A Time Series Analysis-Based Stock Price Prediction Using Machine Learning

PREPARED BY:

- **Jasjappan Singh 209302267**
- **Archit Jain 209302015**

DATE: 10TH November 2022

Index

Abstract

Over the **years,** the stock market has been **viewed** by people around the **world as a very risky investment.** This project aims to **make sense** of stock market **historical data, especially the** nifty **50,** and derive analysis from **it, reducing** the **knowledge** gap between market behavior and **investors. Stock** data **consists** of **many** statistical terms **that** are difficult to understand **for ordinary people** who **want** to **start investing in the** stock **market, so** this project aims **to fill** the **knowledge gaps.** This study aims to **explain future** market **scenarios** by **corroborating** statistical **responses.**

The Project Involves analyzing Nifty50 Data Over the Span of 12 Years. Opening Price, Closing Price, Lowest Price, and Highest Prize are the attributes available The Data is further Explored through Exploratory Data Analysis.

Problem Statement

A Time Series Analysis-Based Stock Price Prediction Using Machine Learning - Random Forest, Decision Trees, Regression, and Classification Algorithms to Predict if Market Goes Up or Down the next Day as well as Predict Future Stock Price through Regression Algorithms.

Literature review

There are many kinds of research works in the area of forecasting using time series analysis. Some of the important tasks are mentioned here. A study deals with the implication of support vector machines (SVMs) regression, XG Boost Regression, in predicting the share price to examine the feasibility of SVM regression in predicting stock price. The study examines the performance of the forecasting ability of the models. It was observed that different models
performed well in different periods

References

https://www.researchgate.net/publication/340938611_A_Time_Series_Analysis-Based_Stock_Price_Prediction_Using_Machine_Learning_and_Deep_Learning_Models

https://www.researchgate.net/publication/348195151_Stock_market_analysis_and_prediction_using_time_series_analysis

https://www.kaggle.com/code/jagannathrk/stock-market-time-series

https://www.analyticsvidhya.com/blog/2021/07/stock-market-forecasting-using-time-series-analysis-with-arima-model/

Data Set From YFinance Nifty Data 2007 - 2022

Code for KNN Imputer

```
check_nan = nifty.isnull().values.any()
print(check_nan)
imputer = KNNImputer(n_neighbors=2)
d1 = imputer.fit_transform(nifty)
```

Convert to Date Time Format

```
import datetime

df['Date'] = pd.to_datetime(df['Date'], format="%Y %m %d")
df['Date'].isnull()
check_nan = df['Date'].isnull().values.any()

# printing the result
print(check_nan)
```

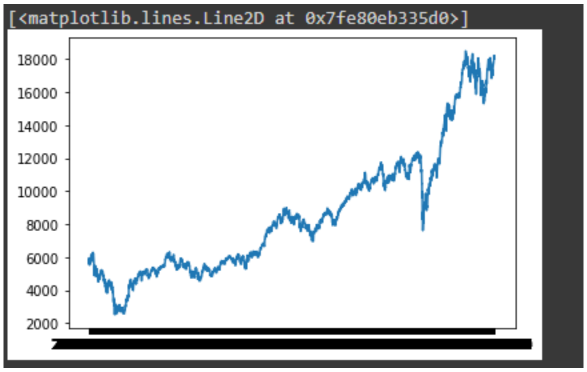
	Open	High	Low	Close	Adj Close	Volume	Tommorow	Target
0	5660.600098	5660.600098	5477.500000	5617.100098	5617.100098	0.0	5695.399902	1
1	5612.350098	5758.850098	5591.600098	5695.399902	5695.399902	0.0	5937.899902	1
2	5703.950195	5950.200195	5700.049805	5937.899902	5937.899902	0.0	5912.100098	0
3	5942.700195	5966.950195	5895.649902	5912.100098	5912.100098	0.0	5906.850098	0
4	5913.149902	5948.049805	5817.399902	5906.850098	5906.850098	0.0	5907.649902	1
...	...	...	...	...	...	...	...	...
3698	17968.349609	18106.300781	17959.199219	18052.699219	18052.699219	213000.0	18117.150391	1
3699	18053.400391	18135.099609	18017.150391	18117.150391	18117.150391	267900.0	18202.800781	1
3700	18211.750000	18255.500000	18064.750000	18202.800781	18202.800781	314800.0	18157.000000	0
3701	18288.250000	18296.400391	18117.500000	18157.000000	18157.000000	307200.0	18028.199219	0
3702	18044.349609	18103.099609	17969.400391	18028.199219	18028.199219	0.0	NaN	0

3703 rows × 8 columns

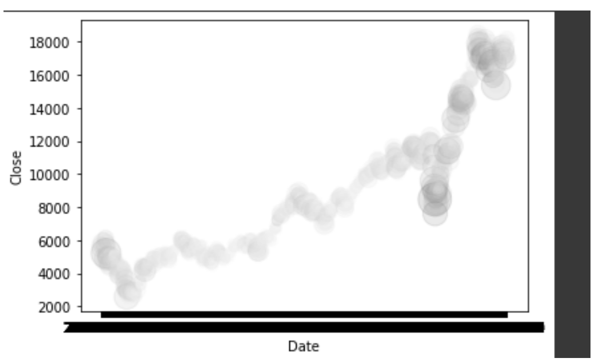
Data Before Imputation with KNN

Data Analysis

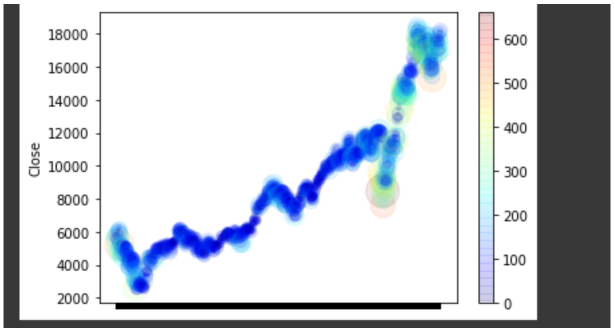
[<matplotlib.lines.Line2D at 0x7fe80eb335d0>]



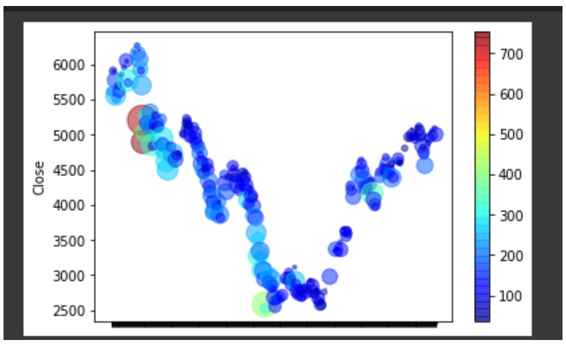
Stock Market Price Plot



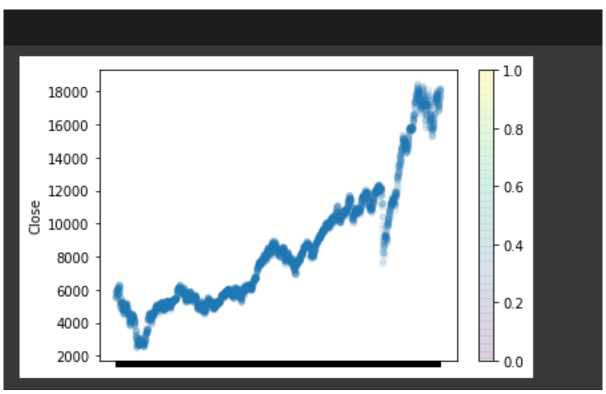
Market Volatility



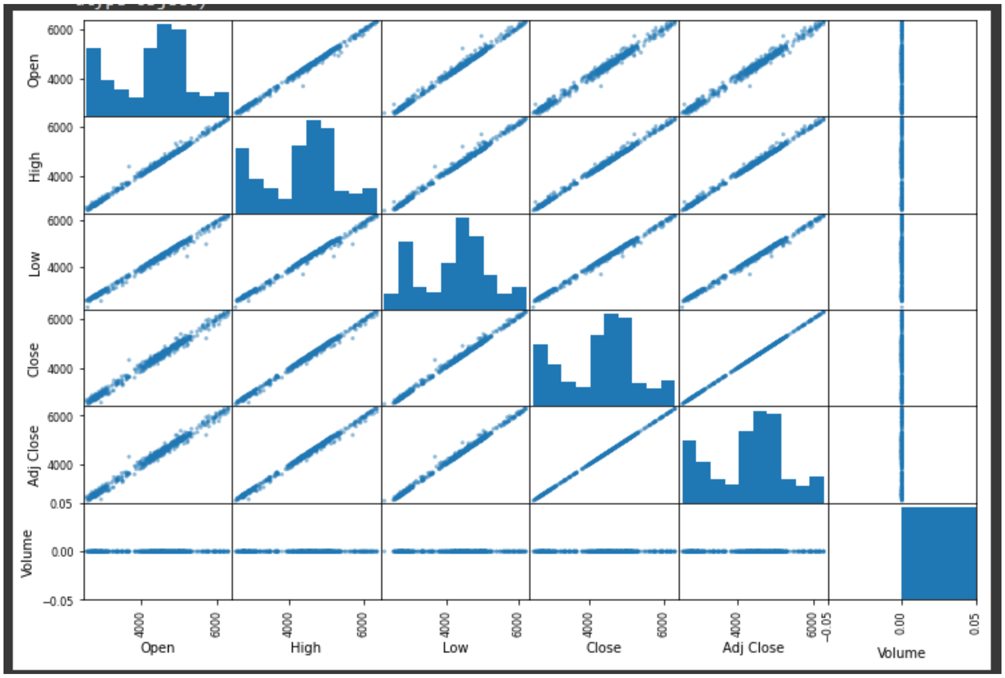
The difference in Opening and Closing Prices as well as Highest and Lowest Value



Plot for 500 Days from Beginning



Jet Scatter Plot for Date vs Closing price



Scatter Matrix for All Attributes

Correlation Coefficient Calculation

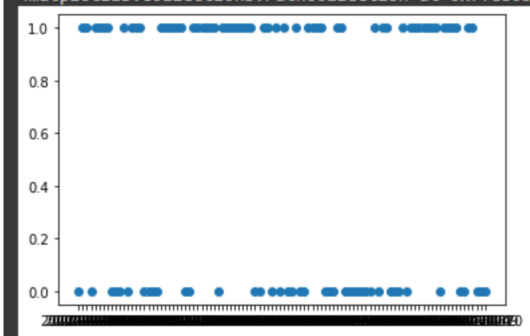
	Open	High	Low	Close	Adj Close	Volume
Open	1.000000	0.999888	0.999841	0.999739	0.999739	0.632835
High	0.999888	1.000000	0.999797	0.999879	0.999879	0.634527
Low	0.999841	0.999797	1.000000	0.999872	0.999872	0.630128
Close	0.999739	0.999879	0.999872	1.000000	1.000000	0.632319
Adj Close	0.999739	0.999879	0.999872	1.000000	1.000000	0.632319
Volume	0.632835	0.634527	0.630128	0.632319	0.632319	1.000000

Random Forest Data Prediction

A random forest is a meta-estimator that fits a number of decision tree classifiers on various sub-samples of the dataset and uses averaging to improve the predictive accuracy and control over-fitting. The sub-sample size is controlled with the parameter if(default), otherwise the whole dataset is used to build each tree.

```
plt.scatter(date_pred,test[7])
```

```
<matplotlib.collections.PathCollection at 0x7fe8055f70d0>
```




```
plt.scatter(date_pred,preds)
```

```
<matplotlib.collections.PathCollection at 0x7fe8052b1150>
```

Model Precision - `0.6896551724137931`

Tweaking Parameters to Improve Performance

Adding Next Days Clossing Price as a Predictor

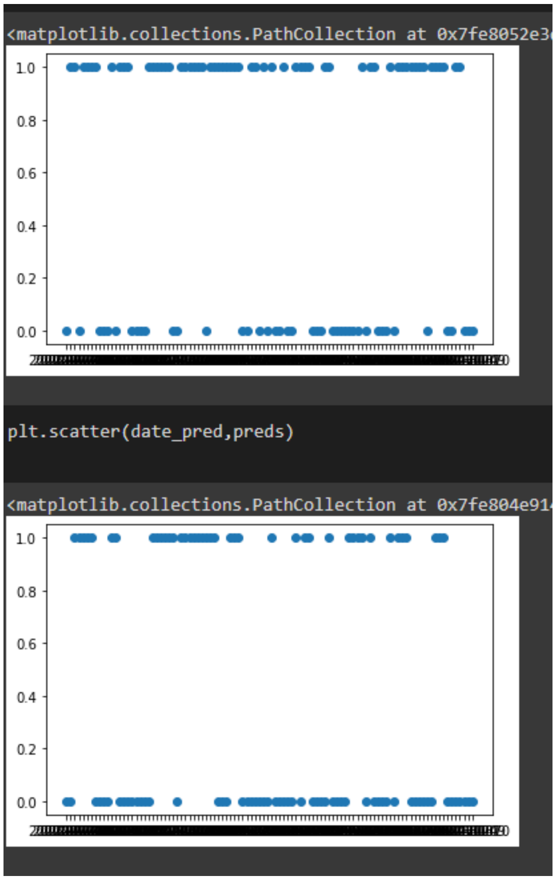
```
[37] model = RandomForestClassifier(n_estimators=50, min_samples_split=50, random_state=1)
     train = d1.iloc[:-100]
     test = d1.iloc[-100:]


     predictors = [0,1,2,3,5,6]
     model.fit(train[predictors], train[7])

     preds = model.predict(test[predictors])
     preds = pd.Series(preds, index=test.index)
     precision_score(test[7], preds)

     0.7916666666666666
```

Max Precision Achieved `0.7916666666666666`

Using XGBoost Classifier

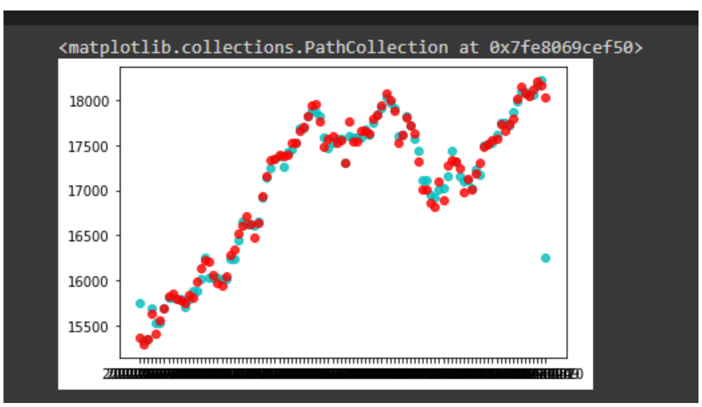
XGBoost minimizes a regularized (L1 and L2) objective function that combines a convex loss function (based on the difference between the predicted and target outputs) and a penalty term for model complexity (in other words, the regression tree functions).

Precision Achieved - `0.8333333333333334`



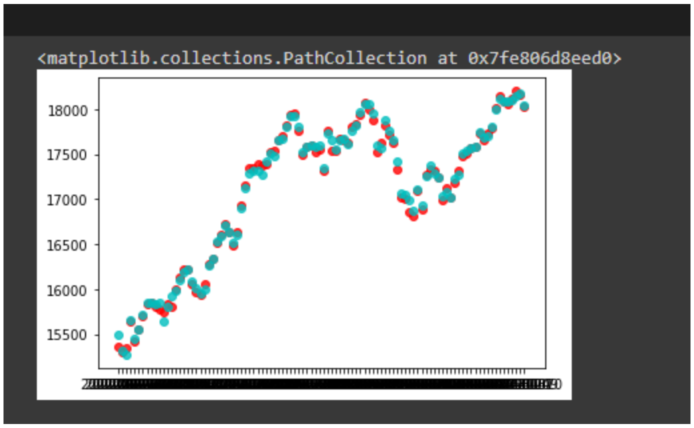
Stock Prediction Using Gradient Boost Regression

Gradient boosting is one of the most popular machine learning algorithms for tabular datasets. It is powerful enough to find **non-linear relationships** between model **targets** and **features,** and **easy to use to handle** missing values, outliers, and **high-cardinality** categorical values **in** features without special **handling. increase.**



Cyan Predicted Data, Red Orignal Data

Stock Prediction Using ARDR Regression



The ARDRegression **considers the model weights as a Gaussian distributed and estimates the lambda and alpha parameters through the iterati**

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.pyplot as plt

import xgboost as xgb

from tensorflow import keras
from tensorflow.keras import layers


from keras.layers import Dense
from sklearn.impute import KNNImputer
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import precision_score

df = pd.read_csv('nifty50.csv')

import datetime

df['Date'] = pd.to_datetime(df['Date'], format="%Y %m %d")
df['Date'].isnull()
check_nan = df['Date'].isnull().values.any()

# printing the result
print(check_nan)

"""# **Exploratory Data Analysis**"""

plt.plot(df['Date'], df['Close'])



df.plot.scatter(x = 'Date', y = 'Close', alpha=0.1, s=df['Open']-df['Close'],c=(df['High']-df['Close']));

df.plot.scatter(x = 'Date', y = 'Close', alpha=0.1, s=df['Open']-df['Close'],c=(df['High']-df['Close']), cmap=plt.get_cmap("jet"), colorbar

df.plot.scatter(x = 'Date', y = 'Close', alpha=0.1, cmap=plt.get_cmap("jet"), colorbar=True);

testdata = df[0:500]
testdata.plot.scatter(x = 'Date', y = 'Close', alpha=0.5,s=testdata["Open"]-testdata["Close"],c=testdata["High"]-testdata["Low"], cmap=plt.

pd.plotting.scatter_matrix(testdata, figsize=(12,8))

corr_mat = df.corr()
corr_mat

nifty = df.iloc[ :, 1:]

nifty

nifty["Tommorow"] = nifty["Close"].shift(-1)
nifty["Target"] = (nifty["Tommorow"] > nifty["Close"]).astype(int)
```

```
"""# **Removing Nan Data with KNN**"""

check_nan = nifty.isnull().values.any()

print(check_nan)

imputer = KNNImputer(n_neighbors=2)
d1 = imputer.fit_transform(nifty)

nifty

d1 = pd.DataFrame(d1)

df.isnull()

check_nan = d1.isnull().values.any()

print(check_nan)

"""#  **Random Fores Classification**

---


"""

model = RandomForestClassifier(n_estimators=100, min_samples_split=100, random_state=1)
train = d1.iloc[:-100]
test = d1.iloc[-100:]





d1

df

predictors = [0,1,2,3,5]
model.fit(train[predictors], train[7])

"""## **Random Fores M1**"""

preds = model.predict(test[predictors])
preds = pd.Series(preds, index=test.index)
precision_score(test[7], preds)

preds

train

test[7]

df

date_pred = df.iloc[-100:, 0]

plt.scatter(date_pred,test[7])

plt.scatter(date_pred,preds)

"""## Random Forest **M2**"""

model = RandomForestClassifier(n_estimators=50, min_samples_split=10, random_state=1)
train = d1.iloc[:-100]
test = d1.iloc[-100:]


predictors = [0,1,2,3,5]
model.fit(train[predictors], train[7])

preds = model.predict(test[predictors])
preds = pd.Series(preds, index=test.index)
precision_score(test[7], preds)

model = RandomForestClassifier(n_estimators=50, min_samples_split=50, random_state=1)
train = d1.iloc[:-100]
test = d1.iloc[-100:]


predictors = [0,1,2,3,5]
```

```
model.fit(train[predictors], train[7])

preds = model.predict(test[predictors])
preds = pd.Series(preds, index=test.index)
precision_score(test[7], preds)

"""Adding Next Days Clossing Price as a Predictor"""

model = RandomForestClassifier(n_estimators=50, min_samples_split=50, random_state=1)
train = d1.iloc[:-100]
test = d1.iloc[-100:]


predictors = [0,1,2,3,5,6]
model.fit(train[predictors], train[7])

preds = model.predict(test[predictors])
preds = pd.Series(preds, index=test.index)
precision_score(test[7], preds)

model = RandomForestClassifier(n_estimators=100, min_samples_split=100, random_state=1)
train = d1.iloc[:-100]
test = d1.iloc[-100:]


predictors = [0,1,2,3,5,6]
model.fit(train[predictors], train[7])

preds = model.predict(test[predictors])
preds = pd.Series(preds, index=test.index)
precision_score(test[7], preds)

"""**XGBoost Classification**"""

import xgboost as xgb
predictors = [0,1,2,3,5,6]

xgb = xgb.XGBClassifier()
xgb.fit(train[predictors], train[7])

preds = xgb.predict(test[predictors])
preds = pd.Series(preds, index=test.index)
precision_score(test[7], preds)

model = RandomForestClassifier(n_estimators=100, min_samples_split=100, random_state=1)

model = RandomForestClassifier(n_estimators=100, min_samples_split=100, random_state=1)
train = d1.iloc[:-100]
test = d1.iloc[-100:]


predictors = [0,1,2,3,5,6]
model.fit(train[predictors], train[7])

preds = model.predict(test[predictors])
preds = pd.Series(preds, index=test.index)
precision_score(test[7], preds)

plt.scatter(date_pred,test[7])

plt.scatter(date_pred,preds)

"""### Decision **Tree**"""

from sklearn import tree
treeplt = tree.DecisionTreeClassifier()
train = d1.iloc[0:2500]
test = d1.iloc[2500:]

predictors = [0,1,2,3,5]
treeplt.fit(train[predictors], train[7])

preds = treeplt.predict(test[predictors])
preds = pd.Series(preds, index=test.index)
precision_score(test[7], preds)

tree.plot_tree(treeplt)

plt.scatter(date_pred,test[7],alpha=0.8,color = 'r')

plt.scatter(date_pred,preds,alpha=0.8,color = 'c')
```

```
"""Cyan for Predicted Data and Red for actual Data

## **Regression Based Prediction of Stock Market**
"""

train = d1.iloc[:-100]
test = d1.iloc[-100:]

from sklearn.ensemble import GradientBoostingRegressor
gbr = GradientBoostingRegressor(random_state=42)

predictors = [0,1,2,5,6,7]

gbr.fit(train[predictors], train[3])
preds = gbr.predict(test[predictors])

preds
print(mean_squared_error(test[3], preds))

test[3]

plt.scatter(date_pred,preds,alpha=0.8,color = 'c')
 plt.scatter(date_pred,test[3],alpha=0.8,color = 'r')

"""Cyan for the Predicted value
Red for the Orignal Data
"""

plt.scatter(date_pred,test[3]-preds)

train = d1.iloc[:-100]
test = d1.iloc[-100:]

predictors = [0,1,2,5,6,7]

from sklearn import linear_model
linear = linear_model.ARDRegression()

linear.fit(train[predictors], train[3])

preds = linear.predict(test[predictors])

preds = pd.DataFrame(preds)

from sklearn.metrics import mean_squared_error
print(mean_squared_error(test[3], preds))

plt.scatter(date_pred,test[3],alpha=0.8,color = 'r')
plt.scatter(date_pred,preds,alpha=0.8,color = 'c')
```