Predicting SP500 with Random Forest

In [ ]:

```
import yfinance as yf
import pandas as pd
from sklearn.ensemble import RandomForestClassifier
from xgboost import XGBClassifier, plot_importance
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, con
import matplotlib.pyplot as plt
```

Importing SP500 Index with the maximum historical data

In [42]:

```
sp500 = yf.Ticker("^GSPC")
```

In [43]:

```
sp500 = sp500.history(period="max")
```

Dropping unnecessary columns (Dividends and Stock Splits are for specific Stocks, not index)

In [44]:

```
sp500 = sp500.drop(['Dividends','Stock Splits'],axis=1)
```

Feature Engineering:

- 'Tomorrow' column will reflect the close price of the following day.
- 'Target' is a boolean value that indicates if the price was higher (1) or lower (0) than yesterday

In [45]:

```
sp500['Tomorrow'] = sp500['Close'].shift(-1)
```

In [46]:

```
sp500['Target'] = (sp500['Tomorrow'] > sp500['Close']).astype(int)
```

Filtering only data after Jan 1st, 1990

In [47]:

```
sp500 = sp500.loc['1990-01-01':].copy()
```

Instantiate **Random Forest Model**

In [48]:

```
model = RandomForestClassifier(n_estimators=100,min_samples_split=100,random_state=42)
```

Fitting the 1st Version of the Model

Train dataset will be all data with exception of the last 100 days, which will be used on the **Test** dataset

In [49]:

```
train = sp500.iloc[:-100]
test = sp500.iloc[-100:]

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

Out[49]:

```
▾               RandomForestClassifier
RandomForestClassifier(min_samples_split=100, random_state=42)
```

In [50]:

```
y_pred = model.predict(test[predictors])
```

In [51]:

```
y_pred = pd.Series(y_pred,index=test.index)
```

In [52]:

```
precision_score(test['Target'],y_pred)
```

Out[52]:

```
0.5483870967741935
```

This result means that we predicted approximately 55% of days in which the SP500 would have a price higher than the day before.

Improving the Model

To improve the model, lets build these functions to perform backtest

In [53]:

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

In [54]:

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

    for i in range(start, data.shape[0],step):
        train = data.iloc[0:i].copy()
        test = data.iloc[i:(i+step)].copy()
        predictions = predict(train,test,predictors,model)
        all_predictions.append(predictions)
    return pd.concat(all_predictions)
```

In [55]:

```
predictions = backtest(sp500,model,predictors)
```

In [56]:

```
predictions['Predictions'].value_counts(normalize=True)
```

Out[56]:

```
Predictions
0    0.578361
1    0.421639
Name: proportion, dtype: float64
```

In [57]:

```
precision_score(predictions['Target'],predictions['Predictions'])
```

Out[57]:

```
0.5217729393468118
```

In [58]:

```
predictions['Target'].value_counts(normalize=True)
```

Out[58]:

```
Target
1    0.534754
0    0.465246
Name: proportion, dtype: float64
```

Adding other predictors

In [59]:

```
#Rolling Averages: 2, 5, 60, 250 and 1000 days
#Trend: Number of days that SP500 went up
horizons = [2,5,60,250,1000]
new_predictors = []

for i in horizons:
    rolling_averages = sp500.rolling(i).mean()
    ratio_column = f'Close_Ratio_{i}'
    sp500[ratio_column] = sp500['Close'] / rolling_averages['Close']

    trend_column = f'Trend_{i}'
    sp500[trend_column] = sp500.shift(1).rolling(i).sum()['Target']

    new_predictors += [ratio_column,trend_column]
```

In [60]:

```
sp500 = sp500.dropna()
```

In [61]:

```
model = RandomForestClassifier(n_estimators=200,min_samples_split=50,random_state =42)
```

In [62]:

```
def predict_new(train,test,predictors,model):
    model.fit(train[predictors],train['Target'])
    pred = model.predict_proba(test[predictors])[:,1]
    pred[pred >= 0.6] = 1
    pred[pred < 0.6] = 0
    pred = pd.Series(pred,index=test.index,name='Predictions')
    combined = pd.concat([test['Target'],pred],axis=1)
    return combined
```

In [63]:

```
new_predictions = backtest(sp500,model,new_predictors)
```

In [64]:

```
new_predictions['Predictions'].value_counts(normalize=True)
```

Out[64]:

```
Predictions
1    0.706021
0    0.293979
Name: proportion, dtype: float64
```

In [65]:

```
new_predictions['Target'].value_counts(normalize=True)
```

Out[65]:

```
Target
1    0.544617
0    0.455383
Name: proportion, dtype: float64
```

In [67]:

```
precision_score(new_predictions['Target'],new_predictions['Predictions'])
```

Out[67]:

```
0.5533333333333333
```

Conclusion

This notebook was intended to showcase a model that had the purpose of trying to predict if the SP500 Index would increase with minimal information input.

- The raw model had a precision of **54.8%**, predicting that in **42.0%** of the days the index would increase against the day before. This is slightly worst than the actual number of days that the index went up (42.2%).
- With the Backtest and the improved model, our precision increased to **55.3%** and the model was able to predict **53.5%** of the days where the index went up.
- This is just the first step of the model. An improved version would have to correlate with foreign indices and other market situation, but just with minimal information and the right parameters we are already better than investing without testing the market.