**Project Name**: Automated Algorithmic Stock Trading System using Machine Learning

**Group Number**: 8

**Index:**

|  |  |
| --- | --- |
| **Title** | **Page No.** |
| 1. Participants | 2 |
| 1. Project Workflow | 3 |
| 1. Project Abstract | 3 - 4 |
| 1. Data Specification | 4 - 5 |
| 1. Project Design | 5 - 10 |
| 1. Project Milestones | 11 |
| 1. Repository / Archive | 11 |
| 1. Code | 11 - 19 |
| 1. Resources and Related Projects | 20 |

**Participants**:

| Participant Name | Email | Roles |
| --- | --- | --- |
| Anjan Shrestha | anjanshrestha@my.unt.edu | Build platform to automate trading by using trained machine learning model |
| Naresh Balla | NareshBalla@my.unt.edu | Clean dataset and perform data preprocessing |
| Bhargav Ram Pushadapu | BhargavRamPushadapu@my.unt.edu | Integrate trading system with Alpaca Trading Broker API |
| Byanagari Rohith | byanagarirohith@my.unt.edu | Perform exploratory data analysis on the dataset to know more about the dataset |
| Balaji Mandava | BalajiMandava@my.unt.edu | Train LSTM model and create output file for trained model |

**Project Workflow:**

We've created a WhatsApp group for this project's teamwork to improve communication. To improve collaboration, we set up a GitHub project to store all our code in one location. Additionally, we have scheduled a zoom meeting among team members for each milestone to assess our progress and address any roadblocks. For tracking tasks, we have used Trello software and a production project plan template in google sheet. Each team member will participate in research, planning, design, model implementation, training, validation, testing, and monitoring so that everyone in the team will have the knowledge necessary to create an effective model right away.

|  |  |
| --- | --- |
| Github Link | <https://github.com/anjanshrestha123/automated-algorithmic-stock-trading-system> |
| Dataset | [Yahoo](https://www.zillow.com/research/data/) Finance API |
| Production Project Plan | [https://docs.google.com/spreadsheets/d/1KnotKDuMn6CdcOR6lmuMMhLGPv7c7GKttz6CBhznAGM/edit - gid=1853352180](https://docs.google.com/spreadsheets/d/1KnotKDuMn6CdcOR6lmuMMhLGPv7c7GKttz6CBhznAGM/edit#gid=1853352180) |
| Trello Link | <https://trello.com/b/DAiV60r9/project-2> |

**Project Abstract:**

The stock market is known for its dynamic and volatile nature, making it extremely challenging to make accurate predictions due to the numerous factors that affect stock prices, such as news, events, financial performance, and sentiment. There are three main types of stock analysis: fundamental analysis, technical analysis, and sentimental analysis. For this project, we will focus on technical analysis for predicting stock prices and automating stock trading using Machine Learning algorithm.

Technical analysis uses historical data such as stock prices, returns, and trading volumes to identify patterns in market movements and trading signals. This type of analysis is typically used for short-term trading, such as hourly, daily, weekly, or monthly, and can result in high returns if predicted accurately. Our goal is to use various python modules to extract stock prices, train and test our model, predict the prices of different stocks using technical analysis, and create a system that automates the stock trading by using prediction from the trained model.

**Data specification:**

**Table

Description automatically generated**

*Fig. Top 5 rows of dataset*

We are using the dataset provided in yahoo finance API that contains stock price data. We want to get stock data in minute range as we are doing the short-term analysis. Since, yahoo finance API only provides minute range stock data for last 30 days, we are going to build our model based on the last 30 days dataset. After getting last 30 days of data, we got 7786 rows and 6 columns. Out of this 6 columns, date and adjusted closing is our feature and target variable. Date is the categorical variable whereas adjusted closing price is the quantitative variable.

We used LSTM as a machine learning model because our problem involves predicting a number, and we used root mean square error to assess our model. Additionally, we used 70% of the data to train the model and 30% for testing.

**Project Design:**

**Diagram

Description automatically generated**

*Fig. High Level Overview*

Our goal is to create three different applications to build an automated algorithmic stock trading system. In project 2, we built the machine learning model and automated trading system. And, in project 3, we will be extending this system to have client-side application and deploy it to AWS cloud.

1. Machine Learning Model:

In project 2, we built Machine Learning Model which utilized data from yahoo finance API, preprocessed it, plotted different graphs for visualization, trained and tested the model using LSTM algorithm, and predicted the stock price using the trained model. We used Google Collaboratory and Jupyter Notebook as an IDE and used Python programming language with different libraries such as NumPy, pandas, sklearn, TensorFlow, matplotlib and so on.

1. Automated Trading System:

The second application is automated trading system which was built in project 2 that runs 24/7 and gets the prediction from trained machine learning model and invokes Alpaca trading broker API to buy and sell stock. For the project, we used paper trading API so that real money won’t be used. PyCharm was used as an IDE and various backend technologies was used such as Python, JSON, and so on.

1. Client-Side Application (Website):

In project 3, we will be building third application i.e., client-side website where users can interact and see their live transactions being traded in Alpaca broker by automated trading system. We will be using Visual Studio Code as an IDE and different front-end technologies such as HTML, CSS, Material Design, JavaScript, Angular, and so on.

Exploratory Data Analysis:

**Chart

Description automatically generated**

*Fig. Exploratory Data Analysis on Apple Stock*

Before running the model, exploratory data analysis has been carried out to learn more about the data. For this research, we have plotted Apple's stock price versus date for the previous 30 days. We can see from this plot that the stock price is going in an upward direction, which indicates that there is a strong likelihood that the stock price will rise.

Model Evaluation:

**Graphical user interface, text, application

Description automatically generated**

*Fig. RMSE score of the trained model.*

The LSTM model has been trained using the training dataset after feature extraction. The model is then assessed using a testing dataset that has had the root mean square error calculated which is found to be 0.2267.

Store trained model:

Graphical user interface, text, application

Description automatically generated

*Fig. ML code to generate output to pickle file.*

The trained model and other necessary variables have been stored to output after the model has been evaluated. It is then picked up by the trading system for further processing.

Run Trading System:

Text

Description automatically generated

*Fig. Running console log of trading system.*

Trading System reads the trained model stored in the output pickle file and makes predictions for the whole day in the minute range. Also, it calls the Yahoo finance API to get the current stock price. Here, if one of the predictions is greater than the current stock price, it places a bracket order by calling Alpaca Trading Broker API. The above figure shows the logs for current price, predicted price and execution of bracket order for Apple and Tesla when the trading system ran.

Generate output file to keep track of traded stock:

Graphical user interface, text

Description automatically generated

*Fig. Output file generated by trading system.*

After trading system places a bracket order, it updates files with the stock name, prices, and quantity to keep track of the stocks that is being traded. If all the stock have been traded, then the trading system stops running for the day.

Order Execution in Alpaca broker by Trading System:

Graphical user interface

Description automatically generated

*Fig. UI of Alpaca broker showing the recent orders placed by Trading System.*

When trading system makes a call via Alpaca Trading Broker API, the bracket order executes immediately. We can see the order in the Alpaca UI as shown in the figure above. We can see that bracket order for Apple and Tesla have been placed by trading system.

**Project Milestones:**

This project was divided into two main milestones that are described below:

|  |  |  |
| --- | --- | --- |
| Milestone | Date | Incremental Feature |
| 1 | April 7th, 2023 | Build Machine learning model |
| 2 | April 14th, 2023 | Build automated trading system and complete project report |

**Repository / Archive:**

GitHub Repository Link:

<https://github.com/anjanshrestha123/automated-algorithmic-stock-trading-system>

How to setup our project:

<https://github.com/anjanshrestha123/automated-algorithmic-stock-trading-system/blob/master/README.md>

**Code: Provide a PDF of all the code:**

* + - 1. Machine Learning Model:

# generate-model.ipynb

### Import and Packages ###

import pandas as pd

import numpy as np

import yfinance as yf

import datetime as dt

from datetime import timedelta

import math

import os

import pickle

import matplotlib.pyplot as plt

import math

from sklearn.preprocessing import MinMaxScaler

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Dense

from tensorflow.keras.layers import LSTM

from sklearn.metrics import mean\_squared\_error

# Dataset Properties

DATE = 'Date'

CLOSE = 'Close'

VOLUME = 'Volume'

# Stock Properties

STOCK\_TICKER = os.environ['STOCK\_TICKER'] if 'STOCK\_TICKER' in os.environ else 'AAPL' # Retrive stock ticker from environment variable

# Target

NUMBER\_OF\_DAYS\_TO\_PREDICT = 30 # Target Stock Trend

# Custom Hyperparameters

NUMBER\_OF\_INTERVAL\_FOR\_PRICE\_PREDICTION = 20

# LSTM Hyperparameters

NUMBER\_OF\_EPOCH = 25

BATCH\_SIZE = 30

df = pd.DataFrame()

# Loop call to yfinance API since data can be fetched for 7 days at once with maximum of 30 days

total\_days = 29

for n in range(math.ceil(total\_days/7)):

days = total\_days - n\*7

start\_date = dt.datetime.today() - dt.timedelta(days=days)

print(start\_date)

end\_date = dt.datetime.today() - dt.timedelta(days= (days - 7 if days > 7 else 0))

df = pd.concat([df, yf.download(STOCK\_TICKER, start=start\_date, end=end\_date, interval='1m')])

df.shape

df.head(5)

# Plotting the graph visualizing price change with date

df.plot(y=[CLOSE],figsize=(15,10), ylabel='Stock Price', title='Stock Price Movement for last 30 days')

# Plotting moving average

plt.figure(figsize=(15,10))

df[CLOSE].rolling(window=20).mean().plot(label='20 Minute Average')

df[CLOSE].plot(label='Closing Price', title='20 Minute Moving Average')

plt.legend()

# Function to extract Stock Closing Price as a Feature from Dataframe

model\_df = df.reset\_index()[CLOSE]

scaler = MinMaxScaler(feature\_range=(0,1))

model\_df = scaler.fit\_transform(np.array(model\_df).reshape(-1,1))

# Splitting first 70% data into training set and last 30% into testing set

train\_index = 0.7 \* model\_df.shape[0]

train\_data = model\_df[:int(train\_index)]

test\_data = model\_df[int(train\_index):]

# Function to create dataset into feature and target

def create\_dataset\_lstm(dataset, time\_step=1):

dataX, dataY = [], []

for i in range(len(dataset) - time\_step-1):

a = dataset[i:(i+time\_step), 0]

dataX.append(a)

dataY.append(dataset[i+time\_step, 0])

return np.array(dataX), np.array(dataY)

# Creating training and testing dataset

X\_train, y\_train = create\_dataset\_lstm(train\_data, NUMBER\_OF\_INTERVAL\_FOR\_PRICE\_PREDICTION)

X\_test, y\_test = create\_dataset\_lstm(test\_data, NUMBER\_OF\_INTERVAL\_FOR\_PRICE\_PREDICTION)

# reshape input to be [samples, time steps, features] which is required for LSTM

X\_train = X\_train.reshape(X\_train.shape[0], X\_train.shape[1], 1)

X\_test = X\_test.reshape(X\_test.shape[0], X\_test.shape[1], 1)

model = Sequential()

model.add(LSTM(50, return\_sequences=True, input\_shape=(NUMBER\_OF\_INTERVAL\_FOR\_PRICE\_PREDICTION,1)))

model.add(LSTM(50, return\_sequences=True))

model.add(LSTM(50))

model.add(Dense(1))

model.compile(loss='mean\_squared\_error', optimizer='adam', metrics=['accuracy'])

model.fit(X\_train, y\_train, validation\_data=(X\_test, y\_test), epochs=NUMBER\_OF\_EPOCH, batch\_size=BATCH\_SIZE, verbose=0)

y\_pred = model.predict(X\_test)

y\_pred = scaler.inverse\_transform(y\_pred)

y\_test = scaler.inverse\_transform(np.array(y\_test).reshape(-1,1))

current\_rmse = math.sqrt(mean\_squared\_error(y\_pred, y\_test))

# Output to pickle file

output = {

'trained\_model': model,

'latest\_interval\_data': np.array(model\_df[len(model\_df) - NUMBER\_OF\_INTERVAL\_FOR\_PRICE\_PREDICTION:]),

'scaler': scaler,

'rmse': current\_rmse

}

with open('./output/output-{}.pickle'.format(STOCK\_TICKER), 'wb') as f:

pickle.dump(output, f)

# config.py

import configparser

# Read App configuration from a file

config = configparser.RawConfigParser()

config.read('app.properties')

# app.properties

[App]

stock.list.file.path=../../resources/stock-list.txt

ipynb.file.path=./generate-model.ipynb

# run.py

import os

import nbformat

from nbconvert.preprocessors import ExecutePreprocessor

from config.config import config

# Setting up jupyter filename to run

with open(config.get('App', 'ipynb.file.path')) as ff:

nb\_in = nbformat.read(ff, nbformat.NO\_CONVERT)

ep = ExecutePreprocessor(timeout=600, kernel\_name='python3')

# Generate trained model for each stock in the list

for stock\_ticker in open(config.get('App', 'stock.list.file.path'), 'r').read().splitlines():

os.environ['STOCK\_TICKER'] = stock\_ticker.strip().upper()

# Run the notebook

print('Generating trained model for ', os.environ['STOCK\_TICKER'])

ep.preprocess(nb\_in)

* + - 1. Automated Trading System:

# config.py

import configparser

import datetime

# Read App configuration from a file

config = configparser.RawConfigParser()

config.read('app.properties')

def get\_model\_output\_path(stock\_ticker):

return config.get('App', 'model.output.path').format(stock\_ticker=stock\_ticker)

def get\_stock\_list\_file\_path():

return config.get('App', 'stock.list.file.path')

def get\_traded\_stock\_info\_list\_file\_path():

return config.get('App', 'traded.stock.info.list.file.path').format(datetime.datetime.now())

def get\_traded\_stock\_info(stock\_ticker, current\_price, threshold\_signal\_price, take\_profit\_signal\_price, stop\_loss\_price, quantity):

return config.get('App', 'traded.stock.info.pattern').format(

stock\_ticker=stock\_ticker,

current\_price=current\_price,

threshold\_signal\_price=threshold\_signal\_price,

take\_profit\_signal\_price=take\_profit\_signal\_price,

stop\_loss\_price=stop\_loss\_price,

quantity=quantity)

def get\_stock\_market\_time\_zone():

return config.get('App', 'stock.market.time.zone')

def get\_stock\_market\_open\_hour():

return int(config.get('App', 'stock.market.open.hour'))

def get\_stock\_market\_open\_minute():

return int(config.get('App', 'stock.market.open.minute'))

def get\_stock\_market\_close\_hour():

return int(config.get('App', 'stock.market.close.hour'))

def get\_stock\_market\_close\_minute():

return int(config.get('App', 'stock.market.close.minute'))

def get\_number\_of\_predictions():

return int(config.get('App', 'number.of.predictions'))

def get\_threshold\_signal\_in\_pct():

return int(config.get('App', 'threshold.signal.in.pct'))

def get\_stop\_loss\_signal\_in\_pct():

return int(config.get('App', 'stop.loss.signal.in.pct'))

def get\_take\_profit\_signal\_in\_pct():

return int(config.get('App', 'take.profit.signal.in.pct'))

def get\_alpaca\_api\_key():

return config.get('Alpaca', 'alpaca.api.key')

def get\_alpaca\_api\_secret():

return config.get('Alpaca', 'alpaca.api.secret')

# alpaca\_broker\_proxy.py

from alpaca.trading.requests import MarketOrderRequest

from alpaca.trading.client import TradingClient

from alpaca.trading.enums import OrderSide, OrderType, OrderClass, TimeInForce

from config import config

def create\_trading\_client():

return TradingClient(config.get\_alpaca\_api\_key(), config.get\_alpaca\_api\_secret(), paper=True)

def place\_bracket\_order(stock\_ticker, stop\_loss\_price, take\_profit\_price, quantity):

# Create trading client

trading\_client = create\_trading\_client()

# Create market order request

market\_order\_request = MarketOrderRequest(

symbol=stock\_ticker,

qty=quantity,

side=OrderSide.BUY,

type=OrderType.MARKET,

time\_in\_force=TimeInForce.GTC,

order\_class=OrderClass.BRACKET,

stop\_loss={'stop\_price': stop\_loss\_price},

take\_profit={'limit\_price': take\_profit\_price}

)

# Execute bracket order

trading\_client.submit\_order(market\_order\_request)

# stock\_price\_ml\_proxy.py

import pickle

from config import config

import numpy as np

def read\_model\_output(stock\_ticker):

with open(config.get\_model\_output\_path(stock\_ticker), 'rb') as f:

model\_output = pickle.load(f)

return model\_output

def predict\_stock\_price(stock\_ticker, no\_of\_predictions):

print('Predicting stock prices for [{}] for next [{}] intervals'.format(stock\_ticker, no\_of\_predictions))

# Read model output

model\_output = read\_model\_output(stock\_ticker)

# Store the values into different variables

model = model\_output['trained\_model']

rmse = model\_output['rmse']

latest\_interval\_data = model\_output['latest\_interval\_data']

scaler = model\_output['scaler']

predicted\_prices = []

for i in range(no\_of\_predictions):

# Make prediction for next interval

last\_n\_days\_data = latest\_interval\_data.reshape(latest\_interval\_data.shape[1], latest\_interval\_data.shape[0], 1)

prediction = model.predict(last\_n\_days\_data)

transformed\_prediction = scaler.inverse\_transform(prediction)

# Prepare data for next prediction

latest\_interval\_data = np.delete(latest\_interval\_data, 0, 0)

latest\_interval\_data = np.append(latest\_interval\_data, prediction, axis=0)

# Store prediction values

predicted\_prices.append(transformed\_prediction[0][0])

print('Predicted stock prices for [{}] are {}'.format(stock\_ticker, predicted\_prices))

return predicted\_prices

# yahoo\_finance\_proxy.py

import yfinance as yf

import datetime as dt

def get\_current\_price(stock\_ticker):

data = yf.download(stock\_ticker, start=dt.datetime.today() - dt.timedelta(days= 1),end=dt.datetime.today(), interval='1m')

return round(data[-1:]['Close'].values[0], 2)

# stock\_trade\_service.py

from proxy import stock\_price\_ml\_proxy

from proxy import yahoo\_finance\_proxy

from proxy import alpaca\_broker\_proxy

from util import file\_util

from pytz import timezone

from config import config

import datetime

predicted\_prices\_by\_stock\_ticker = {}

def is\_stock\_market\_open():

current\_date\_time = datetime.datetime.now(timezone(config.get\_stock\_market\_time\_zone()))

# 0=Monday, 1=Tuesday, 2=Wednesday, 3=Thursday, 4=Friday, 5=Saturday, 6=Sunday

is\_weekday = current\_date\_time.weekday() <= 4

# check for stock market opening and closing time

stock\_market\_open\_time = datetime.time(config.get\_stock\_market\_open\_hour(), config.get\_stock\_market\_open\_minute())

stock\_market\_close\_time = datetime.time(config.get\_stock\_market\_close\_hour(), config.get\_stock\_market\_close\_minute())

is\_stock\_market\_office\_hour = stock\_market\_open\_time < current\_date\_time.time() < stock\_market\_close\_time

return is\_weekday and is\_stock\_market\_office\_hour

def trade\_stock(tradeless\_stock\_ticker\_list):

for stock\_ticker in tradeless\_stock\_ticker\_list:

# Fetching general price information

print('Started the process to trade [{}] stock'.format(stock\_ticker))

current\_price = yahoo\_finance\_proxy.get\_current\_price(stock\_ticker)

threshold\_signal\_price = round(current\_price + ((config.get\_threshold\_signal\_in\_pct() \* current\_price) / 100), 2)

take\_profit\_signal\_price = round(current\_price + ((config.get\_take\_profit\_signal\_in\_pct() \* current\_price) / 100), 2)

stop\_loss\_price = round(current\_price - ((config.get\_stop\_loss\_signal\_in\_pct() \* current\_price) / 100), 2)

print('Current Price: [{}], Threshold Signal Price: [{}], Take Profit Signal Price: [{}], Stop Loss Price: [{}]'

.format(current\_price, threshold\_signal\_price, take\_profit\_signal\_price, stop\_loss\_price))

# Fetching predicted prices from ML model

if stock\_ticker in predicted\_prices\_by\_stock\_ticker:

predicted\_prices = predicted\_prices\_by\_stock\_ticker[stock\_ticker]

else:

predicted\_prices = stock\_price\_ml\_proxy.predict\_stock\_price(stock\_ticker, config.get\_number\_of\_predictions())

predicted\_prices\_by\_stock\_ticker[stock\_ticker] = predicted\_prices

# Execute order if certain condition is satisfied

if any(predicted\_price >= threshold\_signal\_price for predicted\_price in predicted\_prices):

quantity = 1 # This need to be modified to have some calculation based approach

print('Executing bracket order for [{}] with quantity [{}]'.format(stock\_ticker, quantity))

alpaca\_broker\_proxy.place\_bracket\_order(stock\_ticker, stop\_loss\_price, take\_profit\_signal\_price, quantity)

file\_util.update\_traded\_stock\_to\_file(stock\_ticker, current\_price, threshold\_signal\_price, take\_profit\_signal\_price, stop\_loss\_price, quantity)

else:

print('Predicted prices are not more than threshold signal price')

# file\_util.py

from config import config

import os.path

def fetch\_stock\_ticker\_list():

return [stock\_ticker.strip().upper() for stock\_ticker in open(config.get\_stock\_list\_file\_path(), 'r').read().splitlines()]

def fetch\_tradeless\_stock\_ticker\_list():

stock\_ticker\_list = fetch\_stock\_ticker\_list()

# Populate stock ticker list based on the traded stock info list from the file

traded\_stock\_ticker\_list = []

traded\_stock\_info\_list\_file\_path = config.get\_traded\_stock\_info\_list\_file\_path()

if os.path.isfile(traded\_stock\_info\_list\_file\_path):

traded\_stock\_ticker\_list = [traded\_stock\_info.split(' ')[0]

for traded\_stock\_info in open(traded\_stock\_info\_list\_file\_path, 'r').read().splitlines()]

return list(set(stock\_ticker\_list) - set(traded\_stock\_ticker\_list))

def update\_traded\_stock\_to\_file(stock\_ticker, current\_price, threshold\_signal\_price, take\_profit\_signal\_price, stop\_loss\_price, quantity):

with open(config.get\_traded\_stock\_info\_list\_file\_path(), 'a') as file:

traded\_stock\_info = config.get\_traded\_stock\_info(

stock\_ticker,

current\_price,

threshold\_signal\_price,

take\_profit\_signal\_price,

stop\_loss\_price,

quantity

)

file.write('{}\n'.format(traded\_stock\_info))

# app.properties

[App]

model.output.path=../ml/output/output-{stock\_ticker}.pickle

stock.list.file.path=../../resources/stock-list.txt

traded.stock.info.list.file.path=./output/{:%Y-%m-%d}-predicted-stock-list.txt

traded.stock.info.pattern={stock\_ticker} {current\_price} {threshold\_signal\_price} {take\_profit\_signal\_price} {stop\_loss\_price} {quantity}

stock.market.time.zone=US/Eastern

stock.market.open.hour=9

stock.market.open.minute=30

stock.market.close.hour=16

stock.market.close.minute=0

number.of.predictions=420

threshold.signal.in.pct=10

stop.loss.signal.in.pct=5

take.profit.signal.in.pct=8

[Alpaca]

alpaca.api.key=PK7YZYAA8IVPBFXDTV65

alpaca.api.secret=17p8knApEdzmXjr2t8uWeFZ2t7tUlHqCtE2lnNw1

# run.py

from service import stock\_trade\_service

from util import file\_util

# Runs in infinite loop until exit condition occurs

# Exit condition:

# 1. Stock market has not open yet

# 2. All the stocks listed in the file have been traded for current day

number\_of\_iterations = 1

while True:

print('\n\n \*\*\*\*\* Starting Automated Algorithmic Trading System. Iteration Number: [{}] \*\*\*\*\*'.format(number\_of\_iterations))

if stock\_trade\_service.is\_stock\_market\_open():

tradeless\_stock\_ticker\_list = file\_util.fetch\_tradeless\_stock\_ticker\_list()

if len(tradeless\_stock\_ticker\_list) > 0:

stock\_trade\_service.trade\_stock(tradeless\_stock\_ticker\_list)

else:

print('All the stocks have been traded for today')

break

else:

print('Stock market has not open yet')

break

number\_of\_iterations += number\_of\_iterations

**Resources and Related Projects:**

1. The article discusses building a simple trading system that uses real-time stock price data from Yahoo Finance in one-minute intervals. Instead of personal rules, an ARIMA model is used to make predictions. The system connects with brokers like Robin Hood and Alpaca to execute trades. The system is deployed on AWS, and the user receives Telegram notifications for every action performed by the system.

Reference link: <https://towardsdatascience.com/how-to-create-a-fully-automated-ai-based-trading-system-with-python-708503c1a907>

1. The article discusses using Python machine learning in algorithmic trading and focuses on building a Simple Moving Average (SMA) trading strategy. The SMA is used as a technical indicator to create trading strategies, and a simple crossover strategy is built by calculating two SMAs (shorter and longer) and triggering a trade when the shorter period SMA crosses above the longer period SMA. The article also mentions back testing the strategy.

Reference link: <https://www.analyticsvidhya.com/blog/2022/04/how-to-use-algorithmic-trading-with-machine-learning-in-python/>