NASDAQ Stock Exchange 28-Day Prediction (2022)

Christopher M. Frutos*, Andrew B. Garcia*, Kayleigh E. Movalli*

**Virginia Polytechnic Institute and State University – Graduate School,* cfrutos@vt.edu, agarcia1296@vt.edu, kayleighm@vt.edu

***Abstract*— In this paper, we will employ a novel approach to predicting General Motors' stock prices by experimenting with four machine learning models' and determine which model proves best performance for this application.**

The four models we will be comparing are linear regression, logistic regression model, neural-networks model, and an ensemble model that is comprised of decision tree regressors. These models utilize supervised learning which means they will be training on a labeled dataset that was provided by the NASDAQ official website. The dataset contains market records dating back 5 years from current day of project: approximately May 1st 2017 to April 28th 2022. A dataset of this size should suffice in providing information to train an accurate model for predicting the price of General Motor's stock with a high accuracy. We will also utilize data from Market Insider to incorporate other features with which the models will train on. These features include coal and gas prices, as well as the inflation rates per each year.

***Index Terms*— Machine Learning, Linear Regression, Logistic Linear Regression, Learning Model, Neural Networks, Ensemble Model, Stock Exchange**

I. Introduction

Being able to accurately predict outcomes of the stock market using a machine learning algorithm can prove to be a very profitable and practical for Americans, especially since investing is at an all-time high[1]. The stock market can often be confusing due to the number of volatile factors that go into predicting an outcome of a stock. Many times these factors are unrelated to the company's performance, such as global pandemics, wars, and natural disasters, all of which can have a significant effect on the behavior of certain stocks. Other factors to consider are pop-culture trends, rapidly developing technologies, and legislative bills, since these all can determine what naïve investors may be persuaded to purchase. All of these things are difficult to keep in mind for a naïve investor, but with the help of a learning model we believe we can simplify this process into one that will lower the risk of investing and automate some of the decision process. With Wall Street already incorporating machine learning for predicting stock prices, there is no question that this concept will become a recurring topic in conversations regarding stock exchange prediction methodologies [2] in the future. In this project we hope to implement a simple and straightforward model that can compare and, hopefully, compete with the current standards in order to get ahead of the curve. For our model we will be focusing on predicting General Motor (GM)'s stock prices, using gas and oil prices, inflation rates, and the overall state of health of the economy as primary determinants of what the price will be 28 days after purchasing it. The profits will be calculated using equation (3)*:

Where n is a unit day and *ClosedPrice* is the price of a stock at closing time. The time the stock market opens and closes is 6:30am – 1:00pm (PST). The purpose of this project will be to determine whether buying a stock generated a profit or not, and to see how much profit can be generated.

The use of Artificial Intelligence and Machine Learning has become more prominent in the world for almost any sort of statistics-based predicting models (the projected market increasing 800% from 2016 to 2022 [2]), and we believe that it will only continue to have an impact in many fields, including the stock exchange.

II. Data Preparation & Background on General Motors

A. Data Origins

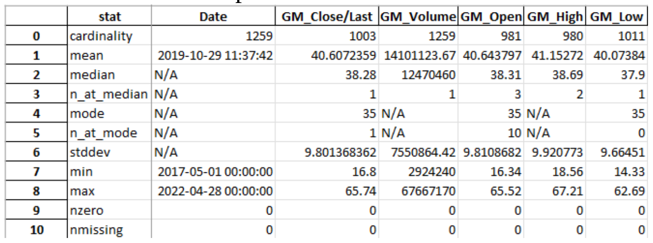
For this experiment, we will only be looking at the prices of General Motors (GM). Founded in 1908, General Motors is known today for being the greatest American auto manufacturer. General Motors has a volatile history of coming in and out of the market, changing ownership, and held an existence that has withstood the test of time. As of 2010, GM has remained public but for constraint purposes, for this experiment we are only concerned for April 2017 and to the present.

B. Data Quality Reports

Using the raw data provided, the following quality reports (**Table 1**) was generated pre-data preparation and will serve as a comparison for data modifications and validation for the classifiers. It is clear in the quality report shown below that there is data missing (N/A) that we will have to repair.

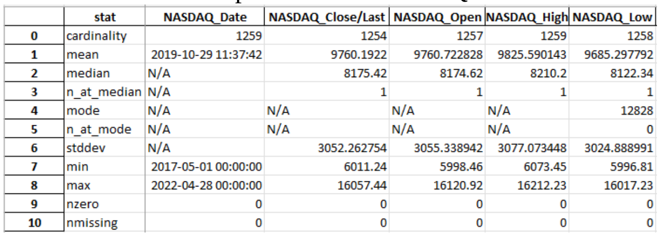
General Motors (GM)

Table 1.1 The data report on the General Motors data set.



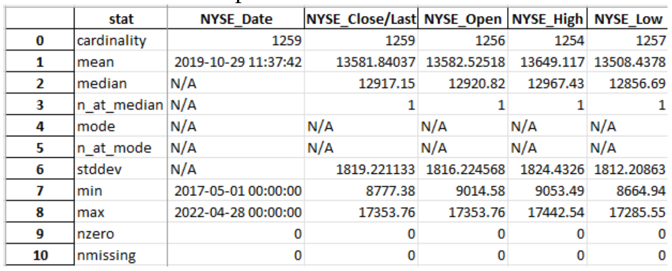
	stat	Date	GM_Close/Last	GM_Volume	GM_Open	GM_High	GM_Low
0	cardinality	1259	1003	1259	981	980	1011
1	mean	2019-10-29 11:37:42	40.6072359	14101123.67	40.643797	41.15272	40.07384
2	median	N/A	38.28	12470460	38.31	38.69	37.9
3	n_at_median	N/A	1	1	3	2	1
4	mode	N/A	35	N/A	35	N/A	35
5	n_at_mode	N/A	1	N/A	10	N/A	0
6	stddev	N/A	9.801368362	7550864.42	9.8108682	9.920773	9.66451
7	min	2017-05-01 00:00:00	16.8	2924240	16.34	18.56	14.33
8	max	2022-04-28 00:00:00	65.74	67667170	65.52	67.21	62.69
9	nzero	0	0	0	0	0	0
10	nmissing	0	0	0	0	0	0

Table 1.2 The data report on the NASDAQ health data set.



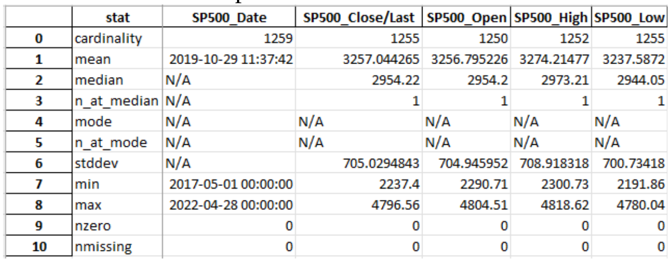
	stat	NASDAQ_Date	NASDAQ_Close/Last	NASDAQ_Open	NASDAQ_High	NASDAQ_Low
0	cardinality	1259	1254	1257	1259	1258
1	mean	2019-10-29 11:37:42	9760.1922	9760.722828	9825.590143	9685.297792
2	median	N/A	8175.42	8174.62	8210.2	8122.34
3	n_at_median	N/A	1	1	1	1
4	mode	N/A	N/A	N/A	N/A	12828
5	n_at_mode	N/A	N/A	N/A	N/A	0
6	stddev	N/A	3052.262754	3055.338942	3077.073448	3024.888991
7	min	2017-05-01 00:00:00	6011.24	5998.46	6073.45	5996.81
8	max	2022-04-28 00:00:00	16057.44	16120.92	16212.23	16017.23
9	nzero	0	0	0	0	0
10	nmissing	0	0	0	0	0

Table 1.3 The data report on the NYSE health data set.



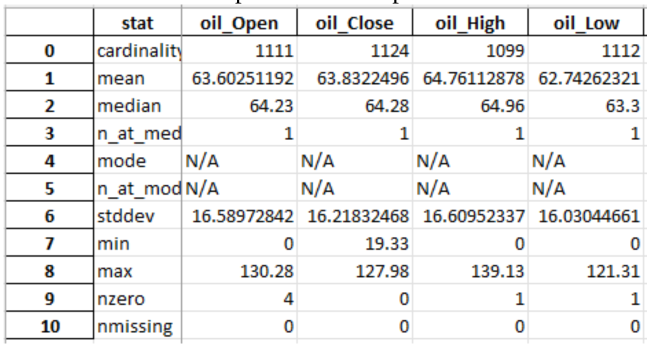
	stat	NYSE_Date	NYSE_Close/Last	NYSE_Open	NYSE_High	NYSE_Low
0	cardinality	1259	1259	1256	1254	1257
1	mean	2019-10-29 11:37:42	13581.84037	13582.52518	13649.117	13508.4378
2	median	N/A	12917.15	12920.82	12967.43	12856.69
3	n_at_median	N/A	1	1	1	1
4	mode	N/A	N/A	N/A	N/A	N/A
5	n_at_mode	N/A	N/A	N/A	N/A	N/A
6	stddev	N/A	1819.221133	1816.224568	1824.4326	1812.20863
7	min	2017-05-01 00:00:00	8777.38	9014.58	9053.49	8664.94
8	max	2022-04-28 00:00:00	17353.76	17353.76	17442.54	17285.55
9	nzero	0	0	0	0	0
10	nmissing	0	0	0	0	0

Table 1.4 The data report on the SP500 health data set.



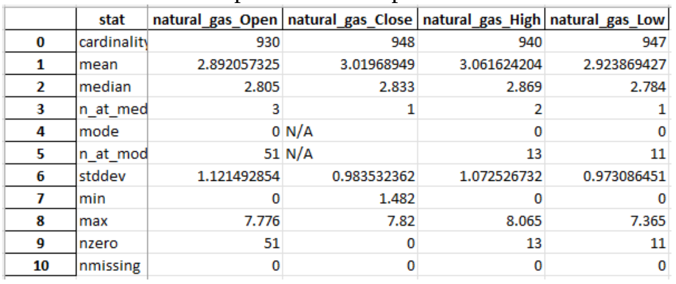
	stat	SP500_Date	SP500_Close/Last	SP500_Open	SP500_High	SP500_Low
0	cardinality	1259	1255	1250	1252	1255
1	mean	2019-10-29 11:37:42	3257.044265	3256.795226	3274.21477	3237.5872
2	median	N/A	2954.22	2954.2	2973.21	2944.05
3	n_at_median	N/A	1	1	1	1
4	mode	N/A	N/A	N/A	N/A	N/A
5	n_at_mode	N/A	N/A	N/A	N/A	N/A
6	stddev	N/A	705.0294843	704.945952	708.918318	700.73418
7	min	2017-05-01 00:00:00	2237.4	2290.71	2300.73	2191.86
8	max	2022-04-28 00:00:00	4796.56	4804.51	4818.62	4780.04
9	nzero	0	0	0	0	0
10	nmissing	0	0	0	0	0

Table 1.5 The data report on the oil price data set.



	stat	oil_Open	oil_Close	oil_High	oil_Low
0	cardinality	1111	1124	1099	1112
1	mean	63.60251192	63.8322496	64.76112878	62.74262321
2	median	64.23	64.28	64.96	63.3
3	n_at_med	1	1	1	1
4	mode	N/A	N/A	N/A	N/A
5	n_at_mod	N/A	N/A	N/A	N/A
6	stddev	16.58972842	16.21832468	16.60952337	16.03044661
7	min	0	19.33	0	0
8	max	130.28	127.98	139.13	121.31
9	nzero	4	0	1	1
10	nmissing	0	0	0	0

Table 1.6 The data report on the oil price data set.



	stat	natural_gas_Open	natural_gas_Close	natural_gas_High	natural_gas_Low
0	cardinality	930	948	940	947
1	mean	2.892057325	3.01968949	3.061624204	2.923869427
2	median	2.805	2.833	2.869	2.784
3	n_at_med	3	1	2	1
4	mode	0	N/A	0	0
5	n_at_mod	51	N/A	13	11
6	stddev	1.121492854	0.983532362	1.072526732	0.973086451
7	min	0	1.482	0	0
8	max	7.776	7.82	8.065	7.365
9	nzero	51	0	13	11
10	nmissing	0	0	0	0

C. Analysis of Data Preparation

The data did not contain a lot of missing values, which should be the case as the stock exchange is expected to report all prices. In order to replace the missing values we saw in the dataset, we took the mean of each feature and replaced the missing variables with it so as not to add any bias. Once done with that, we ran another quality report to check over the data again.

We noted from our datasets that within the past 5 years, the General Motors stock had a low of $14.33, a high of $67.52, and an average of around $40.63 (**Table 1.1)**. The NASDAQ index had a low of $5996.81, a high of $16212.23, and an average of around $9760.20 (**Table 1.2),** the NYSE index had a low of $8664.94, a high of $17442.54, and an average of around $13581.84 (**Table 1.3)**., the S&P 500 index had a low of $2191.86, a high of $4818.62, and an average of around $3257.04 (**Table 1.4), t**he oil price index had a low of $0.0, a high of $139.13, and an average of around $63.83 (**Table 1.5)**, and, finally, the natural gas price index had a low of $0.0, a high of $8.06, and an average of around $3.01 (**Table 1.6)**.

D. Data Sources

Table 2.1 Links from which data was retrieved

General Motors	https://www.nasdaq.com/market-activity/stocks/gm
NASDAQ	https://www.nasdaq.com/market-activity/index/comp
NYSE	https://www.nasdaq.com/market-activity/index/nya
SP500	https://www.nasdaq.com/market-activity/futures/sp
Oil Index	https://markets.businessinsider.com/commodities/oil-price?type=brent
Gas Index	https://markets.businessinsider.com/commodities/natural-gas-price

IV. Setup and Prediction Models Results

A. Setup

Before we can begin training and testing our models, the data must be separated into feature and target data. To enhance the performance and prediction of our models, several days' worth of data was combined into each row (i.e., GM Stock Price in 1 Day, NASDAQ Stock Price in 1 Day, …), this was done for 3 days on each row of data for all features. Our target data is the price of GM stock in 28 days, because our data ends on April 28th, 2022, there is no future stock price in 28 days to compare to, taking this into account, our data was cut ton March 31st,

2022. Lastly, our feature data was normalized to [0,1] and both feature and testing data were split into 70% training and 30% for testing. All of the resulting models' performances and income generated are shown in **Table 4.3.**

B. Linear Regression

In this project, a Linear Regression model was created using the default parameters from sci-kit learn. This model attempts to draw a correlation between our target and feature variables in a linear way. Our training data was given to the model to predict the future GM stock price and it resulted in the worst performing model as seen in **Table 4.3.** This model scored an R^2 score of 83.78% and a Mean Squared Error (MSE) score of 14.0828, but there was a model that had a higher score than this model. Ultimately, we would not recommend using a Linear Regression as there were better models suitable for the stock exchange. The reason for its poor performance is because stock prices are almost never linear so this makes it difficult for this model to predict.

C. Logistic Regression

Similar to the Linear Regression, the model determines the probability of a discrete outcome given an input variable in a linear fashion. The most practical applications and common uses for these models generate a binary outcome. In order to make predictions with this model, a binary target had to be made. The binary target was a 1 if the stock price in 28 days was greater than the current price and a 0 if it wasn't. When evaluating a Logistic Regression model, an R-squared statistic does not exist. In place of this, we used the sci-kit learn built-in "score" feature for this model. Using our test data, our accuracy resulted in 83.33% and an MSE score of 0.1667 which is great for an MSE score. However, this was not our best performing model, this is probably because it has the lowest scoring accuracy.

D. Multi-Layer Perceptron (MLP) Regressor

An MLP Regressor is a Neural Network that has multiple hidden layers each with multiple neurons. What is unique to this neural network compared to others is that it trains iteratively at each time step the partial derivatives of the loss function with respect to the model parameters are computed to update the parameters [4]. Our model was trained with a variety in 2 different parameters, number of neurons in each hidden layer and activation type. For hidden layers, we chose to go with 3 hidden layers and all possible combinations of 10, 50, 100, 150, and 200 neurons. This was done to save training time while also making the most comprehensive model before venturing into deep learning. For activation types, we cycled through relu, logistic, identity, and tanh. Each possible model was trained, tested, and their respective results were saved into **Table 4.1** as the Top 10 Best Models and **Table 4.2** as the Top 10 Worst Models. From these two tables, it can be seen that tanh activation type produces most of the best performing models and logistic produces all of the worst. Our best performing model in this regressor is tanh with a hidden layer structure of (200,150,10). This model resulted in our highest accuracy model, but not by much. It's mean squared error is still decently larger than others. In the next section, we will discuss why this isn't our best performing model and why Extra Tress performs better.

Table 4.1: MLP Regressor Best Test Accuracy

MLP Regressor Best Test Accuracy					
	Hidden Layer	Activation	Mean Squared Error	Train Accuracy	Test Accuracy
490	(200, 150, 10)	tanh	6.677244899	0.95129477	0.923089498
485	(200, 100, 10)	tanh	6.987313824	0.958198482	0.919518031
496	(200, 200, 50)	tanh	7.092575256	0.940964637	0.918305599
491	(200, 150, 50)	tanh	7.274859278	0.944430899	0.916205997
495	(200, 200, 10)	tanh	7.577275113	0.945440966	0.912722681
480	(200, 50, 10)	tanh	7.692724937	0.945545666	0.911392895
73	(100, 200, 150)	relu	7.766359401	0.918195926	0.910544752
486	(200, 100, 50)	tanh	7.772872661	0.925756581	0.91046973
92	(150, 150, 100)	relu	7.916654226	0.91823613	0.908813611
455	(150, 50, 10)	tanh	8.001045356	0.939895525	0.907841569

Table 4.2: MLP Regressor Worst Test Accuracy

MLP Regressor Worst Test Accuracy					
	Hidden Layer	Activation	Mean Squared Error	Train Accuracy	Test Accuracy
142	(10, 150, 100)	logistic	86.8454688	1.36E-05	-3.12E-04
181	(100, 50, 50)	logistic	86.8412184	2.09E-05	-2.63E-04
244	(200, 150, 200)	logistic	86.8410050	2.75E-05	-2.61E-04
217	(150, 150, 100)	logistic	86.8404409	1.55E-05	-2.54E-04
242	(200, 150, 100)	logistic	86.8341544	2.45E-05	-1.82E-04
222	(150, 200, 100)	logistic	86.8329157	7.35E-05	-1.67E-04
238	(200, 100, 150)	logistic	86.8306338	5.14E-05	-1.41E-04
214	(150, 100, 200)	logistic	86.8299463	1.28E-04	-1.33E-04
212	(150, 100, 100)	logistic	86.8296635	4.12E-05	-1.30E-04
219	(150, 150, 200)	logistic	86.8286240	1.39E-04	-1.18E-04

E. Ensemble Models

The Ensemble model takes advantage of using different models in parallel. We tested 3 different Ensemble Model types: Decision Tree Regressor, Random Forest Regressor, and Extra Tree Regressor; These models scored 85.72%, 94.77%, and 94.79% respectively. Out of these three models, Extra Tree Regressor had the best performance compared to every model presented.

Table 4.3: Model Results

Model Results			
Model	**Accuracy (R2)**	**MSE**	**Total Income**
Linear Regression	83.78%	14.0828	$ 43,785.70
Logistic Regression	83.33%	0.166667	$ 53,637.77
MLP Regressor	95.01%	6.677244	$ 53,254.66
Decision Tree Regressor	85.72%	12.39338	$ 55,535.93
Random Forest Regressor	93.77%	5.404949	$ 56,594.26
Extra Trees Regressor	94.79%	4.520924	$ 57,482.95

General Motors (GM)

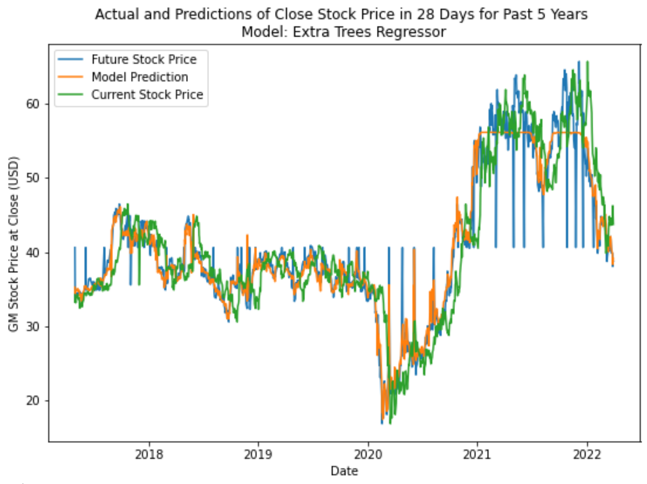


Figure 4.1

Presented in **Figure 4.1** are the actual and predicted prices of the closing stock market price for the past 5 years. The blue represents the actual future stock price, the orange is the prediction from the model, and the green is the current stock price. It can be seen that the predicted price is close to the actual. This graph shows how well our model predictions match up with the actual data for the future stock prices, and only on rare occasions does our model appear to not invest on days it could have profited. This can be seen in the trends in late 2021 where we see some higher spikes in the actual data but our predictor stays neutral.

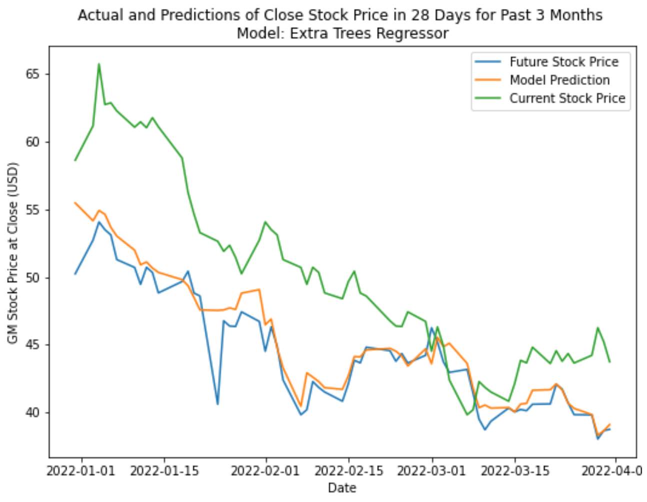


Figure 4.2

Presented in **Figure 4.2** are the actual and predictions of the closing stock market price for the past 3 months. The blue represents the actual future stock price, the orange is the prediction from the model, and the green is the current stock price. Based on this graph, the stock market has an overall downward trend and our model predicts that as well. Using our model on any given day, it can be seen that the current stock price is predicted to be lower, so there was little to no investing during this period. This leads to a better overall income because any time the model is wrong then that is when we lose money.

V. Equations

A. Mean Square Error

The Mean Square Error (MSE) (2) equation is an equation used in linear regression models in which the smaller the rate, the better.

$$\text{MSE} = \frac{1}{|TestSet|}\sum_{1}^{|TestSet|}(RETURNEDVALUE - MODELOUTPUT)^2 \quad (1)$$

B. Income Calculation Formula

The income calculation formula is supposed to be used on the closed price for the day invested.

$$\text{Income} = \frac{\$1000}{ClosedPrice_n} * ClosedPrice_{n+28} - \$1000 \ (2)$$

VI. Discussion and Conclusion

In conclusion, when reviewing all our outputs, we were able to select one learning model that performed higher than the others.

The model we identified as the best-fit for our data was the extra trees regressor which we ran in a parallel ensemble with two other decision tree based regressors. This model’s R2 score was the second highest but had a lower MSE compared to the model with the leading R2. We felt that because the MSE was significantly lower, while the change in R2 wasn’t that great, we would rather work with a model that had a lower loss score since we’d rather have less risk when investing. This is what made the extra trees model stand out from the rest. Using the extra trees regressor we were able to predict the closing stock price with a 94.8% correct prediction and an MSE of 4.5, which is shown in Table 5.3. This model also turns a higher profit than the other models, $57482.95, which we feel constitutes it as a good model since it achieves our desired goal- which was to predict and profit off the stock market.

One thing that we would do to improve this model is add more features such as more commodities, more importantly, if we took some time to analyze driving factors and commodities more correlated with the stock that we were assigned; for example, perhaps the price of steel has a direct impact on a company like General Motors, that heavily relies on this commodity as well as other metals (i.e., copper, aluminum). Another factor that we noticed was that adding more resolution to our step sizes in models that had a setting for more recursion, would usually produce a more optimized solution, However, higher resolution requires more time spent in processing, or a machine with higher processing speeds. Some settings we were hoping to use required either faster multithreading methodologies along with high-end computing processors or spending hundreds of hours waiting for a result. Had we started our project sooner, perhaps we would have had more time to run more robust models.

Overall, given the high profit, high R2, and low(er) MSE (in comparison to the other models), we would continue to use the extra trees regressor model if we intended to further our research. However, any of these models, other than Linear Regression and Logistic Regression, are more than acceptable for the purpose of predicting future stock prices. It's clear that this application of machine learning is very promising and could be very useful to the everyday investor, or to someone who is interested in beginning.

VII. References

[1] JeffCoxCNBCcom, "Investors have put more money into stocks in the last 5 months than the previous 12 years combined," *CNBC*, 09-Apr-2021. [Online]. Available: https://www.cnbc.com/2021/04/09/investors-have-put-more-money-into-stocks-in-the-last-5-months-than-the-previous-12-years-combined.html. [Accessed: 03-May-2022].

[2] "Commodity prices | commodity market | markets insider," *Business Insider*. [Online]. Available: https://markets.businessinsider.com/commodities. [Accessed: 03-May-2022].

[3] "Futures," *Nasdaq*. [Online]. Available: https://www.nasdaq.com/market-activity/futures/. [Accessed: 03-May-2022].

[4] "Sklearn.neural_network.Mlpregressor," *scikit*. [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPRegressor.html. [Accessed: 03-May-2022].

VIII. Appendix

project2.py

```
# -*- coding: utf-8 -*-
"""
Created on Fri Apr 29 15:24:34 2022

@author: cfrut, agarc, kmova
"""

import pandas as pd
import os
from tqdm import tqdm
import numpy as np

# Setup
# Create Full Path - This is the OS agnostic way of doing so
dir_name = os.getcwd()
filename = 'GM_5Y_HistoricalData.csv'
full_path = os.path.join(dir_name, filename)

#
# Create the Main Data Frame
#
df_main = pd.read_csv(full_path) # read Excel spreadsheet
df_main.name = 'df_main' # name it
df_main['Date'] = pd.to_datetime(df_main['Date']) # Convert to Datetime


# - Create Data Frames for the other Features -
# COMP (NASDAQ) Index
NASDAQ_df          =               pd.read_csv(os.path.join(dir_name,
'COMP_5Y_HistoricalData.csv'))
NASDAQ_df['Date']  =  pd.to_datetime(NASDAQ_df['Date'])  #  Convert  to
Datetime
NASDAQ_df = NASDAQ_df.drop(columns = ['Volume']) # Drop Volume
NASDAQ_df.name = "NASDAQ" # name it

# NYSE Index
NYSE_df = pd.read_csv(os.path.join(dir_name, 'NYA_5Y_HistoricalData.csv'))
NYSE_df['Date'] = pd.to_datetime(NYSE_df['Date'])
NYSE_df = NYSE_df.drop(columns = ['Volume'])
NYSE_df.name = 'NYSE'

#S&P 500
SP500_df               =               pd.read_csv(os.path.join(dir_name,
'SPX_5Y_HistoricalData.csv'))
SP500_df['Date'] = pd.to_datetime(SP500_df['Date'])
SP500_df = SP500_df.drop(columns = ['Volume'])
SP500_df.name = 'SP500'

# Oil
oil_df = pd.read_csv(os.path.join(dir_name, 'Oil_04_28_22-05_01_17.csv'))
oil_df['Date'] = pd.to_datetime(oil_df['Date'])
oil_df = oil_df.drop(columns = ['Volume'])
oil_df.name = 'oil'

# Natural Gas
natural_gas_df = pd.read_csv(os.path.join(dir_name, 'Natural_Gas_04_28_22-
05_01_17.csv'))
natural_gas_df['Date'] = pd.to_datetime(natural_gas_df['Date'])
natural_gas_df = natural_gas_df.drop(columns = ['Volume'])
natural_gas_df.name = 'natural_gas'

# Commodities
commodity_df  =  pd.read_excel(os.path.join(dir_name,  'commodity-price-
index-60.xlsx'))
commodity_df.name = 'commodity'
commodity_df['Month'] = pd.to_datetime(commodity_df['Month'])

# Inflation
inflation_df           =           pd.read_excel(os.path.join(dir_name,
'Inflation_rate_per_year.xlsx'))
inflation_df  =  inflation_df[inflation_df['Country  Name']  ==  'United
States']
inflation_df = inflation_df.reset_index(drop = True)
inflation_df = inflation_df.drop(columns = ['Country Name','Country Code',
'Indicator Name', 'Indicator Code'])
inflation_df.name = 'inflation'


# Combine Stock Data to df_main
item_list = [NASDAQ_df, NYSE_df, SP500_df, oil_df, natural_gas_df]

for df in tqdm(item_list):
    for idx in df_main.index:
        this_series = df[df['Date'] == df_main['Date'][idx]]
        for this_column in this_series.columns:
            try:
                df_main.loc[idx,str(df.name)+'_'+str(this_column)]      =
this_series[this_column].values[0]
            except:
                df_main.loc[idx,str(df.name)+'_'+str(this_column)]      =
np.NAN

# Drop Dates from Features

for item in item_list:
    df_main = df_main.drop(columns = [item.name+'_Date'])


# Generating a Report for RAW
import sys
stats_path = os.path.join(dir_name,'..', 'Homework_2')
sys.path.append(stats_path)
from stats_report import StatsReport

labels = df_main.columns
report = StatsReport()

# Create a simple data set summary for the console
for thisLabel in tqdm(labels): # for each column, report stats
    thisCol = df_main[thisLabel]
    report.addCol(thisLabel, thisCol)
```

```
   #print(report.to_string())
   report.statsdf.to_excel("Quality_Report_Before_Prep.xlsx")


   # Replace all Missing Values then Make Report - Should have no missing
values
   labels = df_main.columns
   report = StatsReport()

   for this_label in labels:
       df_main[this_label].fillna(df_main[this_label].mean(), inplace = True)

   # Create a simple data set summary for the console
   for thisLabel in tqdm(labels): # for each column, report stats
       thisCol = df_main[thisLabel]
       report.addCol(thisLabel, thisCol)

   report.statsdf.to_excel("Quality_Report.xlsx")

   # Add Several Day Data

   labels = df_main.columns
   for this_label in labels:
       print(f'\nUsing Label: {this_label}')
       for idx in tqdm(df_main.index):
           if idx < 3 or idx > len(df_main):
               pass
           else:
               df_main.loc[idx, this_label + '_in_1_day'] = df_main.loc[idx-
1][this_label]
               df_main.loc[idx, this_label + '_in_2_days'] = df_main.loc[idx-
2][this_label]
               df_main.loc[idx, this_label + '_in_3_days'] = df_main.loc[idx-
3][this_label]
   # Add Commodities and Inflation

   for c_idx in commodity_df.index:
       for df_idx in tqdm(df_main.index):
           if           commodity_df['Month'][c_idx].month            ==
df_main['Date'][df_idx].month:
               #print(f'\nAdding Here: c_idx = {c_idx}, df_idx = {df_idx}')
               df_main.loc[df_idx,str(commodity_df.name)+'_'+str('Price')] =
commodity_df['Price'][c_idx]
               df_main.loc[df_idx,str(commodity_df.name)+'_'+str('Change')]
= commodity_df['Change'][c_idx]
           else:
               #print("Didn't Work")
               pass

   for year in inflation_df.columns:
       for df_idx in tqdm(df_main.index):
           if year == df_main['Date'][df_idx].year:
               print('\nworked')
               df_main.loc[df_idx,'Inflation_Rate'] = inflation_df[year][0]

   # Replace Inflation Rate of 2022 with Avg
   df_main['Inflation_Rate'].fillna(df_main['Inflation_Rate'].mean(),
inplace = True)

   # Create Target Value - Stock Price 28 days later
   import datetime

   start = df_main['Date'][0] - datetime.timedelta(days = 28)
   idx = df_main[df_main['Date'] == start].index[0]

   for this_idx in tqdm(range(idx,len(df_main))):
       future_date   =   df_main['Date']   ==   df_main['Date'][this_idx]   +
datetime.timedelta(days = 28)
       if future_date.sum() == 0:
           df_main.loc[this_idx,       'GM_Close/Last_in_28_Days']       =
df_main['GM_Close/Last'].mean()
       else:
           future_idx = df_main[future_date].index[0]
           df_main.loc[this_idx,       'GM_Close/Last_in_28_Days']       =
df_main['GM_Close/Last'][future_idx]

   df_main = df_main.drop(index = range(0,idx)).reset_index(drop = True)

   #%%
   # Assuming I buy GM stock every time model says yes to profit in 28 days
   # Assuming over 3 Months period
   def calculate_income(clf, X, df):
       predY = clf.predict(X)
       total_income = 0
       for   current_price,   pred_future_price,   actual_future_price   in
zip(df['GM_Close/Last'], predY, df['GM_Close/Last_in_28_Days'] ):
           if current_price < pred_future_price:
               income = (1000/current_price)*actual_future_price - 1000
           else:
               income = 0
           total_income = total_income+income
       return total_income
```

```
   #%% Imports
   from sklearn.linear_model import LinearRegression
   from sklearn.metrics import mean_squared_error
   from sklearn.preprocessing import MinMaxScaler
   from sklearn.linear_model import LogisticRegression
   from itertools import product
   from sklearn.neural_network import MLPClassifier
   from sklearn import metrics
   from sklearn.model_selection import train_test_split
   from sklearn.preprocessing import PolynomialFeatures

   #%% Target Data
   # Setting up Training Data
   # Data
   predictors = df_main.drop(columns = ['GM_Close/Last_in_28_Days']).columns
   X = df_main[predictors].to_numpy(np.float64)
   min_max_scaler = MinMaxScaler()
   X_norm = min_max_scaler.fit_transform(X)

   Y = df_main['GM_Close/Last_in_28_Days'].to_numpy(np.float64)

   # Split Testing and Training Data
   X_train,   X_test,   y_train,   y_test   =   train_test_split(X_norm,   Y,
test_size=0.3,
                                                  train_size=0.7,
random_state=22222,
                                                  shuffle=True,
stratify=None)

   #%%
   # Linear Regression
   linreg_model = LinearRegression().fit(X_train, y_train)
   y_pred = linreg_model.predict(X_test)
   print(f'Linear     Regression     Score:     {linreg_model.score(X_test,
y_test):.4f}')
   print(f'Mean    squared    error    (MSE):    {mean_squared_error(y_test,
y_pred):.4f}')
   #print(f"Accuracy: {metrics.accuracy_score(y_test, y_pred)}")

   print(f"Income: {calculate_income(linreg_model, X_norm, df_main)}")

   #%% Create Binary Target DataFrame
   df_bin_target = df_main.copy()
   for idx in df_bin_target.index:
       if  df_bin_target.loc[idx,  'GM_Close/Last']  <  df_bin_target.loc[idx,
'GM_Close/Last_in_28_Days']:
           df_bin_target.loc[idx, 'Profit'] = True
       else:
           df_bin_target.loc[idx, 'Profit'] = False

   # Setting up Training Data
   predictors = df_bin_target.drop(columns = ['Profit']).columns
   X_bin = df_bin_target[predictors].to_numpy(np.float64)
   min_max_scaler = MinMaxScaler()
   X_bin_norm = min_max_scaler.fit_transform(X_bin)

   Y_bin = df_bin_target['Profit'].to_numpy(np.float64)

   # Split Testing and Training Data
   X_bin_train,      X_bin_test,      y_bin_train,      y_bin_test      =
train_test_split(X_bin_norm, Y_bin, test_size=0.3,

train_size=0.7, random_state=22222,

shuffle=True, stratify=None)
   #%% Logistic Regression
   modelLog = LogisticRegression()
   clf_Log = modelLog.fit(X_bin_train, y_bin_train)
   Ypred = clf_Log.predict(X_bin_test)
   Ypredclass = 1*(Ypred > 0.5)
   print("Logistic Regression \nR2 = %f,  MSE = %f,  Classification Accuracy
=        %f"        %        (metrics.r2_score(y_bin_test,        Ypred),
metrics.mean_squared_error(y_bin_test,                                Ypred),
metrics.accuracy_score(y_bin_test, Ypredclass)))

   Ypred = clf_Log.predict(X_bin_norm)
   total_income = 0
   for       current_price,       pred,       actual_future_price       in
zip(df_main['GM_Close/Last'], Ypred, df_main['GM_Close/Last_in_28_Days'] ):
       if pred == 1:
           income = (1000/current_price)*actual_future_price - 1000
       else:
           income = 0
       total_income = total_income+income

   print(f'Income: {total_income}')
   #%% - MIGHT NOT NEED - DELETE LATER
   '''
   # Poly
   poly  =  PolynomialFeatures(2)  #  object  to  generate  polynomial  basis
functions
   #X1_train = df_main.drop([targetName], axis=1).to_numpy()
   bigTrainX = poly.fit_transform(X1_train)
```

```
    mlrf = LogisticRegression() # creates the regressor object
    mlrf.fit(bigTrainX, y_train)
    Ypred1 = mlrf.predict(bigTrainX)
    Ypredclass1 = 1*(Ypred1 > 0.5)
    print("R2   =   %f,   MSE   =   %f,   Classification   Accuracy   =   %f"  %
(metrics.r2_score(y1_test,    Ypred1),    metrics.mean_squared_error(y1_test,
Ypred1), metrics.accuracy_score(y1_test, Ypredclass1)))
    print("W: ", np.append(np.array(mlrf.intercept_), mlrf.coef_))
    '''
    #%% - Neural Network
    from sklearn.neural_network import MLPRegressor

    from datetime import datetime
    t1 = datetime.now()
    resultsDF = pd.DataFrame(data=None, columns=['Hidden Layer', 'Activation',
'Mean Squared Error', 'Train Accuracy', 'Test Accuracy'])
    #firstHL = list(range(150,200))
    #secondHL = list(product(range(100,200),repeat=2))

    neurons = [10,50,100,150,200]
    thirdHL = list(product(neurons,repeat=3))
    hiddenLayers = thirdHL

    activations = ('relu', 'logistic', 'identity', 'tanh')
    # Hidden Layers
    for act in activations:
        for hl in hiddenLayers:
            #hl = i
            #currActivate = k
            #regpenalty = 0.0003 #according to forums on the internet, this is
an optimal adam solver learning rate
            clf = MLPRegressor(hidden_layer_sizes=(hl)
                                    , activation=act
                                    , solver='adam'
                                    , alpha=0.04
                                    , max_iter=10000
                                    ,
validation_fraction=0.42).fit(X_train,y_train)
            annPredY = clf.predict(X_test)

            # Get  Scores
            train_accuracy = clf.score(X_train, y_train)
            test_accuracy = clf.score(X_test, y_test)
            mse = metrics.mean_squared_error(y_test, annPredY)

            print("\n ###### MPL Classifier #######")
            print(f"\n Activation Type: {act}")
            #print(f"\nLearning Rate: {learning_rate}")
            print(f"\nHidden Layers: {hl}")
            print("\n\rANN: MSE = %f" % mse)
            print(f"\nTrain Accuracy = {train_accuracy}")
            print(f"\nTest Accuracy = {test_accuracy}")

            resultsDF = resultsDF.append({'Hidden Layer': hl,
                                          'Activation': act,
                                          'Mean Squared Error': mse,
                                          'Train Accuracy': train_accuracy,
                                          'Test      Accuracy':test_accuracy},
ignore_index=True)

    t2 = datetime.now()
    total_time = t2-t1
    print("Started: ", t1)
    print("Ended: ", t2)
    print("Total Time for ANN: ", t2-t1)


    #%% - Save ANN DF
    best_test = resultsDF['Test Accuracy'].nlargest(10)
    #best_misslabeled = resultsDF['misclass'].nsmallest(10)
    worst_test = resultsDF['Test Accuracy'].nsmallest(10)
    #worst_mislabeled = resultsDF['misclass'].nlargest(10)

    best_test_df = pd.DataFrame([resultsDF.loc[best_test.index[idx]] for idx
in range(len(best_test.index))])
    #best_misslabeled_df                                                       =
pd.DataFrame([resultsDF.loc[best_misslabeled.index[idx]]    for    idx    in
range(len(best_misslabeled.index))])
    worst_test_df = pd.DataFrame([resultsDF.loc[worst_test.index[idx]] for idx
in range(len(worst_test.index))])
    #worst_mislabeled_df                                                       =
pd.DataFrame([resultsDF.loc[worst_mislabeled.index[idx]]    for    idx    in
range(len(worst_mislabeled.index))])

    best_test_df.to_excel('Best_Test_Accuracy.xlsx')
    #best_misslabeled_df.to_excel('Best_Mislabeled_Models.xlsx')
    worst_test_df.to_excel('Worst_Test_Accuracy.xlsx')
    #worst_mislabeled_df.to_excel('Worst_Mislabeled_Models.xlsx')


    #%%
    # Ensemble (TY SCIKIT LEARN DOCUMENTATION!)
    # Random Forest Bc That Sounds The Most Useful (Did I Profit?)
    from sklearn.model_selection import cross_val_score
    from sklearn.datasets import make_blobs
    from sklearn.ensemble import RandomForestRegressor
    from sklearn.ensemble import ExtraTreesRegressor
    from sklearn.tree import DecisionTreeRegressor

    X,       y       =       make_blobs(n_samples=10000,       n_features=10,
centers=100,random_state=0)

    clf                =                   DecisionTreeRegressor(max_depth=None,
min_samples_split=2,random_state=0).fit(X_train, y_train)
    predY = clf.predict(X_test)
    scores = cross_val_score(clf, X_test, y_test, cv=5)
    print(scores.mean())
    print(f'Decision Tree \nR2 Score: {metrics.r2_score(y_test, predY)}')
    print(f'MSE: {metrics.mean_squared_error(y_test, predY)}')
    print(f'Income: {calculate_income(clf, X_norm, df_main)}')

    clf                =                 RandomForestRegressor(n_estimators=10,
max_depth=None,min_samples_split=2, random_state=0).fit(X_train, y_train)
    predY = clf.predict(X_test)
    scores = cross_val_score(clf, X_test, y_test, cv=5)
    print(scores.mean())
    print(f'Random Forest \nR2 Score: {metrics.r2_score(y_test, predY)}')
    print(f'MSE: {metrics.mean_squared_error(y_test, predY)}')
    print(f'Income: {calculate_income(clf,X_norm, df_main)}')

    clf = ExtraTreesRegressor(n_estimators=10, max_depth=None,
                              min_samples_split=2,
random_state=0).fit(X_train, y_train)
    predY = clf.predict(X_test)
    scores = cross_val_score(clf, X_test, y_test, cv=5)
    print(scores.mean())
    print(f'Extra Tree \nR2 Score: {metrics.r2_score(y_test, predY)}')
    print(f'MSE: {metrics.mean_squared_error(y_test, predY)}')
    print(f'Income: {calculate_income(clf,X_norm, df_main)}')
    #%% Create ANN Best
    clf_best = MLPRegressor(hidden_layer_sizes=(200,150,10)
                            , activation='tanh'
                            , solver='adam'
                            , alpha=0.04
                            , max_iter=10000
                            , validation_fraction=0.42).fit(X_train,y_train)
    clf_best = clf_best.fit(X_train, y_train.ravel())
    annPredY = clf_best.predict(X_norm)

    clf_best.predict(X_test)
    print(f'R2      Score       for       ANN:       {metrics.r2_score(y_test,
clf_best.predict(X_test))}')
    print(f'Income: {calculate_income(clf_best,X_norm, df_main)}')
    #%%
    import matplotlib.pyplot as plt

    clf = ExtraTreesRegressor(n_estimators=10, max_depth=None,
                              min_samples_split=2,
random_state=0).fit(X_train, y_train)
    predY = clf.predict(X_norm)


    #%%
    # Plot 5 Years
    plt.figure()
    plt.plot(df_main['Date'].values, Y, label = 'Future Stock Price')
    plt.plot(df_main['Date'].values, annPredY,label = 'Model Prediction')
    plt.plot(df_main['Date'].values,   df_main['GM_Close/Last'],   label   =
'Current Stock Price')
    plt.xlabel('Date')
    plt.ylabel('GM Stock Price at Close (USD)')
    plt.title('Actual and Predictions of Close Stock Price in 28 Days for Past
5 Years \nModel: Extra Trees Regressor')
    plt.legend()
    plt.show()

    ## - 3 Months
    df_3months = df_main.drop(index = range(63, len(df_main)))
    #predictors           =           df_3months.drop(columns           =
['GM_Close/Last_in_28_Days']).columns
    #X_3months = df_3months[predictors].to_numpy(np.float64)
    #min_max_scaler = MinMaxScaler()
    #X_norm_3months = min_max_scaler.fit_transform(X_3months)
    #Y_3months = df_3months['GM_Close/Last_in_28_Days'].to_numpy(np.float64)
    #print(Y_3months)

    # Prediction
    Y_3months = Y[:63]
    annPredY_3 = clf_best.predict(X_norm[:63])
    print(annPredY_3)

    # Plot 3 Months
    plt.figure()
    plt.plot(df_3months['Date'].values,  Y_3months,  label  =  'Future  Stock
Price')
```

```
   plt.plot(df_3months['Date'].values,    annPredY_3,    label    =    'Model
Prediction')
   plt.plot(df_3months['Date'].values, df_main['GM_Close/Last'][:63], label =
'Current Stock Price')
   plt.xlabel('Date')
   plt.ylabel('GM Stock Price at Close (USD)')
   plt.title('Actual and Predictions of Close Stock Price in 28 Days for Past
3 Months \nModel: Extra Trees Regressor')
   plt.legend()
   plt.show()
```