**Integer Programming**:

**PuLP**

PuLP is an open-source [linear programming](https://en.wikipedia.org/wiki/Linear_programming) (LP) package, it is an LP modeler written in python. PuLP can generate MPS or LP files and call GLPK, COIN CLP/CBC, CPLEX, and GUROBI to solve linear problems.

1. **Define Problem**

Define the problem by giving a suitable name to your problem. Also, specify your aim for the objective function of whether to LpMaximize or LpMinimize.

1. **Define Integer variables for each strategy (Decision Variables)**

Define LpVariable to hold the variables of the objective functions. The next argument specifies the lower bound and the upper bound of the defined variable. After that select type of category as per your requirement which can be LpContinuous or LpBinary or LpInteger.

1. **Create the objective function**

We start building the IP problem by adding the main objective function. We are defining objective function using decision variables and return values (which is difference between first and last value of return value of each strategies) of each strategies. By taking sum of product decision variables and return values, we created our objective function.

s1\*r1 + s2\*r2+ s3\*r3 + s4\*r4+..............................+s15\*r15

1. **Create constraints**

The constraints are added in a similar way to the objective.

1. **Run problem**

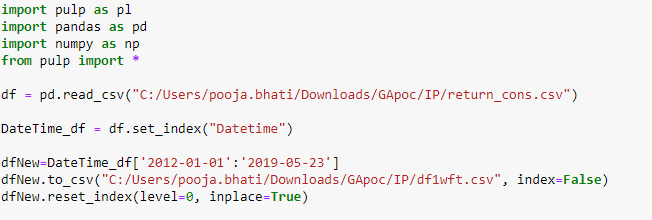
Run the problem using default solver.

1. **Getting Solution**

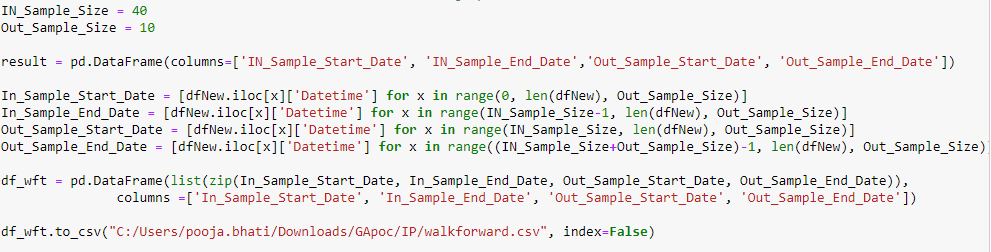
Now get the solution using variable.value() function and the objective value using problem.objective.value()

**Code Go through:**

Define library and the necessary packages required, read the return consolidated file from 2012 to 2019.

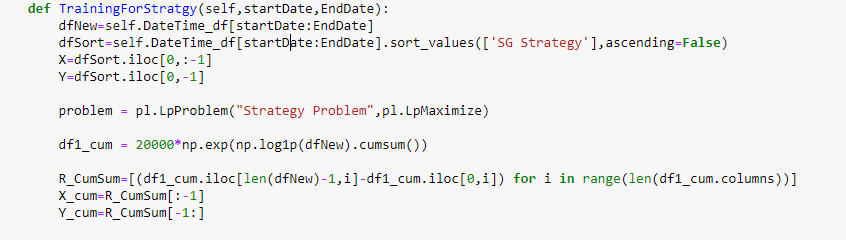


Using Walk forward testing create different iteration to run the problem statement multiple time of the sequenced data. In\_Sample\_Size is for training whereas Out\_Sample\_Size is of testing our problem statement.

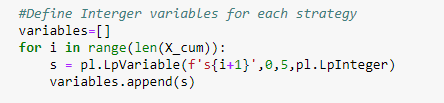


For particular iteration created by Walk forward testing, consider all 15 strategies return value as X and SG strategy as Y. We are problem as "problem" and the objective function as LpMaximize.

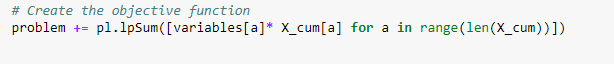
Now take cumulative sum of X and multiply be 20000 and after that take difference between first and last value for each strategies. Store that in X\_cum for all 15 strategies.



We defined variables as s1, s2, s3, s4,.....,s15, bound between 0 to 5 and Category as LpInteger. LpInteger forces all of the variables to assume only integer values.

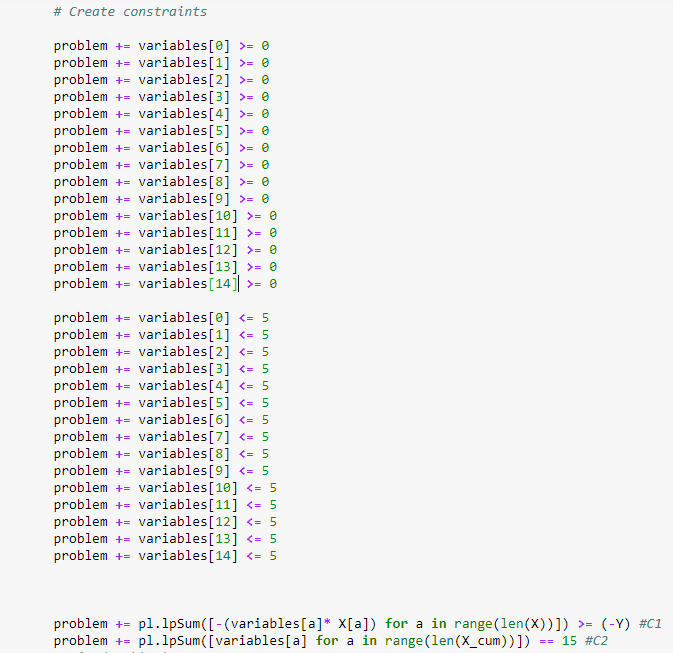


Now, create objective function using decision variables and X\_cum defined above.



We defined following constraints:

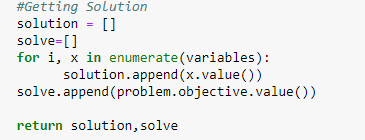
* Hard constraints for all the variables lies between 0 to 5.
* Main objective function value must be less than equal to SG Strategy return value(Sort values High to Low based on SG Strategy and use highest value to create constraint).
* Sum of all weights/objective value must be equal to 15.



Here we solve the problem,



Getting the solution/weight for each strategies and the objective value.



**Trend Prediction:**

1. **Linear Regression:**

**Background:**

The linear regression model returns an equation that determines the relationship between the independent variables and the dependent variable.

The equation for linear regression can be written as:

Here, x1, x2,….xn represent the independent variables while the coefficients θ1, θ2, …. θn represent the weights.

**Implementation:**

1. Sorting: We have sorted the data ascending order and then create a separate dataset so that any new feature created does not affect the original data.

2. Create features: we use the date column to extract features like – day, month, year, mon/fri etc. and then fit a linear regression model with the help of the library fastai.

This creates features such as:

‘Year’, ‘Month’, ‘Week’, ‘Day’, ‘Dayofweek’, ‘Dayofyear’, ‘Is\_month\_end’, ‘Is\_month\_start’, ‘Is\_quarter\_end’, ‘Is\_quarter\_start’, ‘Is\_year\_end’, and ‘Is\_year\_start’.

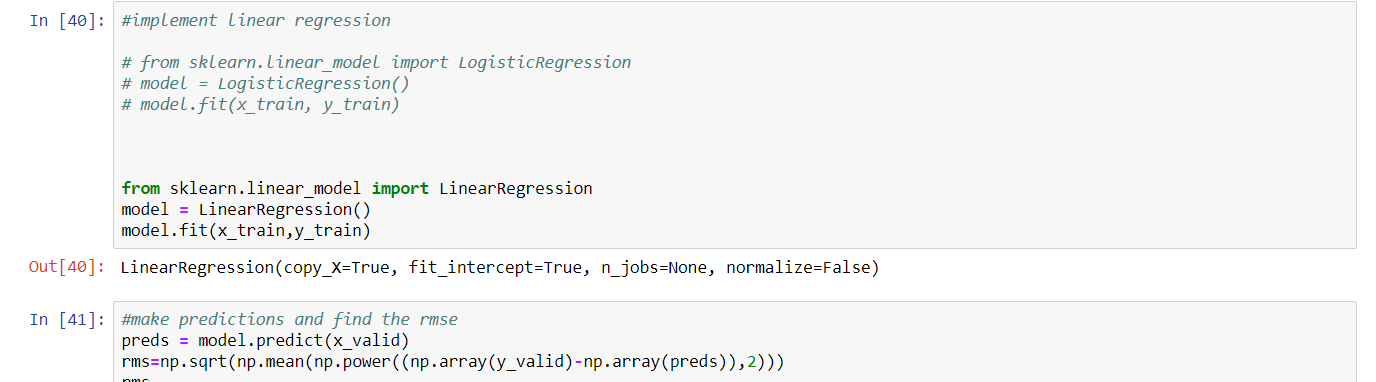
3. Traning and testing data split: We splited the data in training and testing in 80:20 ratio.

4. Applied Regression Model: After importing the Linear Regression from linear model, we have created the object of Linear regressor and then fit the traing data into regressor model through fit method.

5. Predictions and RMSE calculation: After fitting the model, we have predicted the values for x\_valid and then calculated the Root Mean Square Error(RMSE) for checking the accuracy.

**RMSE : 906.24**

**Code Snippet:**



1. **k-Nearest Neighbors:**

**background:**

Based on the independent variables, kNN finds the similarity between new data points and old data points.

K-nearest neighbors (KNN) algorithm is a type of supervised ML algorithm which can be used for both classification as well as regression predictive problems. However, it is mainly used for classification predictive problems in industry. The following two properties would define KNN well −

Lazy learning algorithm − KNN is a lazy learning algorithm because it does not have a specialized training phase and uses all the data for training while classification.

**Implementation:**

1. Sorting: We have sorted the data ascending order and then create a separate dataset so that any new feature created does not affect the original data.

2. Training and testing data split: We splitted the data in training and testing in 80:20 ratio.

3. Applied K Nearest Neighbors Model: After importing Nearest Neighbors from model, we have created the object of Nearest Neighbors and then fit the training data into Nearest Neighbors model through fit method.

5. Predictions and RMSE calculation: After fitting the model, we have predicted the values for x\_valid and then calculated the Root Mean Square Error(RMSE) for checking the accuracy.

**RMSE : 846.16**

**Code Snippet:**



1. **Auto ARIMA:**

**Background:**

ARIMA is a very popular statistical method for time series forecasting. ARIMA models take into account the past values to predict the future values. There are three important parameters in ARIMA:

p (past values used for forecasting the next value)

q (past forecast errors used to predict the future values)

d (order of differencing)

Parameter tuning for ARIMA consumes a lot of time. So we will use auto ARIMA which automatically selects the best combination of (p,q,d) that provides the least error.

**Implementation:**

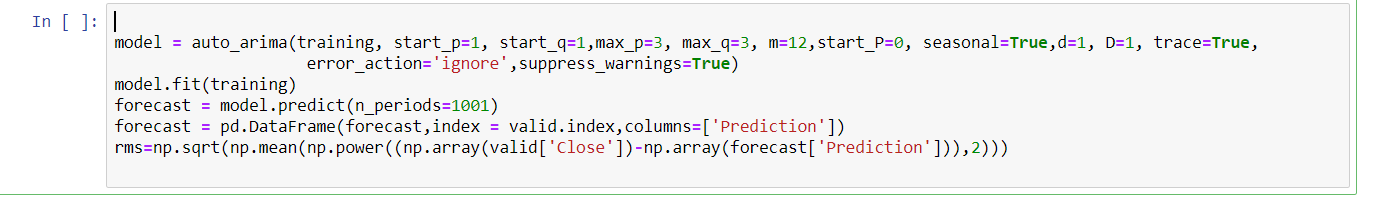
1. Sorting: We have sorted the data ascending order and then create a separate dataset so that any new feature created does not affect the original data.

2. Training and testing data split: We splitted the data in training and testing in 80:20 ratio.

3. Applied Auto ARIMA Model: After importing the Auto ARIMA from Pyramid ARIMA, we have created the object of Auto ARIMA and then fit the training data into Auto model through fit method.

4. Predictions and RMSE calculation: After fitting the model, we have predicted the values for test data and then calculated the Root Mean Square Error(RMSE) for checking the accuracy.

**Code Snippet:**



1. **Prophet:**

**Background:**

There are a number of time series techniques that can be implemented on the stock prediction dataset, but most of these techniques require a lot of data preprocessing before fitting the model. Prophet, designed and pioneered by Facebook, is a time series forecasting library that requires no data preprocessing and is extremely simple to implement. The input for Prophet is a data frame with two columns: date and target (ds and y).

Prophet tries to capture the seasonality in the past data and works well when the dataset is large.

**Implementation:**

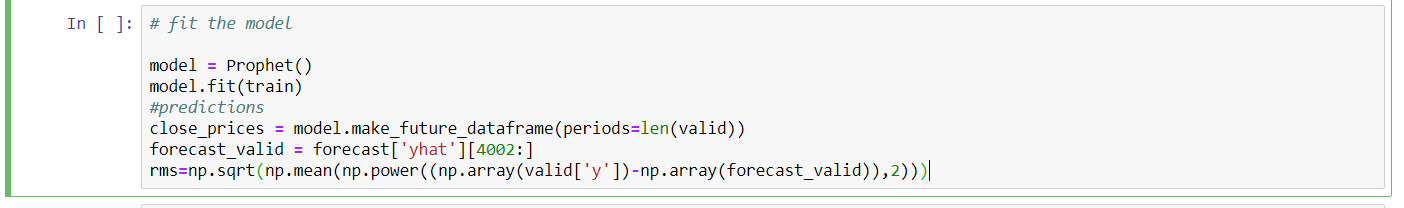
1. Data Preprocessing: We have created a separate dataset with Date and Close price so that any new feature created does not affect the original data and then rename Close price as y and Date as ds.

2. Training and testing data split: We splitted the data in training and testing in 80:20 ratio.

3. Applied Prophet Model: After importing the Prophet machine learning Model from fbprophet library, we have created the object of Prophet and then fit the training data into Prophet model through fit method.

4. Predictions and RMSE calculation: After fitting the model, we have predicted the values for x\_valid and then calculated the Root Mean Square Error(RMSE) for checking the accuracy.

**Code Snippet:**



1. **Long Short Term Memory (LSTM):**

**Background:**

LSTMs are widely used for sequence prediction problems and have proven to be extremely effective. The reason they work so well is because LSTM is able to store past information that is important, and forget the information that is not. LSTM has three gates:

The input gate: The input gate adds information to the cell state

The forget gate: It removes the information that is no longer required by the model

The output gate: Output Gate at LSTM selects the information to be shown as output

**Key Terms:**

**Vanishing Gradient:** Vanishing Gradient problem arises while training an Artificial Neural Network. This mainly occurs when the network parameters and hyper parameters are not properly set. Parameters could be weights and biases while hyper parameters could be learning rate, the number of epochs, the number of batches, etc.

**Activation function:** In artificial neural networks, the activation function of a node defines the output of that node given an input or set of inputs.

**Sigmoid function:** A sigmoid function is a mathematical function having a characteristic "S"-shaped curve or sigmoid curve.

**Libraries:** Keras sequential, Drop, Dropout, LSTM

**Implementation:**

1. Sorting: We have sorted the data ascending order and then create a separate dataset so that any new feature created does not affect the original data.

2. Training and testing data split: We splitted the data in training and testing in 80:20 ratio.

3. Add LSTM Model: After importing the LSTM from Sequential model, we have created the object of Sequential and add Sequential object in Model.

4. Compile the Model: Compilation is prerequisite prior to fit the model so we have compiled the model.

5. Fit the Model: After compilation model is ready for fitting. we have fitted the model with training data and default parameters epochs=1, batch\_size=1, verbose=2

6. Predictions and RMSE calculation: After fitting the model, we have predicted the values for x\_valid and then calculated the Root Mean Square Error(RMSE) for checking the accuracy.

**Code Snippet:**

