked at different minimum tree depths (how far the branches of the tree can go), n\_estimators(the number of trees to be used in the forest), and finally max features. Originally, we experimented with values of n\_estimators greater than 100. However, upon some testing, there were few improvements in the model and it did not get much better after 100, even though computationally it took significantly longer. The depths we looked at included 1,2,3 as we believe any deeper and the tree would be over complex as there is a bias-variance tradeoff with models. We also looked at only 3 or 5 features to be careful with not overfitting the model.

1. Gradient Boosting Regression

Similar to Random Forest, Gradient Boosting Regression generates a randomly generated number of trees. However, each subsequent tree is built off of the residuals from the tree before it. As a regressor, the gradient boosting algorithm can be used for predicting the continuous target variable, and the cost function is Mean Square Error (MSE). All of the hyperparameters that were used are the same as the ones used in the random forest, as the models are similar with the only difference being that new trees are built off of the residuals, not randomness.

1. Neural Networks

Neural networks are the ultimate “black box” of machine learning algorithms. Similar to their inspiration, human brains, we do not know exactly what is happening under the hood of complex neural networks. Essentially, neural networks have a forward run, in which a loss is calculated, and then this loss is tuned through backpropagation, where weights are changed in the direction that is expected to reduce the loss. This occurs through several nodes and neural network layers, hence the difficulty to understand and interpret results. Running a neural network often means sacrificing nearly all interpretability for the sake of better prediction, which they can absolutely excel at.

Fortunately, stock market price prediction does not require interpreting the reasoning behind the model, hence neural networks are worth trying out. Unfortunately, for time series data, recurrent neural networks would be the best-known choice to use over simple dense neural networks, however, limited knowledge of implementing RNNs and LSTMs, forces us to delve into the art of constructing a network architecture and tuning hyperparameters using only dense layers.

After attempting multiple initial architectures, we found the model drastically overfitting at 5 dense layers and slightly underfitting at 3 layers or less, hence we begin our hyperparameter tuning at 4 dense layers with sigmoid activation functions. The number of neurons per hidden layer we start with is a standard 16, 32, and 64 respectively. When we noticed that the model was overfitting (as the training r-squared was significantly larger than the validation r-squared), we added dropouts of 0.3 and batch normalization after each hidden layer. Afterward, the model seemed to be slightly underfitting this time, where we increased the neurons per hidden layer to 32, 64, and 128 respectively. At this point, we also decided to try ReLU activation functions and add a learning rate decay function, with an initial learning rate of 0.02 and a decay rate of 0.7. Both of these implementations led to our largest validation r-squared of around 0.7%, which is not the largest our other machine learning models have provided. Only 0.7% of the variability of smoothed returns is explained by the variables given.

Overall, the usage of only dense neural networks is limited to predicting stock prices in the energy sector. Tuning hyperparameters hardly increases the validation r-squared, thus they have limited influence on the predictive power of dense neural networks in this context. We believe that recurrent neural networks, which have some sense of time series memory in the construction, would have a better chance of predicting. Given the correct variables, this machine learning method has the potential to beat our benchmark, however, this would come at the sacrifice of understanding which of these variables are most significant and which are mostly irrelevant to the energy market and stock prices in general.

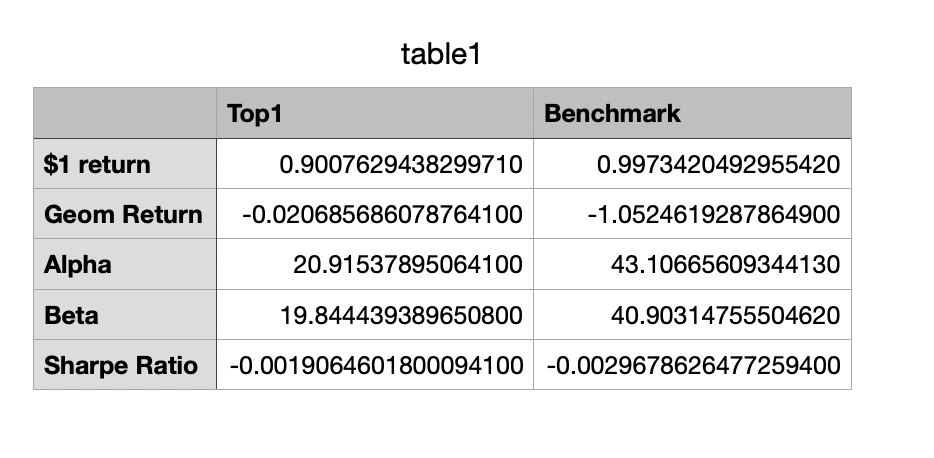
**Results**

**Best Hyperparameters for Random Forest:** {'max\_depth': 3, 'max\_features': 5, 'n\_estimators': 100, 'random\_state': 42}

**Best Test-sample R-squared:** -0.09516261999191356

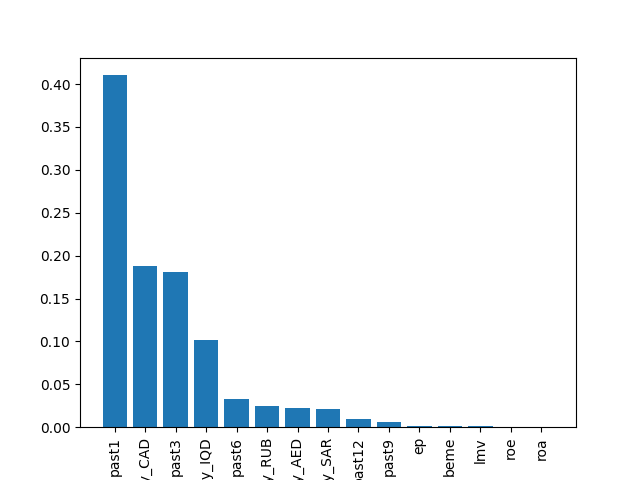
**Best Validation-sample R-squared:** 0.055864250766311074

The test r squared is much worse than the validation r squared. This may be due to overfitting the validation hyperparameter and the training data. This also could be because our test set was 2020 which was an odd year due to covid and all stocks plummeted. In retrospect, we should have not included 2020 in our analysis to get better returns. 2020 shows using our model that negative returns are being predicted.

****Compared to the benchmark, even the best portfolio predicted from this algorithm performs worse in every regard. The returns are expected to be negative as you can expect a 90-cent return on a dollar.

Random Forest Best Predictors:

The random forest importance plot shows that the most important variables out of our data set for the prediction of returns in the energy sector are recent past returns and, unlike our linear regression results, currency volatilities for the five largest oil exporters against the US dollar. The plot shows that excluding the ‘past1’ variable from the model would result in around a 40% loss in its accuracy.



Gradient Boosting Regression Best Predictors:

Similar to the importance plot of the random forest model, the top 10 most important variables are the same. Once again, ‘past1’ appears to be the most important variable: if it is excluded, then the model will lose around 25% of its accuracy. This reinforces the correlation between recent past data and major oil exporters’ currencies’ volatilities with the stock returns of the energy sector.



**Predictions**

The companies that we recommend be held onto if one must hold onto one are the ones that appeared in decile 0 most frequently for all dates of the portfolios created by the random forest model. The ticker symbols of these companies include ('PPL', 'AE', 'MTRX', 'NOV', 'HLX', 'SO', 'ETR, 'PNW', 'PCG', 'AVA', 'NI', 'SR', 'ATO', 'NWN', 'SWX', 'SWN', 'CTRA', 'WMB', 'SJW', 'MRO'). The overall return of investing in these stocks is 0.9 compared to a benchmark of 0.99. However, this is a worse performance than the benchmark, so even our best ML algorithm still gave a worse performance than the benchmark. This also may be due to the time series graph we looked at for returns vs time for the data showing a decreasing return in this sector for the past few years which makes it hard to predict using a machine learning algorithm if the stocks are dipping. Additionally, since we are attempting to predict returns on an industry largely affected by the political climate, there is not a sufficient number of political events that occurred during our given training time frame.

**Code Appendix**

Tukey Smoothing (R):

data <- read.csv("joined\_signals\_currency.csv")

tic\_data = list()

for(i in 1:length(unique(data$tic))){

df = data.frame(random = NA)

tic\_data[[i]] = df }

tickers <- data$tic

for (tic in 1:length(unique(data$tic))) {

tic\_data[[tic]] <- data[grep(tickers[tic], data$tic),]

smoothed <- smooth(x = tic\_data[[tic]]$RET)

tic\_data[[tic]]$RET <- smoothed }

library(dplyr)

tic\_data <- bind\_rows(tic\_data, .id = "column\_label")

summary(tic\_data)

Linear Regression:

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import statsmodels.formula.api as sm

import io

import warnings

from sklearn.linear\_model import LinearRegression

warnings.filterwarnings("ignore")

iyear = 2010

window = 6

num\_groups = 10

valid\_window = 3

gridpoints = 100

def wavg(group, avg\_name, weight\_name):

d = group[avg\_name]

w = group[weight\_name]

try:

return (d \* w).sum()/ w.sum()

except ZeroDivisionError:

return np.nan

filepath = '/content/signals1.csv'

df = pd.read\_csv('smoothed\_data\_wNA.csv', parse\_dates=['DATE'])

df = df.dropna()

df['smooth\_ret'] = df['smooth\_ret']\*100

df.drop(columns=['RET'])

df['year'] = pd.DatetimeIndex(df['DATE']).year

column\_name\_list = df.columns.tolist()

column\_name\_list.remove('mvlag')

column\_name\_list.remove('PERMNO')

column\_name\_list.remove('DATE')

column\_name\_list.remove('RET')

column\_name\_list.remove('smooth\_ret')

column\_name\_list.remove('tic')

column\_name\_list.remove('Date\_x')

column\_name\_list.remove('Open\_x')

column\_name\_list.remove('High\_x')

column\_name\_list.remove('Low\_x')

column\_name\_list.remove('Close\_x')

column\_name\_list.remove('Adj.Close\_x')

column\_name\_list.remove('Volume\_x')

column\_name\_list.remove('Date\_y')

column\_name\_list.remove('Open\_y')

column\_name\_list.remove('High\_y')

column\_name\_list.remove('Low\_y')

column\_name\_list.remove('Close\_y')

column\_name\_list.remove('Adj.Close\_y')

column\_name\_list.remove('Volume\_y')

column\_name\_list.remove('Date\_x.1')

column\_name\_list.remove('Open\_x.1')

column\_name\_list.remove('High\_x.1')

column\_name\_list.remove('Low\_x.1')

column\_name\_list.remove('Close\_x.1')

column\_name\_list.remove('Adj.Close\_x.1')

column\_name\_list.remove('Volume\_x.1')

column\_name\_list.remove('Date\_y.1')

column\_name\_list.remove('Open\_y.1')

column\_name\_list.remove('High\_y.1')

column\_name\_list.remove('Low\_y.1')

column\_name\_list.remove('Close\_y.1')

column\_name\_list.remove('Adj.Close\_y.1')

column\_name\_list.remove('Volume\_y.1')

column\_name\_list.remove('Date')

column\_name\_list.remove('Open')

column\_name\_list.remove('High')

column\_name\_list.remove('Low')

column\_name\_list.remove('Close')

column\_name\_list.remove('Adj.Close')

column\_name\_list.remove('Volume')

print(column\_name\_list)

out\_of\_sample = df[df['year'] >= iyear + valid\_window]

window = 0

pred\_reg\_expanding = []

#Expanding window regression

for i in range(iyear,2021-valid\_window):

if window > 0:

X\_train = df[(df['year']<i) & (df['year']>=(i-window))][column\_name\_list]

y\_train = df[(df['year']<i) & (df['year']>=(i-window))]['smooth\_ret'].values

else:

X\_train = df[df['year']<i][column\_name\_list]

y\_train = df[df['year']<i]['smooth\_ret'].values

X\_valid = df[(df['year']>=i) & (df['year']<i+valid\_window)][column\_name\_list]

y\_valid = df[(df['year']>=i) & (df['year']<i+valid\_window)]['smooth\_ret']

X\_test = df[df['year']==i+valid\_window][column\_name\_list]

y\_test = df[df['year']==i+valid\_window]['smooth\_ret']

reg = LinearRegression()

reg.fit(X\_train, y\_train)

print(reg.score(X\_test, y\_test))

y\_pred\_reg = reg.predict(X\_test)

pred\_reg\_expanding.extend(y\_pred\_reg)

window = 3

pred\_reg\_roll = []

for i in range(iyear,2021-valid\_window):

if window > 0:

X\_train = df[(df['year']<i) & (df['year']>=(i-window))][column\_name\_list]

y\_train = df[(df['year']<i) & (df['year']>=(i-window))]['smooth\_ret'].values

else:

X\_train = df[df['year']<i][column\_name\_list]

y\_train = df[df['year']<i]['smooth\_ret'].values

X\_valid = df[(df['year']>=i) & (df['year']<i+valid\_window)][column\_name\_list]

y\_valid = df[(df['year']>=i) & (df['year']<i+valid\_window)]['smooth\_ret']

X\_test = df[df['year']==i+valid\_window][column\_name\_list]

y\_test = df[df['year']==i+valid\_window]['smooth\_ret']

reg = LinearRegression()

reg.fit(X\_train,y\_train)

print(reg.score(X\_valid, y\_valid))

y\_pred\_reg = reg.predict(X\_test)

pred\_reg\_roll.extend(y\_pred\_reg)

out\_of\_sample['pred\_reg\_expand']=np.reshape(np.array(pred\_reg\_expanding),(-1,1))

out\_of\_sample['pred\_reg\_roll']=np.reshape(np.array(pred\_reg\_roll),(-1,1))

out\_of\_sample['ERDecile1']=out\_of\_sample.groupby(['DATE'])['pred\_reg\_expand'].\

transform(lambda x: pd.qcut(x,num\_groups, labels=False, duplicates='drop'))

out\_of\_sample['ERDecile1']=np.abs(out\_of\_sample['ERDecile1']-(num\_groups-1))

out\_of\_sample['ERDecile2']=out\_of\_sample.groupby(['DATE'])['pred\_reg\_roll'].\

transform(lambda x: pd.qcut(x,num\_groups, labels=False, duplicates='drop'))

out\_of\_sample['ERDecile2']=np.abs(out\_of\_sample['ERDecile2']-(num\_groups-1))

benchmark = out\_of\_sample.groupby(['DATE'],as\_index=False).apply(wavg,'RET','mvlag')

benchmark = benchmark.rename(columns={None:'benchmark'})

erret1 = out\_of\_sample.groupby(['DATE','ERDecile1'],as\_index=False).apply(wavg,

'RET','mvlag')

erret1 = erret1.rename(columns={None:'erret1','ERDecile1':'Decile'})

erret2 = out\_of\_sample.groupby(['DATE','ERDecile2'],as\_index=False).apply(wavg,

'RET','mvlag')

erret2 = erret2.rename(columns={None:'erret2','ERDecile2':'Decile'})

port\_ret = benchmark.merge(erret1, how='inner', on='DATE')

port\_ret = port\_ret.merge(erret2, how='inner', on=['DATE','Decile'])

port\_ret['rb\_gross']=1+port\_ret['benchmark']/100

port\_ret['r1\_gross']=1+port\_ret['erret1']/100

port\_ret['r2\_gross']=1+port\_ret['erret2']/100

port\_ret=port\_ret.set\_index('DATE')

port\_ret['rcumb'] = port\_ret.groupby(['Decile'])['rb\_gross'].transform('cumprod')

port\_ret['rcum1'] = port\_ret.groupby(['Decile'])['r1\_gross'].transform('cumprod')

port\_ret['rcum2'] = port\_ret.groupby(['Decile'])['r2\_gross'].transform('cumprod')

port\_ret = port\_ret.sort\_values(['DATE','Decile'])

port\_ret[port\_ret['Decile']==0]['rcumb'].plot(label='Benchmark')

port\_ret[port\_ret['Decile']==0]['rcum1'].plot(label='Regression Expanding')

port\_ret[port\_ret['Decile']==0]['rcum2'].plot(label='Regression Rolling')

plt.title('Cumulative Strategy Return')

plt.legend()

plt.show()

winner = port\_ret[port\_ret['Decile']==0]

rf = pd.read\_csv('rf.csv',parse\_dates=['DATE'])

winner = winner.merge(rf,how='inner',on='DATE')

winner['rfp1']=1+winner['rf']

winner['cumrf']=winner['rfp1'].transform('cumprod')

newreg1 = sm.ols('erret1~benchmark',data=winner).fit()

beta1 = newreg1.params[1]

newreg2 = sm.ols('erret2~benchmark',data=winner).fit()

beta2 = newreg2.params[1]

geom\_avg1 = winner['rcum1'][len(winner)-1]\*\*(1/(2021-iyear))-1

geom\_avg2 = winner['rcum2'][len(winner)-1]\*\*(1/(2021-iyear))-1

geom\_avgb = winner['rcumb'][len(winner)-1]\*\*(1/(2021-iyear))-1

geom\_avgf = winner['cumrf'][len(winner)-1]\*\*(1/(2021-iyear))-1

alpha1 = geom\_avg1 - geom\_avgf - beta1\*(geom\_avgb-geom\_avgf)

alpha2 = geom\_avg2 - geom\_avgf - beta2\*(geom\_avgb-geom\_avgf)

print("Strategy Compounded Return: {}".format([geom\_avg1,geom\_avg2]))

print("Benchmark Compounded Return: {}".format(geom\_avgb))

print("Alpha: {}".format([alpha1,alpha2]))

print("R-squared: {}".format([newreg1.rsquared\_adj, newreg2.rsquared\_adj]))

drawdown1 = []

drawdown3 = []

drawdown12 = []

for i in range(12,len(winner)):

retm = winner['cumrf'][i]/winner['cumrf'][i-1]-1

retq = winner['cumrf'][i]/winner['cumrf'][i-3]-1

reta = winner['cumrf'][i]/winner['cumrf'][i-12]-1

drawdown1 = np.append(drawdown1,retm)

drawdown3 = np.append(drawdown3,retq)

drawdown12 = np.append(drawdown12, reta)

print("Maximum 1-Month Drawdown: {}".format(min(drawdown1)))

print("Maximum 3-Month Drawdown: {}".format(min(drawdown3)))

print("Maximum 12-Month Drawdown: {}".format(min(drawdown12)))

column\_name\_list.remove('column\_label')

column\_name\_list.remove('X')

column\_name\_list.remove('group')

column\_name\_list.remove('year')

ols\_formula = 'smooth\_ret~' + "+".join(column\_name\_list)

cluster\_date\_ols = sm.ols(formula = ols\_formula,

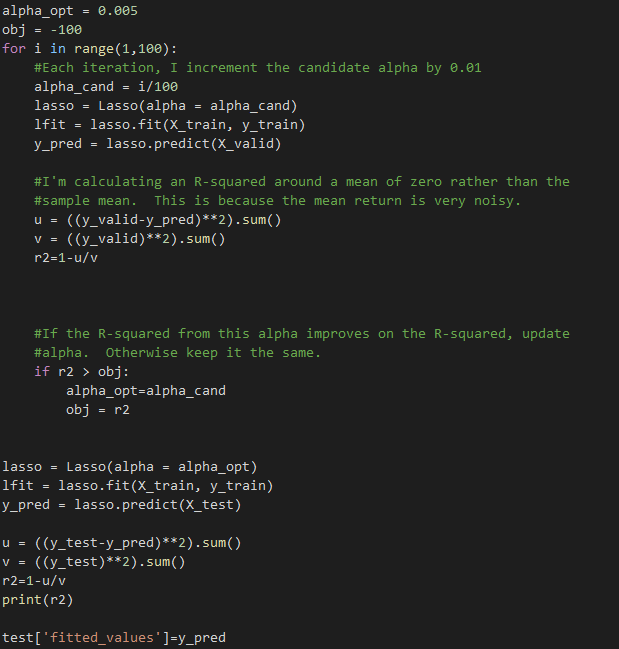
data=df).fit(cov\_type='cluster',

cov\_kwds={'groups': df['DATE']},

use\_t=True)

print(cluster\_date\_ols.summary())

LASSO Penalized Regression:



alpha\_opt = 0.00005

obj = -1

for j in range(1,100):

alpha\_cand = j/100

lasso = Lasso(alpha = alpha\_cand)

lfit = lasso.fit(X\_train, y\_train)

y\_pred = lasso.predict(X\_valid)

u = ((y\_valid-y\_pred)\*\*2).sum()

v = ((y\_valid)\*\*2).sum()

r2 = 1-u/v

if r2 > obj:

alpha\_opt = alpha\_cand

obj = r2

lasso = Lasso(alpha=alpha\_opt)

lfit = lasso.fit(X\_train, y\_train)

y\_pred = lasso.predict(X\_test)

u = ((y\_test-y\_pred)\*\*2).sum()

v = ((y\_test)\*\*2).sum()

r2=1-u/v

print(r2)

pred = df[df['year']==i+valid\_window]

pred['fitted\_values']=lasso.predict(X\_test)

test=test.append(pred)

pred\_reg\_expanding.extend(pred)

Neural Network:

import pandas as pd

import matplotlib.pyplot as plt

import numpy as np

import tensorflow as tf

from tensorflow import keras

from tensorflow.keras import layers

from keras.models import Sequential

from keras.layers import Dense

from keras.callbacks import EarlyStopping

from keras.callbacks import ModelCheckpoint

from keras.models import load\_model

from sklearn.metrics import r2\_score

df = pd.read\_csv('smoothed\_data\_wNA.csv', parse\_dates=['DATE'])

print(df)

df = df.dropna()

df['smooth\_ret'] = df['smooth\_ret']\*100

df['year'] = pd.DatetimeIndex(df['DATE']).year

column\_name\_list = df.columns.tolist()

column\_name\_list.remove('mvlag')

column\_name\_list.remove('PERMNO')

column\_name\_list.remove('DATE')

column\_name\_list.remove('RET')

column\_name\_list.remove('smooth\_ret')

column\_name\_list.remove('tic')

column\_name\_list.remove('Date\_x')

column\_name\_list.remove('Open\_x')

column\_name\_list.remove('High\_x')

column\_name\_list.remove('Low\_x')

column\_name\_list.remove('Close\_x')

column\_name\_list.remove('Adj.Close\_x')

column\_name\_list.remove('Volume\_x')

column\_name\_list.remove('Date\_y')

column\_name\_list.remove('Open\_y')

column\_name\_list.remove('High\_y')

column\_name\_list.remove('Low\_y')

column\_name\_list.remove('Close\_y')

column\_name\_list.remove('Adj.Close\_y')

column\_name\_list.remove('Volume\_y')

column\_name\_list.remove('Date\_x.1')

column\_name\_list.remove('Open\_x.1')

column\_name\_list.remove('High\_x.1')

column\_name\_list.remove('Low\_x.1')

column\_name\_list.remove('Close\_x.1')

column\_name\_list.remove('Adj.Close\_x.1')

column\_name\_list.remove('Volume\_x.1')

column\_name\_list.remove('Date\_y.1')

column\_name\_list.remove('Open\_y.1')

column\_name\_list.remove('High\_y.1')

column\_name\_list.remove('Low\_y.1')

column\_name\_list.remove('Close\_y.1')

column\_name\_list.remove('Adj.Close\_y.1')

column\_name\_list.remove('Volume\_y.1')

column\_name\_list.remove('Date')

column\_name\_list.remove('Open')

column\_name\_list.remove('High')

column\_name\_list.remove('Low')

column\_name\_list.remove('Close')

column\_name\_list.remove('Adj.Close')

column\_name\_list.remove('Volume')

X\_train = df[df['year']<2016][column\_name\_list]

y\_train = df[df['year']<2016]['smooth\_ret']

X\_valid = df[(df['year']>=2016) & (df['year']<2020)][column\_name\_list]

y\_valid = df[(df['year']>=2016) & (df['year']<2020)]['smooth\_ret']

X\_test = df[df['year']==2020][column\_name\_list]

y\_test = df[df['year']==2020]['smooth\_ret']

from numpy.random import seed

seed(1)

from tensorflow.random import set\_seed

set\_seed(2)

nnet = Sequential()

nnet.add(Dense(32, input\_dim = X\_train.shape[1],activation='relu'))

nnet.add(layers.Dropout(0.3))

nnet.add(layers.BatchNormalization())

nnet.add(Dense(64, input\_dim = X\_train.shape[1],activation='relu'))

nnet.add(layers.Dropout(0.3))

nnet.add(layers.BatchNormalization())

nnet.add(Dense(128, input\_dim = X\_train.shape[1],activation='relu'))

nnet.add(layers.Dropout(0.3))

nnet.add(layers.BatchNormalization())

nnet.add(layers.Dense(1, activation='linear'))

initial\_learning\_rate = 0.02

lr\_schedule = tf.keras.optimizers.schedules.ExponentialDecay(

initial\_learning\_rate,

decay\_steps=100000,

decay\_rate=0.70,

staircase=True)

opt = keras.optimizers.Adam(learning\_rate= lr\_schedule)

nnet.compile(optimizer=opt, loss='mse')

stop = EarlyStopping(monitor='val\_loss', patience = 10, verbose=1)

mc = ModelCheckpoint('best\_model.h5', monitor='val\_loss', mode='min',

save\_best\_only=True, verbose=1)

history = nnet.fit(X\_train, y\_train, validation\_data = (X\_valid, y\_valid),

epochs = 100, batch\_size=1000, callbacks = [stop, mc])

best\_model=load\_model('best\_model.h5')

train\_preds = best\_model.predict(X\_train)

valid\_preds = best\_model.predict(X\_valid)

test\_preds = best\_model.predict(X\_test)

print(r2\_score(y\_train, train\_preds))

print(r2\_score(y\_valid, valid\_preds))

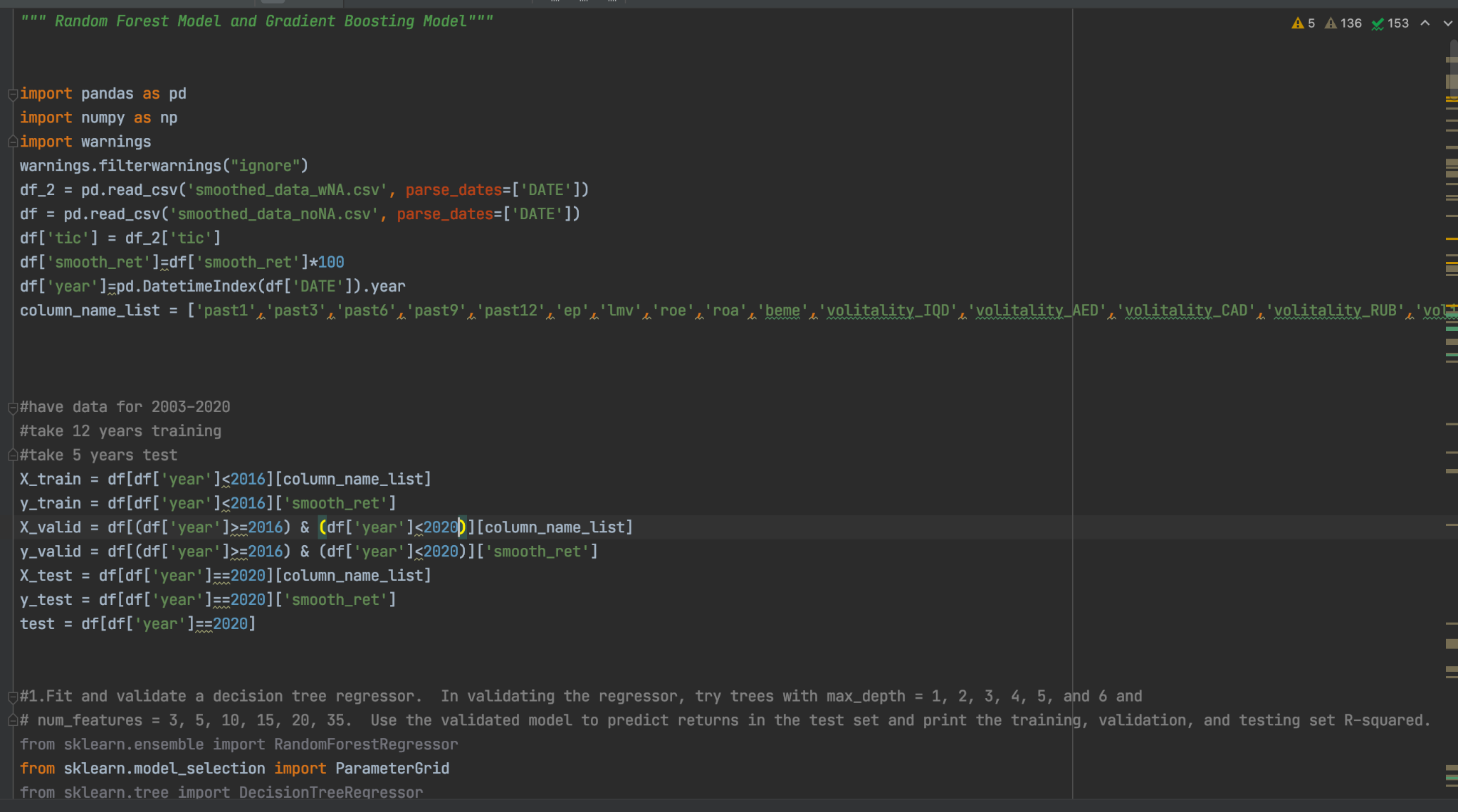
print(r2\_score(y\_test, test\_preds))

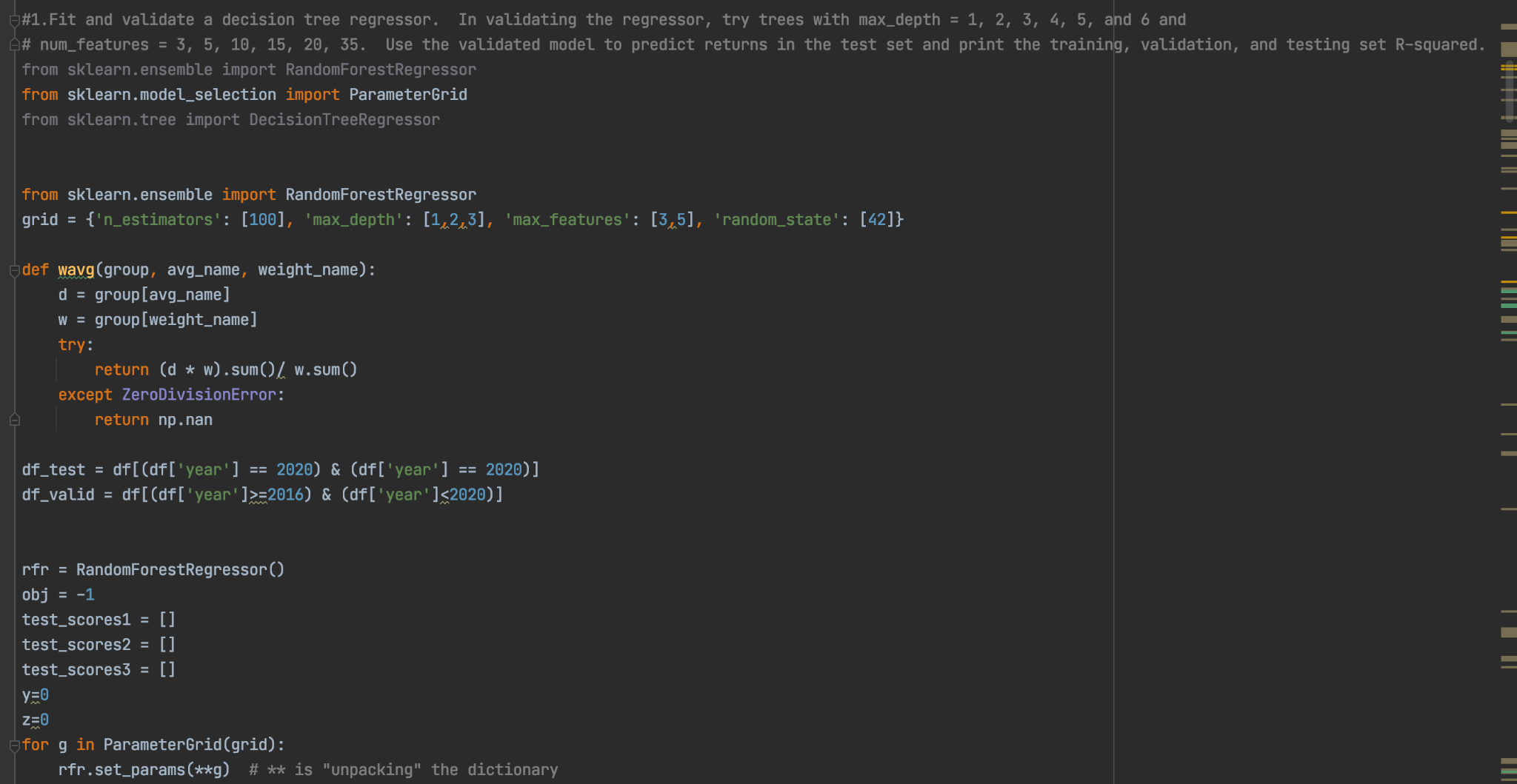
plt.plot(history.history['loss'])

plt.plot(history.history['val\_loss'])

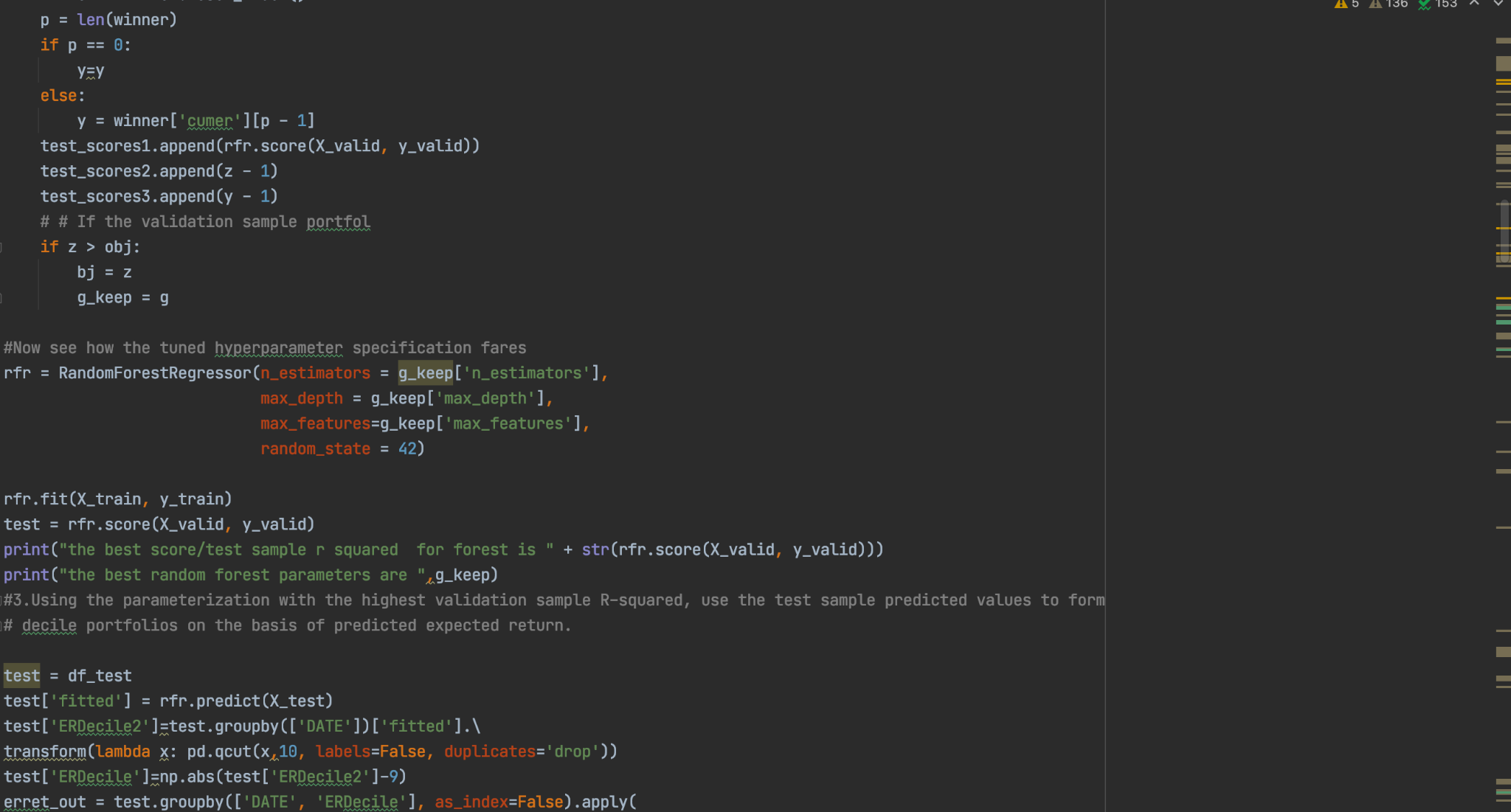
plt.show()

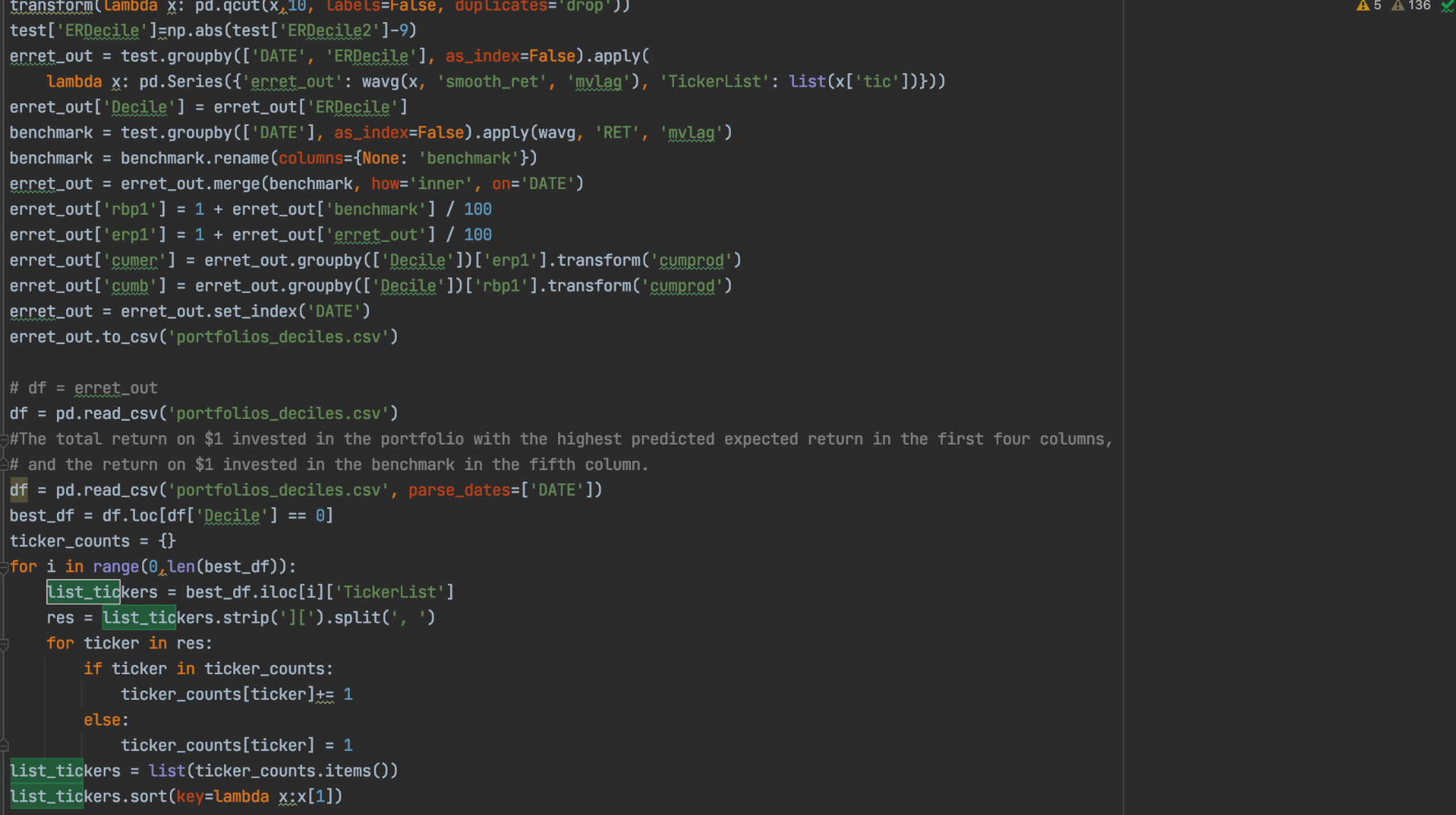
Random Forest and Gradient Boosting Code

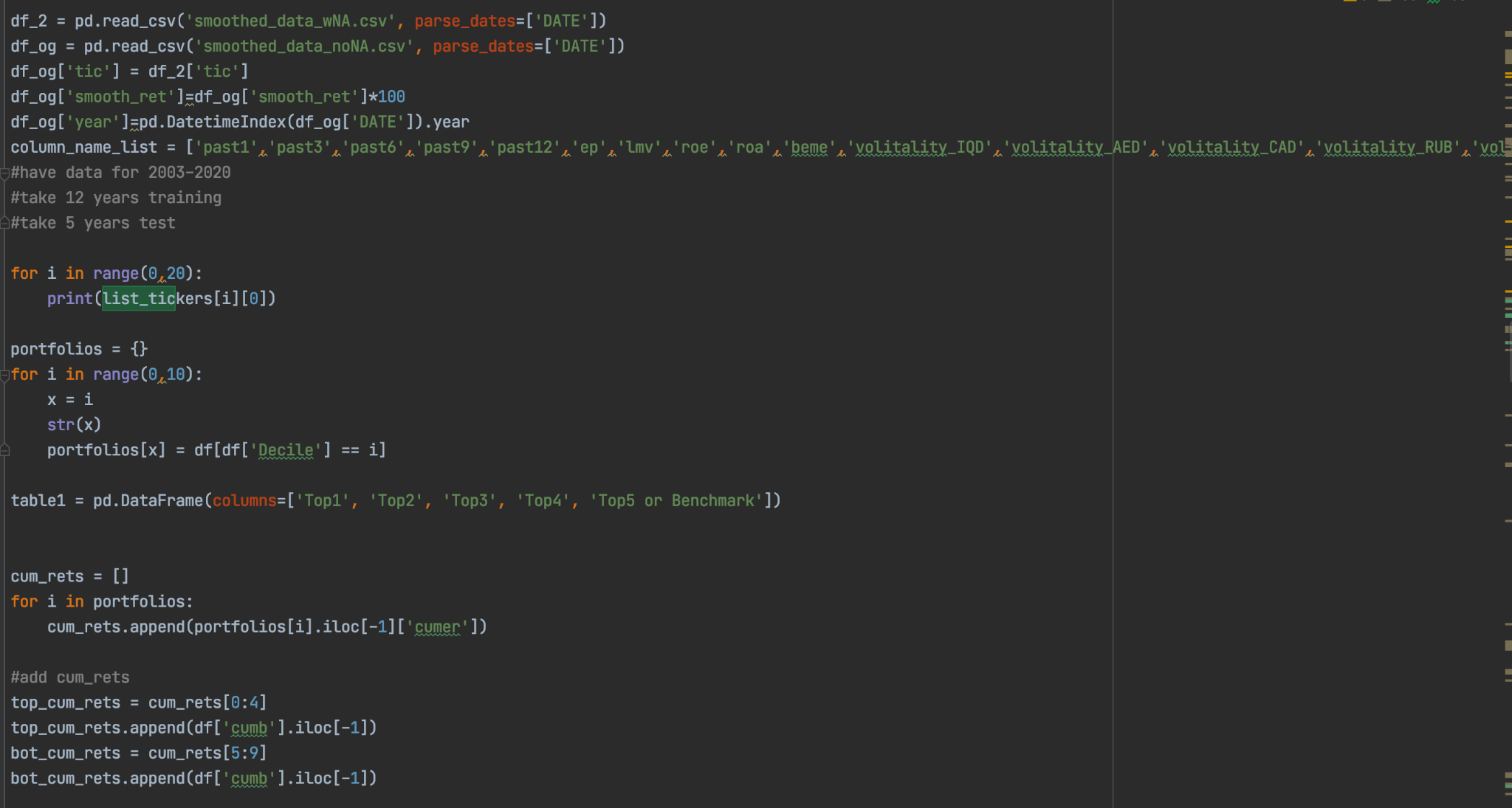


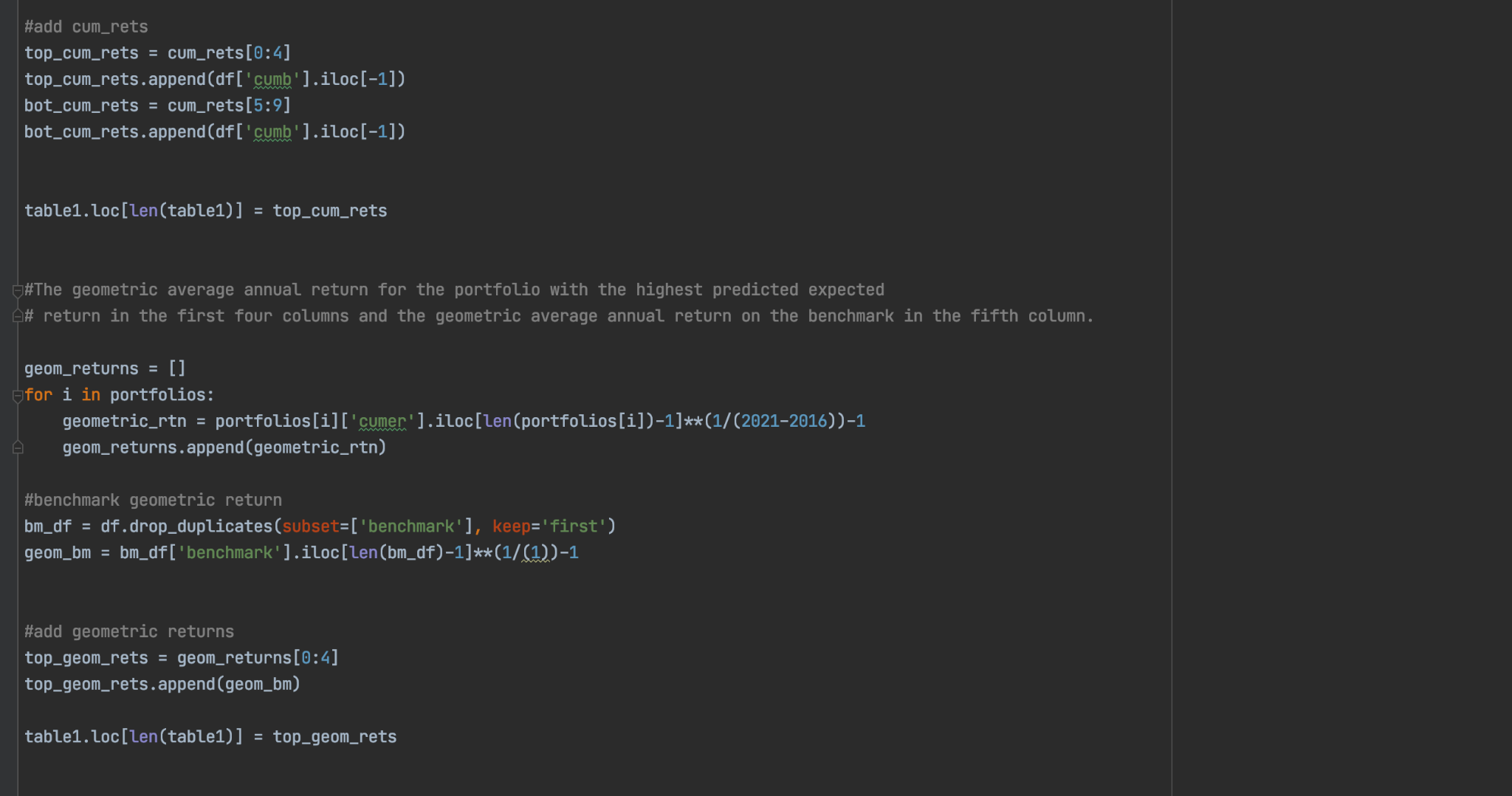


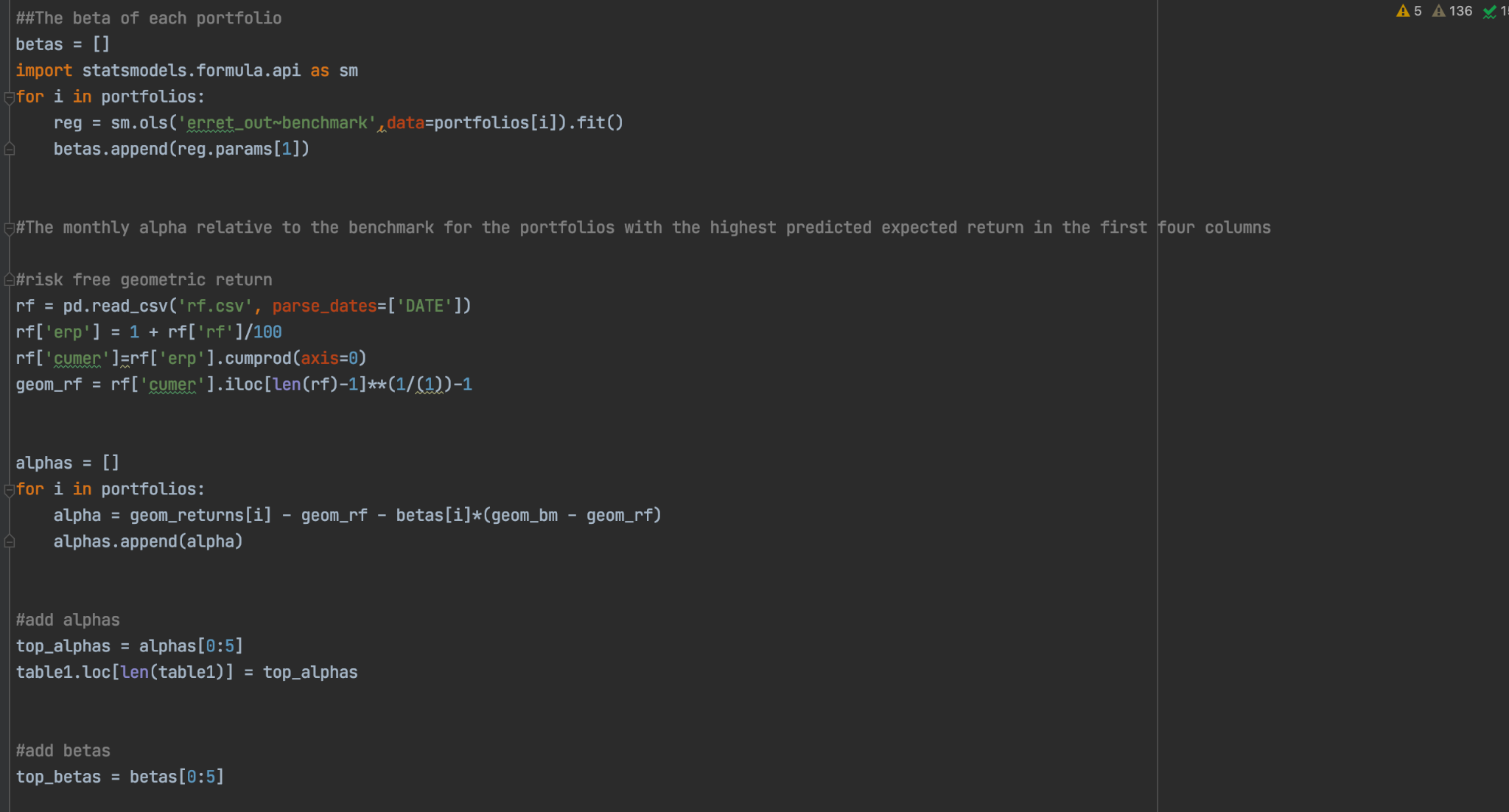




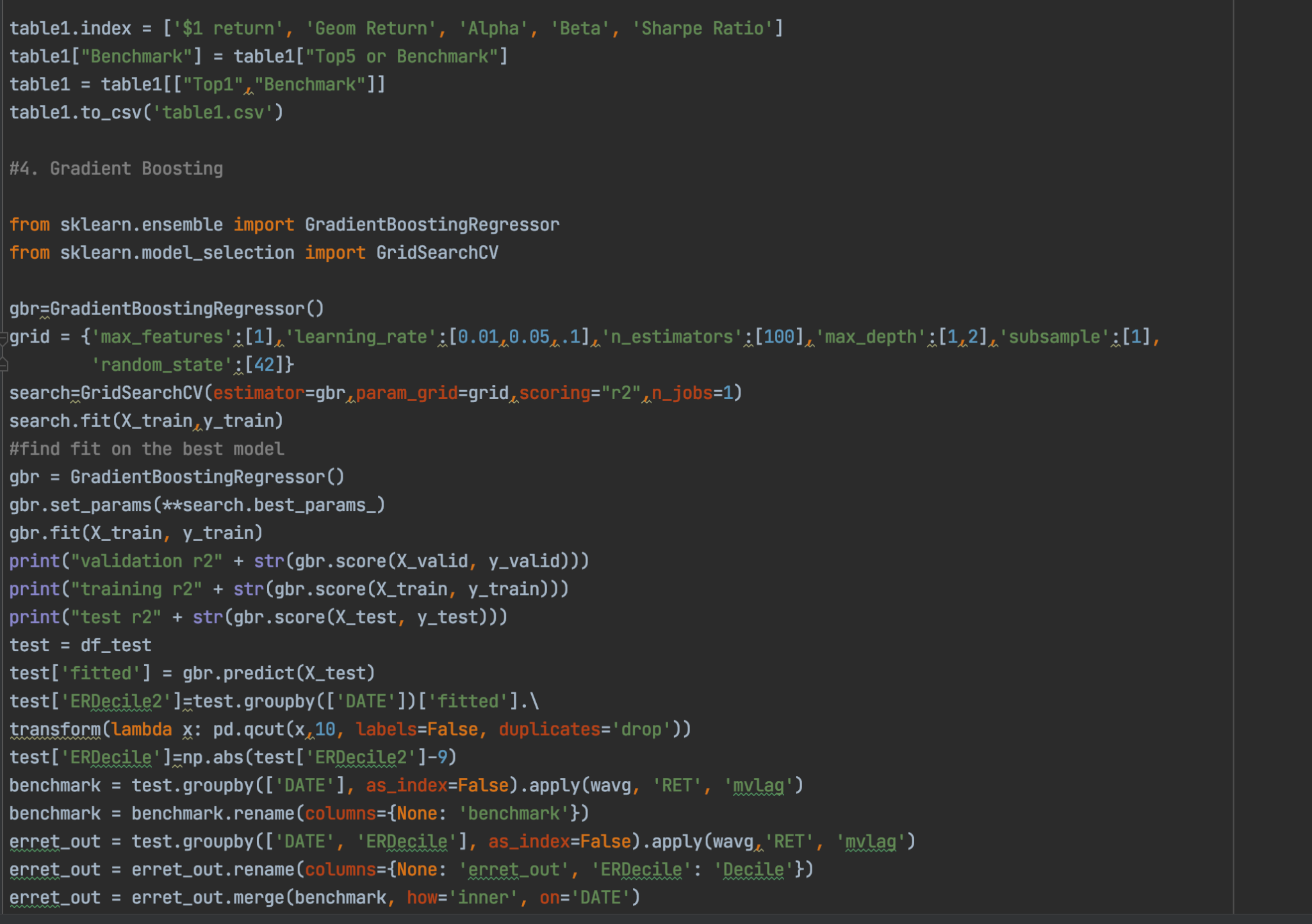


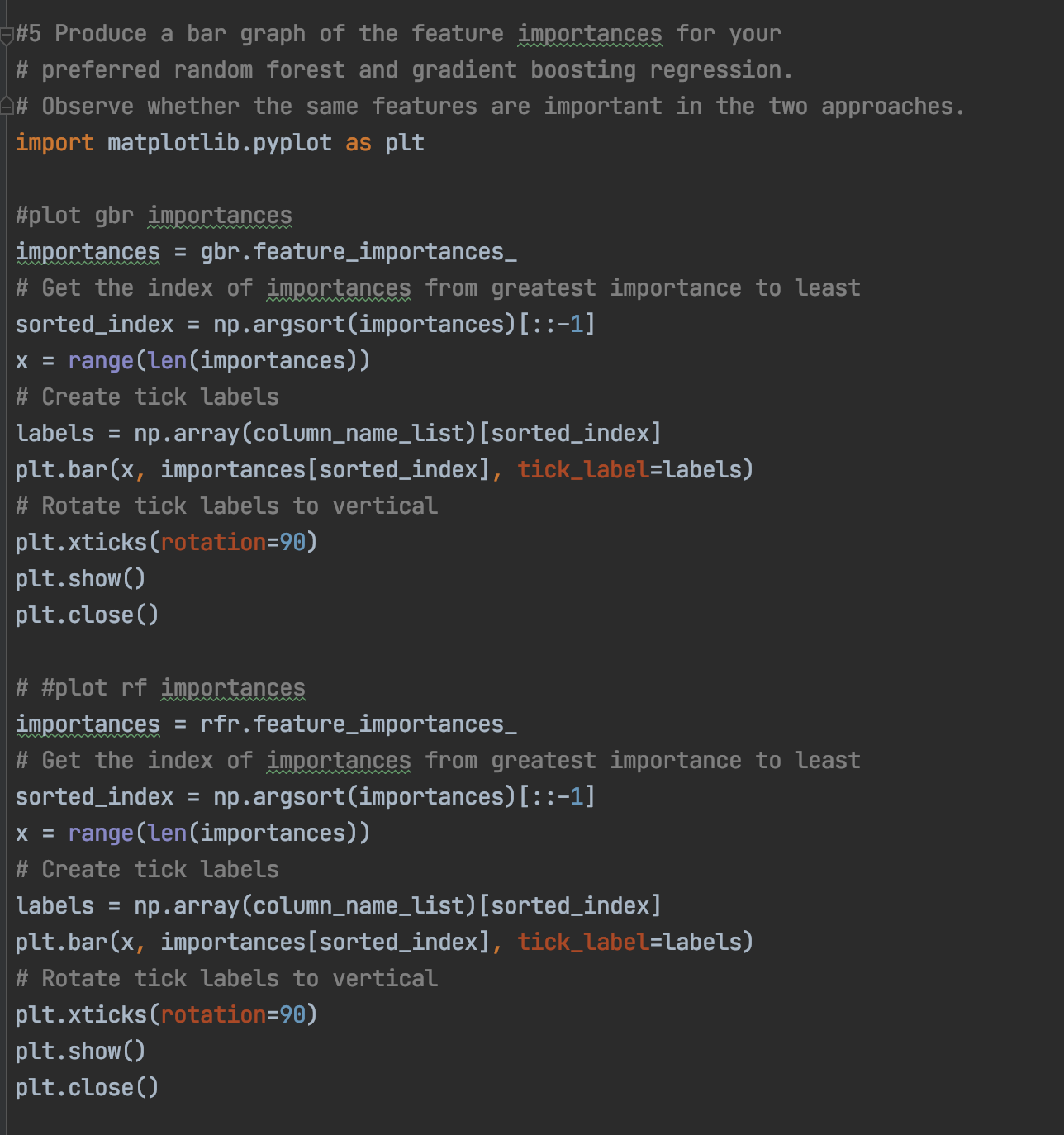




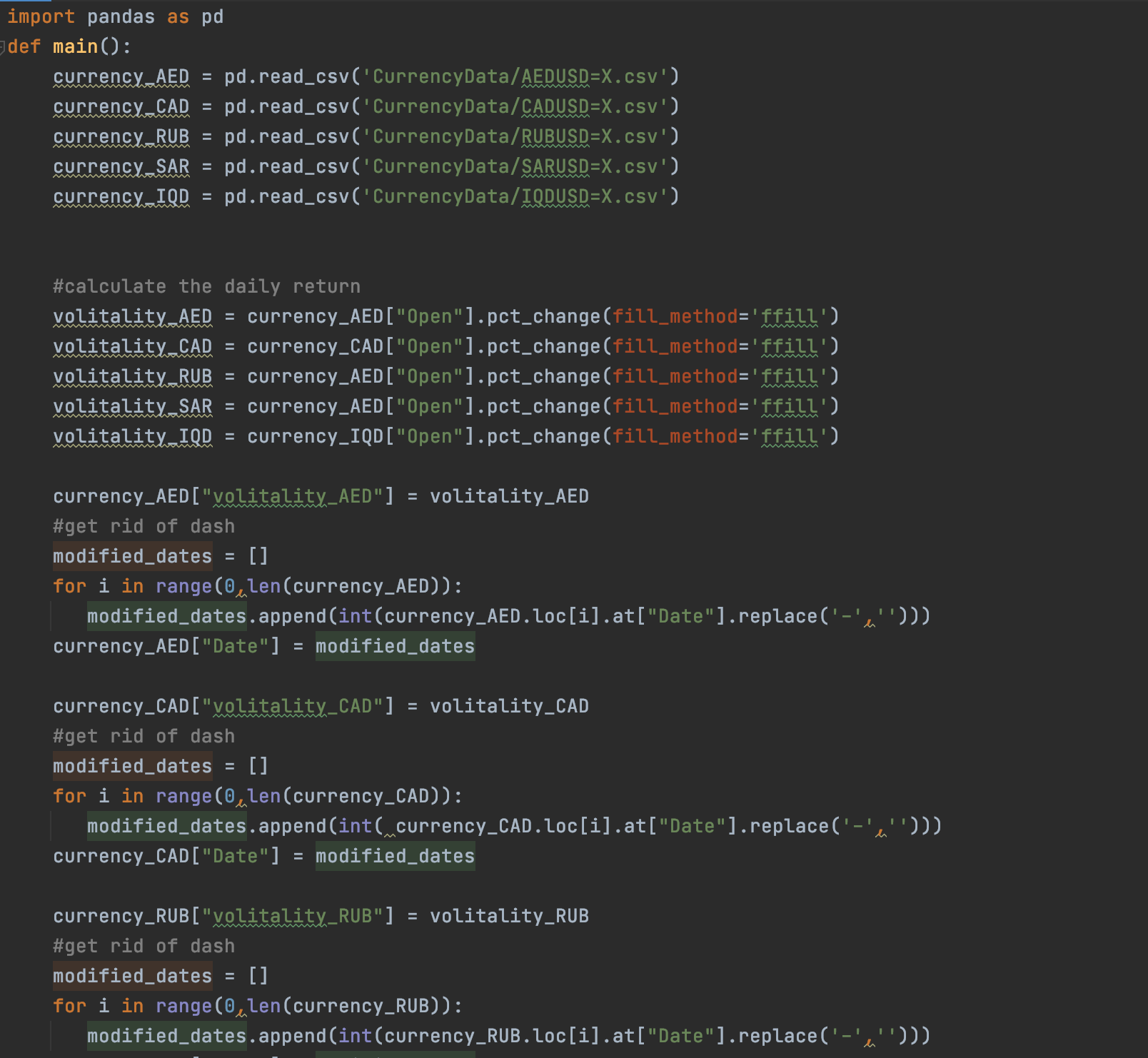


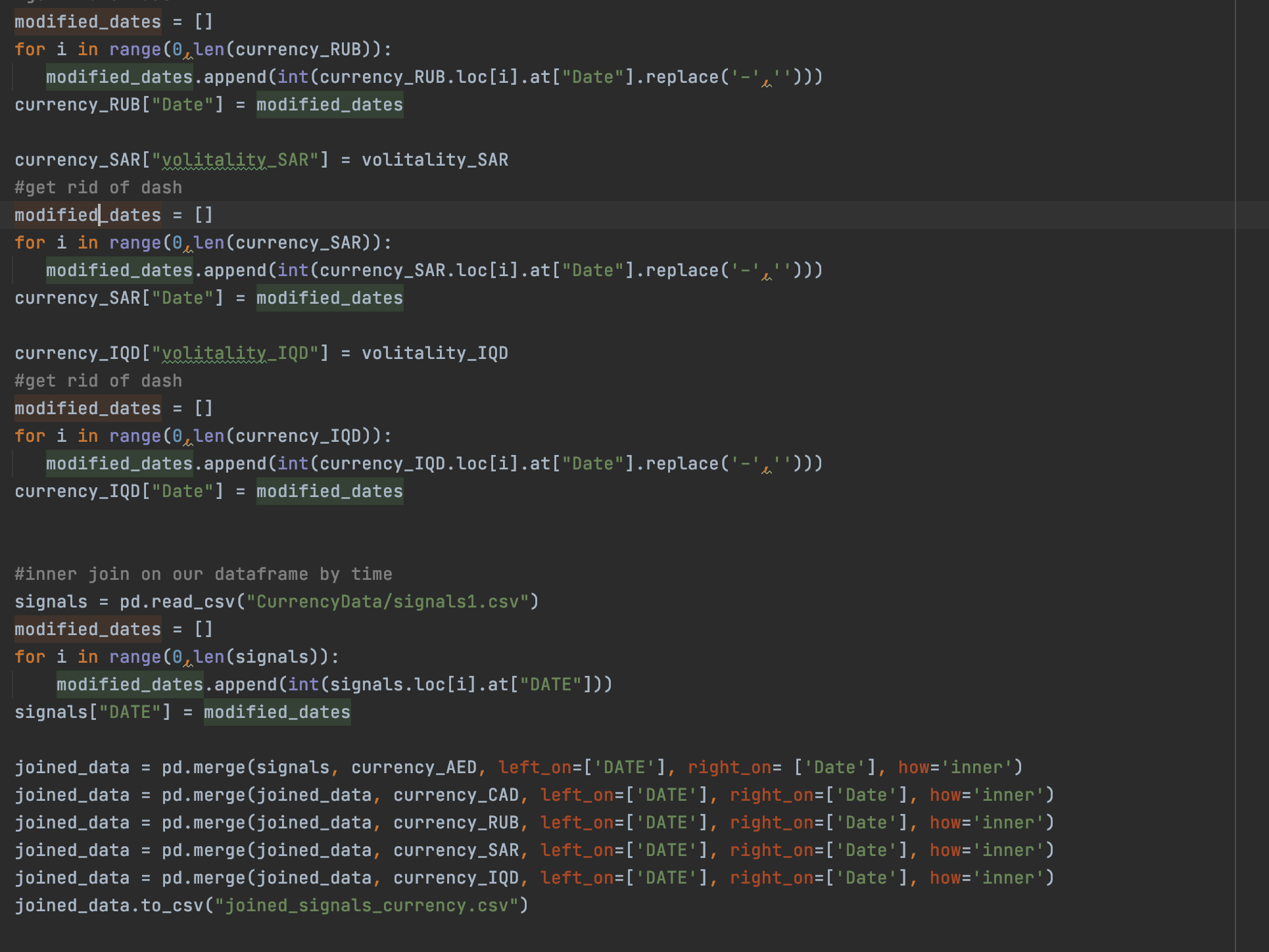






Joining the Currencies and Volatility Calculation Code





1. https://towardsdatascience.com/what-are-the-best-metrics-to-evaluate-your-regression-model-418ca481755b [↑](#footnote-ref-0)
2. Ibid. [↑](#footnote-ref-1)
3. http://www.stat.yale.edu/Courses/1997-98/101/linreg.htm [↑](#footnote-ref-2)