final algo (1)

In [48]:

import numpy as np  
import pandas as pd  
import matplotlib.pyplot as plt  
import matplotlib.pyplot as plt  
from sklearn.ensemble import RandomForestRegressor  
from sklearn.metrics import mean\_squared\_error, r2\_score  
from sklearn.model\_selection import train\_test\_split  
from sklearn.ensemble import RandomForestClassifier  
from sklearn.metrics import accuracy\_score, confusion\_matrix  
import time  
import datetime  
import pandas as pd  
import numpy as np  
import pyotp  
import logging  
from sklearn.model\_selection import train\_test\_split  
from sklearn.metrics import accuracy\_score, mean\_squared\_error  
from sklearn.preprocessing import StandardScaler  
import joblib  
from sklearn.model\_selection import train\_test\_split  
from sklearn.metrics import mean\_squared\_error  
from sklearn.model\_selection import train\_test\_split  
from sklearn.preprocessing import StandardScaler  
from sklearn.metrics import accuracy\_score  
import joblib  
from sklearn.pipeline import Pipeline

In [49]:

import requests # Install requests module first.  
  
url = "https://public.coindcx.com/market\_data/candles?pair=B-BTC\_USDT&interval=1m" # Replace 'SNTBTC' with the desired market pair.  
  
response = requests.get(url)  
data = response.json()  
data\_f=pd.DataFrame(data)  
data\_f

Out[49]:

|  | open | high | low | volume | close | time |
| --- | --- | --- | --- | --- | --- | --- |
| 0 | 109464.82 | 109464.83 | 109463.01 | 1.08416 | 109463.01 | 1749533160000 |
| 1 | 109459.92 | 109477.06 | 109459.91 | 5.54338 | 109464.82 | 1749533100000 |
| 2 | 109449.09 | 109460.00 | 109400.00 | 26.41650 | 109459.92 | 1749533040000 |
| 3 | 109500.00 | 109500.01 | 109449.09 | 9.06334 | 109449.09 | 1749532980000 |
| 4 | 109572.04 | 109572.05 | 109500.00 | 12.03132 | 109500.00 | 1749532920000 |
| ... | ... | ... | ... | ... | ... | ... |
| 495 | 109192.06 | 109374.99 | 109132.00 | 91.67599 | 109281.68 | 1749503460000 |
| 496 | 108996.87 | 109316.33 | 108996.87 | 346.19421 | 109192.06 | 1749503400000 |
| 497 | 108808.33 | 108996.88 | 108808.33 | 129.54848 | 108996.87 | 1749503340000 |
| 498 | 108777.82 | 108808.34 | 108766.76 | 8.92314 | 108808.33 | 1749503280000 |
| 499 | 108771.26 | 108786.14 | 108771.25 | 2.93656 | 108777.82 | 1749503220000 |

500 rows × 6 columns

In [50]:

def ewm(data\_f):  
 data\_f["close"] = pd.to\_numeric(data\_f["close"], errors='coerce')  
  
 data\_f["ewm9"] = data\_f["close"].ewm(com=9).mean()  
 data\_f["ewm15"] = data\_f["close"].ewm(com=15).mean()  
  
 return data\_f["ewm9"], data\_f["ewm15"]

In [51]:

def william\_p():  
 close\_p=data\_f["close"].iloc[0]  
 h\_high=data\_f["high"].rolling(window=14).max()  
 l\_low=data\_f["low"].rolling(window=14).min()  
 william =((h\_high-close\_p)/(h\_high-l\_low))\*(-100)  
 return william

In [52]:

def bollinger\_band(data\_f):  
  
 data\_f["close"] = pd.to\_numeric(data\_f["close"], errors='coerce')  
 data\_f["MB"] = data\_f["close"].rolling(window=20).mean() # Middle Band (Exponential Moving Average)  
 data\_f["LB"] = data\_f["MB"] - 2 \* data\_f["close"].ewm(com=9).std() # Lower Band  
 data\_f["HB"] = data\_f["MB"] + 2 \* data\_f["close"].ewm(com=9).std() # Higher Band  
 data\_f["b\_width"] = data\_f["HB"] - data\_f["LB"] # Band Width  
 data\_f.dropna(subset=["MB", "HB", "LB"], inplace=True)  
   
  
 return data\_f["LB"], data\_f["HB"], data\_f["MB"]

In [53]:

def volatility\_percentage():  
 range=data\_f['high']-data\_f['low']  
 normal\_range=range/data\_f['close']  
 percen\_volatility=normal\_range\*100  
 return percen\_volatility

In [54]:

def ichimoku\_cloud():  
 conversion\_line=(data\_f["high"].rolling(window=9).max()+data\_f["low"].rolling(window=9).min())/2  
 base\_line=(data\_f["high"].rolling(window=26).max()+data\_f["low"].rolling(window=26).min())/2  
 leading\_span\_A=((base\_line+conversion\_line)/2).shift(26)  
 leading\_span\_B=((data\_f["high"].rolling(window=52).max()+data\_f["low"].rolling(window=52).min())/2).shift(26)  
 lagging\_span=data\_f["close"].shift(-26)  
 return conversion\_line,base\_line,leading\_span\_A,leading\_span\_B,lagging\_span

In [55]:

def macd(data\_f):  
  
 data\_f["MACD\_LINE"] = data\_f["close"].ewm(span=12).mean() - data\_f["close"].ewm(span=26).mean()  
  
  
 data\_f["SIGNAL\_LINE"] = data\_f["MACD\_LINE"].ewm(span=9).mean()  
  
  
 data\_f["MACD\_HIST"] = data\_f["MACD\_LINE"] - data\_f["SIGNAL\_LINE"]  
  
  
 return data\_f["MACD\_LINE"], data\_f["SIGNAL\_LINE"], data\_f["MACD\_HIST"]

In [56]:

def parabolic\_sar(data\_f, af\_start=0.02, af\_step=0.02, af\_max=0.2):  
 high = data\_f['high'].values  
 low = data\_f['low'].values  
 close = data\_f['close'].values  
   
 length = len(data\_f)  
 sar = [None] \* length  
 trend = [None] \* length  
  
 # Initialization  
 af = af\_start  
 ep = high[0]  
 sar[1] = low[0] # Assume starting trend is up  
 trend[1] = 'up'  
  
 for i in range(2, length):  
 prev\_sar = sar[i - 1]  
 prev\_ep = ep  
 prev\_af = af  
 prev\_trend = trend[i - 1]  
  
 if prev\_trend == 'up':  
 sar\_raw = prev\_sar + prev\_af \* (prev\_ep - prev\_sar)  
 sar[i] = min(sar\_raw, low[i - 1], low[i - 2])  
  
 if high[i] > prev\_ep:  
 ep = high[i]  
 af = min(prev\_af + af\_step, af\_max)  
 else:  
 ep = prev\_ep  
 af = prev\_af  
  
 if low[i] < sar[i]:  
 # Trend reversal to down  
 trend[i] = 'down'  
 sar[i] = prev\_ep  
 ep = low[i]  
 af = af\_start  
 else:  
 trend[i] = 'up'  
  
 else: # prev\_trend == 'down'  
 sar\_raw = prev\_sar + prev\_af \* (prev\_ep - prev\_sar)  
 sar[i] = max(sar\_raw, high[i - 1], high[i - 2])  
  
 if low[i] < prev\_ep:  
 ep = low[i]  
 af = min(prev\_af + af\_step, af\_max)  
 else:  
 ep = prev\_ep  
 af = prev\_af  
  
 if high[i] > sar[i]:  
 # Trend reversal to up  
 trend[i] = 'up'  
 sar[i] = prev\_ep  
 ep = high[i]  
 af = af\_start  
 else:  
 trend[i] = 'down'  
  
 data\_f['parabolic\_sar'] = sar  
 data\_f['trend'] = trend  
 return data\_f

In [57]:

def stochastic(data\_f):  
 # Ensure the required columns are numeric  
 data\_f["high"] = pd.to\_numeric(data\_f["high"], errors="coerce")  
 data\_f["low"] = pd.to\_numeric(data\_f["low"], errors="coerce")  
 data\_f["close"] = pd.to\_numeric(data\_f["close"], errors="coerce")  
  
  
 # Calculate 14-day high and low  
 data\_f["high14"] = data\_f["high"].rolling(window=14).max()  
 data\_f["low14"] = data\_f["low"].rolling(window=14).min()  
  
 denominator = data\_f["high14"] - data\_f["low14"]  
 denominator.replace(0, pd.NA)  
  
 # Calculate %K  
 data\_f["%k"] = (  
 ((data\_f["close"] - data\_f["low14"]) / (data\_f["high14"] - data\_f["low14"]))  
 \* 100  
 ).rolling(window=3).mean()  
  
 # Handle cases where the denominator is zero  
 data\_f["%k"].where(data\_f["high14"] != data\_f["low14"], other=0)  
  
 # Calculate %D as the 3-day rolling mean of %K  
 data\_f["%d"] = data\_f["%k"].rolling(window=3).mean()  
  
 # Return %K and %D  
 return data\_f["%k"],data\_f["%d"]

In [58]:

def donchain\_channel():  
 highest\_high=data\_f["high"].rolling(window=20).max()  
 lowest\_low=data\_f["low"].rolling(window=20).min()  
 middle\_band=(highest\_high+lowest\_low)/2  
 return highest\_high,lowest\_low,middle\_band

In [59]:

def roc(data\_f):  
 roc=((data\_f["close"]-data\_f["close"].shift(10))/data\_f["close"].shift(10)\*100)  
 return roc

In [60]:

def momentum\_indi():  
 momentum=(data\_f["close"]/data\_f["close"].shift(10)-1)\*100  
 return momentum

In [61]:

def commodity\_ci():  
 typical\_price=(data\_f["close"]+data\_f["low"]+data\_f["high"])/3  
 sm\_typical\_price=typical\_price.rolling(window=20).mean()  
 mean\_deviation = typical\_price.rolling(window=20).apply(lambda x: (abs(x - x.mean())).mean(), raw=False)  
 commodity\_channel\_index=(typical\_price-sm\_typical\_price)/(0.015\*mean\_devation)  
 return commodity\_channel\_index

In [62]:

def rsi(data\_f):  
 data\_f["change"]=data\_f["close"]-data\_f["close"].shift(1)  
 data\_f["gain"]=np.where(data\_f["change"]>=0,data\_f["change"],0)  
 data\_f["loss"]=np.where(data\_f["change"]<0,-1\*data\_f["change"],0)  
 data\_f["avg\_gain"]=data\_f["gain"].ewm(alpha=1/14,min\_periods=14).mean()  
 data\_f["avg\_loss"]=data\_f["loss"].ewm(alpha=1/14,min\_periods=14).mean()  
 data\_f["rsi"]=100-(100/(1+data\_f["avg\_gain"]/data\_f["avg\_loss"]))  
 return data\_f["rsi"]

In [63]:

def atr\_adx(data\_f):  
 # Ensure all data is numeric, coercing errors to NaN  
 data\_f = data\_f.apply(pd.to\_numeric, errors='coerce')  
  
 data\_f["h-l"] = data\_f["high"] - data\_f["low"]  
 data\_f["h-cp"] = data\_f["high"] - data\_f["close"].shift(1)  
 data\_f["l-cp"] = data\_f["low"] - data\_f["close"].shift(1)  
 data\_f["tr"] = data\_f[["h-l", "h-cp", "l-cp"]].max(axis=1)  
 data\_f["$High"]=data["high"]-data\_f["high"].shift(1)  
 data\_f["$Low"]=data["low"].shift(1)-data\_f["low"]  
 data\_f["atr"] = data\_f["tr"].rolling(14).mean()  
  
 return data\_f["atr"], data\_f["tr"]

In [85]:

def pivot\_standard():  
 central\_pivot=(data\_f["low"]+data\_f["high"]+data\_f["close"])/3  
 first\_resistance=2\*central\_pivot-data\_f["low"]  
 first\_support=2\*central\_pivot-data\_f["high"]  
 second\_resistance=central\_pivot+data\_f["high"]-data\_f["low"]  
 second\_support=central\_pivot-data\_f["high"]+data\_f["low"]  
 third\_resistance=2\*central\_pivot+data\_f["high"]-2\*data\_f["low"]  
 third\_support=2\*central\_pivot-2\*data\_f["high"]+data\_f["low"]  
 return central\_pivot,first\_support,second\_support,third\_support,first\_resistance,second\_resistance,third\_resistance

In [87]:

def pivot\_cammerilla():  
 delta=data\_f["high"]-data\_f["low"]  
 h1=data\_f["close"]+delta\*(1.1/12)  
 h2=data\_f["close"]+delta\*(1.1/6)  
 h3=data\_f["close"]+delta\*(1.1/4)  
 h4=data\_f["close"]+delta\*(1.1/2)  
 l1=data\_f["close"]-delta\*(1.1/12)  
 l2=data\_f["close"]-delta\*(1.1/6)  
 l3=data\_f["close"]-delta\*(1.1/4)  
 l4=data\_f["close"]-delta\*(1.1/2)  
 return l1,l2,l3,l4,h1,h2,h3,h4

In [91]:

def target\_close(data\_f):  
 # Ensure 'intc' column is numeric  
 data\_f["close"] = pd.to\_numeric(data\_f["close"], errors='coerce')  
  
 # Initialize 'signal' column with 0  
 data\_f['signal'] = 0  
  
 # Assign signals based on correct interpretation of percentage change  
 data\_f.loc[data\_f['close'].pct\_change(2) > 0.0007, 'signal'] = 2 # Buy Put  
 data\_f.loc[data\_f['close'].pct\_change(2) < -0.0007, 'signal'] = 1 # Buy Call  
  
 return data\_f["signal"]

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]: