OBJECTIVE

The project is aimed to design a Algorithm bot which can be used to buy and sell stock from stock broker automatically.

ABSTRACT

We need to build up a script which can help a professional or part-time trader to extract all the stock data from financial site and apart from that they can use various indicator and implement in the script to automatically buy or sell stock from the official stock broker.

Software requirement

- Operating System-Windows 7 or above
- Software/Platform used- Anaconda Python Distribution

Hardware requirement

- *Processor-Intel core i3 5th gen*
- *Ram-4GB or above*
- *System Type-64 bit OS*
- Constant Internet Connection

PROGRESSIVE REPORT

My approach towards building this Algorithmic trading script using Python include various steps which will be explained in the upcoming slide. Apart from the explanation I have provide the code which I have used to build up this model. This will give the better understanding of my project.

Step 1:- GETTING DATA USING YFINANCE

Explanation:-

So let's get into how we can use my finance library to get data into your system. The first thing that we need to do is to import the finance library.

This library gets us the all financial data of any stocks which may include intraday data, 6 month data, 1 year data or many more. In the given code I have used one stock MSFT to get 1mont data, a 3 year financial data and intraday data from yfinance.

Installation:-

Install yfinance using pip:-

- $ pip install yfinance --upgrade --no-cache-dir

Code:-

```
import yfinance as yf

# get ohlcv data for any ticker by period.
data = yf.download("MSFT", period='1mo', interval="5m")

# get ohlcv data for any ticker by start date and end date
data = yf.download("MSFT", start="2017-01-01", end="2020-04-24")

# get intraday data for any ticker by period.
data = yf.download("MSFT", period='1mo', interval="5m")
```

Step 2:- GETTING DATA FOR MULTIPLE STOCKS

Explanation:-

We will be using finance library to extract data for a number of tickers. Here we have used a new library is date and time library as dt. And panda data frame is a python dictionaries which I am trying to store temporary information which are then later imported into pandas data frame. One thing should be noticed that where ever I have used tickers it means stock name.

Code:-

```
import datetime as dt

import yfinance as yf

import pandas as pd

stocks = ["AMZN","MSFT","INTC","GOOG","INFY.NS","3988.HK"]

start = dt.datetime.today()-dt.timedelta(360)

end = dt.datetime.today()

cl_price = pd.DataFrame() # empty dataframe which will be filled with closing prices of each stock

ohlcv_data = { } # empty dictionary which will be filled with ohlcv dataframe for each ticker

# looping over tickers and creating a dataframe with close prices

for ticker in stocks:

    cl_price[ticker] = yf.download(ticker,start,end)["Adj Close"]

    # looping over tickers and storing OHLCV dataframe in dictionary

for ticker in stocks:

    ohlcv_data[ticker] = yf.download(ticker,start,end)
```

Step 3:- WEB SCRAPPING to Extract Stock Fundamental Data

Explanation:-

Two library which I'll used to perform web scrapping is :-

- Requests :- Request is a fairly popular python library which helps you establish a connection with a web server.

  Installing Requests:-
  - $ python -m pip install requests

- Beautiful soup:- Its utility lies in the ease with which it lets us to parse HTML data.

  Installing Beautifulsoup4:-
  - $pip install beautifulsoup4

What I am doing here is that I am running a loop and each pass through of the loop will perform the operation or perform the web scraping for one ticker. We will loop through every single ticker in the tickers list and that's how my code will work. So for example the first code block is scraping the balance sheet part of the Yahoo Finance Web site. Then the second code block is the financials so that's income statement.

Code:-

```
import requests
from bs4 import BeautifulSoup
import pandas as pd
tickers = ["AAPL","MSFT"] #list of tickers whose financial data needs to be extracted
financial_dir = {}
for ticker in tickers:
    #getting balance sheet data from yahoo finance for the given ticker
    temp_dir = {}
    url = 'https://in.finance.yahoo.com/quote/'+ticker+'/balance-sheet?p='+ticker
    page = requests.get(url)
    page_content = page.content
    soup = BeautifulSoup(page_content,'html.parser')
    tabl = soup.find_all("div", {"class" : "M(0) Whs(n) BdEnd Bdc($seperatorColor) D(itb)"})
    for t in tabl:
        rows = t.find_all("div", {"class" : "rw-expnded"})
        for row in rows:
```

```
        temp_dir[row.get_text(separator='|').split("|")[0]]=row.get_text(separator='|').split("|")[1]
        #getting income statement data from yahoo finance for the given ticker
    url = 'https://in.finance.yahoo.com/quote/'+ticker+'/financials?p='+ticker
    page = requests.get(url)
    page_content = page.content
    soup = BeautifulSoup(page_content,'html.parser')
    tabl = soup.find_all("div", {"class" : "M(0) Whs(n) BdEnd Bdc($seperatorColor) D(itb)"})
    for t in tabl:
        rows = t.find_all("div", {"class" : "rw-expnded"})
        for row in rows:
           temp_dir[row.get_text(separator='|').split("|")[0]]=row.get_text(separator='|').split("|")[1]
    #getting cashflow statement data from yahoo finance for the given ticker
    url = 'https://in.finance.yahoo.com/quote/'+ticker+'/cash-flow?p='+ticker
  page = requests.get(url)
    page_content = page.content
    soup = BeautifulSoup(page_content,'html.parser')
    tabl = soup.find_all("div", {"class" : "M(0) Whs(n) BdEnd Bdc($seperatorColor) D(itb)"})
    for t in tabl:
        rows = t.find_all("div", {"class" : "rw-expnded"})
        for row in rows:
         temp_dir[row.get_text(separator='|').split("|")[0]]=row.get_text(separator='|').split("|")[1]
     #getting key statistics data from yahoo finance for the given ticker
    url = 'https://in.finance.yahoo.com/quote/'+ticker+'/key-statistics?p='+ticker
    page = requests.get(url)
    page_content = page.content
    soup = BeautifulSoup(page_content,'html.parser')
    tabl = soup.findAll("table", {"class": "W(100%) Bdcl(c) "})
    for t in tabl:
        rows = t.find_all("tr")
        for row in rows:
            if len(row.get_text(separator='|').split("|")[0:2])>0:
             temp_dir[row.get_text(separator='|').split("|")[0]]=row.get_text(separator='|').split("|")[-1]
    #combining all extracted information with the corresponding ticker
    financial_dir[ticker] = temp_dir
#storing information in pandas dataframe
combined_financials = pd.DataFrame(financial_dir)
tickers = combined_financials.columns
for ticker in tickers:combined_financials =combined_financials[~combined_financials[ticker].str.contains("[a-
z]").fillna(False)]
```

Step 4:- Visualization Techniques

Explanation:-

Visualization techniques which means drawing plots etc. for my data.The library that is used for visualization in Python is Matplotlib.

Installing Matplotlib:-

- -m pip install -U matplotlib

Code:-

```
import pandas as pd
import yfinance as yf
import datetime
import matplotlib.pyplot as plt

# Download historical data for required stocks
tickers = ["MSFT","AMZN","AAPL","CSCO","IBM","FB"]

close_prices = pd.DataFrame() # empty dataframe which will be filled with closing prices of each stock
start = dt.datetime.today()-dt.timedelta(3650)
end = dt.datetime.today()

# looping over tickers and creating a dataframe with close prices
for ticker in tickers:
    close_prices[ticker] = yf.download(ticker,start,end)["Adj Close"]

# Handling NaN Values
close_prices.fillna(method='bfill',axis=0,inplace=True) # Replaces NaN values with the next valid value along
the column
daily_return = close_prices.pct_change() # Creates dataframe with daily return for each stock

# Data vizualization
close_prices.plot() # Plot of all the stocks superimposed on the same chart

cp_standardized = (close_prices - close_prices.mean())/close_prices.std() # Standardization
cp_standardized.plot() # Plot of all the stocks standardized and superimposed on the same chart

close_prices.plot(subplots=True, layout = (3,2), title = "Tech Stock Price Evolution", grid =True) # Subplots of
the stocks

# Pyplot demo
fig, ax = plt.subplots()
```

```
plt.style.available
plt.style.use('ggplot')
ax.set(title="Daily return on tech stocks", xlabel="Tech Stocks", ylabel = "Daily Returns")
plt.bar(daily_return.columns,daily_return.mean())
```

Step 5:- Technical Indicators

Explanation:-

Mathematical calculations which is based on the historical price, volume, open interest etc.and thes mathematical calculations try to predict or forecast what the asset price is going to be in the future. In my project I have used MACD as the techinal indicator for predicting my stock buy and sell stock. MACD is also know as moving average covergence divergence. MACD is calculated is that we calculate a fast moving average which is typically a 12 period moving average and a slow moving average which is typically a 26 period moving average. And then we take the difference of these two which is then called the MACD line.

MACD Implementation in Python:-

Explanation:-

In this I'll be using Microsoft's stock price. I have already extracted. OHLCV data using Panda's data reader get_data_yahoo module. This is the OHLCV data. I have used five years data.

Code:-

```
import yfinance as yf
import datetime as dt
import matplotlib.pyplot as plt

# Download historical data for required stocks
ticker = "MSFT"
ohlcv = yf.download(ticker,dt.date.today()-dt.timedelta(1825),dt.datetime.today())

def MACD(DF,a,b,c):
    """function to calculate MACD
       typical values a = 12; b =26, c =9"""
    df = DF.copy()
    df["MA_Fast"]=df["Adj Close"].ewm(span=a,min_periods=a).mean()
    df["MA_Slow"]=df["Adj Close"].ewm(span=b,min_periods=b).mean()
    df["MACD"]=df["MA_Fast"]-df["MA_Slow"]
    df["Signal"]=df["MACD"].ewm(span=c,min_periods=c).mean()
    df.dropna(inplace=True)
    return df

# Visualization - plotting MACD/signal along with close price and volume for last 100 data points
df = MACD(ohlcv, 12, 26, 9)
```

```
plt.subplot(311)
plt.plot(df.iloc[-100:,4])
plt.title('MSFT Stock Price')
plt.xticks([])

plt.subplot(312)
plt.bar(df.iloc[-100:,5].index, df.iloc[-100:,5].values)
plt.title('Volume')
plt.xticks([])

plt.subplot(313)
plt.plot(df.iloc[-100:,[-2,-1]])
plt.title('MACD')
plt.legend(('MACD','Signal'),loc='lower right')

plt.show()

# Visualization - Using object orient approach
# Get the figure and the axes
fig, (ax0, ax1) = plt.subplots(nrows=2,ncols=1, sharex=True, sharey=False, figsize=(10, 6), gridspec_kw =
{'height_ratios':[2.5, 1]})
df.iloc[-100:,4].plot(ax=ax0)
ax0.set(ylabel='Adj Close')

df.iloc[-100:,[-2,-1]].plot(ax=ax1)
ax1.set(xlabel='Date', ylabel='MACD/Signal')

# Title the figure
fig.suptitle('Stock Price with MACD', fontsize=14, fontweight='bold')
```

Step 6:- CAGR Implementation in Python

Explanation:-

Before we get into back testing our strategies it's important that we have a fairly good understanding of performance measurement and by performance measurement. I mean how we are going to quantify how a particular trading style strategy performed.
CAGR also known as compounded annual growth rate.
When you are interested in investing in a mutual fund or any fund the first thing you look at is what was the annualized return that fund has been generating over the past year.

Code:-

```
import yfinance as yf
import numpy as np
import datetime as dt

# Download historical data for required stocks
ticker = "^GSPC"
SnP = yf.download(ticker,dt.date.today()-dt.timedelta(1825),dt.datetime.today())


def CAGR(DF):
    "function to calculate the Cumulative Annual Growth Rate of a trading strategy"
    df = DF.copy()
    df["daily_ret"] = DF["Adj Close"].pct_change()
    df["cum_return"] = (1 + df["daily_ret"]).cumprod()
    n = len(df)/252
    CAGR = (df["cum_return"][-1])**(1/n) - 1
    return CAGR
```

<u>Step 7</u>:- Backtest Your Strategies

<u>Explanation</u>:-

In this section we'll talk about back testing so till now we have got the basic tools to build a creating strategy or creating system. We know how to get data from various sources we know how to perform basic statistical analysis on that data how do we know how to code. Technical indicators and use them for developing a strategy. However before we bring all these tools together and formulate a strategy it's very important that we back test them before deploying our strategy or creating system on the real world data or in the real world scenarios. Although I'll be skipping this part and will be using my strategies and script in the demo account of FXCM. So I'll be directly jumping to my last part of this project where I'll directly implement my script on the authorized stock broker demo account FXCM.

Step 8:- Implementation of my algorithm Script on FXCM.

Explanation:-

Basically FXCM is a trading platform where a Forex market is listed.Therefore I’ll be using demo account of FXCM where I’ll implement my startegy. This would be done by generating an API from an FXCM terminal. Before jumping into this part we have to install FXCM library in python

Installation of the fxcm Python package:--

- $ pip install fxcmpy

The final code will be shown below where I have connected my terminal with FXCM terminal using API here you will see that I have used indicator(MACD & Renko) also apart from that I have used basic measure step like handling NaN value, value investing.

Code:-

```
import fxcmpy
import numpy as np
from stocktrends import Renko
import statsmodels.api as sm
import time
import copy

#initiating API connection and defining trade parameters
token_path = "D:\\Udemy\\API\\api.txt"
con = fxcmpy.fxcmpy(access_token = open(token_path,'r').read(), log_level = 'error', server='demo')

#defining strategy parameters
pairs = ['EUR/USD','GBP/USD','USD/CHF','AUD/USD','USD/CAD'] #currency pairs to be included in
    the strategy
#pairs = ['EUR/JPY','USD/JPY','AUD/JPY','AUD/NZD','NZD/USD']
pos_size = 10 #max capital allocated/position size for any currency pair


def MACD(DF,a,b,c):
    """function to calculate MACD
       typical values a = 12; b =26, c =9"""
    df = DF.copy()
    df["MA_Fast"]=df["Close"].ewm(span=a,min_periods=a).mean()
    df["MA_Slow"]=df["Close"].ewm(span=b,min_periods=b).mean()
    df["MACD"]=df["MA_Fast"]-df["MA_Slow"]
    df["Signal"]=df["MACD"].ewm(span=c,min_periods=c).mean()
    df.dropna(inplace=True)
    return (df["MACD"],df["Signal"])

def ATR(DF,n):
    "function to calculate True Range and Average True Range"
    df = DF.copy()
    df['H-L']=abs(df['High']-df['Low'])
    df['H-PC']=abs(df['High']-df['Close'].shift(1))
    df['L-PC']=abs(df['Low']-df['Close'].shift(1))
    df['TR']=df[['H-L','H-PC','L-PC']].max(axis=1,skipna=False)
    df['ATR'] = df['TR'].rolling(n).mean()
    #df['ATR'] = df['TR'].ewm(span=n,adjust=False,min_periods=n).mean()
    df2 = df.drop(['H-L','H-PC','L-PC'],axis=1)
    return df2

def slope(ser,n):
    "function to calculate the slope of n consecutive points on a plot"
    slopes = [i*0 for i in range(n-1)]
    for i in range(n,len(ser)+1):
        y = ser[i-n:i]
        x = np.array(range(n))
        y_scaled = (y - y.min())/(y.max() - y.min())
        x_scaled = (x - x.min())/(x.max() - x.min())
        x_scaled = sm.add_constant(x_scaled)
        model = sm.OLS(y_scaled,x_scaled)
        results = model.fit()
        slopes.append(results.params[-1])
    slope_angle = (np.rad2deg(np.arctan(np.array(slopes))))
    return np.array(slope_angle)

def renko_DF(DF):
    "function to convert ohlc data into renko bricks"
    df = DF.copy()
    df.reset_index(inplace=True)
    df = df.iloc[:,[0,1,2,3,4,5]]
    df.columns = ["date","open","close","high","low","volume"]
    df2 = Renko(df)
    df2.brick_size = round(ATR(DF,120)["ATR"][-1],4)
    renko_df = df2.get_bricks()
    renko_df["bar_num"] =
    np.where(renko_df["uptrend"]==True,1,np.where(renko_df["uptrend"]==False,-1,0))
    for i in range(1,len(renko_df["bar_num"])):
        if renko_df["bar_num"][i]>0 and renko_df["bar_num"][i-1]>0:
            renko_df["bar_num"][i]+=renko_df["bar_num"][i-1]
        elif renko_df["bar_num"][i]<0 and renko_df["bar_num"][i-1]<0:
            renko_df["bar_num"][i]+=renko_df["bar_num"][i-1]
    renko_df.drop_duplicates(subset="date",keep="last",inplace=True)
    return renko_df

def renko_merge(DF):
    "function to merging renko df with original ohlc df"
    df = copy.deepcopy(DF)
    df["Date"] = df.index
    renko = renko_DF(df)
    renko.columns = ["Date","open","high","low","close","uptrend","bar_num"]
    merged_df = df.merge(renko.loc[:,["Date","bar_num"]],how="outer",on="Date")
    merged_df["bar_num"].fillna(method='ffill',inplace=True)
    merged_df["macd"]= MACD(merged_df,12,26,9)[0]
    merged_df["macd_sig"]= MACD(merged_df,12,26,9)[1]
    merged_df["macd_slope"] = slope(merged_df["macd"],5)
    merged_df["macd_sig_slope"] = slope(merged_df["macd_sig"],5)
    return merged_df

def trade_signal(MERGED_DF,l_s):
    "function to generate signal"
    signal = ""
    df = copy.deepcopy(MERGED_DF)
    if l_s == "":
        if df["bar_num"].tolist()[-1]>=2 and df["macd"].tolist()[-1]>df["macd_sig"].tolist()[-1] and
      df["macd_slope"].tolist()[-1]>df["macd_sig_slope"].tolist()[-1]:
            signal = "Buy"
        elif df["bar_num"].tolist()[-1]<=-2 and df["macd"].tolist()[-1]<df["macd_sig"].tolist()[-1] and
      df["macd_slope"].tolist()[-1]<df["macd_sig_slope"].tolist()[-1]:
            signal = "Sell"

    elif l_s == "long":
        if df["bar_num"].tolist()[-1]<=-2 and df["macd"].tolist()[-1]<df["macd_sig"].tolist()[-1] and
      df["macd_slope"].tolist()[-1]<df["macd_sig_slope"].tolist()[-1]:
            signal = "Close_Sell"
        elif df["macd"].tolist()[-1]<df["macd_sig"].tolist()[-1] and df["macd_slope"].tolist()[-
      1]<df["macd_sig_slope"].tolist()[-1]:
            signal = "Close"

    elif l_s == "short":
        if df["bar_num"].tolist()[-1]>=2 and df["macd"].tolist()[-1]>df["macd_sig"].tolist()[-1] and
      df["macd_slope"].tolist()[-1]>df["macd_sig_slope"].tolist()[-1]:
            signal = "Close_Buy"
        elif df["macd"].tolist()[-1]>df["macd_sig"].tolist()[-1] and df["macd_slope"].tolist()[-
      1]>df["macd_sig_slope"].tolist()[-1]:
            signal = "Close"
    return signal


def main():
    try:
        open_pos = con.get_open_positions()
        for currency in pairs:
            long_short = ""
            if len(open_pos)>0:
                open_pos_cur = open_pos[open_pos["currency"]==currency]
                if len(open_pos_cur)>0:
                    if open_pos_cur["isBuy"].tolist()[0]==True:
                        long_short = "long"
                    elif open_pos_cur["isBuy"].tolist()[0]==False:
                        long_short = "short"
            data = con.get_candles(currency, period='m5', number=250)
            ohlc = data.iloc[:,[0,1,2,3,8]]
            ohlc.columns = ["Open","Close","High","Low","Volume"]
            signal = trade_signal(renko_merge(ohlc),long_short)
            if signal == "Buy":
                con.open_trade(symbol=currency, is_buy=True, is_in_pips=True, amount=pos_size,
                               time_in_force='GTC', stop=-8, trailing_step =True, order_type='AtMarket')
                print("New long position initiated for ", currency)
            elif signal == "Sell":
                con.open_trade(symbol=currency, is_buy=False, is_in_pips=True, amount=pos_size,
                               time_in_force='GTC', stop=-8, trailing_step =True, order_type='AtMarket')
                print("New short position initiated for ", currency)
            elif signal == "Close":
                con.close_all_for_symbol(currency)
                print("All positions closed for ", currency)
            elif signal == "Close_Buy":
                con.close_all_for_symbol(currency)
                print("Existing Short position closed for ", currency)
                con.open_trade(symbol=currency, is_buy=True, is_in_pips=True, amount=pos_size,
                               time_in_force='GTC', stop=-8, trailing_step =True, order_type='AtMarket')
                print("New long position initiated for ", currency)
            elif signal == "Close_Sell":
                con.close_all_for_symbol(currency)
                print("Existing long position closed for ", currency)
                con.open_trade(symbol=currency, is_buy=False, is_in_pips=True, amount=pos_size,
                               time_in_force='GTC', stop=-8, trailing_step =True, order_type='AtMarket')
                print("New short position initiated for ", currency)
    except:
        print("error encountered... skipping this iteration")


# Continuous execution
starttime=time.time()
timeout = time.time() + 60*60*1 # 60 seconds times 60 meaning the script will run for 1 hr
while time.time() <= timeout:
    try:
        print("passthrough at ",time.strftime('%Y-%m-%d %H:%M:%S', time.localtime(time.time())))
        main()
        time.sleep(300 - ((time.time() - starttime) % 300.0)) # 5 minute interval between each new
      execution
    except KeyboardInterrupt:
        print('\n\nKeyboard exception received. Exiting.')
        exit()

# Close all positions and exit
for currency in pairs:
    print("closing all positions for ",currency)
    con.close_all_for_symbol(currency)
con.close()
```

CONCLUSION

Finally the project came to an end by finally implementing it into broker terminal and checking my script and I came to conclusion that this system can be beneficial for professional trader who simply have to implement their indicator algorithm in code and execute it to buy and sell stock. Lastly we want to thank our mentor under whose guidance we were able to make this innovative project.

FUTURE SCOPE

From my point view if some modification or some innovative algorithm is used or implemented this project can bring a big change in the stock market or financial world. Not only this a perfect implementation can bring a huge profit to a trader.

REFERENCES

- **https://pypi.org/project/yfinance/**
- **https://www.fxcm.com/markets/algorithmic-trading/forex-python/**
- **https://in.finance.yahoo.com/**
- **https://matplotlib.org/**
- **https://pandas.pydata.org/**
- **https://www.udemy.com/**