STOCK MARKET PREDICTION

MACHINE LEARNING

AYAAN,DIVIJ,PRANAV,PRITHVI

Problem Statement

Introduction:

The stock market is a dynamic and complex environment influenced by numerous factors. Investors face challenges in analyzing historical trends and predicting future stock values.

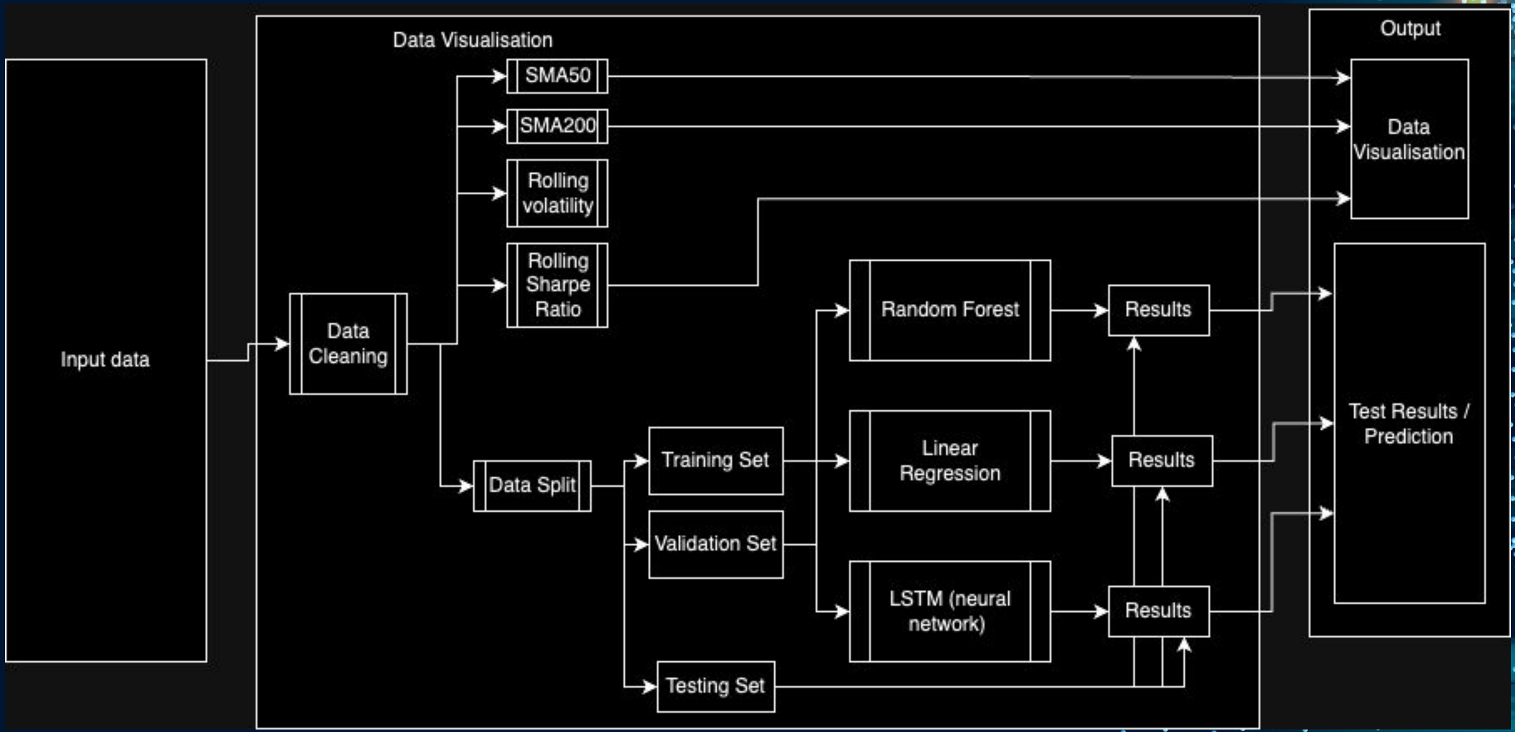
Features

Using the **yfinance** stock-market data library to predict future stock values

Validation and exploration of the data

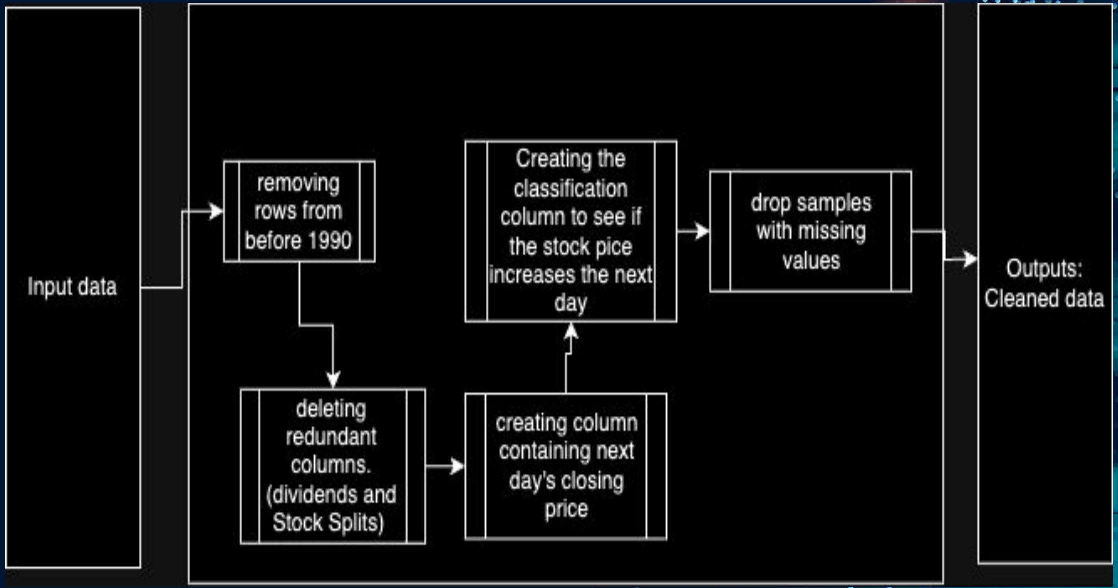
i. SMA_50
ii. SMA_200
iii. Rolling Volatility
iv. Rolling Sharpe Ratio

Architectural/Layout diagram



Data Download and Cleaning

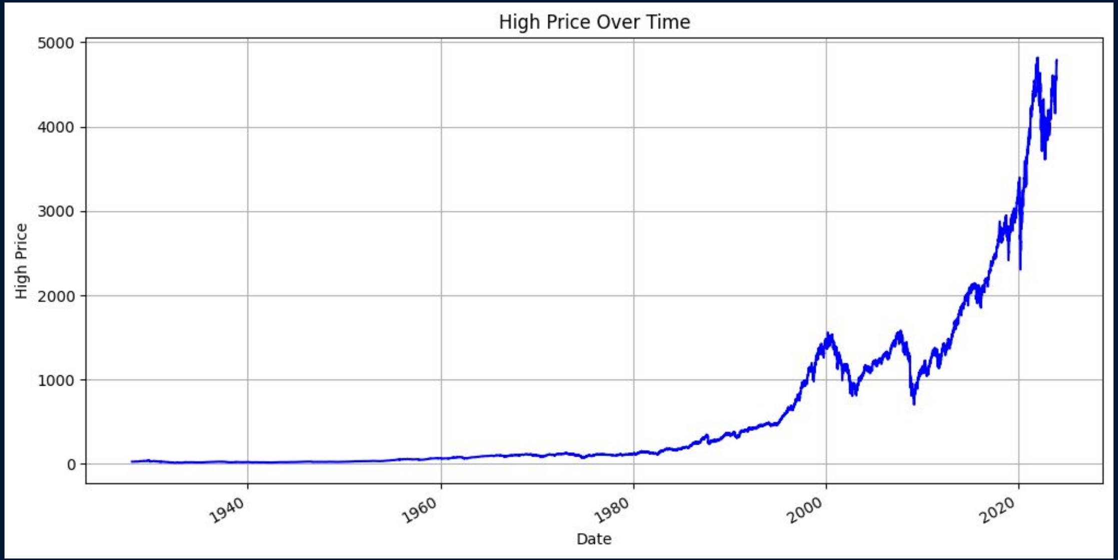
- Removal of unnecessary columns and missing values
- Creation of new columns
    - Tomorrow
    - Target
    - SMA50 and SMA200
- Making sure all types are correct



Data Visualization

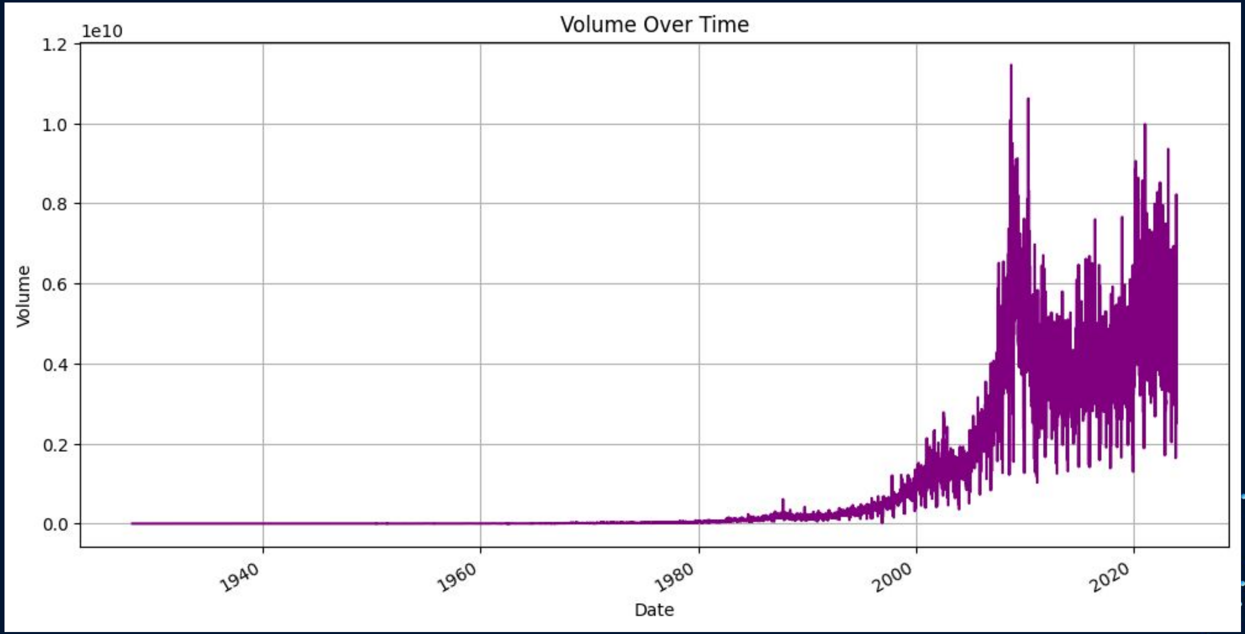
Data Visualization

Data Visualization

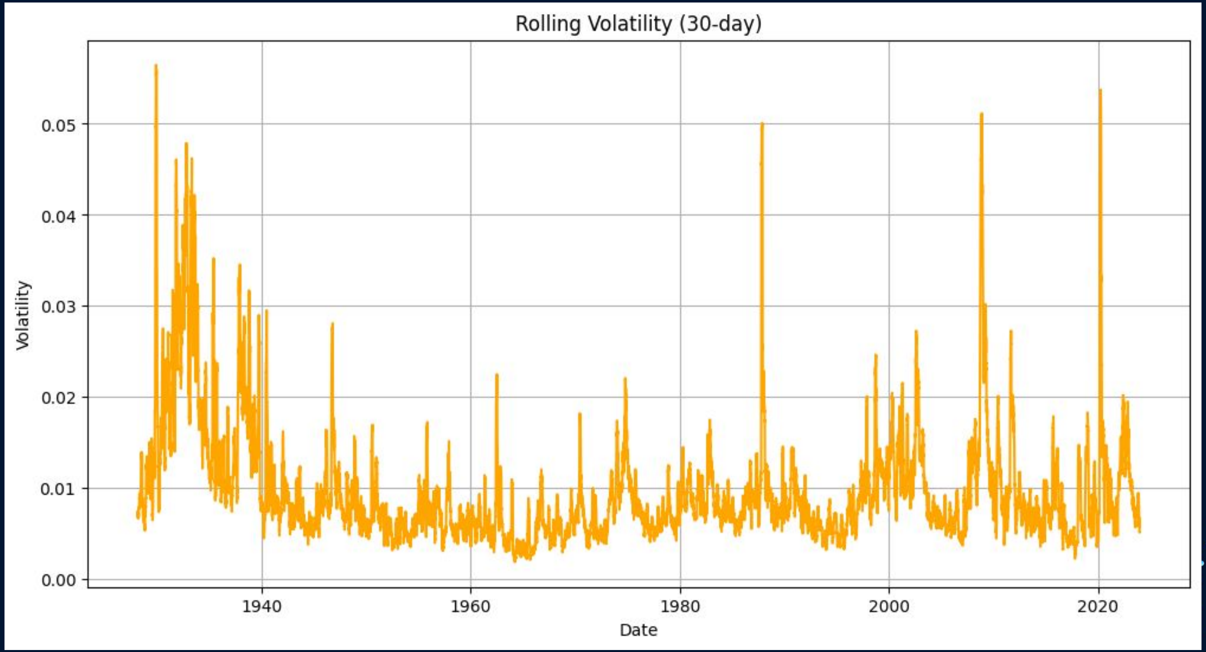


Data Visualization

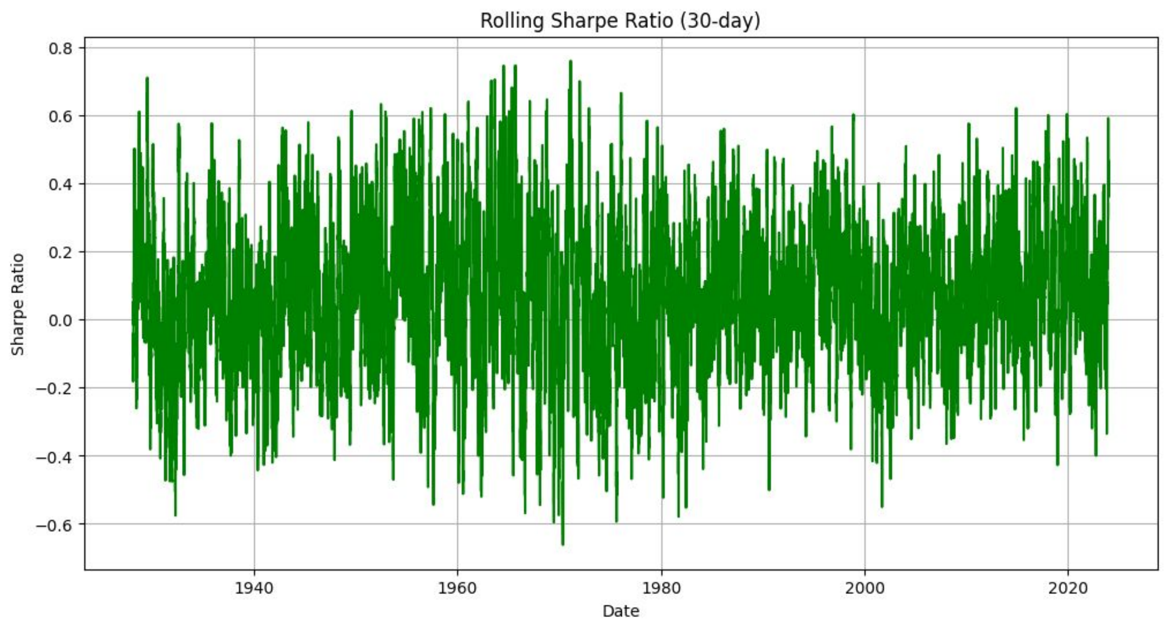
Data Visualization



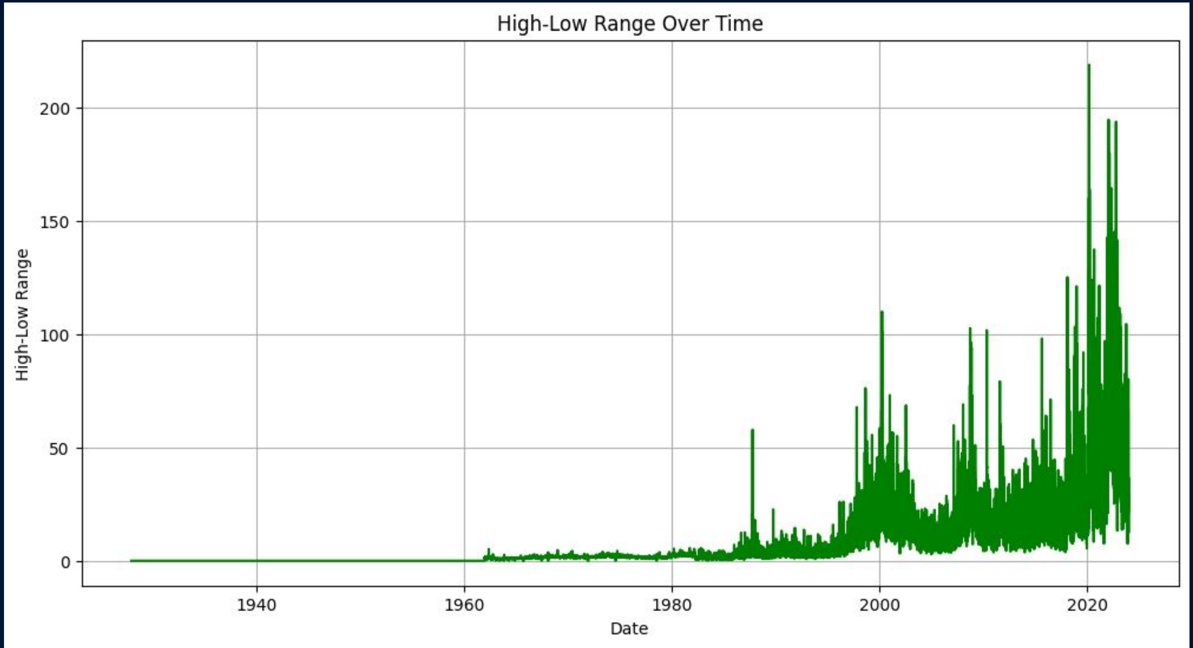
Data Visualization



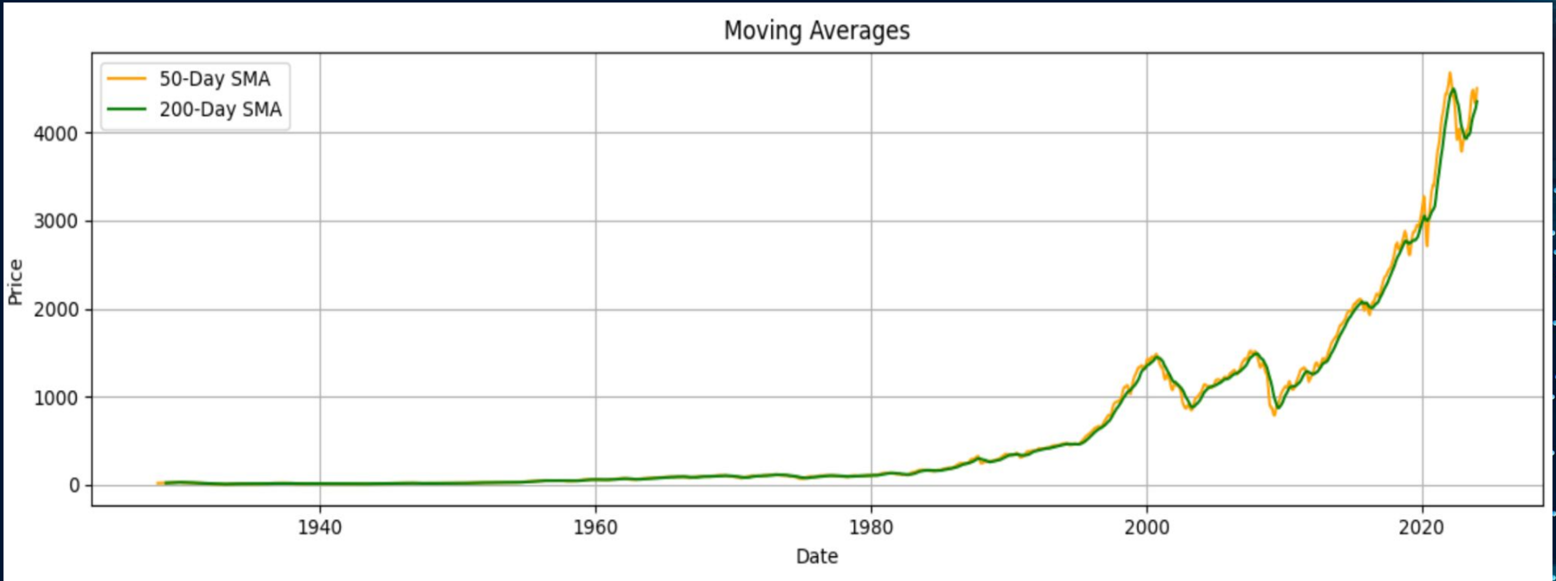
Data Visualization



Data Visualization

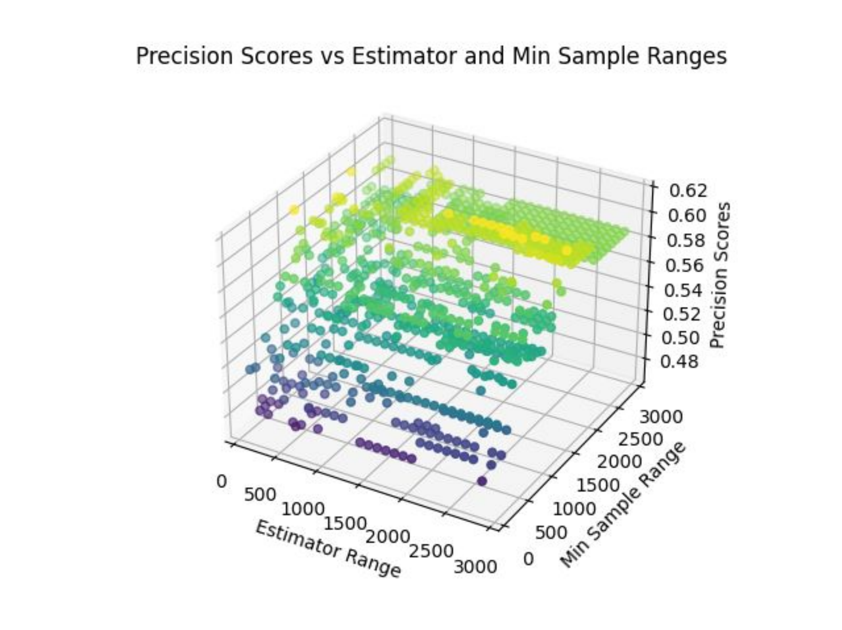


Data Visualization



Random Forest (Closing Price)

- With our initial model, accuracy was quite low
- The best precision score we could reach after parameter tuning was 0.625

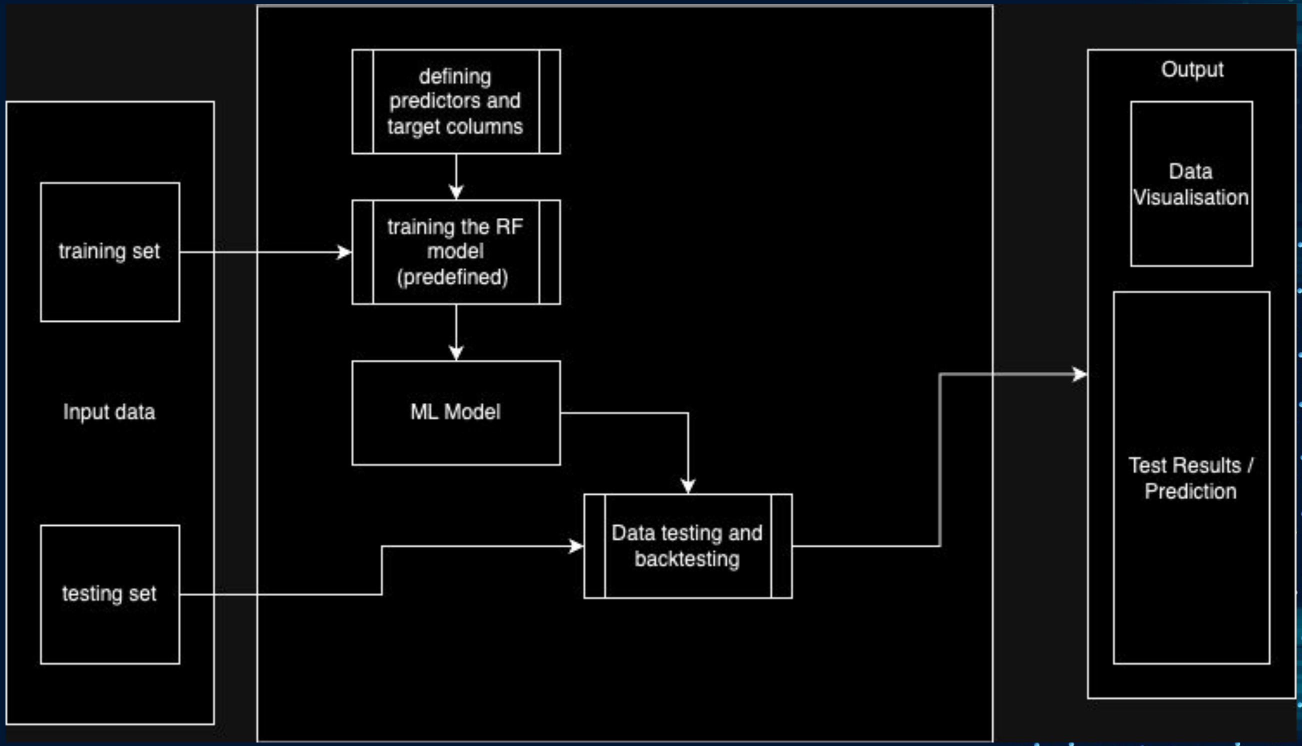


Backtesting

Initially we were only predicting for the last 100 days, but we want to be able to test across multiple years of data.

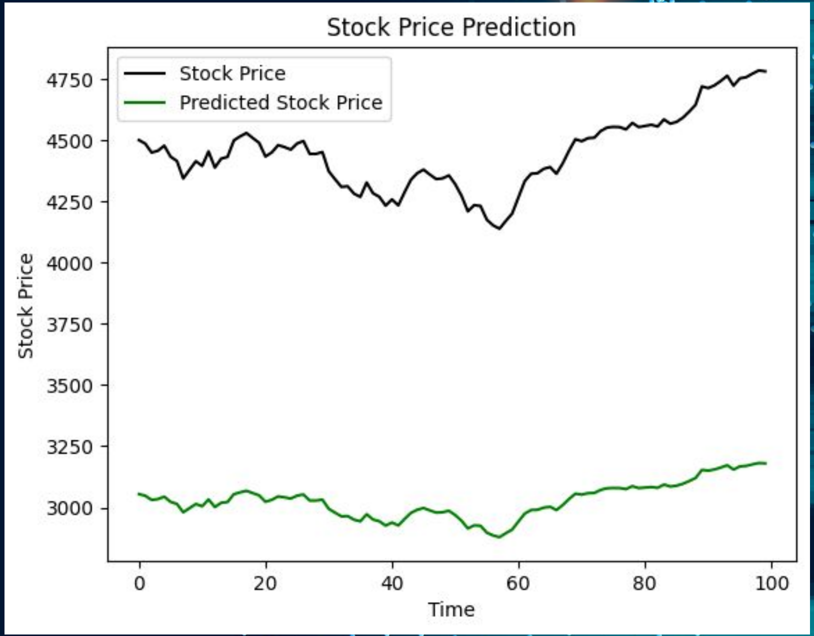
Therefore we implemented backtesting to better test our model for handling different situations.

Architectural/Layout diagram for Random Forest



LSTM

- Model gets prices wrong but trends right
- Model details:
    - Timesteps of 60
    - Epochs: 100
    - Batch size: 32
    - Dropout: 0.2
- Loss: 3.5688e-04



TEST CASES AND RESULTS

TEST CASE 1 : RANDOM FOREST (500 DECISION TREES)

[5]
```
#Looking at too much data would make the model more inaccurate.
#The market changes over time and the market shift would make old data worsen the model
#To prevent this, data from after 1990 is taken

specificDate = pd.to_datetime("1990-01-01").date()
sp500_downloaded = sp500_downloaded.loc[specificDate:].copy()

#Deleting these columns from the data frame as they are for individual stocks and not required to index
del sp500_downloaded["Dividends"]
del sp500_downloaded["Stock Splits"]

# creating a column that shows the next day's closing price
sp500_downloaded["Tomorrow"] = sp500_downloaded["Close"].shift(-1)
# creating the target column that uses the "Tomorrow" column to see if the price increases the next day
sp500_downloaded["Target"] = (sp500_downloaded["Tomorrow"] > sp500_downloaded["Close"]).astype(int)
```

Setting up the initial ML model

Using a random forest classifier to predict whether the stock price would increase.

Using random forest because they are more resistant to overfitting than most models, run quick and pick up non-linear tendencies in the data.

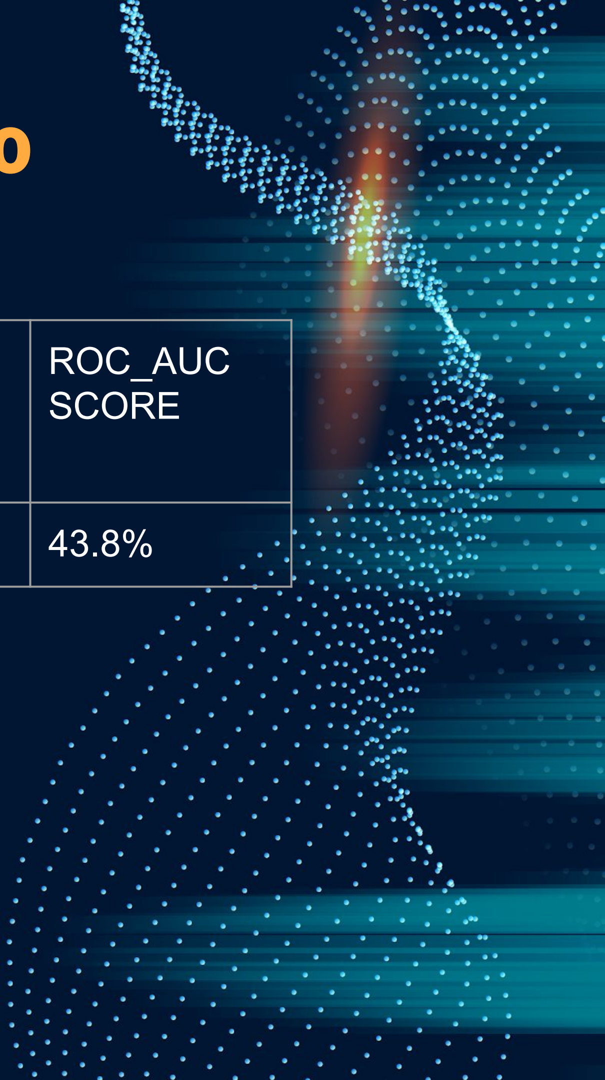
Picking up non-linearity is important for our data. A higher closing price than from a year ago doesn't mean the target will also be higher, the target could be 0, implying non-linearity.

[6]
```
# Creating a random forest with 500 decision trees ('n_estimators')
model = RandomForestClassifier(n_estimators=500, min_samples_split=100, random_state=1)

# Creating a test set of the last 100 items
train = sp500_downloaded.iloc[:-100]
test = sp500_downloaded.iloc[-100:]

predictors = ["Close", "Volume", "Open", "High", "Low"]
model.fit(train[predictors], train["Target"])
```

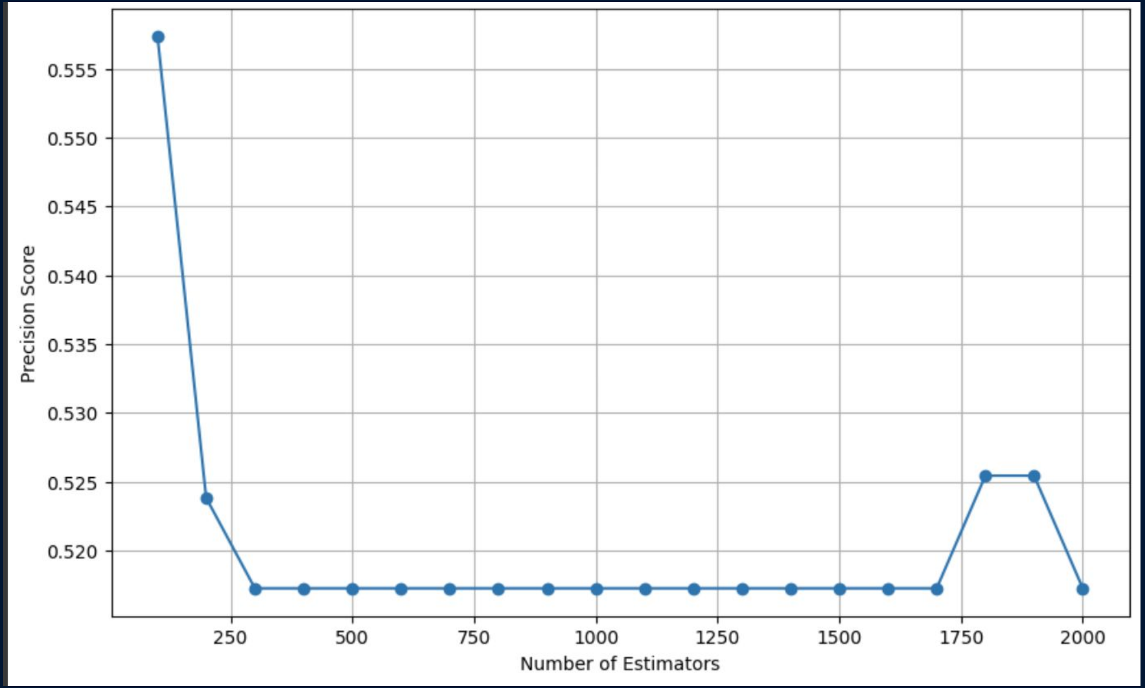
```
RandomForestClassifier
RandomForestClassifier(min_samples_split=100, n_estimators=500, random_state=1)
```

TEST CASE 1 : RANDOM FOREST (500 DECISION TREES)

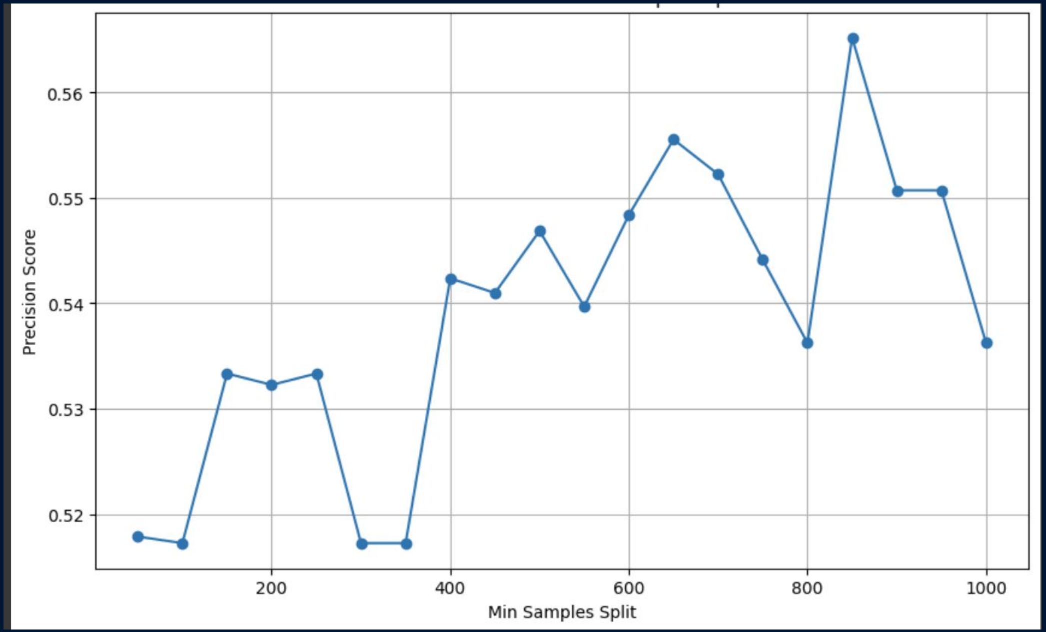
PARAMETERS	ACCURACY	PRECISION	RECALL	F1-SCORE	ROC_AUC SCORE
SCORE	45%	52%	52.6%	52.1%	43.8%



TEST CASE 2 : TUNING NUMBER OF ESTIMATORS



TEST CASE 3 : TUNING MINIMUM SAMPLES SPLIT



TEST CASE 4 : AFTER BACK TESTING

```
[15] def predict(train, test, predictors, model):
         model.fit(train[predictors], train["Target"])
         preds = model.predict(test[predictors])
         preds = pd.Series(preds, index=test.index, name="Predictions")
         combined = pd.concat([test["Target"], preds], axis=1)
         return combined

     def predict(train, test, predictors, model):
         # Fit the model on the training data
         model.fit(train[predictors], train["Target"])

         # Make predictions on the test data
         preds = model.predict(test[predictors])

         # Create a Series of predictions with index from the test data
         preds = pd.Series(preds, index=test.index, name="Predictions")

         # Combine the actual target values and the predictions into a DataFrame
         combined = pd.concat([test["Target"], preds], axis=1)
         return combined
```

```
def backtest(data, model, predictors, start=2500, step=250):
    all_predictions = []

    # Iterate through the data with a given start index and step size
    for i in range(start, data.shape[0], step):
        # Create train and test sets based on the current index and step size
        train = data.iloc[0:i].copy()
        test = data.iloc[i:(i+step)].copy()

        # Generate predictions for the current test set using the predict function
        predictions = predict(train, test, predictors, model)

        # Append the predictions to the list
        all_predictions.append(predictions)

    # Concatenate all the predictions into a single DataFrame and return it
    return pd.concat(all_predictions)
```

TEST CASE 4 : AFTER BACK TESTING

PARAMETERS	ACCURACY	PRECISION	RECALL	F1-SCORE	ROC_AUC SCORE
SCORE	49%	54%	42%	46%	49%

TEST CASE 5 : 2008 MARKET CRASH (SMA_200)

```python
#2008-2010 Market Crash

from sklearn.metrics import accuracy_score, precision_score


df_2008_2010 = sp500_downloaded[(sp500_downloaded.index.year >= 2008) & (sp500_downloaded.index.year <= 2010)]

# Drop samples with missing values
df_2008_2009 = df_2008_2010.dropna()
# Splitting data into train and test sets
train_size = int(0.8 * len(df_2008_2010))  # 80% train, 20% test
train = df_2008_2010[:train_size]
test = df_2008_2010[train_size:]

# Drop samples with missing values
train = train.dropna()
test = test.dropna()

predictors = ["Close", "Volume", "Open", "High", "Low","SMA_200"]

# Fit the model to the training data
model.fit(train[predictors], train["Target_SMA200"])

# Predict using the model
predictions = model.predict(test[predictors])

# Evaluate the model
accuracy = accuracy_score(test["Target_SMA200"], predictions)
print("Accuracy:", accuracy)

precision = precision_score(test["Target_SMA200"], predictions)
print("Precision:", precision)
```

```
Accuracy: 0.7171052631578947
Precision: 0.872
```

TEST CASE 6 : 2008 MARKET CRASH (TARGET)

```
#2008-2010 Market Crash

from sklearn.metrics import accuracy_score, precision_score


df_2008_2010 = sp500_downloaded[(sp500_downloaded.index.year >= 2008) & (sp500_downloaded.index.year <= 2010)]

# Drop samples with missing values
df_2008_2009 = df_2008_2010.dropna()
# Splitting data into train and test sets
train_size = int(0.8 * len(df_2008_2010))  # 80% train, 20% test
train = df_2008_2010[:train_size]
test = df_2008_2010[train_size:]

# Drop samples with missing values
train = train.dropna()
test = test.dropna()

predictors = ["Close", "Volume", "Open", "High", "Low","SMA_200"]

# Fit the model to the training data
model.fit(train[predictors], train["Target"])

# Predict using the model
predictions = model.predict(test[predictors])

# Evaluate the model
accuracy = accuracy_score(test["Target"], predictions)
print("Accuracy:", accuracy)

precision = precision_score(test["Target"], predictions)
print("Precision:", precision)
```

```
Accuracy: 0.5328947368421053
Precision: 0.559322033898305
```

BIBLIOGRAPHY

- https://ieeexplore.ieee.org/document/8703332
- https://www.diva-portal.org/smash/get/diva2:1672304/FULLTEXT01.pdf
- https://ceur-ws.org/Vol-3283/Paper86.pdf
- https://www.sciencedirect.com/science/article/pii/S187705092030792 4