```
!pip install ta
!pip install xgboost
!pip install keras
!pip install tensorflow
import warnings
warnings.filterwarnings('ignore')
```

```
Requirement already satisfied: ta in d:\new folder (3)\lib\site-packages (0.10.2)
Requirement already satisfied: numpy in c:\users\rushi\appdata\roaming\python\python310\site-packages (from ta) (1.22.3)
Requirement already satisfied: pandas in c:\users\rushi\appdata\roaming\python\python310\site-packages (from ta) (1.4.2)
Requirement already satisfied: python-dateutil>=2.8.1 in c:\users\rushi\appdata\roaming\python\python310\site-packages (from pandas->ta) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in c:\users\rushi\appdata\roaming\python\python310\site-packages (from pandas->ta) (2022.1)
Requirement already satisfied: six>=1.5 in c:\users\rushi\appdata\roaming\python\python310\site-packages (from python-dateutil>=2.8.1->pandas->ta) (1.16.0)

WARNING: There was an error checking the latest version of pip.

Requirement already satisfied: xgboost in d:\new folder (3)\lib\site-packages (1.7.6)
Requirement already satisfied: numpy in c:\users\rushi\appdata\roaming\python\python310\site-packages (from xgboost) (1.22.3)
Requirement already satisfied: scipy in c:\users\rushi\appdata\roaming\python\python310\site-packages (from xgboost) (1.8.0)

WARNING: There was an error checking the latest version of pip.

Requirement already satisfied: keras in d:\new folder (3)\lib\site-packages (2.13.1)

WARNING: There was an error checking the latest version of pip.

Collecting tensorflow
  Downloading tensorflow-2.13.0-cp310-cp310-win_amd64.whl (1.9 kB)
Collecting tensorflow-intel==2.13.0
  Downloading tensorflow_intel-2.13.0-cp310-cp310-win_amd64.whl (276.5 MB)
     -------------------------------------- 276.5/276.5 MB 2.4 MB/s eta 0:00:00
Collecting grpcio<2.0,>=1.24.3
  Downloading grpcio-1.56.0-cp310-cp310-win_amd64.whl (4.2 MB)
     ---------------------------------------- 4.2/4.2 MB 13.5 MB/s eta 0:00:00
Collecting google-pasta>=0.1.1
  Using cached google_pasta-0.2.0-py3-none-any.whl (57 kB)
```

```
Requirement already satisfied: typing-extensions<4.6.0,>=3.6.6 in d:\
new folder (3)\lib\site-packages (from tensorflow-intel==2.13.0-
>tensorflow) (4.4.0)
Collecting flatbuffers>=23.1.21
  Using cached flatbuffers-23.5.26-py2.py3-none-any.whl (26 kB)
Collecting tensorboard<2.14,>=2.13
  Downloading tensorboard-2.13.0-py3-none-any.whl (5.6 MB)
     ---------------------------------------- 5.6/5.6 MB 11.9 MB/s eta
0:00:00
Requirement already satisfied: wrapt>=1.11.0 in d:\new folder (3)\lib\
site-packages (from tensorflow-intel==2.13.0->tensorflow) (1.14.1)
Collecting tensorflow-io-gcs-filesystem>=0.23.1
  Using cached tensorflow_io_gcs_filesystem-0.31.0-cp310-cp310-
win_amd64.whl (1.5 MB)
Requirement already satisfied: numpy<=1.24.3,>=1.22 in c:\users\rushi\
appdata\roaming\python\python310\site-packages (from tensorflow-
intel==2.13.0->tensorflow) (1.22.3)
Collecting astunparse>=1.6.0
  Using cached astunparse-1.6.3-py2.py3-none-any.whl (12 kB)
Collecting termcolor>=1.1.0
  Using cached termcolor-2.3.0-py3-none-any.whl (6.9 kB)
Collecting tensorflow-estimator<2.14,>=2.13.0
  Downloading tensorflow_estimator-2.13.0-py2.py3-none-any.whl (440
kB)
     -------------------------------------- 440.8/440.8 kB 6.8 MB/s
eta 0:00:00
Collecting absl-py>=1.0.0
  Using cached absl_py-1.4.0-py3-none-any.whl (126 kB)
Collecting protobuf!=4.21.0,!=4.21.1,!=4.21.2,!=4.21.3,!=4.21.4,!
=4.21.5,<5.0.0dev,>=3.20.3
  Downloading protobuf-4.23.3-cp310-abi3-win_amd64.whl (422 kB)
     -------------------------------------- 422.5/422.5 kB 6.5 MB/s
eta 0:00:00
Requirement already satisfied: keras<2.14,>=2.13.1 in d:\new folder
(3)\lib\site-packages (from tensorflow-intel==2.13.0->tensorflow)
(2.13.1)
Collecting gast<=0.4.0,>=0.2.1
  Using cached gast-0.4.0-py3-none-any.whl (9.8 kB)
Requirement already satisfied: setuptools in d:\new folder (3)\lib\
site-packages (from tensorflow-intel==2.13.0->tensorflow) (65.6.3)
Collecting libclang>=13.0.0
  Using cached libclang-16.0.0-py2.py3-none-win_amd64.whl (24.4 MB)
Collecting opt-einsum>=2.3.2
  Using cached opt_einsum-3.3.0-py3-none-any.whl (65 kB)
Requirement already satisfied: six>=1.12.0 in c:\users\rushi\appdata\
roaming\python\python310\site-packages (from tensorflow-intel==2.13.0-
>tensorflow) (1.16.0)
Requirement already satisfied: h5py>=2.9.0 in d:\new folder (3)\lib\
site-packages (from tensorflow-intel==2.13.0->tensorflow) (3.7.0)
```

```
Requirement already satisfied: packaging in c:\users\rushi\appdata\
roaming\python\python310\site-packages (from tensorflow-intel==2.13.0-
>tensorflow) (21.3)
Requirement already satisfied: wheel<1.0,>=0.23.0 in d:\new folder
(3)\lib\site-packages (from astunparse>=1.6.0->tensorflow-
intel==2.13.0->tensorflow) (0.38.4)
Requirement already satisfied: markdown>=2.6.8 in d:\new folder (3)\
lib\site-packages (from tensorboard<2.14,>=2.13->tensorflow-
intel==2.13.0->tensorflow) (3.4.1)
Collecting tensorboard-data-server<0.8.0,>=0.7.0
  Downloading tensorboard_data_server-0.7.1-py3-none-any.whl (2.4 kB)
Requirement already satisfied: requests<3,>=2.21.0 in c:\users\rushi\
appdata\roaming\python\python310\site-packages (from
tensorboard<2.14,>=2.13->tensorflow-intel==2.13.0->tensorflow)
(2.28.1)
Collecting google-auth-oauthlib<1.1,>=0.5
  Using cached google_auth_oauthlib-1.0.0-py2.py3-none-any.whl (18 kB)
Requirement already satisfied: werkzeug>=1.0.1 in d:\new folder (3)\
lib\site-packages (from tensorboard<2.14,>=2.13->tensorflow-
intel==2.13.0->tensorflow) (2.2.2)
Collecting google-auth<3,>=1.6.3
  Downloading google_auth-2.21.0-py2.py3-none-any.whl (182 kB)
     ------------------------------------ 182.1/182.1 kB 11.5 MB/s
eta 0:00:00
Requirement already satisfied: pyparsing!=3.0.5,>=2.0.2 in c:\users\
rushi\appdata\roaming\python\python310\site-packages (from packaging-
>tensorflow-intel==2.13.0->tensorflow) (3.0.9)
Requirement already satisfied: pyasn1-modules>=0.2.1 in d:\new folder
(3)\lib\site-packages (from google-auth<3,>=1.6.3-
>tensorboard<2.14,>=2.13->tensorflow-intel==2.13.0->tensorflow)
(0.2.8)
Requirement already satisfied: urllib3<2.0 in c:\users\rushi\appdata\
roaming\python\python310\site-packages (from google-auth<3,>=1.6.3-
>tensorboard<2.14,>=2.13->tensorflow-intel==2.13.0->tensorflow)
(1.26.12)
Collecting cachetools<6.0,>=2.0.0
  Using cached cachetools-5.3.1-py3-none-any.whl (9.3 kB)
Collecting rsa<5,>=3.1.4
  Using cached rsa-4.9-py3-none-any.whl (34 kB)
Collecting requests-oauthlib>=0.7.0
  Using cached requests_oauthlib-1.3.1-py2.py3-none-any.whl (23 kB)
Requirement already satisfied: charset-normalizer<3,>=2 in c:\users\
rushi\appdata\roaming\python\python310\site-packages (from
requests<3,>=2.21.0->tensorboard<2.14,>=2.13->tensorflow-
intel==2.13.0->tensorflow) (2.1.1)
Requirement already satisfied: idna<4,>=2.5 in c:\users\rushi\appdata\
roaming\python\python310\site-packages (from requests<3,>=2.21.0-
>tensorboard<2.14,>=2.13->tensorflow-intel==2.13.0->tensorflow) (3.4)
Requirement already satisfied: certifi>=2017.4.17 in c:\users\rushi\
```

```
appdata\roaming\python\python310\site-packages (from
requests<3,>=2.21.0->tensorboard<2.14,>=2.13->tensorflow-
intel==2.13.0->tensorflow) (2022.9.24)
Requirement already satisfied: MarkupSafe>=2.1.1 in c:\users\rushi\
appdata\roaming\python\python310\site-packages (from werkzeug>=1.0.1-
>tensorboard<2.14,>=2.13->tensorflow-intel==2.13.0->tensorflow)
(2.1.1)
Requirement already satisfied: pyasn1<0.5.0,>=0.4.6 in d:\new folder
(3)\lib\site-packages (from pyasn1-modules>=0.2.1->google-
auth<3,>=1.6.3->tensorboard<2.14,>=2.13->tensorflow-intel==2.13.0-
>tensorflow) (0.4.8)
Collecting oauthlib>=3.0.0
  Using cached oauthlib-3.2.2-py3-none-any.whl (151 kB)
Installing collected packages: libclang, flatbuffers, termcolor,
tensorflow-io-gcs-filesystem, tensorflow-estimator, tensorboard-data-
server, rsa, protobuf, opt-einsum, oauthlib, grpcio, google-pasta,
gast, cachetools, astunparse, absl-py, requests-oauthlib, google-auth,
google-auth-oauthlib, tensorboard, tensorflow-intel, tensorflow
Successfully installed absl-py-1.4.0 astunparse-1.6.3 cachetools-5.3.1
flatbuffers-23.5.26 gast-0.4.0 google-auth-2.21.0 google-auth-
oauthlib-1.0.0 google-pasta-0.2.0 grpcio-1.56.0 libclang-16.0.0
oauthlib-3.2.2 opt-einsum-3.3.0 protobuf-4.23.3 requests-oauthlib-
1.3.1 rsa-4.9 tensorboard-2.13.0 tensorboard-data-server-0.7.1
tensorflow-2.13.0 tensorflow-estimator-2.13.0 tensorflow-intel-2.13.0
tensorflow-io-gcs-filesystem-0.31.0 termcolor-2.3.0

WARNING: There was an error checking the latest version of pip.
```

```python
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score
from ta import add_all_ta_features
from ta.utils import dropna
from sklearn.metrics import precision_score, recall_score, f1_score,
roc_auc_score
import warnings

df = pd.read_csv(r'C:\Users\rushi\Downloads\BTC-USD 2021.csv')
warnings.filterwarnings('ignore')

df = pd.read_csv(r'C:\Users\rushi\Downloads\BTC-USD 2021.csv')
print(df.head())
```

```
         Date          Open          High           Low         Close
\
0  2021-01-01  28994.009766  29600.626953  28803.585938  29374.152344

1  2021-01-02  29376.455078  33155.117188  29091.181641  32127.267578
```

```
2  2021-01-03  32129.408203  34608.558594  32052.316406  32782.023438

3  2021-01-04  32810.949219  33440.218750  28722.755859  31971.914063

4  2021-01-05  31977.041016  34437.589844  30221.187500  33992.429688


      Adj Close       Volume
0  29374.152344  40730301359
1  32127.267578  67865420765
2  32782.023438  78665235202
3  31971.914063  81163475344
4  33992.429688  67547324782
```

```
from sklearn.preprocessing import MinMaxScaler
# Handling missing values
df.dropna(inplace=True)

# Below code trains a Random Forest classifier model to predict
whether the return will be positive or negative,
# using time series cross-validation. It then applies a simple trading
strategy based on these predictions
# and tracks the capital over time. At each step, the root mean
squared error (RMSE) between the model's
# predicted probabilities and the actual outcomes is calculated and
printed. It also plots the predicted return
# probabilities and the cumulative profit/loss over time.

import pandas as pd
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report, mean_squared_error
from sklearn.model_selection import TimeSeriesSplit, train_test_split
from sklearn.metrics import accuracy_score
from math import sqrt
import matplotlib.pyplot as plt
import warnings
from sklearn.exceptions import DataConversionWarning
warnings.filterwarnings(action='ignore', category=UserWarning)

# Load the data
df = pd.read_csv(r'C:\Users\rushi\Downloads\BTC-USD 2021.csv')

df['Date'] = pd.to_datetime(df['Date'])

# Sort the values by 'Date'
df = df.sort_values('Date')

# Calculate the 'Return'
df['Return'] = df['Close'].pct_change()

# Define the target variable
```

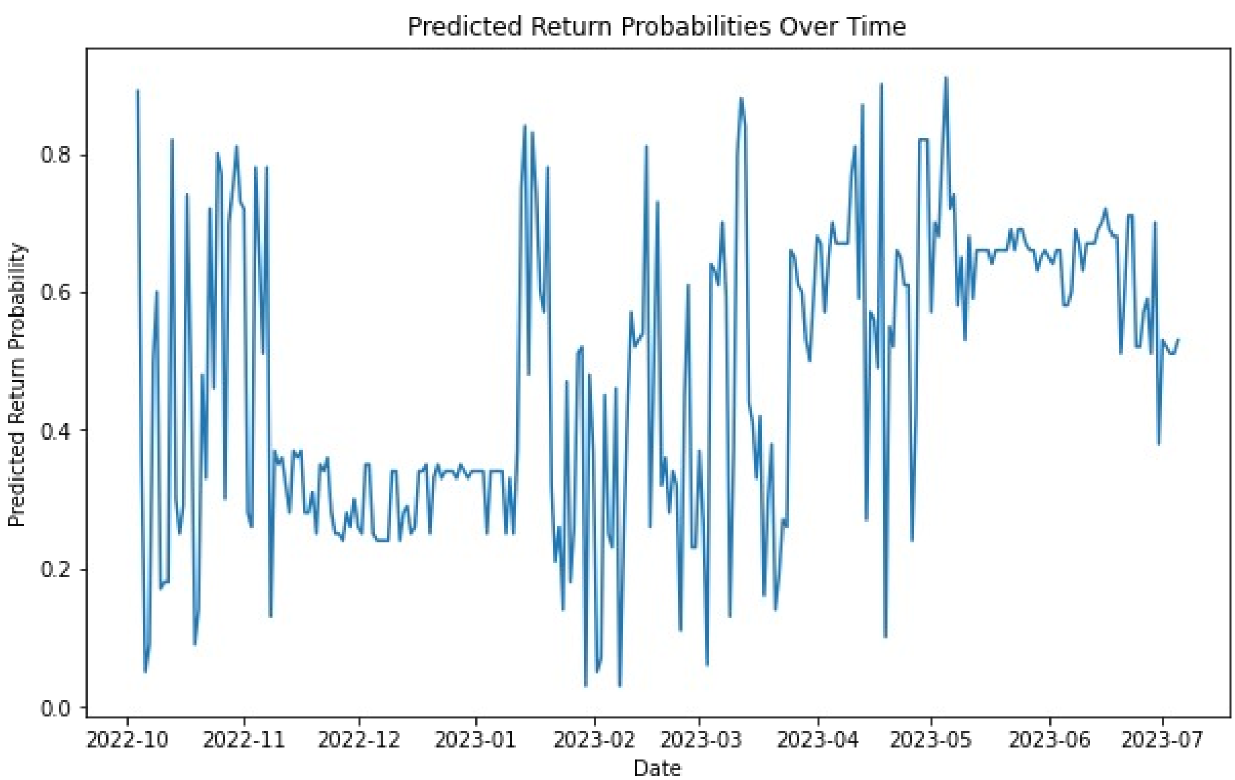
```
y = df['Return']

X = df.drop(columns=['Date', 'Return', 'Close'])

# Create a binary target variable
y_binary = (y > 0).astype(int)

# Initialize the model
model = RandomForestClassifier(n_estimators=100, random_state=42)

# Split the data into training (70%) and test (30%) sets
X_full_train, X_test, y_full_train, y_test = train_test_split(X,
y_binary, test_size=0.3, shuffle=False)

tscv = TimeSeriesSplit(n_splits=5)

# List to store final capitals for each training window
final_capitals = []
training_window_sizes = [0.5, 0.6, 0.7, 0.8]

for train_index, val_index in tscv.split(X_full_train):
    X_train, X_val = X_full_train.iloc[train_index],
X_full_train.iloc[val_index]
    y_train, y_val = y_full_train.iloc[train_index],
y_full_train.iloc[val_index]

    # Train the model
    model.fit(X_train, y_train)

    # Make predictions on the validation set
    y_pred = model.predict(X_val)
    y_pred_proba = model.predict_proba(X_val)[:, 1]  # We only need
the probability for the class '1'

    # Calculate the RMSE
    rmse = sqrt(mean_squared_error(y_val, y_pred_proba))
    print('RMSE:', rmse)

    # Calculate the metrics
    print(classification_report(y_val, y_pred))

    # Calculate accuracy
    acc = accuracy_score(y_val, y_pred)
    print(f"Validation Accuracy: {acc}")

    # Implement the trading strategy on the validation set
    capitals_window = [10000000]  # Starting with 10 million capital
    for i in range(len(y_val)):
        y_pred_i = model.predict([X_val.iloc[i]])[0]
        if y_pred_i == 1:
```

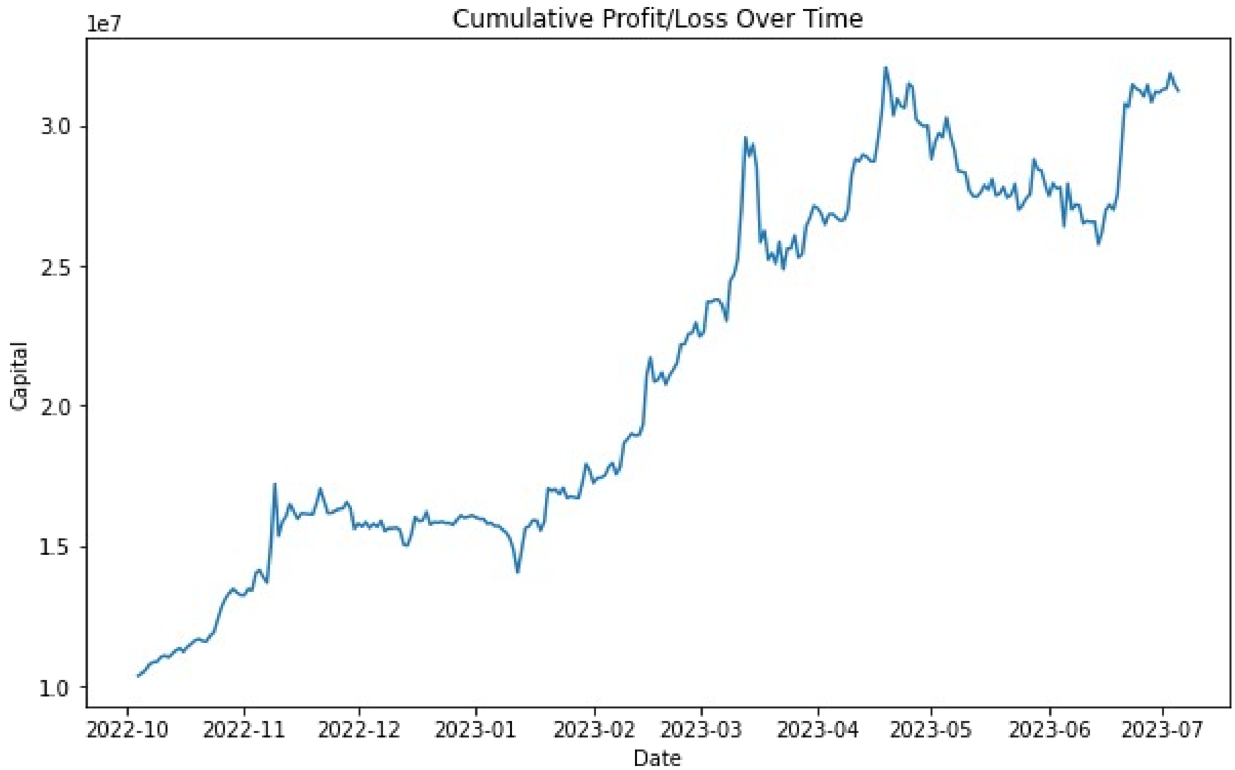
```
            capitals_window.append(capitals_window[-1] * (1 +
df['Return'].iloc[val_index[i]]))
        else:
            capitals_window.append(capitals_window[-1] * (1 -
df['Return'].iloc[val_index[i]]))

    # Store final capital for this training window
    final_capitals.append(capitals_window[-1])
    training_window_sizes.append(len(train_index) / len(X_full_train)
* 100)  # as a percentage

# After cross-validation, evaluate the model on the test set
model.fit(X_full_train, y_full_train)
y_test_pred = model.predict(X_test)
print("Test Classification Report:")
print(classification_report(y_test, y_test_pred))

# Implement a basic trading strategy
capitals = [10000000]  # Starting with 10 million capital
dates = df['Date'].iloc[y_test.index].tolist()

for i in range(len(y_test_pred)):
    if y_test_pred[i] == 1:  # If the model predicts the price will go
up
        # Buy at the close price
        capitals.append(capitals[-1] * (1 +
df['Return'].iloc[y_test.index[i]]))
    else:  # If the model predicts the price will go down
        # Sell at the close price
        capitals.append(capitals[-1] * (1 -
df['Return'].iloc[y_test.index[i]]))

# Calculate the total profit/loss
total_profit = capitals[-1] - capitals[0]
print(f"Total Profit: ${total_profit}")

# Create a DataFrame for visualization
viz_df = pd.DataFrame({
    'Date': dates,
    'Predicted_Return_Proba': model.predict_proba(X_test)[:, 1],
    'Capital': capitals[1:]
})


# Plotting predicted return probabilities
plt.figure(figsize=(10, 6))
plt.plot(viz_df['Date'], viz_df['Predicted_Return_Proba'])
plt.title('Predicted Return Probabilities Over Time')
plt.xlabel('Date')
plt.ylabel('Predicted Return Probability')
```

```
plt.show()

# Plotting cumulative profit/loss curve
plt.figure(figsize=(10, 6))
plt.plot(viz_df['Date'], viz_df['Capital'])
plt.title('Cumulative Profit/Loss Over Time')
plt.xlabel('Date')
plt.ylabel('Capital')
plt.show()
```

```
RMSE: 0.46035773620415993
              precision    recall  f1-score   support

           0       0.57      0.91      0.70        53
           1       0.77      0.32      0.45        53

    accuracy                           0.61       106
   macro avg       0.67      0.61      0.58       106
weighted avg       0.67      0.61      0.58       106

Validation Accuracy: 0.6132075471698113
RMSE: 0.49445414919927344
              precision    recall  f1-score   support

           0       0.54      0.94      0.69        51
           1       0.82      0.25      0.39        55

    accuracy                           0.58       106
   macro avg       0.68      0.60      0.54       106
weighted avg       0.69      0.58      0.53       106

Validation Accuracy: 0.5849056603773585
RMSE: 0.4083156843830611
              precision    recall  f1-score   support

           0       0.76      0.85      0.80        55
           1       0.82      0.71      0.76        51

    accuracy                           0.78       106
   macro avg       0.79      0.78      0.78       106
weighted avg       0.79      0.78      0.78       106

Validation Accuracy: 0.7830188679245284
RMSE: 0.47137694726735874
              precision    recall  f1-score   support

           0       0.65      0.89      0.75        54
           1       0.81      0.50      0.62        52

    accuracy                           0.70       106
```

```
   macro avg       0.73      0.69      0.68       106
weighted avg       0.73      0.70      0.69       106

Validation Accuracy: 0.6981132075471698
RMSE: 0.5260093621377891
              precision    recall  f1-score   support

           0       0.57      0.47      0.51        58
           1       0.47      0.58      0.52        48

    accuracy                           0.52       106
   macro avg       0.52      0.52      0.52       106
weighted avg       0.53      0.52      0.52       106

Validation Accuracy: 0.5188679245283019
Test Classification Report:
              precision    recall  f1-score   support

           0       0.58      0.57      0.58       142
           1       0.55      0.56      0.56       133

    accuracy                           0.57       275
   macro avg       0.57      0.57      0.57       275
weighted avg       0.57      0.57      0.57       275

Total Profit: $21237145.50199619
```






```
# below code Implement the "Long or Short" strategy with  different
training window sizes

# import pandas as pd
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report, mean_squared_error
import matplotlib.pyplot as plt

# Load the data
df = pd.read_csv(r'C:\Users\rushi\Downloads\BTC-USD 2021.csv')

df['Date'] = pd.to_datetime(df['Date'])
df = df.sort_values('Date')
df['Return'] = df['Close'].pct_change()
y = df['Return']
X = df.drop(columns=['Date', 'Return', 'Close'])
y_binary = (y > 0).astype(int)
model = RandomForestClassifier(n_estimators=100, random_state=42)

# Different training window sizes as fractions of the total data
length
training_windows = [0.5, 0.6, 0.7]
final_capitals = []

for window in training_windows:
    train_size = int(window * len(X))
```

```
    X_train, X_test = X.iloc[:train_size], X.iloc[train_size:]
    y_train, y_test = y_binary.iloc[:train_size],
y_binary.iloc[train_size:]

    # Train the model
    model.fit(X_train, y_train)

    # Make predictions on the test set
    y_test_pred = model.predict(X_test)

    # Implement the "Long or Short" strategy
    capitals = [10000000]  # Starting with 10 million capital

    for i in range(len(y_test_pred)):
        if y_test_pred[i] == 1:  # If the model predicts the price
will go up
            capitals.append(capitals[-1] * (1 +
df['Return'].iloc[y_test.index[i]]))
        else:  # If the model predicts the price will go down
            capitals.append(capitals[-1] * (1 -
df['Return'].iloc[y_test.index[i]]))

    final_capital = capitals[-1]
    final_capitals.append(final_capital)

    print(f"Training Window: {window*100}% -> Final Capital: $
{final_capital}")

# Visualization of final capitals for different training window sizes
plt.figure(figsize=(10, 6))
plt.plot(training_windows, final_capitals, marker='o', linestyle='-',
color='red')
plt.title('Final Capital vs. Training Window Size')
plt.xlabel('Training Window Size (%)')
plt.ylabel('Final Capital ($)')
plt.grid(True)
plt.show()
```

```
Training Window: 50.0% -> Final Capital: $14897367.792707803
Training Window: 60.0% -> Final Capital: $24119758.892387908
Training Window: 70.0% -> Final Capital: $31237145.50199619
```

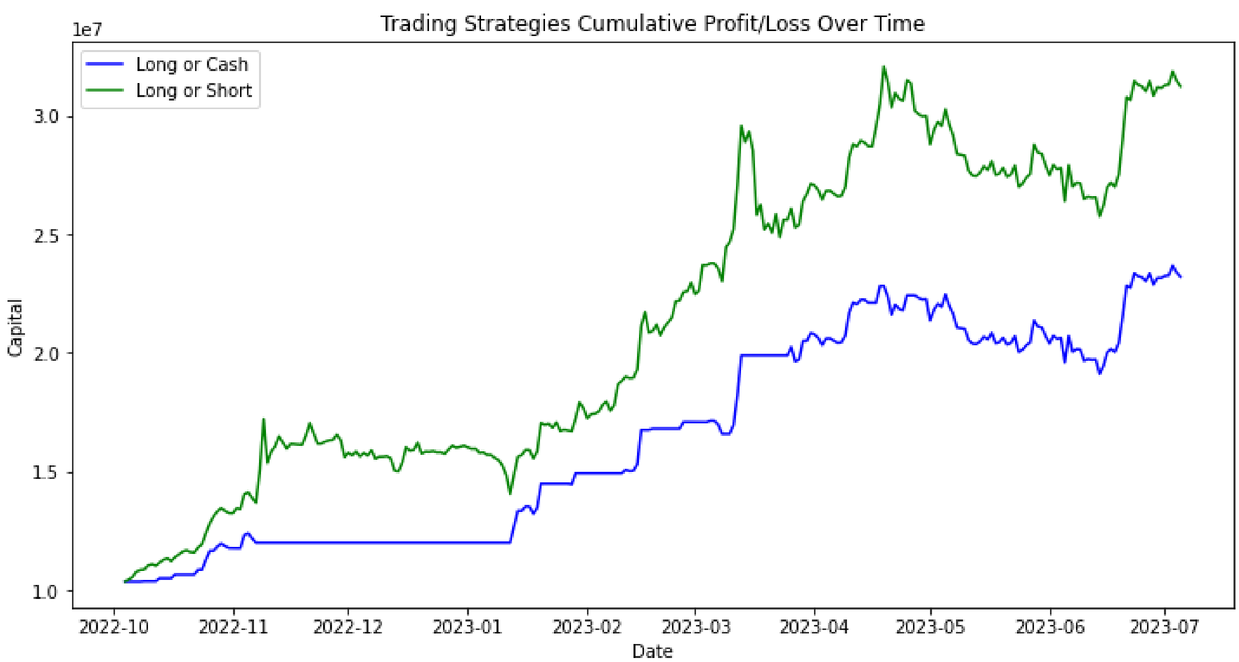

```
# below code is the comparision of which trading strategy is best
"long or cash" "long or Short" In random forest

import pandas as pd
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report
from sklearn.model_selection import TimeSeriesSplit, train_test_split
import matplotlib.pyplot as plt

# Load the data
df = pd.read_csv(r'C:\Users\rushi\Downloads\BTC-USD 2021.csv')

df['Date'] = pd.to_datetime(df['Date'])

# Sort the values by 'Date'
df = df.sort_values('Date')

# Calculate the 'Return'
df['Return'] = df['Close'].pct_change()

# Define the target variable
y = df['Return']

X = df.drop(columns=['Date', 'Return', 'Close'])

# Create a binary target variable
```

```
y_binary = (y > 0).astype(int)

# Initialize the model
model = RandomForestClassifier(n_estimators=100, random_state=42)

# Split the data into training (70%) and test (30%) sets
X_full_train, X_test, y_full_train, y_test = train_test_split(X,
y_binary, test_size=0.3, shuffle=False)

# Implement TimeSeriesSplit for cross-validation on the training set
tscv = TimeSeriesSplit(n_splits=5)

for train_index, val_index in tscv.split(X_full_train):
    X_train, X_val = X_full_train.iloc[train_index],
X_full_train.iloc[val_index]
    y_train, y_val = y_full_train.iloc[train_index],
y_full_train.iloc[val_index]

    # Train the model
    model.fit(X_train, y_train)

    # Make predictions on the validation set
    y_pred = model.predict(X_val)

    # Calculate the metrics
    print(classification_report(y_val, y_pred))

# After cross-validation, evaluate the model on the test set
model.fit(X_full_train, y_full_train)
y_test_pred = model.predict(X_test)

# Implement the "Long or Cash" strategy
capitals_long_or_cash = [10000000]  # Starting with 10 million capital

for i in range(len(y_test_pred)):
    if y_test_pred[i] == 1:  # If the model predicts the price will go
up
        # Buy at the close price
        capitals_long_or_cash.append(capitals_long_or_cash[-1] * (1 +
df['Return'].iloc[y_test.index[i]]))
    else:  # If the model predicts the price will go down, hold cash
(no change in capital)
        capitals_long_or_cash.append(capitals_long_or_cash[-1])

# Implement the "Long or Short" strategy
capitals_long_or_short = [10000000]  # Starting with 10 million
capital

for i in range(len(y_test_pred)):
    if y_test_pred[i] == 1:  # If the model predicts the price will go
```

```
up
        # Buy at the close price
        capitals_long_or_short.append(capitals_long_or_short[-1] * (1
+ df['Return'].iloc[y_test.index[i]]))
    else:  # If the model predicts the price will go down
        # Sell (short) at the close price
        capitals_long_or_short.append(capitals_long_or_short[-1] * (1
- df['Return'].iloc[y_test.index[i]]))

# Visualization
dates = df['Date'].iloc[y_test.index].tolist()
plt.figure(figsize=(12, 6))
plt.plot(dates, capitals_long_or_cash[1:], label="Long or Cash",
color='blue')
plt.plot(dates, capitals_long_or_short[1:], label="Long or Short",
color='green')
plt.title('Trading Strategies Cumulative Profit/Loss Over Time')
plt.xlabel('Date')
plt.ylabel('Capital')
plt.legend()
plt.show()

# Calculate and print profits
profit_long_or_cash = capitals_long_or_cash[-1] -
capitals_long_or_cash[0]
profit_long_or_short = capitals_long_or_short[-1] -
capitals_long_or_short[0]
print(f"Total Profit (Long or Cash): ${profit_long_or_cash}")
print(f"Total Profit (Long or Short): ${profit_long_or_short}")
```

```
              precision    recall  f1-score   support

           0       0.57      0.91      0.70        53
           1       0.77      0.32      0.45        53

    accuracy                           0.61       106
   macro avg       0.67      0.61      0.58       106
weighted avg       0.67      0.61      0.58       106

              precision    recall  f1-score   support

           0       0.54      0.94      0.69        51
           1       0.82      0.25      0.39        55

    accuracy                           0.58       106
   macro avg       0.68      0.60      0.54       106
weighted avg       0.69      0.58      0.53       106

              precision    recall  f1-score   support
```

```
               0       0.76      0.85      0.80        55
               1       0.82      0.71      0.76        51

        accuracy                           0.78       106
       macro avg       0.79      0.78      0.78       106
    weighted avg       0.79      0.78      0.78       106

                  precision    recall  f1-score   support

               0       0.65      0.89      0.75        54
               1       0.81      0.50      0.62        52

        accuracy                           0.70       106
       macro avg       0.73      0.69      0.68       106
    weighted avg       0.73      0.70      0.69       106

                  precision    recall  f1-score   support

               0       0.57      0.47      0.51        58
               1       0.47      0.58      0.52        48

        accuracy                           0.52       106
       macro avg       0.52      0.52      0.52       106
    weighted avg       0.53      0.52      0.52       106
```




```
Total Profit (Long or Cash): $13207241.542338066
Total Profit (Long or Short): $21237145.50199619
```

```python
# below code is  the "long or cash" trading strategy on the test set

#LSTM

import numpy as np
import pandas as pd
from sklearn.preprocessing import MinMaxScaler
from keras.models import Sequential
from keras.layers import Dense, LSTM
import matplotlib.pyplot as plt

# Load the data
df = pd.read_csv(r'C:\Users\rushi\Downloads\BTC-USD 2021.csv')
df['Date'] = pd.to_datetime(df['Date'])

# Sort the values by 'Date'
df = df.sort_values('Date')

# Calculate the 'Return'
df['Return'] = df['Close'].pct_change()

# Prepare the data
new_data = df[['Close']].copy()
scaled_data = MinMaxScaler().fit_transform(new_data.values)

# Define the LSTM model
def create_lstm_model():
    model = Sequential()
    model.add(LSTM(units=50, return_sequences=True, input_shape=(1,
1)))
    model.add(LSTM(units=50))
    model.add(Dense(1))
    model.compile(loss='mean_squared_error', optimizer='adam')
    return model

# Split data into training and test sets
train_data_len = int(0.7 * len(scaled_data))
X_full_train = scaled_data[:train_data_len]
X_test = scaled_data[train_data_len:]

window_sizes = [0.5, 0.6, 0.7, 0.8]
final_capitals = []

# We'll use only the last window for demonstration
window = window_sizes[-1]
train_end = int(train_data_len * window)
train_data = X_full_train[:train_end]
val_data = X_full_train[train_end:]

model = create_lstm_model()
model.fit(train_data[:-1], train_data[1:], epochs=10, batch_size=1,
```

```
verbose=0)

# Predicting on the test set
predicted_prices = model.predict(X_test[:-1])
predicted_prices =
MinMaxScaler().fit(new_data[:train_data_len]).inverse_transform(predic
ted_prices)

# Create a dataframe for visualization
valid_data_df = new_data[train_data_len:-1].copy()
valid_data_df['Predictions'] = predicted_prices

# Implement the "long or cash" trading strategy on the test set
test_capitals = [10000000]
for i in range(len(predicted_prices)):
    if predicted_prices[i] > valid_data_df['Close'].iloc[i]:
        test_capitals.append(test_capitals[-1] * (1 +
df['Return'].iloc[train_data_len + i + 1]))
    else:
        test_capitals.append(test_capitals[-1])

valid_data_df['Portfolio Value'] = test_capitals[1:]

# Visualizing the prediction and trading simulation
plt.figure(figsize=(16,8))
train_data_plot = new_data[:train_end]
plt.plot(train_data_plot.index, train_data_plot["Close"],
label="Train")
plt.plot(valid_data_df.index, valid_data_df['Close'], label="Actual
Price")
plt.plot(valid_data_df.index, valid_data_df['Predictions'],
label="Predicted Price")
plt.title('Bitcoin Price Prediction with Trading Simulation')
plt.xlabel('Time')
plt.ylabel('Price/Value')
plt.legend(loc="lower right")
plt.show()

# Print the final portfolio value
final_portfolio_value = valid_data_df['Portfolio Value'].iloc[-1]
print(f'Final Portfolio Value: {final_portfolio_value}')
```

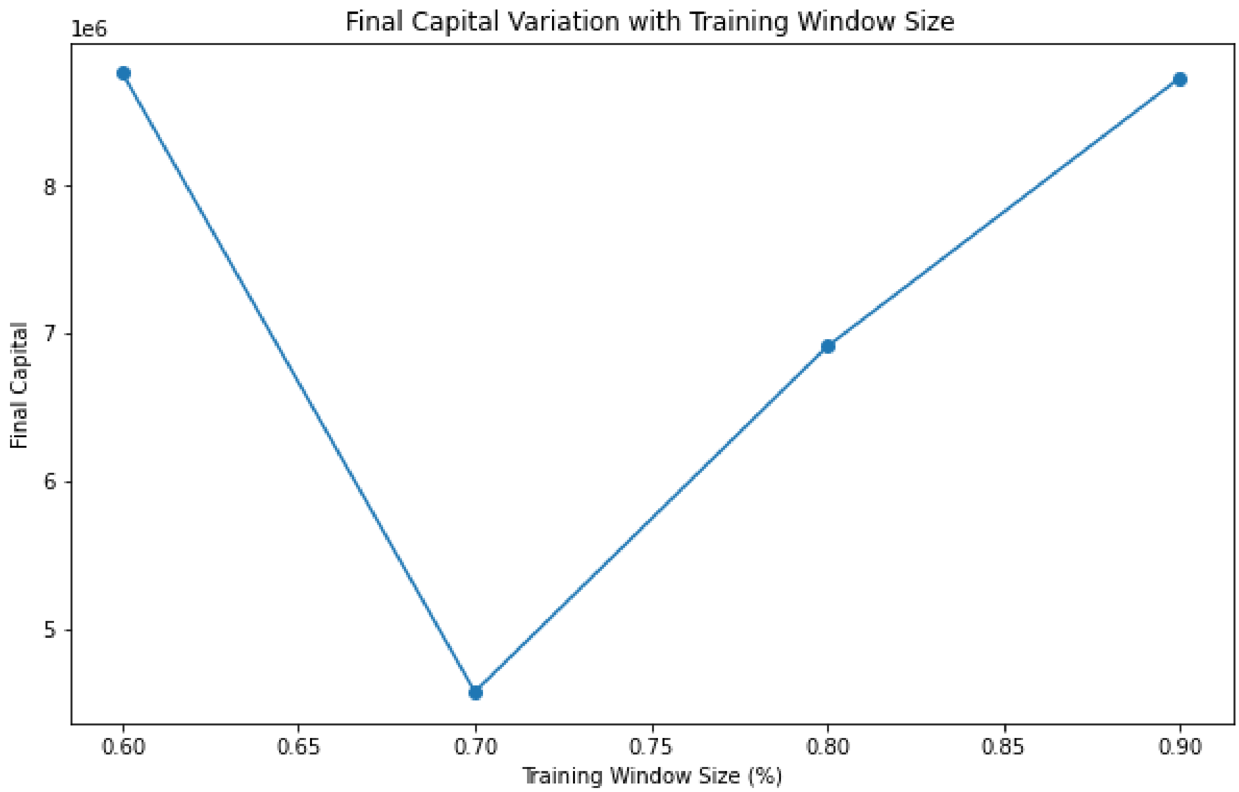
```
9/9 [==============================] - 1s 3ms/step
```


```
Final Portfolio Value: 15018765.010671802

# below code is  the "long or Short" trading strategy on the test set

import numpy as np
import pandas as pd
from sklearn.preprocessing import MinMaxScaler
from keras.models import Sequential
from keras.layers import Dense, LSTM
import matplotlib.pyplot as plt

# Load the data
df = pd.read_csv(r'C:\Users\rushi\Downloads\BTC-USD 2021.csv')
df['Date'] = pd.to_datetime(df['Date'])

# Sort the values by 'Date'
df = df.sort_values('Date')

# Calculate the 'Return'
df['Return'] = df['Close'].pct_change()

# Prepare the data
new_data = df[['Close']].copy()
scaled_data = MinMaxScaler().fit_transform(new_data.values)

# Define the LSTM model
def create_lstm_model():
    model = Sequential()
    model.add(LSTM(units=50, return_sequences=True, input_shape=(1,
1)))
    model.add(LSTM(units=50))
```

```
    model.add(Dense(1))
    model.compile(loss='mean_squared_error', optimizer='adam')
    return model

# Split data into training and test sets
X_full_train = scaled_data[:-int(0.3*len(scaled_data))]
X_test = scaled_data[-int(0.3*len(scaled_data)):]

# Define window sizes
window_sizes = [ 0.6, 0.7, 0.8,0.9]
final_capitals = []

for window in window_sizes:
    train_end = int(len(X_full_train) * window)

    train_data = X_full_train[:train_end]
    val_data = X_full_train[train_end:]

    model = create_lstm_model()
    model.fit(train_data[:-1], train_data[1:], epochs=10,
batch_size=1, verbose=0)

    predictions = model.predict(val_data[:-1])
    predictions = np.where(predictions > val_data[:-1], 1, 0)

    # Implement the "long or short" trading strategy
    capitals = [10000000]
    for i in range(len(predictions)):
        if predictions[i] == 1:
            capitals.append(capitals[-1] * (1 +
df['Return'].iloc[train_end + i + 1]))
        else:
            capitals.append(capitals[-1])
    final_capitals.append(capitals[-1])

# Plot how the final capital varies with training window size
plt.figure(figsize=(10, 6))
plt.plot(window_sizes, final_capitals, marker='o')
plt.title('Final Capital Variation with Training Window Size')
plt.xlabel('Training Window Size (%)')
plt.ylabel('Final Capital')
plt.show()

# After evaluating different window sizes, train the model on the full
training set and evaluate on the test set
model = create_lstm_model()
model.fit(X_full_train[:-1], X_full_train[1:], epochs=10,
batch_size=1, verbose=0)

test_predictions = model.predict(X_test[:-1])
```

```
test_predictions = np.where(test_predictions > X_test[:-1], 1, 0)

# Implement the "long or short" trading strategy on the test set
test_capitals = [10000000]
for i in range(len(test_predictions)):
    if test_predictions[i] == 1:
        # Long
        test_capitals.append(test_capitals[-1] * (1 +
df['Return'].iloc[-int(0.3*len(df)) + i + 1]))
    else:
        # Short
        test_capitals.append(test_capitals[-1] * (1 -
df['Return'].iloc[-int(0.3*len(df)) + i + 1]))

# Print final portfolio value
print(f"Final Portfolio Value: ${test_capitals[-1]:,.2f}")

# Visualizing the prediction and trading simulation
plt.figure(figsize=(16,8))
train_data_plot = new_data[:train_end]
plt.plot(train_data_plot.index, train_data_plot["Close"],
label="Train")
plt.plot(valid_data_df.index, valid_data_df['Close'], label="Actual
Price")
plt.plot(valid_data_df.index, valid_data_df['Predictions'],
label="Predicted Price")
plt.title('Bitcoin Price Prediction with Long/Short Trading
Simulation')
plt.xlabel('Time')
plt.ylabel('Price/Value')
plt.legend(loc="lower right")
plt.show()
```

```
8/8 [==============================] - 1s 4ms/step
6/6 [==============================] - 1s 3ms/step
4/4 [==============================] - 1s 5ms/step
2/2 [==============================] - 1s 0s/step
```




```
9/9 [==============================] - 1s 4ms/step
Final Portfolio Value: $19,002,979.05
```


```
import numpy as np
import pandas as pd
```

```python
from sklearn.preprocessing import MinMaxScaler
from keras.models import Sequential
from keras.layers import Dense, LSTM
import matplotlib.pyplot as plt

# Load the data
df = pd.read_csv(r'C:\Users\rushi\Downloads\BTC-USD 2021.csv')
df['Date'] = pd.to_datetime(df['Date'])

# Sort the values by 'Date'
df = df.sort_values('Date')

# Calculate the 'Return'
df['Return'] = df['Close'].pct_change()

# Prepare the data
new_data = df[['Close']].copy()
scaler = MinMaxScaler()
scaled_data = scaler.fit_transform(new_data.values)

# Define the LSTM model
def create_lstm_model():
    model = Sequential()
    model.add(LSTM(units=50, return_sequences=True, input_shape=(1,
1)))
    model.add(LSTM(units=50))
    model.add(Dense(1))
    model.compile(loss='mean_squared_error', optimizer='adam')
    return model

# Adjust the number of days for training here
num_days = 500
train_data_len = num_days
X_train = scaled_data[:train_data_len]
X_valid = scaled_data[train_data_len:]

model = create_lstm_model()
model.fit(X_train[:-1], X_train[1:], epochs=10, batch_size=1,
verbose=0)

predictions = model.predict(X_valid[:-1])

# Convert the scaled predictions back to original scale
predicted_prices = scaler.inverse_transform(predictions)

# Implement the "long or short" trading strategy
capitals = [10000000]  # Starting capital
for i in range(len(predictions)):
    if predictions[i] > X_valid[i]:
        # Long
```

```
        capitals.append(capitals[-1] * (1 +
df['Return'].iloc[train_data_len + i + 1]))
    else:
        # Short
        capitals.append(capitals[-1] * (1 -
df['Return'].iloc[train_data_len + i + 1]))

# Print final portfolio value
print(f"Final Portfolio Value: ${capitals[-1]:,.2f}")

# Visualizing the prediction and trading simulation
plt.figure(figsize=(16,8))
train_data_plot = new_data[:train_data_len]
plt.plot(train_data_plot.index, train_data_plot["Close"],
label="Train")
plt.plot(new_data[train_data_len:].index, new_data[train_data_len:]
['Close'], label="Actual Price")
plt.plot(new_data[train_data_len:].index[:-1], predicted_prices,
label="Predicted Price", alpha=0.6)
plt.legend(loc="lower right")
plt.title('Bitcoin Price Prediction with Long/Short Trading
Simulation')
plt.xlabel('Time')
plt.ylabel('Price/Value')
plt.show()
```

```
13/13 [==============================] - 1s 3ms/step
Final Portfolio Value: $10,227,877.85
```