Documentation of Code

Zak Micallef

The code spit into files in this document and over of the code will occur

The first several files covers the LSTM and how the stores the results.

Todo list

an array of dictionary used to refaired to the save locations and names of each asset being used. All stored in ‘dataset\’.

```
todo = [
    {
        'stockName': 'Amazon (AMZN)',
        'savePlace': 'amazon',
        'fileName': 'AMZN.csv'
    },
    {
        'stockName': 'SPN500 (SPN)',
        'savePlace': 'SPN500',
        'fileName': 'SPN.csv'
    },
    ...
    {
        'stockName': 'Tesla (TSLA)',
        'savePlace': 'tesla',
        'fileName': 'TSLA.csv'
    },
    {
        'stockName': 'Amazon (AMZN)',
        'savePlace': 'amazon',
        'fileName': 'AMZN.csv'
    }
]
```

Main.py – (for LSMT)

Used to run all the files put together for the final results of the LSTM.

```
import os
import pandas as pd

# importing python files made for this project
import indicator_handler
import data_handler
import algo_trader
import trade_handler

def train_model_and_save_history(save_path_folder, tek_ind, epochs, split):
    # For the first technical indicators
    train_, test_, x_train, y_train, x_test, y_test, scaler = data_handler.prep_data(tek_ind, split)

    history_, results = algo_trader.fit_model(x_train, y_train, x_test, y_test, save_path_folder, epochs, scaler)

    # saving training history
    pd.DataFrame(history_.history).to_csv(save_path_folder + '/history.csv', index=False)

    # save the loss and validation loss on graph
    data_handler.save_loss_graph(history_, save_path_folder)

    # # load results from model
    results = algo_trader.lode_results(x_test, x_train, scaler, save_path_folder)

    data_handler.compare_predictions_to_results(results, test_[3], save_path_folder)

    trade_handler.save_trades(results, save_path_folder, test_)


todo = [
    ...
]

# make of see if there
if not (os.path.isdir('results')):
    os.mkdir('results')

# split of in-sampling and out-of-sampling
split = 0.95

_100_epochs = 100
_120_epochs = 100

for info in todo:
    # Get data set
    stockName = info['stockName']
    fileName = info['fileName']
    save_path_main_folder = 'results/' + info['savePlace']
    # see if file exists ask to over wright
    if True: #os.path.isdir(save_path_main_folder): FIXX
        # do_over_wright = input("Overwrite "+info['stockName']+" 'Y' or 'N'")
        # if do_over_wright == 'Y':
        if True:
            # making missing folder to save ...
            if not os.path.isdir(save_path_main_folder):
                os.mkdir(save_path_main_folder)
            tek_ind_1, tek_ind_2, tek_ind_all = indicator_handler.get_data_ind(fileName)
            # stock with split of data and moving averages
            data_handler.save_stock_graph_with_ma(tek_ind_1, stockName, save_path_main_folder, split)
            # training and saving the the model weights and in-sampling and out-of-sampling progression
            train_model_and_save_history(save_path_main_folder + '/training_1', tek_ind_1, _100_epochs, split)
            train_model_and_save_history(save_path_main_folder + '/training_2', tek_ind_2, _100_epochs, split)
            train_model_and_save_history(save_path_main_folder + '/training_all', tek_ind_all, _120_epochs, split)
            # Getting results of trading ..
            # cutting of at 70 day mark so that results are equal length
            # buy and hold for each day trading at the 70 day mark
            cutting = 70
            cash_for_simulation = 1000

            ledger_buy_only = trade_handler.only_buy(tek_ind_all, cash_for_simulation, split)[:cutting]
            # buying at random each day at the 70 day mark
            ledger_random = trade_handler.random_buy(tek_ind_all, cash_for_simulation, split)[:cutting]
            # buy and hold earnings at the 70 day mark
            buy_hold_earnings = trade_handler.buy_hold_strategy(tek_ind_all[:cutting], cash_for_simulation, split)
            # loading all results from model
            ledger_1 = pd.read_csv(save_path_main_folder + '/training_1/ledger.csv')
            ledger_2 = pd.read_csv(save_path_main_folder + '/training_2/ledger.csv')
            ledger_all = pd.read_csv(save_path_main_folder + '/training_all/ledger.csv')
            # comparing then and saving them in the correct folder
            data_handler.compare_and_save_results(stockName, save_path_main_folder, ledger_1, ledger_2, ledger_all,
ledger_buy_only, ledger_random, buy_hold_earnings)
```

algo_trader.py

Used to handle all the neural network model

```
import tensorflow as tf
import tensorflow.compat.v1 as tf

# importing python files made for this project
import data_handler

tf.disable_v2_behavior()
tf.keras.callbacks.TensorBoard(log_dir='./Graph', histogram_freq=0,
                               write_graph=True, write_images=True)


def set_model(x_train):
    # Creating model
    model_ = tf.keras.Sequential()
    model_.add(tf.keras.layers.LSTM(
        units=75,
        return_sequences=True,
        input_shape=(x_train.shape[1], x_train.shape[2])
    ))
    model_.add(tf.keras.layers.LSTM(
        units=30,
        return_sequences=True
    ))
    model_.add(tf.keras.layers.LSTM(
        units=30,
        return_sequences=True
    ))
    model_.add(tf.keras.layers.Dense(units=1))
    model_.compile(loss='mae', optimizer='adam')
    return model_


def fit_model(x_train, y_train, x_test, y_test, save_path_folder, epochs, scaler):
    save_path = save_path_folder + "/model_weights/cp.ckpt"


    # Create a callback that saves the model's weights
    cp_callback = tf.keras.callbacks.ModelCheckpoint(filepath=save_path,
                                                     save_weights_only=True,
                                                     verbose=1)

    sess = tf.Session(config=tf.ConfigProto())
    with sess.as_default():
        model_ = set_model(x_train)
        # fit netwrok
        tf.compat.v1.keras.backend.set_session(sess)
        history = model_.fit(
            x_train,
            y_train,
            epochs=epochs,
            batch_size=x_test.shape[1],
            validation_data=(x_test, y_test),
            shuffle=False,
            callbacks=[cp_callback]
        )
        model_.save_weights(save_path_folder + '/final_weights/checkpoint')
        predictions = model_.predict(x_test)
    sess.close()

    return history, data_handler.get_mean_results(predictions, x_test, scaler)


# Loads the weights
def lode_results(x_test, x_train, scaler, checkpoint_path):
    sess = tf.Session(config=tf.ConfigProto())

    with sess.as_default():
        tf.compat.v1.keras.backend.set_session(sess)
        model_ = set_model(x_train)
        model_.load_weights(checkpoint_path + "/final_weights/checkpoint").expect_partial()
        predictions = model_.predict(x_test)

    results_ = data_handler.get_mean_results(predictions, x_test, scaler)

    return results_
```

Trade_handler.py

Used to simulate the trades by storing it in a ledger and save for later use (part 1)

```
import math
import statistics

# importing python files made for this project
import data_handler

import pandas as pd

from random import seed
from random import random

seed(1)


# ledger object to keep track of trades
class Ledger:
    def __init__(self, cash):
        self.ledger = pd.DataFrame([['hold', cash]], columns=['buy/sell/hold', 'cash'])

    def add(self, decision, cash):
        self.ledger.loc[self.ledger.shape[0]] = [decision, cash]

    def get_ledger(self):
        self.ledger['dayly return/losses percentage'] = self.ledger['cash'].pct_change().fillna(0) * 100
        self.ledger['dayly return/losses'] = self.ledger['cash'].diff().fillna(0)
        self.ledger['good trade'] = self.ledger['dayly return/losses percentage'] > 0
        return self.ledger


# cal buy profits or losses
def buy(open_value, close_value, cash):
    return (1 + ((close_value - open_value) / close_value)) * cash


# cal short profits or losses
def short(open_value, close_value, cash):
    return (1 + ((open_value - close_value) / close_value)) * cash


# make trade to see the profits/losses and then save
def save_trades(results, save_path_folder, stock):
    open_ = stock.iloc[:, 1]
    close_ = stock.iloc[:, 0]

    day_change = results - open_[:len(results)]
    a_10th_of_the_sd_results = statistics.stdev(results) / 100

    #  if in the range of a 10th of the standard deviation then dont trade
    des = day_change.apply(lambda x: 'hold' if -a_10th_of_the_sd_results < x < a_10th_of_the_sd_results else (
        'sell' if x < 0 else 'buy'))
    cash = 1000
    ledger_obj = Ledger(cash)
    for i, x in enumerate(des.iteritems()):
        open_cur = open_[i]
        close_cur = close_[i]
        if x[1] == 'buy':
            cash = buy(open_cur, close_cur, cash)
        elif x[1] == 'sell':
            cash = short(open_cur, close_cur, cash)
        ledger_obj.add(x[1], cash)

    # save ledger
    ledger_obj.get_ledger().to_csv(save_path_folder + '/ledger.csv', index=False)
```

(part 2)

```
# cal the profits or losses with a buy and hold strategy
def buy_hold_strategy(stock, cash, split):
    stock_hist = data_handler.prep_data(stock, split)
    stock_hist = stock_hist[1]
    open_value = stock_hist.iloc[0, 1]
    close_value = stock_hist.iloc[-1, 0]
    # saving starting value
    buy_hold_for_graph = [cash]

    # no trade happening in between
    for _ in range(68):
        buy_hold_for_graph.append(None)

    # appending end value
    end = buy(open_value, close_value, cash)
    buy_hold_for_graph.append(end)
    return buy_hold_for_graph


# Buy everyday and sell at the end of the day simulation
def only_buy(stock, cash, split):
    stock_hist = data_handler.prep_data(stock, split)
    stock_hist = stock_hist[1]
    ledger_obj = Ledger(cash)

    for i, day in stock_hist.iloc[:, :2].iterrows():
        open = day[1]
        close = day[0]
        cash = buy(open, close, cash)
        ledger_obj.add('buy', cash)

    return ledger_obj.get_ledger()


# Short or Buy randomly and sell for each day simulation
def random_buy(stock, cash, split):
    stock_hist = data_handler.prep_data(stock, split)
    stock_hist = stock_hist[1]
    ledger_obj = Ledger(cash)

    # generate random numbers between 0-1
    for i, day in stock_hist.iloc[:, :2].iterrows():
        value = math.floor(random() * 3)
        open = day[1]
        close = day[0]
        if value == 0:
            cash = buy(open, close, cash)
            ledger_obj.add('buy', cash)
        elif value == 1:
            cash = short(open, close, cash)
            ledger_obj.add('sell', cash)
        else:
            ledger_obj.add('hold', cash)

    return ledger_obj.get_ledger()
```

Data_handler.py

Used to edit and organize data (part 1)

```
import numpy as np
import pandas as pd
from sklearn.preprocessing import MinMaxScaler
import math
import matplotlib.pyplot as plt
import os

# find sharp radio
def sharpe(y):
    return np.sqrt(y.count()) * (y.mean() / y.std())


# Split into train and test sets
def split_data(data, split_point):
    val = np.array(data)
    train_sample = int(len(data) * split_point)
    train = pd.DataFrame(val[: train_sample, :])
    test = pd.DataFrame(val[train_sample:, :])
    return train, test


# shaping data for training the neural network
def shape_for_nn(data):
    x_ = []
    y_ = []

    for i in range(1, data.shape[0]):
        x_.append(data[i, 1:])
        y_.append(data[i, 0])

    x_, y_ = np.array(x_), np.array(y_)

    x_ = x_.reshape((x_.shape[0], x_.shape[1], 1))

    # Trimming data to fit model
    trim = (x_.shape[1] * math.floor(x_.shape[0] / x_.shape[1]))

    if trim != 0:
        x_ = x_[-trim:]
        y_ = y_[:trim]

    return x_, y_


def prep_data(ind, split):
    train, test = split_data(ind, split)

    scaler = MinMaxScaler()
    scaler.fit_transform(ind)
    train_ = scaler.transform(train)
    test_ = scaler.transform(test)
    x_train, y_train = shape_for_nn(train_)
    x_test, y_test = shape_for_nn(test_)

    return train, test, x_train, y_train, x_test, y_test, scaler


def save_stock_graph_with_ma(df, stockName, save_path_folder, split):
    df.reset_index(inplace=True, drop=True)
    _ = prep_data(df, split)
    train = _[0][2]
    test = pd.concat([pd.Series(np.repeat(None, train.shape[0])), _[1][2]]).reset_index(drop=True)

    # check if files exits and remove file if stock chart exists
    if os.path.isfile(save_path_folder + '/stock_chart.png'):
        os.remove(save_path_folder + '/stock_chart.png')

    plt.figure(figsize=(24, 16))
    plt.plot(train, label='Approx Train')
    plt.plot(test, label='Approx Test')
    plt.plot(df['10day MA'], label='10 Day moving avarage')
    plt.plot(df['50day MA'], label='50 Day moving avarage')
    plt.plot(df['200day MA'], label='200 Day moving avarage')
    plt.legend(loc='best')
    plt.title(stockName)
    plt.savefig(save_path_folder + '/stock_chart.png')
    plt.close('all')


def save_loss_graph(history, save_path_folder):
    # Plot history

    # check if files exits and remove file if stock chart exists
    if os.path.isfile(save_path_folder + '/trading_loss.png'):
        os.remove(save_path_folder + '/trading_loss.png')

    plt.figure(figsize=(10, 6), dpi=100)
    plt.plot(history.history['loss'], label='LSTM train', color='red')
    plt.plot(history.history['val_loss'], label='LSTM test', color='green')
    plt.xlabel('epochs')
    plt.ylabel('loss')
    plt.legend()
    plt.title('Training and Validation loss')
    plt.savefig(save_path_folder + '/trading_loss.png')
    plt.close('all')
```

(part 2)

```
def compare_and_save_results(stock_name, save_path_folder, ledger_1, ledger_2, ledger_all, ledger_buy_only,
ledger_random,
                             buy_hold_earnings):
    # check if files exits and remove file if stock chart exists
    if os.path.isfile(save_path_folder + '/results.png'):
        os.remove(save_path_folder + '/results.png')

    # showing results on a graph
    plt.figure(figsize=(12, 7))
    plt.plot(ledger_1['cash'][:70], label='Technical Ind 1')
    plt.plot(ledger_2['cash'][:70], label='Technical Ind 2')
    plt.plot(ledger_all['cash'], label='All Technical Ind')
    plt.plot(ledger_buy_only['cash'], label='Buy Only Strategy')
    plt.plot(ledger_random['cash'], label='Random Buying and Selling')
    plt.plot(buy_hold_earnings, label='Buy and Hold Strategy', marker="X", linestyle='None')
    plt.title('stock:' + stock_name)
    plt.xlabel('Days')
    plt.ylabel('USD')
    plt.legend(loc='best')
    plt.savefig(save_path_folder + '/results.png')
    plt.close('all')

    updays = ledger_buy_only[ledger_buy_only['good trade'] == True].count()[0]
    downdays = ledger_buy_only[ledger_buy_only['good trade'] == False].count()[0]

    updown_days = pd.DataFrame([
        ['Up trading Days', updays],
        ['Down trading Days', downdays],
    ], columns=['description', 'Frequency'])
    updown_days.to_csv(save_path_folder + '/up_down_days_detail.csv', index=False)

    description = pd.DataFrame([
        [
            'Technical Ind 1',
            ledger_1[ledger_1['buy/sell/hold'] == 'hold'].count()[0],
            ledger_1[ledger_1['buy/sell/hold'] == 'buy'].count()[0],
            ledger_1[ledger_1['buy/sell/hold'] == 'sell'].count()[0],
            ledger_1[(ledger_1['buy/sell/hold'] != 'hold') & (ledger_1['good trade'] == True)].count()[0],
            ledger_1[(ledger_1['buy/sell/hold'] != 'hold') & (ledger_1['good trade'] == False)].count()[0],
            ledger_1['cash'].iloc[-1],
            sharpe(ledger_1['dayly return/losses'])
        ], [
            'Technical Ind 2',
            ledger_2[ledger_2['buy/sell/hold'] == 'hold'].count()[0],
            ledger_2[ledger_2['buy/sell/hold'] == 'buy'].count()[0],
            ledger_2[ledger_2['buy/sell/hold'] == 'sell'].count()[0],
            ledger_2[(ledger_2['buy/sell/hold'] != 'hold') & (ledger_2['good trade'] == True)].count()[0],
            ledger_2[(ledger_2['buy/sell/hold'] != 'hold') & (ledger_2['good trade'] == False)].count()[0],
            ledger_2['cash'].iloc[-1],
            sharpe(ledger_2['dayly return/losses'])
        ], [
            'All Technical Ind',
            ledger_all[ledger_all['buy/sell/hold'] == 'hold'].count()[0],
            ledger_all[ledger_all['buy/sell/hold'] == 'buy'].count()[0],
            ledger_all[ledger_all['buy/sell/hold'] == 'sell'].count()[0],
            ledger_all[(ledger_all['buy/sell/hold'] != 'hold') & (ledger_all['good trade'] == True)].count()[0],
            ledger_all[(ledger_all['buy/sell/hold'] != 'hold') & (ledger_all['good trade'] == False)].count()[0],
            ledger_all['cash'].iloc[-1],
            sharpe(ledger_all['dayly return/losses'])
        ], [
            'Buy Only',
            ledger_buy_only[ledger_buy_only['buy/sell/hold'] == 'hold'].count()[0],
            ledger_buy_only[ledger_buy_only['buy/sell/hold'] == 'buy'].count()[0],
            ledger_buy_only[ledger_buy_only['buy/sell/hold'] == 'sell'].count()[0],
            ledger_buy_only[
                (ledger_buy_only['buy/sell/hold'] != 'hold') & (ledger_buy_only['good trade'] == True)].count()[
                0],
            ledger_buy_only[
                (ledger_buy_only['buy/sell/hold'] != 'hold') & (ledger_buy_only['good trade'] == False)].count()[0],
            ledger_buy_only['cash'].iloc[-1],
            sharpe(ledger_buy_only['dayly return/losses'])
        ], [
            'Random',
            ledger_random[ledger_random['buy/sell/hold'] == 'hold'].count()[0],
            ledger_random[ledger_random['buy/sell/hold'] == 'buy'].count()[0],
            ledger_random[ledger_random['buy/sell/hold'] == 'sell'].count()[0],
            ledger_random[(ledger_random['buy/sell/hold'] != 'hold') & (ledger_random['good trade'] == True)].count()[
                0],
            ledger_random[(ledger_random['buy/sell/hold'] != 'hold') & (ledger_random['good trade'] == False)].count()[
                0],
            ledger_random['cash'].iloc[-1],
            sharpe(ledger_random['dayly return/losses'])
        ]
    ], columns=['description', 'No trading days', 'Buy trades', 'Short trades', 'profit days', 'loss days',
                'End Amount (USD)', 'Sharp Radio'])

    description.to_csv(save_path_folder + '/description.csv', index=False)


def compare_predictions_to_results(results, y_test, save_path_folder):
    # check if files exits and remove file if stock chart exists
    if os.path.isfile(save_path_folder + '/results_vs_predicted.png'):
        os.remove(save_path_folder + '/results_vs_predicted.png')

    # showing results on a graph
    plt.figure(figsize=(12, 7))
    plt.plot(results, label='Predicted results')
    plt.plot(y_test, label='Real Results')
    plt.xlabel('Days')
    plt.ylabel('USD')
    plt.legend(loc='best')
    plt.savefig(save_path_folder + '/results_vs_predicted.png')
    plt.close('all')
```

(part 3)

```
# used to inverse the results from of the first col
def inverse_transform(y, x_test, scaler):
    y_ = pd.DataFrame(np.array(y).reshape((len(y), 1)))
    x_ = pd.DataFrame(np.zeros((len(x_test), len(x_test[0]))))
    x_[-1] = y_
    cols = list(x_.columns)
    cols = [cols[-1]] + cols[:-1]
    x_ = x_[cols]
    return scaler.inverse_transform(x_)[:, 0]


def get_mean_results(predictions, x_test, scaler):
    results_ = []
    for i in range(0, predictions.shape[0]):
        results_.append(predictions[i].mean())

    return inverse_transform(results_, x_test, scaler)
```

Indicator_handler.py

Used to handle all for the indicators (part 1)

```
import matplotlib.pyplot as plt
import copy
import talib
import numpy as np
import pandas as pd


def get_range_of_dates(data):
    end_date = '2019-12-31'
    data['Date'] = pd.to_datetime(data['Date'])
    to_ = data[data['Date'] == end_date].index[0]
    from_ = (to_ - 2000)
    return data[from_:to_].reset_index(drop=True)


def get_data_ind(csv_file_name):
    # opening csv on stock
    df = pd.read_csv("dataset/" + csv_file_name, parse_dates=True)

    # filling in missing values with previous files
    df = df.fillna(method='pad')

    df = get_all_ind(df)
    # add if length not long with range .. not long at all .. trow error
    # df = get_range_of_dates(df)
    df = df[-2000:]
    df = df.drop(['Date'], axis=1)

    tek_ind_1 = get_tek_ind_set_1(df)
    tek_ind_2 = get_tek_ind_set_2(df)
    tek_ind_all = copy.deepcopy(df)

    return tek_ind_1, tek_ind_2, tek_ind_all
```

```
def get_all_ind(data):
    all_ind = copy.deepcopy(data)
    all_ind['daily_return'] = all_ind.Close.pct_change().fillna(0)
    all_ind['cum_daily_return'] = (1 + all_ind['daily_return']).cumprod()

    all_ind['H-L'] = all_ind.High - all_ind.Low

    all_ind['C-O'] = all_ind.Close - all_ind.Open

    all_ind['10day MA'] = all_ind.Close.shift(1).rolling(window=10).mean().fillna(0)
    all_ind['50day MA'] = all_ind.Close.shift(1).rolling(window=50).mean().fillna(0)
    all_ind['200day MA'] = all_ind.Close.shift(1).rolling(window=200).mean().fillna(0)

    all_ind['rsi'] = talib.RSI(all_ind.Close.values, timeperiod=14)

    all_ind['Williams %R'] = talib.WILLR(
        all_ind.High.values,
        all_ind.Low.values,
        all_ind.Close.values,
        14
    )

    # creating 7 and 21 day moving avarages
    all_ind['ma7'] = all_ind.Close.rolling(window=7).mean().fillna(0)
    all_ind['ma21'] = all_ind.Close.rolling(window=21).mean().fillna(0)

    # Creating MACD < do research
    all_ind['ema_26'] = all_ind.Close.ewm(span=26).mean().fillna(0)
    all_ind['ema_12'] = all_ind.Close.ewm(span=12).mean().fillna(0)
    all_ind['macd'] = (all_ind['ema_12'] - all_ind['ema_26'])

    # Creating Bollinger bands
    # Set number of days and standard deviations to use for rolling lookback period for bollinger band calulation
    window = 21
    no_of_std = 2
    # Calculation rolling mean and standard deviation using number of days set above
    rolling_mean = all_ind.Close.rolling(window).mean()
    rolling_std = all_ind.Close.rolling(window).std()
    # Creating two new Dataframe columns to hold valuse of upper and lower bolling bands
    # B['Rolling Mean'] = rolling_mean.fillna(0)
    all_ind['bb_high'] = (rolling_mean + (rolling_std * no_of_std)).fillna(0)
    all_ind['bb_low'] = (rolling_mean - (rolling_std * no_of_std)).fillna(0)

    # create Expontial Moving Avarage
    all_ind['ema'] = all_ind.Close.ewm(com=0.5).mean()

    # create momentum
    all_ind['momentum'] = all_ind.Close - 1
    all_ind.fillna(0, inplace=True)

    # Calculation of Stochastic Oscillator (%K and %D)

    def stok(n):
        all_ind['stok'] = ((all_ind['Close'] - all_ind['Low'].rolling(window=n, center=False).mean()) /
                           (all_ind['High'].rolling(window=n, center=False).max() -
                            all_ind['Low'].rolling(window=n, center=False).min())) * 100
        all_ind['stod'] = all_ind['stok'].rolling(window=3, center=False).mean()


    stok(4)

    all_ind.fillna(0, inplace=True)

    # Calulate of Price Rate of Change
    # ROC = [(Close - Close n perods ago) / (Close n perods ago)] * 100

    all_ind['ROC'] = ((all_ind['Close'] - all_ind['Close'].shift(12)) /
                      (all_ind['Close'].shift(12))) * 100
    all_ind.fillna(0, inplace=True)

    # Calculate of Momentum
    all_ind['Momentum'] = all_ind['Close'] - all_ind['Close'].shift(4)
    all_ind.fillna(0, inplace=True)

    # Calculate of Commodity Channel Index
    tp = (all_ind['High'] + all_ind['Low'] + all_ind['Close']) / 3
    ma = all_ind.Close.rolling(window=20).mean().fillna(0)
    md = all_ind.Close.rolling(window=20).std().fillna(0)
    all_ind['CCI'] = (tp - ma) / (0.015 * md)
    all_ind['CCI'] = all_ind['CCI'].replace(np.inf, np.nan).fillna(0)

    # Calculate of Triple Exponential Moving Average
    # Triple Exponential NA Formula:
    # T-EMA = (3EMA - 3EMA(EMA)) + EMA(EMA(EMA))
    # Where:
    # EMA = EMA(1) + alfa * (Close - EMA(1))
    # alfa = 2 / ( N + 1 )
    # N = The smoothing period

    all_ind['ema'] = all_ind['Close'].ewm(span=3, min_periods=0, adjust=True, ignore_na=False).mean()
    all_ind.fillna(0, inplace=True)

    all_ind['tema'] = (3 * all_ind['ema'] - 3 * all_ind['ema'] * all_ind['ema']) + (all_ind['ema'] *
                                                                                     all_ind['ema'] *
                                                                                     all_ind['ema'])

    # Turning Line
    high = all_ind['High'].rolling(window=9, center=False).max()
    low = all_ind['Low'].rolling(window=9, center=False).min()
    all_ind['turning_line'] = (high + low) / 2
    all_ind.fillna(0, inplace=True)

    # Standerd Line
    p26_high = all_ind['High'].rolling(window=26, center=False).max()
    p26_low = all_ind['Low'].rolling(window=26, center=False).min()
    all_ind['standard_line'] = (p26_high + p26_low) / 2

    # Leading Span 1
    all_ind['ichimoku_span1'] = ((all_ind['turning_line'] + all_ind['standard_line']) / 2).shift(26)

    # tek_ind_2.fillna(0, inplace=True) is this needed ??

    # Leading Span 2
    p53_high = all_ind['High'].rolling(window=52).max()
    p53_low = all_ind['Low'].rolling(window=52).min()
    all_ind['ichimoku_span2'] = ((p53_high + p53_low) / 2).shift(26)

    # tek_ind_2.fillna(0, inplace=True) is this needed ??

    # The most current closing price plotted 22 time periods behind (option days behind)
    all_ind['chikou_span'] = all_ind['Close'].shift(-22)  # 22 suggested by investopedia

    # arranged that data so that the close price of that data is ready to be used appropriately
    all_ind['Close that day'] = all_ind['Close'].shift(-1)
    all_ind['Open that day'] = all_ind['Open'].shift(-1)

    all_ind.fillna(0, inplace=True)

    return (all_ind[['Date', 'Close that day',
        'Open that day', 'Open', 'High', 'Low', 'Close', 'Volume', 'daily_return',
        'cum_daily_return', 'H-L', 'C-O', '10day MA', '50day MA', '200day MA',
        'rsi', 'Williams %R', 'ma7', 'ma21', 'ema_26', 'ema_12', 'macd',
        'bb_high', 'bb_low', 'ema', 'momentum', 'stok', 'stod', 'ROC',
        'Momentum', 'CCI', 'tema', 'turning_line', 'standard_line',
        'ichimoku_span1', 'ichimoku_span2', 'chikou_span']])
```

```
# get daily return on each day
def get_tek_ind_set_1(data):
    return(data[['Close that day', 'Open that day', 'Close', 'Open', 'High', 'Low', 'Volume', 'daily_return', 'cum_daily_return',
      'H-L', 'C-O', '10day MA', '50day MA', '200day MA', 'rsi', 'Williams %R',
      'ma7', 'ma21', 'ema_26', 'ema_12', 'macd', 'bb_high', 'bb_low', 'ema',
      'momentum']])


# second set of technical indicates
def get_tek_ind_set_2(data):
    return(data[['Close that day', 'Open that day', 'Open', 'Close', 'High', 'Low', 'Volume', 'stok', 'stod', 'ROC', 'Momentum',
      'CCI', 'ema', 'tema', 'turning_line', 'standard_line', 'ichimoku_span1',
      'ichimoku_span2', 'chikou_span']])
```

main.py (where the trading strategy happen) BB strategy (part 1)

```
import pandas as pd
import talib
import numpy as np
import matplotlib.pyplot as plt

class Ledger:
    def __init__(self, cash):
        self.cash = cash
        self.ledger = pd.DataFrame([['hold', cash]], columns=['buy/sell/hold', 'cash'])

    def add(self, decision, cash):
        self.cash = cash
        self.ledger.loc[self.ledger.shape[0]] = [decision, self.cash]

    def add_pct(self, decision, pct):
        self.cash = self.cash * pct
        self.ledger.loc[self.ledger.shape[0]] = [decision, self.cash]

    def get_ledger(self):
        self.ledger['dayly return/losses percentage'] = self.ledger['cash'].pct_change().fillna(0) * 100
        self.ledger['dayly return/losses'] = self.ledger['cash'].diff().fillna(0)
        self.ledger['good trade'] = self.ledger['dayly return/losses percentage'] > 0
        return self.ledger


todo = [
    ...
]

# Set number of days and standard deviations to use for rolling lookback period for Bollinger band calculation
def bollinger_strat(df, window, std):
    # Calculate rolling mean and standard deviation using number of days set above
    rolling_mean = df['Open'].rolling(window).mean()
    rolling_std = df['Open'].rolling(window).std()
    # create two new DataFrame columns to hold values of upper and lower Bollinger bands
    df['Rolling Mean'] = rolling_mean
    df['Bollinger High'] = rolling_mean + (rolling_std * std)
    df['Bollinger Low'] = rolling_mean - (rolling_std * std)

    df['Position'] = None
    df = df[-500:]
    # Fill our newly created position column - set to sell (-1) when the price hits the upper band, and set to buy (1) when it
hits the lower band
    for row in range(len(df)):

        if (df['Open'].iloc[row] > df['Bollinger High'].iloc[row]) and (
                df['Open'].iloc[row - 1] < df['Bollinger High'].iloc[row - 1]):
            df['Position'].iloc[row] = -1

        if (df['Open'].iloc[row] < df['Bollinger Low'].iloc[row]) and (
                df['Open'].iloc[row - 1] > df['Bollinger Low'].iloc[row - 1]):
            df['Position'].iloc[row] = 1
        # Forward fill our position column to replace the "None" values with the correct long/short positions to represent the
"holding" of our position

    # forward through time
    df['Position'].fillna(method='ffill', inplace=True)
    # Calculate the daily market return and multiply that by the position to determine strategy returns
    df['Market Return'] = np.log(df['Open'] / df['Open'].shift(1))
    df['Strategy Return'] = df['Market Return'] * df['Position']
    # Plot the strategy returns
    df['Strategy Return'].cumsum().plot(label='window ' + str(window) + ' std ' + str(std))
    # returning the last day to compare results
    return df['Strategy Return'].cumsum().iloc[-1]

 # apply the strategy to all stocks for a range of windows and gap of BB lows and highs
ranges = []
for do in todo:
    df = pd.read_csv('dataset/' + do['fileName'])

    plt.figure(figsize=(12, 8))
    windows = [10, 20, 100]
    stds = [2, 3, 5]
    res = []
    for window in windows:
        for std in stds:
            res.append(bollinger_strat(df, window, std))
    plt.legend(loc='best')
    plt.title(do['stockName'])
    plt.savefig('resultsBB' + '/' + do['savePlace'] + '.png')
    plt.close('all')
    ranges.append((min(res), max(res)))

print(ranges)
```

(part 2) The RSI and MACD Double Confirmation Momentum Strategy

```
# the RSI set at 7 day rolling window and the MACD Momentum strategy
def day_trader(df, title):
    df['rsi7'] = talib.RSI(df.Close.values, timeperiod=7)
    macd, macdsignal, df['macdhist'] = talib.MACD(df.Close.values, fastperiod=12, slowperiod=26, signalperiod=9)
    df = df[-70:]
    df['rsi7 over 50'] = df['rsi7'] > 50
    df['macdhist cross over'] = ((df['macdhist'] > 0).shift(-1)) & (df['macdhist'] < 0)
    df['macdhist cross to under'] = ((df['macdhist'] < 0)) & (df['macdhist'] > 0).shift(1)
    df['buy'] = df['rsi7 over 50'] & df['macdhist cross over']
    df['sell'] = (df['rsi7'] < 50) & df['macdhist cross to under']

    df['position'] = np.zeros(df.shape[0])
    position = 0
    for row in range(len(df)):
        if position == 2:
            position = 0

        if position == 0:
            if df['buy'].iloc[row]:
                position = 1
            elif df['sell'].iloc[row]:
                position = -1
        elif position == 1:
            if df['sell'].iloc[row]:
                position = -1
        elif position == -1:
            if df['sell'].iloc[row]:
                position = 1

        df['position'].iloc[row] = position

    df['action'] = df['position'].shift(1).fillna(0) != df['position']
    df['change'] = df['Close'] - df['Close'].shift(-1)
    df['pct prof/loss'] = np.zeros(df.shape[0])
    df['pct prof/loss'] = (df['change'] * df['position']) / df['Close']
    df['pct prof/loss'][df['action']] = 0

    leger_obj = Ledger(1000)

    des_pre = None
    for row in range(len(df)):
        pct = df['pct prof/loss'].iloc[row]
        des = df['position'].iloc[row]
        if des_pre is None:
            if des == 0:
                des_ = 'hold'
            elif des == 1:
                des_ = 'buy'
            elif des == -1:
                des_ = 'sell'
        else:
            if des == des_pre:
                des_ = 'hold'
            else:
                if des == 0:
                    des_ = 'hold'
                elif des == 1:
                    des_ = 'buy'
                elif des == -1:
                    des_ = 'sell'
        des_pre = des
        leger_obj.add_pct(des_, (pct + 1))

    leg = leger_obj.get_ledger()
    leg['cash'].plot(label=title)
    return leg


def sharpe(y):
    return np.sqrt(y.count()) * (y.mean() / y.std())

plt.figure(figsize=(24, 16))
for do in todo:
    df = pd.read_csv('dataset/' + do['fileName'])
    ledger = day_trader(df, do['stockName'])[:-1]

    description = pd.DataFrame([
        [
            'Technical Ind 1',
            ledger[ledger['buy/sell/hold'] == 'hold'].count()[0],
            ledger[ledger['buy/sell/hold'] == 'buy'].count()[0],
            ledger[ledger['buy/sell/hold'] == 'sell'].count()[0],
            ledger[(ledger['buy/sell/hold'] != 'hold') & (ledger['good trade'] == True)].count()[0],
            ledger[(ledger['buy/sell/hold'] != 'hold') & (ledger['good trade'] == False)].count()[0],
            ledger['cash'].iloc[-1],
            sharpe(ledger['dayly return/losses'])
        ]
    ], columns=['description', 'No trading days', 'Buy trades', 'Short trades', 'profit days', 'loss days',
                'End Amount (USD)', 'Sharp Radio'])

    ledger.to_csv('resultsRs7Macd/' + do['fileName'] + '.csv')
    description.to_csv('resultsRs7Macd/description_' + do['fileName'] + '.csv')

plt.title('compare RS7 with macdhist cross over')
plt.legend(loc="best")
plt.savefig('resultsRs7Macd/profit.png')
plt.close('all')
```