In [ ]:
```
#virtual machine
from google.cloud import compute_v1


#tensorflow
import tensorflow as tf
from tensorflow import keras


# Common imports
import numpy as np
import pandas as pd
import os
import os.path
import urllib

# To plot pretty figures
import matplotlib.pyplot as plt
```

In [ ]:
```
from google.cloud import storage


def authenticate_implicit_with_adc(project_id="your-google-cloud-project-id"):
    """
    When interacting with Google Cloud Client libraries, the library can auto-detect t
    credentials to use.

    // TODO(Developer):
    //  1. Before running this sample,
    //  set up ADC as described in https://cloud.google.com/docs/authentication/exterr
    //  2. Replace the project variable.
    //  3. Make sure that the user account or service account that you are using
    //  has the required permissions. For this sample, you must have "storage.buckets.
    Args:
        project_id: The project id of your Google Cloud project.
    """

    # This snippet demonstrates how to list buckets.
    # *NOTE*: Replace the client created below with the client required for your appli
    # Note that the credentials are not specified when constructing the client.
    # Hence, the client library will look for credentials using ADC.
    storage_client = storage.Client(project=project_id)
    buckets = storage_client.list_buckets()
    print("Buckets:")
    for bucket in buckets:
        print(bucket.name)
    print("Listed all storage buckets.")


    authenticate_implicit_with_adc(project_id="agent-model")
```

In [ ]:
```
from __future__ import print_function

import os.path

from google.auth.transport.requests import Request
from google.oauth2.credentials import Credentials
```

```
from google_auth_oauthlib.flow import InstalledAppFlow
from googleapiclient.discovery import build
from googleapiclient.errors import HttpError

# If modifying these scopes, delete the file token.json.
SCOPES = ['https://www.googleapis.com/oauth2/v1/certs"']

# The ID of a sample document.
DOCUMENT_ID = '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'


def main():
    """Shows basic usage of the Docs API.
    Prints the title of a sample document.
    """
    creds = None
    # The file token.json stores the user's access and refresh tokens, and is
    # created automatically when the authorization flow completes for the first
    # time.
    if os.path.exists(r'C:\Users\zsamach\Documents\client_secret_894200421030-tmlp7b17
        creds = Credentials.from_authorized_user_file(r'C:\Users\zsamach\Documents\cli
    # If there are no (valid) credentials available, let the user log in.
    if not creds or not creds.valid:
        if creds and creds.expired and creds.refresh_token:
            creds.refresh(Request())
        else:
            flow = InstalledAppFlow.from_client_secrets_file(
                open(r'C:\Users\zsamach\Documents\client_secret_894200421030-tmlp7b17g
            creds = flow.run_local_server(port=0)
        # Save the credentials for the next run
        with open('token.json', 'w') as token:
            token.write(creds.to_json())

    try:
        service = build('docs', 'v1', credentials=creds)

        # Retrieve the documents contents from the Docs service.
        document = service.documents().get(documentId=DOCUMENT_ID).execute()

        print('The title of the document is: {}'.format(document.get('title')))
    except HttpError as err:
        print(err)


if __name__ == '__main__':
    main()
```

In [ ]:
```
networks_client = compute_v1.NetworksClient()
for network in networks_client.list(project='agent-model'):
    print(network)
```

In [ ]:
```
from polygon import RESTClient
```

In [ ]:
```
from datetime import date, datetime
from typing import Any, Optional
import pandas as pd
from requests.adapters import HTTPAdapter
from urllib3.util.retry import Retry
```

```
markets = ['crypto', 'stocks', 'fx']

class MyRESTClient(RESTClient):
    def __init__(self, auth_key: str='xz0v5qmSd2ZFCW1Q8A4rOIsIWtk5cht2', timeout:int=5
        super().__init__(auth_key)
        retry_strategy = Retry(total=10,
                               backoff_factor=10,
                               status_forcelist=[429, 500, 502, 503, 504])
        adapter = HTTPAdapter(max_retries=retry_strategy)
        self._session.mount('https://', adapter)
```

In [ ]:
```
client = MyRESTClient(['xz0v5qmSd2ZFCW1Q8A4rOIsIWtk5cht2'])
```

In [ ]:
```
class MyRESTClient(RESTClient):
    def __init__(self, auth_key: str=['xz0v5qmSd2ZFCW1Q8A4rOIsIWtk5cht2'], timeout:int
        super().__init__(auth_key)
        retry_strategy = Retry(total=10,
                               backoff_factor=10,
                               status_forcelist=[429, 500, 502, 503, 504])
        adapter = HTTPAdapter(max_retries=retry_strategy)
        self._session.mount('https://', adapter)

    def get_tickers(self, market:str=None) -> pd.DataFrame:
        if not market in markets:
            raise Exception(f'Market must be one of {markets}.')

        resp = self.reference_tickers_v3(market=market)
        if hasattr(resp, 'results'):
            df = pd.DataFrame(resp.results)

            while hasattr(resp, 'next_url'):
                resp = self.reference_tickers_v3(next_url=resp.next_url)
                df = df.append(pd.DataFrame(resp.results))

            if market == 'crypto':
                # Only use USD pairings.
                df = df[df['currency_symbol'] == 'USD']
                df['name'] = df['base_currency_name']
                df = df[['ticker', 'name', 'market', 'active']]

            df = df.drop_duplicates(subset='ticker')
            return df
        return None
```

In [ ]:
```
client = MyRESTClient(['xz0v5qmSd2ZFCW1Q8A4rOIsIWtk5cht2'])
df = client.get_tickers(market='crypto')
df
```

In [ ]:
```
class MyRESTClient(RESTClient):
    def __init__(self, auth_key: str=['xz0v5qmSd2ZFCW1Q8A4rOIsIWtk5cht2'], timeout:int
        super().__init__(auth_key)
        retry_strategy = Retry(total=10,
                               backoff_factor=10,
                               status_forcelist=[429, 500, 502, 503, 504])
        adapter = HTTPAdapter(max_retries=retry_strategy)
        self._session.mount('https://', adapter)

    def get_tickers(self, market:str=None) -> pd.DataFrame:
```

```
        if not market in markets:
            raise Exception(f'Market must be one of {markets}.')

        resp = self.reference_tickers_v3(market=market)
        if hasattr(resp, 'results'):
            df = pd.DataFrame(resp.results)

            while hasattr(resp, 'next_url'):
                resp = self.reference_tickers_v3(next_url=resp.next_url)
                df = df.append(pd.DataFrame(resp.results))

            if market == 'crypto':
                # Only use USD pairings.
                df = df[df['currency_symbol'] == 'USD']
                df['name'] = df['base_currency_name']
                df = df[['ticker', 'name', 'market', 'active']]

            df = df.drop_duplicates(subset='ticker')
            return df
        return None

    def get_bars(self, market:str=None, ticker:str=None, multiplier:int=1,
                 timespan:str='minute', from_:date=None, to:date=None) -> pd.DataFrame

        if not market in markets:
            raise Exception(f'Market must be one of {markets}.')

        if ticker is None:
            raise Exception('Ticker must not be None.')

        from_ = from_ if from_ else date(2000,1,1)
        to = to if to else date.today()

        if market == 'crypto':
            resp = self.crypto_aggregates(ticker, multiplier, timespan,
                                          from_.strftime('%Y-%m-%d'), to.strftime('%Y-
                                          limit=50000)
            df = pd.DataFrame(resp.results)
            last_minute = 0
            while resp.results[-1]['t'] > last_minute:
                last_minute = resp.results[-1]['t'] # Last minute in response
                last_minute_date = datetime.fromtimestamp(last_minute/1000).strftime('
                resp = self.crypto_aggregates(ticker, multiplier, timespan,
                                              last_minute_date, to.strftime('%Y-%m-%d'),
                                              limit=50000)
                new_bars = pd.DataFrame(resp.results)
                df = df.append(new_bars[new_bars['t'] > last_minute])

            df['date'] = pd.to_datetime(df['t'], unit='ms')
            df = df.rename(columns={'o':'open',
                                    'h':'high',
                                    'l':'low',
                                    'c':'close',
                                    'v':'volume',
                                    'vw':'vwap',
                                    'n':'transactions'})
            df = df[['date','open','high','low','close','volume']]

            return df
        return None
```

In [ ]:
```
start = datetime(2020,1,1)
end = datetime(2022,12,1)
client = MyRESTClient(['xz0v5qmSd2ZFCW1Q8A4rOIsIWtk5cht2'])
df1 = client.get_bars(market='crypto', ticker='X:BTCUSD', from_=start,to=end)
```

In [ ]:
```
data=df1

data
```

In [ ]:
```
## Calculate the MACD and Signal Line indicators
## Calculate the Short Term Exponential Moving Average
ShortEMA = data.close.ewm(span=30, adjust=False).mean()
## Calculate the Long Term Exponential Moving Average
LongEMA = data.close.ewm(span=60, adjust=False).mean()
## Calculate the Moving Average Convergence/Divergence (MACD)
data['MACD'] = ShortEMA - LongEMA
## Calcualte the signal line
data['signal'] = data['MACD'].ewm(span=15, adjust=False).mean()
```

In [ ]:
```
from statistics import median

#classical momentum gauge at t=0 day tenor

ChangeInHigh=data.high.shift(+60)/data.close.shift(+1)
ChangeInLow=data.close.shift(+1)/data.low.shift(+60)
TrailingStd=data.close.shift(+1).rolling(60).std()/data.close.shift(+1).rolling(60).me
HighDifferential=ChangeInHigh/TrailingStd
LowDifferential=ChangeInLow/TrailingStd

Momentum=(LowDifferential-HighDifferential)
data['Momentum%']=Momentum/100
```

In [ ]:
```
from statistics import median

#classical momentum gauge at 1 day tenor (T=1)

ChangeInHighMinus_1=data.high.shift(+90)/data.close.shift(+2)
ChangeInLowMinus_1=data.close.shift(+2)/data.low.shift(+90)
TrailingStdMinus_1=data.close.shift(+2).rolling(60).std()/data.close.shift(+2).rolling
HighDifferentialMinus_1=ChangeInHighMinus_1/TrailingStdMinus_1
LowDifferentialMinus_1=ChangeInLowMinus_1/TrailingStdMinus_1

MomentumMinus_1=(LowDifferentialMinus_1-HighDifferentialMinus_1)
data['Momentum%Minus_1']=Momentum/100
```

In [ ]:
```
from statistics import median

#classical momentum gauge at 2 day tenor (T=2)

ChangeInHighMinus_2=data.high.shift(+120)/data.close.shift(+3)
ChangeInLowMinus_2=data.close.shift(+3)/data.low.shift(+120)
TrailingStdMinus_2=data.close.shift(+3).rolling(60).std()/data.close.shift(+3).rolling
HighDifferentialMinus_2=ChangeInHighMinus_2/TrailingStdMinus_2
LowDifferentialMinus_2=ChangeInLowMinus_2/TrailingStdMinus_2

MomentumMinus_2=(LowDifferentialMinus_2-HighDifferentialMinus_2)
data['Momentum%Minus_2']=MomentumMinus_2/100
```

In [ ]:
```
from statistics import median

#classical momentum gauge at 3 day tenor (T=3)

ChangeInHighMinus_3=data.high.shift(+150)/data.close.shift(+4)
ChangeInLowMinus_3=data.close.shift(+4)/data.low.shift(+150)
TrailingStdMinus_3=data.close.shift(+4).rolling(60).std()/data.close.shift(+4).rolling
HighDifferentialMinus_3=ChangeInHighMinus_3/TrailingStdMinus_3
LowDifferentialMinus_3=ChangeInLowMinus_3/TrailingStdMinus_3

MomentumMinus_3=(LowDifferentialMinus_3-HighDifferentialMinus_3)
data['Momentum%Minus_3']=MomentumMinus_3/100
```

In [ ]:
```
from statistics import median

#classical momentum gauge at 4 day tenor (T=4)

ChangeInHighMinus_4=data.high.shift(+180)/data.close.shift(+5)
ChangeInLowMinus_4=data.close.shift(+5)/data.low.shift(+180)
TrailingStdMinus_4=data.close.shift(+5).rolling(60).std()/data.close.shift(+5).rolling
HighDifferentialMinus_4=ChangeInHighMinus_4/TrailingStdMinus_4
LowDifferentialMinus_4=ChangeInLowMinus_4/TrailingStdMinus_4

MomentumMinus_4=(LowDifferentialMinus_4-HighDifferentialMinus_4)
data['Momentum%Minus_4']=MomentumMinus_4/100
```

In [ ]:
```
#DOWNLOAD_DIR1 = "C:/Users/mary jane/Downloads/"
#filename1 = "BTC_17_21.csv"
#data= pd.read_csv(DOWNLOAD_DIR1+filename1)
#data
```

In [ ]:
```
data=data.iloc[:,:13]
```

In [ ]:
```
import pandas as pd
from matplotlib import pyplot as plt
from statsmodels.tsa.holtwinters import ExponentialSmoothing as HWES
```

In [ ]:
```
#Naive Bayes Predictor
import pandas as pd
from matplotlib import pyplot as plt
from statsmodels.tsa.holtwinters import SimpleExpSmoothing  as HWES
from tqdm import tqdm
import warnings
warnings.filterwarnings("ignore")
tqdm.pandas()
%matplotlib inline
import time
import pylab as pl
from IPython import display
import math


df_train = pd.DataFrame()
MAE=0
i=0
for i in tqdm(range(len(data))):
    df_train =data['close'].iloc[0:i]
```

```
    def mean_absolute_error(Y_actual,Y_Predicted):
        mape = np.mean(np.abs((Y_actual - Y_Predicted)/Y_actual))
        return mape

    if i <1000:
        pass
    else:
        if i==1000:
            model = HWES(df_train).fit()
            forecast_2hr=np.array(model.forecast(60)[1:2])
            MAE=np.array((1*(mean_absolute_error(data['close'][1001:i+2],forecast_2hr[


        else:
            last_forecast=forecast_2hr[-1]
            df_train =data['close'].iloc[i-1000:i]
            model = HWES(df_train).fit()

            if np.isnan(model.forecast(60)[59:].values)==True:
                forecast_2hr=np.append(forecast_2hr,last_forecast)
            else:

                    forecast_2hr=np.append(forecast_2hr,model.forecast(60)[1:2])
            #if i <1012:
                #MAE=np.append(MAE,((1*(mean_absolute_error(data['close'][1001:i+2],fc
            #else:
                #MAE=np.append(MAE,((1*(mean_absolute_error(data['close'][(i+2)-12:i+2



            #if i % 500 == 0:
                #try:
                    #plt.plot(MAE)
                    #display.display(plt.show())
                    #display.clear_output(wait=True)

                #except KeyboardInterrupt:
                    #break
```

In [ ]:
```
naive=pd.DataFrame(forecast_2hr)

naive
```

In [ ]:
```
data=data[1000:]
data=data.reset_index()
data=data.drop('index',axis=1)

data
```

In [ ]:
```
data=data.merge(naive,left_index=True,right_index=True)
data=data.dropna()
data
```

In [ ]:
```
data=data.drop('date', axis=1)
data.to_csv('BTC_17_21.csv',index=False)
```

In [ ]:
```
DOWNLOAD_DIR1 = "C:/Users/mary jane/Downloads/"
filename1 = "BTC_17_21.csv"
data= pd.read_csv(DOWNLOAD_DIR1+filename1)
data
```

In [ ]:
```
data=data.dropna()
cols = data.columns
data=data[cols].apply(lambda row: ','.join(row.values.astype(str)), axis=1)

data
```

In [ ]:
```
import ast

test=pd.DataFrame(data)
test=test[0].apply(ast.literal_eval)
test
```

In [ ]:
```
import random
from collections import deque
```

In [ ]:
```
import random
from collections import deque

def formatPrice(n):
    return("-Rs." if n<0 else "Rs.")+list(map('{:.2f}%'.format,n))

def sigmoid(x):
    return 1/(1+np.exp(-x))
def getState(data, t, n):
    d = t - n + 1
    data=pd.DataFrame(data)
    data=data[0].apply(ast.literal_eval)
    data=list(data)
    block = data[d:t + 1] if d >= 0 else -d * [data[0]] + data[0:t + 1] # pad with t0
    res = []
    for i in range(n - 1):
        array1 = np.array(block[i])
        array2 = np.array(block[i + 1])
        res.append(sigmoid(np.subtract(array2, array1)))

    return np.array([res])
```

In [ ]:
```
from keras.layers import BatchNormalization

class Agent:
    def __init__(self, state_size, is_eval=False, model_name=""):
            self.state_size = state_size # normalized previous days
            self.action_size = 4 # hold, buy,buy all,  sell all
            self.memory = deque(maxlen=720)
            self.NAV=[]
            self.inventory = []
            self.action_list=[]
            self.model_name = model_name
            self.is_eval = is_eval
```

```
            self.gamma = 0.95
            self.epsilon = .25
            self.epsilon_min = 0.01
            self.epsilon_decay = 0.995
            self.model = tf.keras.models.load_model(model_name) if is_eval else self._

    def _model(self):
        model = tf.keras.models.Sequential()
        model.add(tf.keras.layers.GRU(50, return_sequences=True))
        model.add(tf.keras.layers.GRU(30, return_sequences=True))
        model.add(tf.keras.layers.GRU(20, return_sequences=True))
        model.add(tf.keras.layers.GRU(10))
        model.add(keras.layers.Dense(4, activation='softmax'))
        model.compile(loss="sparse_categorical_crossentropy", optimizer=keras.optimize
        return model
    def act(self, state):
        if not self.is_eval and random.random()<= self.epsilon:
            return random.randrange(self.action_size)

        options = self.model.predict(np.array(state))
        return np.argmax(options[0])
    def expReplay(self, batch_size):
        mini_batch = []
        l = len(self.memory)
        for i in range(l - batch_size + 1, l):
            mini_batch.append(self.memory[i])
        for state, action, reward, next_state, done in mini_batch:
            target = reward
            if not done:
                target = reward + self.gamma * np.amax(self.model.predict(next_state)[
            target_f = self.model.predict(state)
            target_f[0][action] = target
            self.model.fit(state, target_f,epochs=5, verbose=0)
        if self.epsilon > self.epsilon_min:
            self.epsilon *= self.epsilon_decay
    def buy(self, initial_money):
        starting_money = initial_money
        states_sell = []
        states_buy = []
        inventory = []
        state = self.getState(0)
        for t in range(0, len(self.trend) - 1, self.skip):
            action = self.act(state)
            next_state = self.getState(t + 1)
            if action == 1 and initial_money >= self.trend[t] and t < (len(self.trend)
                inventory.append(self.trend[t])
                initial_money -= self.trend[t]
                states_buy.append(t)
                print('day %d: buy 1 unit at price %f, total balance %f'% (t, self.tre
            elif action == 2 and len(inventory):
                bought_price = inventory.pop(0)
                initial_money += self.trend[t]
                states_sell.append(t)
                try:
                    invest = ((close[t] - bought_price) / bought_price) * 100
                except:
                    invest = 0
                print(
                    'day %d, sell 1 unit at price %f, investment %f %%, total balance
                    % (t, close[t], invest, initial_money)
```

```
                )
            state = next_state
        invest = ((initial_money - starting_money) / starting_money) * 100
        total_gains = initial_money - starting_money
        return states_buy, states_sell, total_gains, invest
```

In [ ]:
```
test=list(test)
btc = [item[3] for item in test]
btc
```

In [ ]:
```
data=np.array(data)
```

In [ ]:
```
import math
from tqdm import tqdm
import time
batch_size = (10000)
stock_name = "BTC"
window_size = 60*24
episode_count = 5
stock_name = str(stock_name)
window_size = int(window_size)
episode_count = int(episode_count)
trend=btc
agent = Agent(window_size)
%matplotlib inline
l = len(data) - 1


np.random.seed(42)
tf.random.set_seed(42)
keras.backend.clear_session()
for e in range(episode_count + 1):
    print("Episode " + str(e) + "/" + str(episode_count))
    state = getState(data, 0, window_size + 1)
    total_profit = 0
    NAV=(btc[0]*10)
    Liquidity=(btc[0]*10)
    buyallPrice=btc[0]
    LiquidityAll=0
    inventory=0
    agent.inventory = []
    NAV_Series=np.array((((NAV)/(btc[0]*10)))/(((btc[0])/btc[0])))
    Block_Price=btc[0]




    for t in tqdm(range(l)):



        action = agent.act(np.array(state))
        price=btc[t]


        if t==0:
            Liquidity=price*10
            Block_Liquidity=Liquidity
```

```
        elif t>=1:
            Liquidity=((btc[0]*10)-sum(agent.inventory)+total_profit)
        if btc[t]>=Liquidity:
            action = 0 or 3

        # sit
        next_state = getState(data, t + 1, window_size + 1)
        reward = 0


        if action == 1: # buy

            agent.inventory.append(btc[t])
            inventory=len(agent.inventory)
            Liquidity=Liquidity-(agent.inventory[-1])
            NAV=sum(agent.inventory)+Liquidity




        elif action == 2: # buy all

            buyingPower=math.trunc(Liquidity/price)
            Block_Price=btc[t]
            Block_Liquidity=Liquidity
            agent.inventory.append(Block_Price*buyingPower)
            Liquidity=Liquidity-(Block_Price*buyingPower)

            NAV=sum(agent.inventory)+Liquidity



        elif action == 3:
            while sum(agent.inventory) >0 and len(agent.inventory)>0: # sell all
                if agent.memory[-1]!=2 or(agent.memory[-1]!=2 and agent.memory[-2]!=2)
                    Block_Shares_Owned=0
                else:
                    Block_Shares_Owned=math.trunc(Block_Liquidity/Block_Price)
                if Block_Shares_Owned!=0:
                    bought_price =  sum(agent.inventory)/(len(agent.inventory)+Block_S
                else:
                    bought_price =  sum(agent.inventory)/(len(agent.inventory))
                if ((btc[t] - bought_price)*(buyingPower+inventory))<0:
                    total_profit += ((btc[t] - bought_price)*(buyingPower+inventory))
                else:
                    total_profit += ((btc[t] - bought_price)*(buyingPower+inventory)*(


                agent.inventory=[]
                Block_Shares_Owned=0
                inventory=0
                buyingPower=0
                bought_price=0
                buyallPrice=btc[t]
                LiquidityAll=Liquidity
                NAV=(btc[0]*10)+total_profit
```

```
                Liquidity=NAV-sum(agent.inventory)

        elif action ==0:
            while sum(agent.inventory)>0 and t>0:
                if agent.memory[-1]!=2 or(agent.memory[-1]!=2 and agent.memory[-2]!=2)
                    Block_Shares_Owned=0
                else:
                    Block_Shares_Owned=math.trunc(Block_Liquidity/Block_Price)
                if Block_Shares_Owned!=0:
                    bought_price =  sum(agent.inventory)/(len(agent.inventory)+Block_S
                else:
                    bought_price =  sum(agent.inventory)/(len(agent.inventory))
                NAV=sum(agent.inventory)+((btc[t] - bought_price)*(buyingPower+inventc









        NAV_Series=np.append(NAV_Series,(((NAV)/(btc[0]*10)))/(((btc[t])/btc[0])))

        reward = max((((((NAV)/(btc[0]*10)))/(((price)/btc[0])))-1), 0)









        done = True if t == l - 1 else False

        agent.memory.append((action))
        state = next_state




        if done:
            print("---------------------------------")
            print(str(t)+" | "+"Price: " + str(price)+" Total Profit: "+str(total_prof
            print("---------------------------------")
```

In [ ]:
```
agent.model.save('BTC_AgentModel.h5')
```

In [ ]:
```
from datetime import datetime
from time import time, sleep

start = datetime(2022,1,1)

client = MyRESTClient(['xz0v5qmSd2ZFCW1Q8A4rOIsIWtk5cht2'])
df2 = client.get_bars(market='crypto', ticker='X:BTCUSD', from_=start)
df2
```

In [ ]:
```
df2
```

In [ ]:
```
data2=df2
```

In [ ]:
```
## Calculate the MACD and Signal Line indicators
## Calculate the Short Term Exponential Moving Average
```

```
ShortEMA = data2.close.ewm(span=1440, adjust=False).mean()
## Calculate the Long Term Exponential Moving Average
LongEMA = data2.close.ewm(span=43200, adjust=False).mean()
## Calculate the Moving Average Convergence/Divergence (MACD)
data2['MACD'] = ShortEMA - LongEMA
## Calcualte the signal line
data2['signal'] = data2['MACD'].ewm(span=9, adjust=False).mean()
```

In [ ]:
```
from statistics import median

#classical momentum gauge at t=0 day tenor

ChangeInHigh=data2.high.shift(+25)/data2.close.shift(+1)
ChangeInLow=data2.close.shift(+1)/data2.low.shift(+25)
TrailingStd=data2.close.shift(+1).rolling(24).std()/data2.close.shift(+1).rolling(24).
HighDifferential=ChangeInHigh/TrailingStd
LowDifferential=ChangeInLow/TrailingStd

Momentum=(LowDifferential-HighDifferential)
data2['Momentum%']=Momentum/100
```

In [ ]:
```
from statistics import median

#classical momentum gauge at 1 day tenor (T=1)

ChangeInHighMinus_1=data2.high.shift(+26)/data2.close.shift(+2)
ChangeInLowMinus_1=data2.close.shift(+2)/data2.low.shift(+26)
TrailingStdMinus_1=data2.close.shift(+2).rolling(24).std()/data2.close.shift(+2).rolli
HighDifferentialMinus_1=ChangeInHighMinus_1/TrailingStdMinus_1
LowDifferentialMinus_1=ChangeInLowMinus_1/TrailingStdMinus_1

MomentumMinus_1=(LowDifferentialMinus_1-HighDifferentialMinus_1)
data2['Momentum%Minus_1']=Momentum/100
```

In [ ]:
```
from statistics import median

#classical momentum gauge at 2 day tenor (T=2)

ChangeInHighMinus_2=data2.high.shift(+27)/data2.close.shift(+3)
ChangeInLowMinus_2=data2.close.shift(+3)/data2.low.shift(+27)
TrailingStdMinus_2=data2.close.shift(+3).rolling(24).std()/data2.close.shift(+3).rolli
HighDifferentialMinus_2=ChangeInHighMinus_2/TrailingStdMinus_2
LowDifferentialMinus_2=ChangeInLowMinus_2/TrailingStdMinus_2

MomentumMinus_2=(LowDifferentialMinus_2-HighDifferentialMinus_2)
data2['Momentum%Minus_2']=MomentumMinus_2/100
```

In [ ]:
```
from statistics import median

#classical momentum gauge at 3 day tenor (T=3)

ChangeInHighMinus_3=data2.high.shift(+28)/data2.close.shift(+4)
ChangeInLowMinus_3=data2.close.shift(+4)/data2.low.shift(+28)
TrailingStdMinus_3=data2.close.shift(+4).rolling(24).std()/data2.close.shift(+4).rolli
HighDifferentialMinus_3=ChangeInHighMinus_3/TrailingStdMinus_3
LowDifferentialMinus_3=ChangeInLowMinus_3/TrailingStdMinus_3
```

```
MomentumMinus_3=(LowDifferentialMinus_3-HighDifferentialMinus_3)
data2['Momentum%Minus_3']=MomentumMinus_3/100
```

In [ ]:
```
from statistics import median

#classical momentum gauge at 4 day tenor (T=4)

ChangeInHighMinus_4=data2.high.shift(+29)/data2.close.shift(+5)
ChangeInLowMinus_4=data2.close.shift(+5)/data2.low.shift(+29)
TrailingStdMinus_4=data2.close.shift(+5).rolling(24).std()/data2.close.shift(+5).rolli
HighDifferentialMinus_4=ChangeInHighMinus_4/TrailingStdMinus_4
LowDifferentialMinus_4=ChangeInLowMinus_4/TrailingStdMinus_4

MomentumMinus_4=(LowDifferentialMinus_4-HighDifferentialMinus_4)
data2['Momentum%Minus_4']=MomentumMinus_4/100
```

In [ ]:
```
import pandas as pd
from matplotlib import pyplot as plt
from statsmodels.tsa.holtwinters import ExponentialSmoothing as HWES
from tqdm import tqdm
import warnings
warnings.filterwarnings("ignore")
tqdm.pandas()


df_train = pd.DataFrame()

i=0
for i in tqdm(range(5016)):
    if i <100:
        pass
    else:
        if i==100:
            df_train =data2['close'].iloc[0:i]
            model = HWES(df_train, seasonal_periods=24, trend='mul', seasonal='mul').f
            forecast_1hr=model.forecast(1)

        else:
            df_train =data2['close'].iloc[i-100:i]
            model = HWES(df_train, seasonal_periods=24, trend='mul', seasonal='mul').f
            forecast_1hr=np.append(forecast_1hr,model.forecast(1))


forecast_1hr
```

In [ ]:
```
naive=pd.DataFrame(forecast_1hr)
naive
```

In [ ]:
```
data2=data2[100:]
data2=data2.reset_index()
data2=data2.drop('index',axis=1)
data2
```

In [ ]:
```
data2=data2.merge(naive,left_index=True,right_index=True)
```

In [ ]:
```
data2.to_csv('BTC22.csv',index=False)
```

In [ ]:
```
DOWNLOAD_DIR1 = "C:/Users/mary jane/Downloads/"
filename1 = "BTC22.csv"
data2= pd.read_csv(DOWNLOAD_DIR1+filename1)
data2
```

In [ ]:
```
def formatPrice(n):
    return("-Rs." if n<0 else "Rs.")+list(map('{:.2f}%'.format,n))
def getStockDataVec(key):
    vec = []
    lines = open("BTC22.csv","r").read().splitlines()
    for line in lines[1:]:
        #print(line)
        #print(float(line.split(",")[4]))
        vec.append([float(vec) for vec in line.split(',')[1:]])

        #print(vec)
    return vec
def sigmoid(x):
    return 1/(1+np.exp(-x))
def getState(data, t, n):
    d = t - n + 1
    block = data[d:t + 1] if d >= 0 else -d * [data[0]] + data[0:t + 1] # pad with t0
    res = []
    for i in range(n - 1):
        array1 = np.array(block[i])
        array2 = np.array(block[i + 1])
        res.append(sigmoid(np.subtract(array2, array1)))
    return np.array([res])
```

In [ ]:
```
data2 = getStockDataVec(stock_name)
data2
```

In [ ]:
```
btc = [item[3] for item in data2]
btc
```

In [ ]:
```
def flatten(lists):
    results = []
    for numbers in lists:
        for x in numbers:
            results.append(x)
    return results
```

In [ ]:
```
from keras.layers import BatchNormalization

class Agent:
    def __init__(self, state_size, is_eval=False, model_name=""):
            self.state_size = state_size # normalized previous days
            self.action_size = 4 # hold, buy,buy all,  sell all
            self.memory = deque(maxlen=6000)

            self.inventory = []
            self.action_list=[]
            self.model_name = model_name
            self.is_eval = is_eval
            self.gamma = 0.95
            self.epsilon = .25
            self.epsilon_min = 0.01
```

```
            self.epsilon_decay = 0.995
            self.model = tf.keras.models.load_model(model_name) if is_eval else self._

    def _model(self):
        model = keras.models.Sequential()
        model.add(keras.layers.BatchNormalization())
        model.add(keras.layers.Dense(units=1200, activation="relu"))
        model.add(keras.layers.BatchNormalization())
        model.add(keras.layers.Dense(units=600, activation="relu"))
        model.add(keras.layers.BatchNormalization())
        model.add(keras.layers.Dense(units=300, activation="relu"))
        model.add(keras.layers.BatchNormalization())
        model.add(keras.layers.Dense(units=150, activation="relu"))
        model.add(keras.layers.BatchNormalization())
        model.add(keras.layers.Dense(units=75, activation="relu"))
        model.add(keras.layers.BatchNormalization())
        model.add(keras.layers.Dense(units=25, activation="relu"))
        model.add(keras.layers.BatchNormalization())
        model.add(keras.layers.Dense(self.action_size, activation="linear"))
        model.compile(loss="mse", optimizer=keras.optimizers.Adam(learning_rate=0.001)
        return model
    def act(self, state):
        if not self.is_eval and random.random()<= self.epsilon:
            return random.randrange(self.action_size)
        options = self.model.predict(np.array(state))
        return np.argmax(options[0])
    def expReplay(self, batch_size):
        mini_batch = []
        l = len(self.memory)
        for i in range(l - batch_size + 1, l):
            mini_batch.append(self.memory[i])
        for state, action, reward, next_state, done in mini_batch:
            target = reward
            if not done:
                target = reward + self.gamma * np.amax(self.model.predict(next_state)[
            target_f = self.model.predict(state)
            target_f[0][action] = target
            self.model.fit(state, target_f,epochs=5, verbose=0)
        if self.epsilon > self.epsilon_min:
            self.epsilon *= self.epsilon_decay
    def buy(self, initial_money):
        starting_money = initial_money
        states_sell = []
        states_buy = []
        inventory = []
        state = self.getState(0)
        for t in range(0, len(self.trend) - 1, self.skip):
            action = self.act(state)
            next_state = self.getState(t + 1)
            if action == 1 and initial_money >= self.trend[t] and t < (len(self.trend)
                inventory.append(self.trend[t])
                initial_money -= self.trend[t]
                states_buy.append(t)
                print('day %d: buy 1 unit at price %f, total balance %f'% (t, self.tre
            elif action == 2 and len(inventory):
                bought_price = inventory.pop(0)
                initial_money += self.trend[t]
                states_sell.append(t)
                try:
                    invest = ((close[t] - bought_price) / bought_price) * 100
```

```
                except:
                    invest = 0
                print(
                    'day %d, sell 1 unit at price %f, investment %f %%, total balance
                    % (t, close[t], invest, initial_money)
                )
            state = next_state
        invest = ((initial_money - starting_money) / starting_money) * 100
        total_gains = initial_money - starting_money
        return states_buy, states_sell, total_gains, invest
```

In [ ]:

```
stock_name = input("Enter Stock_name")
model_name = 'BTC_AgentModel.h5'
model = tf.keras.models.load_model(model_name)
window_size = model.layers[0].input.shape.as_list()[1]
agent = Agent(window_size, True, model_name)


batch_size = 10000
np.random.seed(42)
tf.random.set_seed(42)
keras.backend.clear_session()
l=len(data2)-1
print(state)
state = getState(data2, 0, window_size + 1)
total_profit = 0
buyallPrice=btc[t]
LiquidityAll=0
invenotry=0
agent.inventory = []
NAV=(btc[0]*10)
for t in range(l):

        action = agent.act(np.array(state))
        price=btc[t]

        if t==0:
            Liquidity=price*10
            buyingPower=[Liquidity/btc[t]]
        elif t>=1:
            Liquidity=((btc[0]*10)-sum(agent.inventory)+total_profit)
        if btc[t]>=Liquidity:
            action = 0 or 3
        # sit
        next_state = getState(data2, t + 1, window_size + 1)
        reward = 0


        if action == 1: # buy
            NAV=(btc[0]*10)+total_profit
            agent.inventory.append(btc[t])
            inventory=len(agent.inventory)

            Liquidity=((btc[0]*10)+total_profit-sum(agent.inventory))


            print("Buy "+str(price)+"------"+str(Liquidity)+"------"+str(sum(agent.inv

        elif action == 2: # buy all
```

```
            NAV=(btc[0]*10)+total_profit
            buyingPower.append(math.trunc(Liquidity/price))
            Block_Price=btc[t]
            Block_Liquidity=Liquidity
            agent.inventory.append(Block_Price*buyingPower[-1])

            Liquidity=((btc[0]*10)+total_profit-sum(agent.inventory))

            print("Buy All "+str(price)+"------"+str(Liquidity)+"------"+str(sum(agent


        elif action == 3:
            while len(agent.inventory) >0 and Liquidity<price: # sell all
                Block_Shares_Owned=Block_Liquidity/Block_Price
                bought_price =  sum(agent.inventory)/(Block_Shares_Owned+len(agent.inv
                if (btc[t] - bought_price)*(Block_Shares_Owned+inventory)<0:
                    total_profit = total_profit+(btc[t] - bought_price)*(Block_Shares_
                else:
                    total_profit = total_profit+(btc[t] - bought_price)*(Block_Shares_
                Liquidity=(Liquidity+total_profit)

                agent.inventory=[]
                Block_Shares_Owned=0
                inventory=0
                buyallPrice=btc[t]
                LiquidityAll=Liquidity
                NAV=(btc[0]*10)+total_profit

                print("Sell All "+"------"+str(price)+"-----> "+str(total_profit)+"---
        elif action ==0:
            NAV=(btc[0]*10)+total_profit
            print("Hold "+str(price)+"------"+str(Liquidity)+"------"+str(sum(agent.ir
            continue



        reward = max(total_profit*(((((total_profit+(btc[0]*10))/(btc[0]*10)))/(((pric
        done = True if t == l - 1 else False
        agent.memory.append((price,NAV,action))
        state = next_state
        if done:
            print("--------------------------------")
            print("Price: " + str(price)+" Total Profit: "+str(total_profit)+" | Inver
            print("--------------------------------")
```

In [ ]:
```
agent.memory
```

In [ ]:
```
len(agent.memory)
```

In [ ]:
```
Performance=pd.DataFrame([item for item in agent.memory])
Performance.columns=['price','NAV','action']
Performance.loc[Performance['action'] > 3.5, 'action'] = 0
```

In [ ]:
```
Performance
```

In [ ]:
```
Performance['NAV_']=(Performance['NAV']/10)

fig = plt.figure(figsize = (300,100))
```

```
plt.plot(Performance['price'], color='y', lw=2.)
plt.plot(Performance['NAV_'], color='b', lw=10,label = 'NAV')
plt.plot(Performance['price'], '-', markersize=20, color='g', label = 'action', markev
plt.plot(Performance['price'], 'o', markersize=20, color='k', label = 'action', markev
plt.plot(Performance['price'], 'o', markersize=20, color='k', label = 'action', markev
plt.plot(Performance['price'], 'v', markersize=20, color='r', label = 'action', markev
plt.title('total gains %f, total investment return %f%%'%(773797.979572-463040.100000,
plt.legend()
plt.savefig('.png')
plt.show()
```

In [ ]: