Capstone Project Code

Contents

1. Data definition, extraction and Pre-processing
2. Volatility Modeling
3. Testing Investment Strategy and Summarizing Results
4. Event Based Testing

```
#Importing libraries
!pip install exchange-calendars --upgrade
!pip install pandas --upgrade
!pip install yfinance
!pip install nsedt
!pip install statsmodels
!pip install numpy -- upgrade
!pip install sklearn

from datetime import datetime, timedelta
import pandas as pd
import yfinance as yf
from nsedt import derivatives
import numpy as np
from statsmodels.tsa.arima.model import ARIMA
from statsmodels.tsa.stattools import adfuller
from sklearn.metrics import mean_squared_error
from google.colab import  drive
drive.mount('/drive')

Drive already mounted at /drive; to attempt to forcibly remount, call
drive.mount("/drive", force_remount=True).
```

Data definition, extraction and Pre-processing

```
#Date ranges
end_date = datetime.now()- timedelta(days=1)
start_date_train = end_date - timedelta(days=365)
end_date_train=end_date - timedelta(days=90)
start_date_val=end_date_train+timedelta(days=1)
end_date_val=end_date - timedelta(days=30)
start_date_test=end_date_val+timedelta(days=1)

ticker_symbols =
["RELIANCE.NS","ADANIENT.NS","BHARTIARTL.NS","SBIN.NS",
                  "ICICIBANK.NS","DRREDDY.NS","ASHOKLEY.NS",
                  "AUROPHARMA.NS","JINDALSTEL.NS","TATAMOTORS.NS"]
index_ticker_symbols=['^NSEI','NIFTY_FIN_SERVICE.NS']
data_dict_train=dict()
```

```python
data_dict_val=dict()
data_dict_test=dict()
```

```python
l=ticker_symbols+index_ticker_symbols
for i in l:
  # Download historical data
  train_data = yf.download(i, start=start_date_train-
timedelta(days=1), end=end_date_train, interval="1d")
  val_data = yf.download(i, start=start_date_val, end=end_date_val,
interval="1d")
  test_data = yf.download(i, start=start_date_test, end=end_date,
interval="1d")

  k=[train_data,val_data,test_data]
  for j in k:
    #adding range data
    j["Range"]=j["High"]-j["Low"]
    #adding COB and PCOB average price data
    j["COB_AvG_Price"]=(j["High"]+j["Low"]+j["Adj Close"])/3
    j["PCOB_AvG_Price"]=(j["High"].shift(1)+j["Low"].shift(1)+j["Adj
Close"].shift(1))/3
    j["PCOB_Adj_Close"]=j["Adj Close"].shift(1)

    #adding volatility measure
    j["%vol_measure"]=(j["Range"]/ j["PCOB_AvG_Price"])*100
    j.dropna(inplace=True)

  data_dict_train[i]=train_data
  data_dict_val[i]=val_data
  data_dict_test[i]=test_data


for i in data_dict_train.keys():
  print(i,"->",data_dict_train[i].shape)
```

```
RELIANCE.NS -> (184, 11)
ADANIENT.NS -> (184, 11)
BHARTIARTL.NS -> (184, 11)
SBIN.NS -> (184, 11)
ICICIBANK.NS -> (184, 11)
DRREDDY.NS -> (184, 11)
ASHOKLEY.NS -> (184, 11)
AUROPHARMA.NS -> (184, 11)
JINDALSTEL.NS -> (184, 11)
TATAMOTORS.NS -> (184, 11)
^NSEI -> (184, 11)
NIFTY_FIN_SERVICE.NS -> (184, 11)
```

```python
vol_df=pd.DataFrame(columns=["Ticker"])
vol_df["Ticker"]=ticker_symbols+index_ticker_symbols
vol=[]
```

```
vol_std=[]
for i in data_dict_train:
  vol.append(data_dict_train[i]["%vol_measure"].mean())
  vol_std.append(data_dict_train[i]["%vol_measure"].std())

vol_df["Avg Volatility"]=vol
vol_df["Std Volatility"]=vol_std

vol_df
```

{"summary":"{\n \"name\": \"vol_df\",\n \"rows\": 12,\n \"fields\": [\n   {\n   \"column\": \"Ticker\",\n   \"properties\": {\n   \"dtype\": \"string\",\n   \"num_unique_values\": 12,\n   \"samples\": [\n   \"^NSEI\",\n   \"TATAMOTORS.NS\",\n   \"RELIANCE.NS\"\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"Avg Volatility\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 0.782429193220397,\n   \"min\": 0.7316693143876727,\n   \"max\": 3.5791894003174103,\n   \"num_unique_values\": 12,\n   \"samples\": [\n   0.7316693143876727,\n   2.0650269000857837,\n   1.4323618250948515\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"Std Volatility\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 0.653381211906111,\n   \"min\": 0.2754258992337595,\n   \"max\": 2.804867271734959,\n   \"num_unique_values\": 12,\n   \"samples\": [\n   0.2754258992337595,\n   0.9380087443198538,\n   0.6410633169986183\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   }\n]\n}","type":"dataframe","variable_name":"vol_df"}

```
def fetch_option_chain(symbol):
    opt=derivatives.get_option_chain(symbol,expiry_date='28-03-2024')
    return opt

cob_option_data=dict()

for i in ticker_symbols:
  print(f"Extracting option data for {i}")
  temp=fetch_option_chain(i[:-3])
  temp["ticker"]=i
  cob_option_data[i]=temp
```

```
Extracting option data for RELIANCE.NS
Extracting option data for ADANIENT.NS
Extracting option data for BHARTIARTL.NS
Extracting option data for SBIN.NS
Extracting option data for ICICIBANK.NS
Extracting option data for DRREDDY.NS
Extracting option data for ASHOKLEY.NS
```

```
Extracting option data for AUROPHARMA.NS
Extracting option data for JINDALSTEL.NS
Extracting option data for TATAMOTORS.NS
```

```
#for i in cob_option_data:
#  cob_option_data[i].to_csv("/drive/My
Drive/option_data/202403/{}.csv".format(i))

#index data
#opt=derivatives.get_option_chain('NIFTY',expiry_date='28-03-2024')
#opt.to_csv("/drive/My Drive/option_data/202403/NIFTY.csv")
#print("NIFTY option data extracted.")
##opt=derivatives.get_option_chain('BANKNIFTY',expiry_date='20-03-
2024')
#opt.to_csv("/drive/My Drive/option_data/202403/BANKNIFTY.csv")
#print("BANKNIFTY option data extracted.")
#opt=derivatives.get_option_chain('FINNIFTY',expiry_date='26-03-2024')
#opt.to_csv("/drive/My Drive/option_data/202403/FINNIFTY.csv")
#print("FINNIFTY option data extracted.")
```

Volatility Modeling

```
all_result_train=dict()
for i in vol_df["Ticker"]:
  print(i)
  mean_vol=vol_df[vol_df["Ticker"]==i]["Avg Volatility"]
[vol_df[vol_df["Ticker"]==i].index[0]]
  std_vol=vol_df[vol_df["Ticker"]==i]["Std Volatility"]
[vol_df[vol_df["Ticker"]==i].index[0]]
  temp_train=data_dict_train[i]
  temp_train["Vol
Zone"]=np.where(np.abs(temp_train['%vol_measure'])>=(mean_vol+(1.25*st
d_vol)),"High
Volatility",np.where(np.abs(temp_train['%vol_measure'])<=(mean_vol+(0.
25*std_vol)),"Low Volatility","Neutral"))
  #standardising the time series
  ts_train=(temp_train["%vol_measure"]-mean_vol)/std_vol
  # Define the ARIMA model
  order=[]
  aic=[]
  bic=[]
  adf_result=[]

result=pd.DataFrame(columns=["order","aic_score","bic_score","adf_resu
lt"])
  for p in range(5):
    for q in range(5):
      model = ARIMA(ts_train, order=(p+1,1,q+1))
      model_fit = model.fit()
```

```
            order.append(("("+str(p+1)+","+"1,"+str(q+1)+")"))
            aic.append(model_fit.aic)
            bic.append(model_fit.bic)
            adf_result.append(adfuller(ts_train)[1])
    result["order"]=order
    result["aic_score"]=aic
    result["bic_score"]=bic
    result["adf_result"]=adf_result
    all_result_train[i]=result

for i in l:
  print(i+"->"+str((all_result_train[i]["adf_result"].unique()[0])))
```

```
RELIANCE.NS->1.209357668503967e-10
ADANIENT.NS->0.0001499230792152034
BHARTIARTL.NS->8.574972536399952e-25
SBIN.NS->2.7708021395392482e-17
ICICIBANK.NS->6.474462155386234e-19
DRREDDY.NS->4.443823364395012e-12
ASHOKLEY.NS->3.939905007445122e-21
AUROPHARMA.NS->6.65562438908639e-18
JINDALSTEL.NS->5.282570920352968e-10
TATAMOTORS.NS->7.697868188105616e-21
^NSEI->2.894882263205805e-15
NIFTY_FIN_SERVICE.NS->3.17996698288298e-14
```

```
all_result_val_test=dict()
pred_result=pd.DataFrame(columns=["mse_aic","mse_bic","decision_criter
ion_adopted","mse_test"])
for i in l:
  pred_aic=[]
  pred_bic=[]
  mean_vol=vol_df[vol_df["Ticker"]==i]["Avg Volatility"]
[vol_df[vol_df["Ticker"]==i].index[0]]
  std_vol=vol_df[vol_df["Ticker"]==i]["Std Volatility"]
[vol_df[vol_df["Ticker"]==i].index[0]]
  temp_train=data_dict_train[i]
  temp_val=data_dict_val[i]

  #standardising the time series
  ts_train=(temp_train["%vol_measure"]-mean_vol)/std_vol
  ts_val=(temp_val["%vol_measure"]-mean_vol)/std_vol

  idx_aic=all_result_train[l[0]][all_result_train[l[0]]
["aic_score"]==all_result_train[l[0]]["aic_score"].max()].index[0]
  idx_bic=all_result_train[l[0]][all_result_train[l[0]]
["bic_score"]==all_result_train[l[0]]["bic_score"].max()].index[0]
  order_aic=all_result_train[i].loc[idx_aic,"order"]
  order_bic=all_result_train[i].loc[idx_bic,"order"]
```

```
  for j in range(len(ts_val)):
    model_aic = ARIMA(ts_train,
order=(int(order_aic[1]),int(order_aic[3]),int(order_aic[5])))
    model_fit_aic = model_aic.fit()
    pred_aic.append(list(model_fit_aic.forecast(steps=1))[0])
    model_bic = ARIMA(ts_train,
order=(int(order_bic[1]),int(order_bic[3]),int(order_bic[5])))
    model_fit_bic = model_bic.fit()
    pred_bic.append(list(model_fit_bic.forecast(steps=1))[0])
    ts_train=np.append(ts_train,ts_val.iloc[j])
  ts_val=ts_val.to_frame()
  ts_val["pred_aic"]=pred_aic
  ts_val["pred_bic"]=pred_bic

  if pred_result.shape[0]==0:
                      pred_result=
pd.DataFrame({"mse_aic":mean_squared_error(ts_val[ts_val.columns[0]],t
s_val[ts_val.columns[1]]),"mse_bic":mean_squared_error(ts_val[ts_val.c
olumns[0]],ts_val[ts_val.columns[2]]),"decision_criterion_adopted":"",
"mse_test":""},index=[i])
  else:

pred_result.loc[i]=[mean_squared_error(ts_val[ts_val.columns[0]],ts_va
l[ts_val.columns[1]]),mean_squared_error(ts_val[ts_val.columns[0]],ts_
val[ts_val.columns[2]]),"",""]


pred_result["decision_criterion_adopted"]=np.where(pred_result[pred_re
sult.columns[0]]<=pred_result[pred_result.columns[1]],"max_aic","max_b
ic")
for i in l:
  mse_test=[]
  mean_vol=vol_df[vol_df["Ticker"]==i]["Avg Volatility"]
[vol_df[vol_df["Ticker"]==i].index[0]]
  std_vol=vol_df[vol_df["Ticker"]==i]["Std Volatility"]
[vol_df[vol_df["Ticker"]==i].index[0]]
  temp_train=data_dict_train[i]
  temp_val=data_dict_val[i]
  temp_test=data_dict_test[i]

  #standardising the time series
  ts_train=(temp_train["%vol_measure"]-mean_vol)/std_vol
  ts_val=(temp_val["%vol_measure"]-mean_vol)/std_vol
  ts_test=(temp_test["%vol_measure"]-mean_vol)/std_vol

  ts_train=np.append(ts_train,ts_val)

  idx_aic=all_result_train[i][all_result_train[i]
["aic_score"]==all_result_train[i]["aic_score"].max()].index[0]
  idx_bic=all_result_train[i][all_result_train[i]
```

```
["bic_score"]==all_result_train[i]["bic_score"].max()].index[0]
  order_aic=all_result_train[i].loc[idx_aic,"order"]
  order_bic=all_result_train[i].loc[idx_bic,"order"]

  if pred_result.loc[i,"decision_criterion_adopted"]=='max_aic':
    idx=idx_aic
    order=order_aic
  else:
    idx=idx_bic
    order=order_bic

  for j in range(len(ts_test)):
    model = ARIMA(ts_train,
order=(int(order[1]),int(order[3]),int(order[5])))
    model_fit = model.fit()
    mse_test.append(list(model_fit.forecast(steps=1))[0])
    ts_train=np.append(ts_train,np.array(ts_test)[j])

  ts_test=ts_test.to_frame()
  ts_test["pred"]=mse_test
  data_dict_test[i]["pred_vol"]=[((k*std_vol)+mean_vol) for k in
mse_test]

pred_result.loc[i,"mse_test"]=mean_squared_error(ts_test[ts_test.colum
ns[0]],ts_test[ts_test.columns[1]])

pred_result
```

{"summary":"{\n \"name\": \"pred_result\",\n \"rows\": 12,\n
\"fields\": [\n   {\n   \"column\": \"mse_aic\",\n
\"properties\": {\n   \"dtype\": \"number\",\n   \"std\":
1.0152273430466472,\n   \"min\": 0.4334263511413505,\n
\"max\": 3.7077977550443912,\n   \"num_unique_values\": 12,\n
\"samples\": [\n   3.569895437111768,\n
2.05541566735582,\n   2.9118936699010813\n   ],\n
\"semantic_type\": \"\",\n   \"description\": \"\"\n   }\
n   },\n   {\n   \"column\": \"mse_bic\",\n   \"properties\":
{\n   \"dtype\": \"number\",\n   \"std\":
1.0152273430466472,\n   \"min\": 0.4334263511413505,\n
\"max\": 3.7077977550443912,\n   \"num_unique_values\": 12,\n
\"samples\": [\n   3.569895437111768,\n
2.05541566735582,\n   2.9118936699010813\n   ],\n
\"semantic_type\": \"\",\n   \"description\": \"\"\n   }\
n   },\n   {\n   \"column\": \"decision_criterion_adopted\",\n
\"properties\": {\n   \"dtype\": \"category\",\n
\"num_unique_values\": 1,\n   \"samples\": [\n
\"max_aic\"\n   ],\n   \"semantic_type\": \"\",\n
\"description\": \"\"\n   }\n   },\n   {\n   \"column\":
\"mse_test\",\n   \"properties\": {\n   \"dtype\": \"date\",\n
\"min\": 0.17731803898796294,\n   \"max\": 3.2144870942826516,\n

```
\"num_unique_values\": 12,\n        \"samples\": [\n
2.811936838900468\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    }\n  ]\
n}","type":"dataframe","variable_name":"pred_result"}
```


Testing Investment Strategy and Summarizing Results

```
correct_prediction_count=0
Total_predictions=0
for i in l:
  data_dict_test[i]
["Actual_Vol_Zone"]=np.where(np.abs(data_dict_test[i]
['%vol_measure'])>=(mean_vol+(0.75*std_vol)),"High
Volatility",np.where(np.abs(data_dict_test[i]
['%vol_measure'])<=(mean_vol+(0.25*std_vol)),"Low
Volatility","Neutral"))
  data_dict_test[i]["Pred_Vol_Zone"]=np.where(np.abs(data_dict_test[i]
['pred_vol'])>=(mean_vol+(0.75*std_vol)),"High
Volatility",np.where(np.abs(data_dict_test[i]
['pred_vol'])<=(mean_vol+(0.25*std_vol)),"Low Volatility","Neutral"))
  data_dict_test[i]["Prediction_accuracy"]=np.where(data_dict_test[i]
["Actual_Vol_Zone"]==data_dict_test[i]["Pred_Vol_Zone"],True,False)
  data_dict_test[i]["Position_taken"]=np.where(data_dict_test[i]
["Pred_Vol_Zone"]=="High Volatility","Short Straddle
position",np.where(data_dict_test[i]["Pred_Vol_Zone"]=="Low
Volatility","Long Straddle position","No position taken"))
  correct_prediction_count+=data_dict_test[i]
["Prediction_accuracy"].sum()
  Total_predictions+=data_dict_test[i].shape[0]
  #print(i+"\n")
  #print(data_dict_test[i]["Actual_Vol_Zone"].value_counts())
  #print(data_dict_test[i]["Pred_Vol_Zone"].value_counts())
print(f"Prediction accuracy of Volatility Zone identification is
{str((correct_prediction_count*100)/Total_predictions)[:5]}%.")
```

```
Prediction accuracy of Volatility Zone identification is 80.61%.
```

```
option_date_list=['20240228','20240229','20240301','20240304','2024030
5','20240306','20240307','20240311']
gain_loss_dict=dict()
for i in l:

gain_loss_summary=pd.DataFrame(columns=["position","strategy_cost","st
rategy_gain","net_gain"])
  for j in option_date_list:
    opt_data=pd.read_csv("/drive/My
Drive/option_data/"+j+"/"+i+".csv")
    underlying_data=data_dict_test[i].iloc[-8:,:]

optimal_invested_option_data=opt_data[((np.abs(opt_data["strikePrice"]
```

```
-
int(round(underlying_data.loc[j,"PCOB_AvG_Price"],0)))==min(np.abs(opt
_data["strikePrice"]-
int(round(underlying_data.loc[j,"PCOB_AvG_Price"],0)))))==True)]
[["strikePrice",'PE.bidprice','PE.askPrice','CE.bidprice','CE.askPrice
']]
    strike=optimal_invested_option_data.iloc[0,0]
    if underlying_data.loc[j,"Position_taken"]=="Long Straddle
position":
      price_CE=optimal_invested_option_data["CE.bidprice"].iloc[0]
      price_PE=optimal_invested_option_data["PE.bidprice"].iloc[0]

gain_loss_entry=[underlying_data.loc[j,"Position_taken"],price_CE+pric
e_PE,int(round(max(underlying_data.loc[j,"High"]-strike,strike-
underlying_data.loc[j,"Low"]),0)),""]
    elif underlying_data.loc[j,"Position_taken"]=="Short Straddle
position":
      price_CE=optimal_invested_option_data["CE.askPrice"].iloc[0]
      price_PE=optimal_invested_option_data["PE.askPrice"].iloc[0]

gain_loss_entry=[underlying_data.loc[j,"Position_taken"],int(round(max
(underlying_data.loc[j,"High"]-strike,strike-
underlying_data.loc[j,"Low"]),0)),price_CE+price_PE,""]

    else:
      gain_loss_entry=[underlying_data.loc[j,"Position_taken"],0,0,0]
    gain_loss_summary.loc[j,gain_loss_summary.columns]=gain_loss_entry
  gain_loss_summary['net_gain']=gain_loss_summary["strategy_gain"]-
gain_loss_summary["strategy_cost"]
  gain_loss_dict[i]=gain_loss_summary

overall_gain=0
overall_cost=0
for i in l:
    print(f"Summary Result:{i}")
    print(f"Net Gain from the strategy for {i} is Rs
{int(round(gain_loss_dict[i]['net_gain'].sum(), 0))}")
    print(f"Percent Net Gain from the strategy for {i} is
{round(((gain_loss_dict[i]['net_gain'].sum() * 100) /
gain_loss_dict[i]['strategy_cost'].sum()), 2)}%")
    print("Overall Summary")
    overall_gain+=int(round(gain_loss_dict[i]['strategy_gain'].sum(),
0))
    overall_cost+=int(round(gain_loss_dict[i]['strategy_cost'].sum(),
0))
net_gains=overall_gain-overall_cost
print(f"Overall Net Gain from the strategy is Rs {net_gains}")
print(f"Percent Net Gain from the strategy is {int(round((net_gains *
100) / overall_cost,2))}%")
```

```
Summary Result:RELIANCE.NS
Net Gain from the strategy for RELIANCE.NS is Rs 675
Percent Net Gain from the strategy for RELIANCE.NS is 157.42%
Overall Summary
Summary Result:ADANIENT.NS
Net Gain from the strategy for ADANIENT.NS is Rs 1569
Percent Net Gain from the strategy for ADANIENT.NS is 287.9%
Overall Summary
Summary Result:BHARTIARTL.NS
Net Gain from the strategy for BHARTIARTL.NS is Rs 239
Percent Net Gain from the strategy for BHARTIARTL.NS is 102.64%
Overall Summary
Summary Result:SBIN.NS
Net Gain from the strategy for SBIN.NS is Rs 241
Percent Net Gain from the strategy for SBIN.NS is 217.16%
Overall Summary
Summary Result:ICICIBANK.NS
Net Gain from the strategy for ICICIBANK.NS is Rs 250
Percent Net Gain from the strategy for ICICIBANK.NS is 163.46%
Overall Summary
Summary Result:DRREDDY.NS
Net Gain from the strategy for DRREDDY.NS is Rs 1849
Percent Net Gain from the strategy for DRREDDY.NS is 211.3%
Overall Summary
Summary Result:ASHOKLEY.NS
Net Gain from the strategy for ASHOKLEY.NS is Rs 63
Percent Net Gain from the strategy for ASHOKLEY.NS is 274.35%
Overall Summary
Summary Result:AUROPHARMA.NS
Net Gain from the strategy for AUROPHARMA.NS is Rs 402
Percent Net Gain from the strategy for AUROPHARMA.NS is 172.62%
Overall Summary
Summary Result:JINDALSTEL.NS
Net Gain from the strategy for JINDALSTEL.NS is Rs 267
Percent Net Gain from the strategy for JINDALSTEL.NS is 113.16%
Overall Summary
Summary Result:TATAMOTORS.NS
Net Gain from the strategy for TATAMOTORS.NS is Rs 257
Percent Net Gain from the strategy for TATAMOTORS.NS is 111.1%
Overall Summary
Summary Result:^NSEI
Net Gain from the strategy for ^NSEI is Rs -1626
Percent Net Gain from the strategy for ^NSEI is -47.28%
Overall Summary
Summary Result:NIFTY_FIN_SERVICE.NS
Net Gain from the strategy for NIFTY_FIN_SERVICE.NS is Rs -480
Percent Net Gain from the strategy for NIFTY_FIN_SERVICE.NS is -43.77%
Overall Summary
```

```
Overall Net Gain from the strategy is Rs 3963
Percent Net Gain from the strategy is 52%
```

Event based testing

1. Demonitisation: Announced on 8th November'2016 ; Model accuracy is studied for month of December based on 1 year training data.
2. COVID Lockdown: Announced on 24th March'2020 ; Model accuracy is studied for month of May based on 1 year training data.
3. JIO : Launched on 5th Sept'2015, Model accuracy is studied for month of October based on 1 year training data.

```
#demonitization
date_demo=datetime(2016,12,1)
date_demo_start=date_demo-timedelta(365)
date_demo_end=date_demo+timedelta(31)
#COVID Lockdown
date_covid=datetime(2020,4,15)
date_covid_start=date_covid-timedelta(365)
date_covid_end=date_covid+timedelta(31)
#JIO launch
date_jio=datetime(2015,10,1)
date_jio_start=date_jio-timedelta(365)
date_jio_end=date_jio+timedelta(31)
events_dict={"demonitisation":
[date_demo_start,date_demo,date_demo_end],"covid_lockdown":
[date_covid_start,date_covid,date_covid_end],"jio_launch":
[date_jio_start,date_jio,date_jio_end]}

l=ticker_symbols+index_ticker_symbols
data_dict_train_e=dict()
data_dict_val_e=dict()
```

Event based testing: Data extraction

```
#demonitisation
e=list(events_dict.keys())[0]
print(e)
for i in l:
  print(i)
  # Download historical data
  train_data = yf.download(i, start=date_demo_start, end=date_demo-
timedelta(1), interval="1d")
  val_data = yf.download(i, start=date_demo, end=date_demo_end,
interval="1d")
  k=[train_data,val_data]
  for j in k:
    #adding range data
    j["Range"]=j["High"]-j["Low"]
```

```
    #adding COB and PCOB average price data
    j["COB_AvG_Price"]=(j["High"]+j["Low"]+j["Adj Close"])/3
    j["PCOB_AvG_Price"]=(j["High"].shift(1)+j["Low"].shift(1)+j["Adj
Close"].shift(1))/3
    j["PCOB_Adj_Close"]=j["Adj Close"].shift(1)

    #adding volatility measure
    j["%vol_measure"]=(j["Range"]/ j["PCOB_AvG_Price"])*100
    j.dropna(inplace=True)

  data_dict_train[i]=k[0]
  data_dict_val[i]=k[1]
data_dict_train_e[e]= data_dict_train
data_dict_val_e[e]= data_dict_val

vol_df_e=pd.DataFrame(columns=["Ticker"])
vol_df_e["Ticker"]=ticker_symbols+index_ticker_symbols
vol=[]
vol_std=[]
for i in data_dict_train_e[e]:
  vol.append(data_dict_train_e[e][l[0]]["%vol_measure"].mean())
  vol_std.append(data_dict_train_e[e][l[0]]["%vol_measure"].std())
vol_df_e["Avg Volatility"]=vol
vol_df_e["Std Volatility"]=vol_std
vol_df_e

all_result_train=dict()
for i in vol_df_e["Ticker"]:
  print(i)
  mean_vol=vol_df_e[vol_df_e["Ticker"]==i]["Avg Volatility"]
[vol_df_e[vol_df_e["Ticker"]==i].index[0]]
  std_vol=vol_df_e[vol_df_e["Ticker"]==i]["Std Volatility"]
[vol_df_e[vol_df_e["Ticker"]==i].index[0]]
  temp_train=data_dict_train_e[e][i]
  #standardising the time series
  ts_train=(temp_train["%vol_measure"]-mean_vol)/std_vol
  # Define the ARIMA model
  order=[]
  aic=[]
  bic=[]
  adf_result=[]

result=pd.DataFrame(columns=["order","aic_score","bic_score","adf_resu
lt"])
  for p in range(5):
      for q in range(5):
        model = ARIMA(ts_train, order=(p+1,1,q+1))
        model_fit = model.fit()
        order.append(("("+str(p+1)+","+"1,"+str(q+1)+")"))
        aic.append(model_fit.aic)
```

```
        bic.append(model_fit.bic)
        adf_result.append(adfuller(ts_train)[1])
  result["order"]=order
  result["aic_score"]=aic
  result["bic_score"]=bic
  result["adf_result"]=adf_result

  all_result_train[i]=result

print(e+"\n")
for i in l:
  print(i+"->"+str((all_result_train[i]["adf_result"].unique()[0])))
```

```
demonitisation

RELIANCE.NS->0.0012041642529197342
ADANIENT.NS->1.8731806422736484e-06
BHARTIARTL.NS->5.620526269157301e-23
SBIN.NS->4.574354640509609e-06
ICICIBANK.NS->7.73951213543398e-12
DRREDDY.NS->3.370143531753657e-05
ASHOKLEY.NS->2.841504517996158e-06
AUROPHARMA.NS->0.00023412062004513008
JINDALSTEL.NS->4.497999956569616e-06
TATAMOTORS.NS->1.0988909856430138e-08
^NSEI->2.07170800573691e-05
NIFTY_FIN_SERVICE.NS->3.531087039375241e-06
```

```
#covid lockdown
e=list(events_dict.keys())[1]
print(e)
for i in l:
  print(i)
  # Download historical data
  train_data = yf.download(i, start=date_covid_start, end=date_covid-
timedelta(1), interval="1d")
  val_data = yf.download(i, start=date_covid, end=date_covid_end,
interval="1d")
  k=[train_data,val_data]
  for j in k:
    #adding range data
    j["Range"]=j["High"]-j["Low"]
    #adding COB and PCOB average price data
    j["COB_AvG_Price"]=(j["High"]+j["Low"]+j["Adj Close"])/3
    j["PCOB_AvG_Price"]=(j["High"].shift(1)+j["Low"].shift(1)+j["Adj
Close"].shift(1))/3
    j["PCOB_Adj_Close"]=j["Adj Close"].shift(1)

    #adding volatility measure
    j["%vol_measure"]=(j["Range"]/ j["PCOB_AvG_Price"])*100
```

```
    j.dropna(inplace=True)

  data_dict_train[i]=k[0]
  data_dict_val[i]=k[1]
data_dict_train_e[e]= data_dict_train
data_dict_val_e[e]= data_dict_val

vol_df_e=pd.DataFrame(columns=["Ticker"])
vol_df_e["Ticker"]=ticker_symbols+index_ticker_symbols
vol=[]
vol_std=[]
for i in data_dict_train_e[e]:
  vol.append(data_dict_train_e[e][l[0]]["%vol_measure"].mean())
  vol_std.append(data_dict_train_e[e][l[0]]["%vol_measure"].std())
vol_df_e["Avg Volatility"]=vol
vol_df_e["Std Volatility"]=vol_std
vol_df_e

all_result_train=dict()
for i in vol_df_e["Ticker"]:
  print(i)
  mean_vol=vol_df_e[vol_df_e["Ticker"]==i]["Avg Volatility"]
[vol_df_e[vol_df_e["Ticker"]==i].index[0]]
  std_vol=vol_df_e[vol_df_e["Ticker"]==i]["Std Volatility"]
[vol_df_e[vol_df_e["Ticker"]==i].index[0]]
  temp_train=data_dict_train_e[e][i]
  #standardising the time series
  ts_train=(temp_train["%vol_measure"]-mean_vol)/std_vol
  # Define the ARIMA model
  order=[]
  aic=[]
  bic=[]
  adf_result=[]

result=pd.DataFrame(columns=["order","aic_score","bic_score","adf_resu
lt"])
  for p in range(5):
      for q in range(5):
        model = ARIMA(ts_train, order=(p+1,1,q+1))
        model_fit = model.fit()
        order.append(("("+str(p+1)+","+"1,"+str(q+1)+")"))
        aic.append(model_fit.aic)
        bic.append(model_fit.bic)
        adf_result.append(adfuller(ts_train)[1])
  result["order"]=order
  result["aic_score"]=aic
  result["bic_score"]=bic
  result["adf_result"]=adf_result

  all_result_train[i]=result
```

```
print(e+"\n")
for i in l:
  print(i+"->"+str((all_result_train[i]["adf_result"].unique()[0])))
```

```
covid_lockdown

RELIANCE.NS->0.008777219570481044
ADANIENT.NS->0.00044773515549302394
BHARTIARTL.NS->0.030177850602491053
SBIN.NS->0.030320323418881488
ICICIBANK.NS->0.04990614238386865
DRREDDY.NS->0.6560617010512816
ASHOKLEY.NS->0.5047168367061341
AUROPHARMA.NS->0.9698971177508432
JINDALSTEL.NS->0.9257774386591413
TATAMOTORS.NS->0.037453113628417856
^NSEI->0.04322598560408522
NIFTY_FIN_SERVICE.NS->0.18507233254435745
```

```
#jio launch
e=list(events_dict.keys())[2]
print(e)
for i in l:
  print(i)
  # Download historical data
  train_data = yf.download(i, start=date_jio_start, end=date_jio-
timedelta(1), interval="1d")
  val_data = yf.download(i, start=date_jio, end=date_jio_end,
interval="1d")
  k=[train_data,val_data]
  for j in k:
    #adding range data
    j["Range"]=j["High"]-j["Low"]
    #adding COB and PCOB average price data
    j["COB_AvG_Price"]=(j["High"]+j["Low"]+j["Adj Close"])/3
    j["PCOB_AvG_Price"]=(j["High"].shift(1)+j["Low"].shift(1)+j["Adj
Close"].shift(1))/3
    j["PCOB_Adj_Close"]=j["Adj Close"].shift(1)

    #adding volatility measure
    j["%vol_measure"]=(j["Range"]/ j["PCOB_AvG_Price"])*100
    j.dropna(inplace=True)

  data_dict_train[i]=k[0]
  data_dict_val[i]=k[1]
data_dict_train_e[e]= data_dict_train
data_dict_val_e[e]= data_dict_val

vol_df_e=pd.DataFrame(columns=["Ticker"])
vol_df_e["Ticker"]=ticker_symbols+index_ticker_symbols
```

```
vol=[]
vol_std=[]
for i in data_dict_train_e[e]:
  vol.append(data_dict_train_e[e][l[0]]["%vol_measure"].mean())
  vol_std.append(data_dict_train_e[e][l[0]]["%vol_measure"].std())
vol_df_e["Avg Volatility"]=vol
vol_df_e["Std Volatility"]=vol_std
vol_df_e

all_result_train=dict()
for i in vol_df_e["Ticker"]:
  print(i)
  mean_vol=vol_df_e[vol_df_e["Ticker"]==i]["Avg Volatility"]
[vol_df_e[vol_df_e["Ticker"]==i].index[0]]
  std_vol=vol_df_e[vol_df_e["Ticker"]==i]["Std Volatility"]
[vol_df_e[vol_df_e["Ticker"]==i].index[0]]
  temp_train=data_dict_train_e[e][i]
  #standardising the time series
  ts_train=(temp_train["%vol_measure"]-mean_vol)/std_vol
  # Define the ARIMA model
  order=[]
  aic=[]
  bic=[]
  adf_result=[]

result=pd.DataFrame(columns=["order","aic_score","bic_score","adf_resu
lt"])
  for p in range(5):
      for q in range(5):
        model = ARIMA(ts_train, order=(p+1,1,q+1))
        model_fit = model.fit()
        order.append(("("+str(p+1)+","+"1,"+str(q+1)+")"))
        aic.append(model_fit.aic)
        bic.append(model_fit.bic)
        adf_result.append(adfuller(ts_train)[1])
  result["order"]=order
  result["aic_score"]=aic
  result["bic_score"]=bic
  result["adf_result"]=adf_result

  all_result_train[i]=result

print(e+"\n")
for i in l:
  print(i+"->"+str((all_result_train[i]["adf_result"].unique()[0])))
```

```
jio_launch

RELIANCE.NS->1.1503264943229835e-24
ADANIENT.NS->3.345431968058091e-28
```

```
BHARTIARTL.NS->1.0906124418542077e-20
SBIN.NS->6.442437644856956e-09
ICICIBANK.NS->0.0026078189605612772
DRREDDY.NS->9.805182045022047e-23
ASHOKLEY.NS->1.1437544809333944e-05
AUROPHARMA.NS->1.2520302052880616e-19
JINDALSTEL.NS->0.0005726663288772431
TATAMOTORS.NS->0.3410238303405554
^NSEI->0.04385295265431146
NIFTY_FIN_SERVICE.NS->1.396139468486274e-05
```