Students

stock price pediction

TEAM MEMBER

# 

**AU732521104019:JAYASURYA.A**

**PHASE 4: SUBMISSION DOCUMENT**

Team member

Project title: stock price prediction

A graph with arrows pointing up

Description automatically generated with medium confidence

**INTRODUCTION:**

In this project, you want to build a stock price prediction model. You can approach this task using various techniques, including machine learning and time series analysis. Here, I'll give you a simplified example using a basic machine learning approach with Python and some popular libraries like NumPy, Pandas, and Scikit-Learn.

**Feature Engineering:**

Feature engineering is the process of selecting and creating relevant features that can help the model make better predictions. For stock price prediction, you can consider the following features:

**Model Training:**

Once you have prepared your features, you can proceed to train a machine learning model. Common algorithms for time series forecasting include Linear Regression, Random Forest, ARIMA, and LSTM (for deep learning).

**EXPLORATORY ANALYSIS**

To begin this exploratory analysis, first import libraries and define functions for plotting the data using matplotlib. Depending on the data, not all plots will be made. (Hey, I'm just a simple kerneling bot, not a Kaggle Competitions Grandmaster!)

**DATA SOURCE:**

**DATASET LINK**:<https://www.kaggle.com/datasets/prasoonkottarathil/microsoft-lifetime-stocks-dataset/>

**Data Collection and Preprocessing:**

* Importing the dataset: Obtain a comprehensive dataset containing relevant features

such as square footage, number of bedrooms, location, amenities, etc.

* Data preprocessing: Clean the data by handling missing values, outliers, and

categorical variables. Standardize or normalize numerical features.

**PROGRAM:**

**STOCK PRICEPREDICTION**

**Project Overview:**

In this project, we aim to predict the future price of a specific stock based on historical data and relevant features. We'll follow these steps:

**Data Collection:** Obtain historical stock price data and other relevant financial data.

**Feature Engineering**: Create meaningful features from the collected data.

**Model Training:** Train a machine learning model on historical data.

**Model Evaluation:** Evaluate the model's performance using appropriate metrics.

**Predictions:** Use the trained model to make future stock price predictions.

**Feature Engineering**

Feature Engineeringhelps to derive some valuable features from the existing ones. These extra features sometimes help in increasing the performance of the model significantly and certainly help to gain deeper insights into the data.

* Python3

|  |
| --- |
| splitted = df['Date'].str.split('/', expand=True)    df['day'] = splitted[1].astype('int')  df['month'] = splitted[0].astype('int')  df['year'] = splitted[2].astype('int')    df.head() |

**Output:**

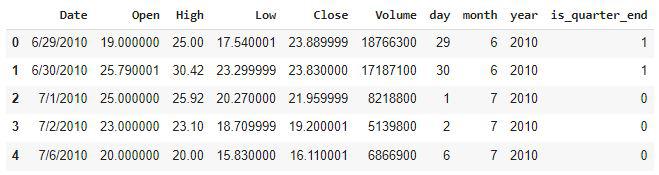


Now we have three more columns namely ‘day’, ‘month’ and ‘year’ all these three have been derived from the ‘Date’ column which was initially provided in the data.

* Python3

|  |
| --- |
| df['is\_quarter\_end'] = np.where(df['month']%3==0,1,0)  df.head() |

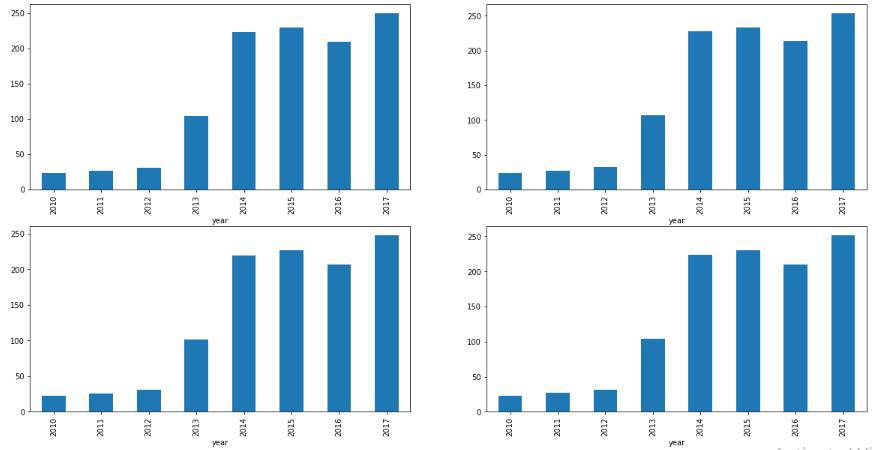
**Output:**



A quarter is defined as a group of three months. Every company prepares its quarterly results and publishes them publicly so, that people can analyze the company’s performance. These quarterly results affect the stock prices heavily which is why we have added this feature because this can be a helpful feature for the learning model.

* Python3

|  |
| --- |
| data\_grouped = df.groupby('year').mean()  plt.subplots(figsize=(20,10))    for i, col in enumerate(['Open', 'High', 'Low', 'Close']):    plt.subplot(2,2,i+1)    data\_grouped[col].plot.bar()  plt.show() |

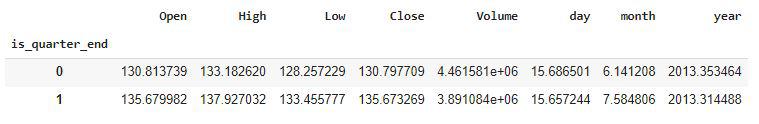
**Output:**

From the above [bar graph](https://www.geeksforgeeks.org/bar-plot-in-matplotlib/), we can conclude that the stock prices have doubled from the year 2013 to that in 2014.

* Python3

|  |
| --- |
| df.groupby('is\_quarter\_end').mean() |

**Output:**



Here are some of the important observations of the above-grouped data:

* Prices are higher in the months which are quarter end as compared to that of the non-quarter end months.
* The volume of trades is lower in the months which are quarter end.
* Python3

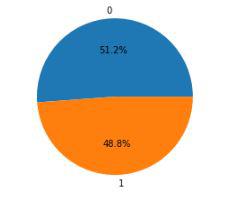
|  |
| --- |
| df['open-close']  = df['Open'] - df['Close']  df['low-high']  = df['Low'] - df['High']  df['target'] = np.where(df['Close'].shift(-1) > df['Close'], 1, 0) |

Above we have added some more columns which will help in the training of our model. We have added the target feature which is a signal whether to buy or not we will train our model to predict this only. But before proceeding let’s check whether the target is balanced or not using a [pie chart](https://www.geeksforgeeks.org/plot-a-pie-chart-in-python-using-matplotlib/).

* Python3

|  |
| --- |
| plt.pie(df['target'].value\_counts().values,          labels=[0, 1], autopct='%1.1f%%')  plt.show() |

**Output:**

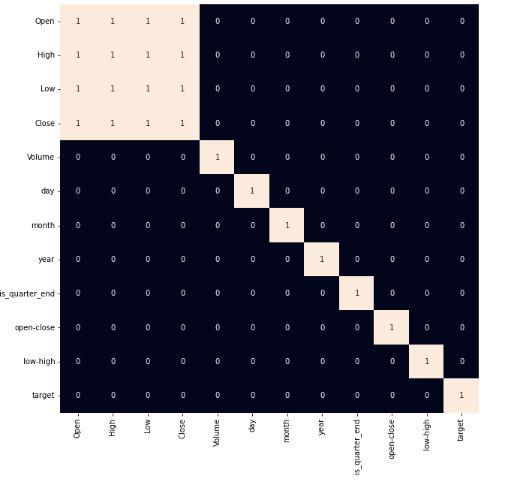


When we add features to our dataset we have to ensure that there are no highly correlated features as they do not help in the learning process of the algorithm.

* Python3

|  |
| --- |
| plt.figure(figsize=(10, 10))    # As our concern is with the highly  # correlated features only so, we will visualize  # our heatmap as per that criteria only.  sb.heatmap(df.corr() > 0.9, annot=True, cbar=False)  plt.show() |

**Output:**



*Heatmap of the correlation between the features*

From the above [heatmap](https://www.geeksforgeeks.org/seaborn-heatmap-a-comprehensive-guide/), we can say that there is a high correlation between OHLC that is pretty obvious, and the added features are not highly correlated with each other or previously provided features which means that we are good to go and build our model.

**Data Splitting and Normalization**

* Python3

|  |
| --- |
| features = df[['open-close', 'low-high', 'is\_quarter\_end']]  target = df['target']    scaler = StandardScaler()  features = scaler.fit\_transform(features)    X\_train, X\_valid, Y\_train, Y\_valid = train\_test\_split(      features, target, test\_size=0.1, random\_state=2022)  print(X\_train.shape, X\_valid.shape) |

**Output:**

(1522, 3) (170, 3)

After selecting the features to train the model on we should [**normalize**](https://www.geeksforgeeks.org/what-is-data-normalization-and-why-is-it-important/)the data because normalized data leads to stable and fast training of the model. After that whole data has been [**split**](https://www.geeksforgeeks.org/how-to-split-a-dataset-into-train-and-test-sets-using-python/)into two parts with a 90/10 ratio so, that we can evaluate the performance of our model on unseen data

**Model Development and Evaluation**

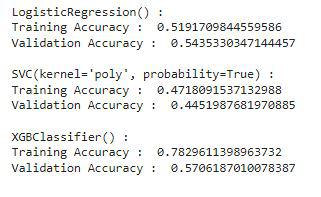
Now is the time to train some state-of-the-art machine learning models([Logistic Regression](https://www.geeksforgeeks.org/understanding-logistic-regression/), [Support Vector Machine](https://www.geeksforgeeks.org/support-vector-machine-algorithm/), [XGBClassifier](https://www.geeksforgeeks.org/ml-xgboost-extreme-gradient-boosting/)), and then based on their performance on the training and validation data we will choose which ML model is serving the purpose at hand better.

For the evaluation metric, we will use the [ROC-AUC curve](https://www.geeksforgeeks.org/auc-roc-curve/) but why this is because instead of predicting the hard probability that is 0 or 1 we would like it to predict soft probabilities that are continuous values between 0 to 1. And with soft probabilities, the ROC-AUC curve is generally used to measure the accuracy of the predictions.

* Python3

|  |
| --- |
| models = [LogisticRegression(), SVC(    kernel='poly', probability=True), XGBClassifier()]    for i in range(3):    models[i].fit(X\_train, Y\_train)      print(f'{models[i]} : ')    print('Training Accuracy : ', metrics.roc\_auc\_score(      Y\_train, models[i].predict\_proba(X\_train)[:,1]))    print('Validation Accuracy : ', metrics.roc\_auc\_score(      Y\_valid, models[i].predict\_proba(X\_valid)[:,1]))    print() |

**Output:**



*Evaluation of the model on training and the testing data.*

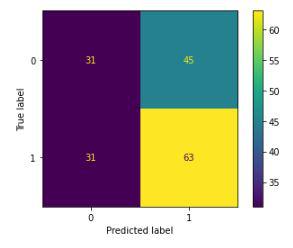
Among the three models, we have trained XGBClassifier has the highest performance but it is pruned to [overfitting](https://www.geeksforgeeks.org/underfitting-and-overfitting-in-machine-learning/) as the difference between the training and the validation accuracy is too high. But in the case of the Logistic Regression, this is not the case.

Now let’s plot a [**confusion matrix**](https://www.geeksforgeeks.org/confusion-matrix-machine-learning/) for the validation data.

* Python3

|  |
| --- |
| metrics.plot\_confusion\_matrix(models[0], X\_valid, Y\_valid)  plt.show() |

**Output:**



*Confusion matrix for the validation data*

**Conclusion:**

We can observe that the accuracy achieved by the state-of-the-art ML model is no better than simply guessing with a probability of 50%. Possible reasons for this may be the lack of data or using a very simple model to perform such a complex task as Stock Market prediction.