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*](mailto:cfrutos@vt.edu)*,* [*agarcia1296@vt.edu*](mailto:agarcia1296@vt.edu)*,* [*kayleighm@vt.edu*](mailto:kayleighm@vt.edu)

***Abstract*—A novel approach to predicting General Motors’ stock prices after 28 days by experimenting with four machine learning models’ and determine which model proves best performance for this application.**

**The four models being linear regression, logistic regression model, neural-networks model, and an ensemble model are supervised learning models so they will train on a labeled dataset. The dataset was provided by the NASDAQ official website, and it is data 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. Data concerning other features with which the models will train on like prices of coal and gas prices, and inflation rates were pulled from Markets Insider’s website.**

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

# I. INTRODUCTION

B

eing 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 already or not already investing in the stock market. The stock market can often be confusing as there are a lot of factors that go into predicting an outcome of a stock not always based solely on a company’s performance. To name a few things: global pandemics, wars, and natural disasters can have a significant effect on the behavior of certain stock prices. Other factors to consider are pop-culture trends, rapidly developing technologies, and legislative bills. These factors can determine what people are often swayed to if not forced to purchase. With Wall Street already incorporating machine learning for predicting stock prices, there is no question that machine learning will become a recurring topic in conversations regarding stock exchange prediction methodologies [1]. In this project we hope to implement a simple and straightforward model that can compare and possibly compete with the current standards. We are using gas and oil prices, inflation rates, and the overall state of health of the economy as primary determinants of what the price of a stock will be 28 days after purchasing it. The profits will be calculated as equation (2)\*:

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

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

# 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 long history of coming in and out of the market, change in ownership, and held an existence that has withstood the test of time. As of 2010, GM has remained public. However, for this experiment, we are only concerned for April 2017 and on.

## B. Data Quality Reports

Using the raw data provided, the following quality report (Table 1) was generated pre-data preparation and will serve as a comparison for data modifications and validation for the classifiers.

Table, Excel

Description automatically generated

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

Table

Description automatically generated

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

Table

Description automatically generated

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

Table

Description automatically generated

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

Table

Description automatically generated

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

Table

Description automatically generated

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

*C. Analysis of Data Preparation*

For the most part, the data was very clean and there were no recorded missing values for any date. This is to be expected as the stock exchange is expected to report these prices. Within the last 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)**. The oil price index had a low of $0.0, a high of $139.13, and an average of around $63.83 (**Table 1.5)**. 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 Book-Keeping*

|  |  |
| --- | --- |
| General Motors |  |
| NASDAQ |  |
| NYSE |  |
| SP500 |  |
| Oil Index | https://markets.businessinsider.com/commodities/oil-price?type=brent |
| Gas Index | https://markets.businessinsider.com/commodities/natural-gas-price |

**Table 2.1** Links from which data was retrieved

# IV. Prediction Model Results

## A. Linear Regression

The linear regression model is the more straightforward model which attempts to draw a correlation between a dependent and independent variable. While the Linear Regression Model proved to have a good performance. With an score of 0.474 and a Mean Squared Error (MSE) score of .131, 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.

## B. Logistic Regression

Similar to the Linear Regression, the model determines the probability of a discrete outcome given an input variable. The most practical applications and common uses for these models generate a binary outcome. We used this model and surprisingly, we got worse results than Linear Regression: an score of 0.331, an MSE score of 0.167 and a Classification Accuracy Score of 0.833

## C. Ensemble Model

The Ensemble model takes advantage of using different models sequentially. For this ensemble, we used a decision tree , a random forest, and a extra tree model; these models produced classifier scores of 0.88, 0.79, and 0.83 respectively. these were much better than the last two models, but the ensemble model did not have a score as good as the artificial neural networks (ANN) model.

# V. Artificial Neural Networks Model

## A. Model Description For ANN

The ANN model was the model that we chose to use as it had an AUROC score of 0.91 and . The best run we used had hidden layer sizes of 9, 8 , and 8. The learning rate was set to 0.0401 and it used an activation type of tanh. Below are the 10 best settings which produced the highest AUROC and misclassification scores.





VI. 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.

(2)

## B. Income Calculation Formula

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

VII. analysis of working model results

1. *ANN Model Analysis*

A picture containing text

Description automatically generated

Text

Description automatically generated

Chart, histogram

Description automatically generated

Chart, line chart

Description automatically generated

VIII. Conclusion

References

1. S. Lazzaro, “Machine Learning's rise, applications, and challenges,” *VentureBeat*, 21-Jun-2021. [Online]. Available: <https://venturebeat.com/2021/06/21/machine-learnings-rise-applications-and-challenges/#:~:text=In%20the%20first%20quarter%20of,to%20%248.81%20billion%20by%202022>. [Accessed: 21-Mar-2022].
2. J. Bowler, “Is weather forecasting getting less accurate?,” *Cosmos*, 04-Feb-2022. [Online]. Available: <https://cosmosmagazine.com/earth/climate/weather-forecasting-less-accurate/>. [Accessed: 21-Mar-2022].
3. [NCEI.Monitoring.info@noaa.gov](mailto:NCEI.Monitoring.info@noaa.gov), “Climate at a glance,” *National Climatic Data Center*. [Online]. Available: <https://www.ncdc.noaa.gov/cag/city/time-series/USW00023066>. [Accessed: 21-Mar-2022].
4. *Index of /*. [Online]. Available: <https://www1.ncdc.noaa.gov/pub/data/ghcn/daily/readme.txt>. [Accessed: 22-Mar-2022].

IX. Appendix

1. ***proj1.py***

import pandas as pd

import os

from tqdm import tqdm

#%% Setup

# Create Full Path - This is the OS agnostic way of doing so

dir\_name = os.getcwd()

filename = 'USW00023066.csv'

full\_path = os.path.join(dir\_name, filename)

#

# Create the Main Data Frame

#

data\_headers = ['ID', 'DATE', 'ELEMENT', 'VALUE1', 'MFLAG1', 'Q\_FLAG1', 'SFLAG1', 'VALUE2']

df\_main = pd.read\_csv(full\_path, names = data\_headers) # read Excel spreadsheet

print('File {0} is of size {1}'.format(full\_path, df\_main.shape))

#%% Generating a Report for RAW

from utils\_project1 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")

#%%

def get\_unique\_column\_values(df):

"""

Identifying Unique Values of each Column in DF

Output is a Dictionary of each Column

"""

headers\_unique = {}

for label in tqdm(df.columns):

headers\_unique[label] = df[label].unique()

#pbar.close()

return headers\_unique

headers\_unique = get\_unique\_column\_values(df\_main)

print(f"List of Dates: {headers\_unique['DATE']}")

#%% Data Preperation - THIS TAKES SEVERAL MINUTES

def prep\_data(df, df\_out, headers\_unique):

"""

Extract Values for Elements and insert into df\_prep

"""

index\_ = 0

for date in tqdm(headers\_unique['DATE']):

date\_idx = df['DATE'] == date

df\_by\_date = df[date\_idx]

df\_out.loc[index\_, 'DATE'] = date

for idx in df\_by\_date['ELEMENT'].index:

df\_out.loc[index\_, df\_by\_date['ELEMENT'][idx]] = df\_by\_date['VALUE1'][idx]

index\_ = index\_+1

df\_prep = pd.DataFrame(columns = ['DATE', \*headers\_unique['ELEMENT']])

prep\_data(df\_main, df\_prep, headers\_unique)

#%% Create Target Columns

#

# Create Columns - PRECIPFLAG and PRECIPAMT

# Create Target Columns - NEXTDAYPRECIPFLAG and NEXTDAYPRECIPAMT

#

for idx in tqdm(df\_prep.index):

rain = df\_prep['PRCP'][idx] # in tenths of mm

snow = df\_prep['SNOW'][idx]

if (rain or snow) > 0:

df\_prep.loc[idx, 'PRECIPFLAG'] = 1 # It rained/snowed

df\_prep.loc[idx, 'PRECIPAMT'] = 0.0393701\*(rain/10) + (0.0393701\*snow)/8 # result is in inches

else:

df\_prep.loc[idx, 'PRECIPFLAG'] = 0 # It did not rain/snow

df\_prep.loc[idx, 'PRECIPAMT'] = 0

if idx > 0:

df\_prep.loc[idx-1, 'NEXTDAYPRECIPFLAG'] = df\_prep.loc[idx, 'PRECIPFLAG']

df\_prep.loc[idx-1, 'NEXTDAYPRECIPAMT'] = df\_prep.loc[idx, 'PRECIPAMT']

#%% Generating a Report

labels\_post = df\_prep.columns

report\_post = StatsReport()

# Create a simple data set summary for the console

for thisLabel in tqdm(labels\_post): # for each column, report stats

thisCol = df\_prep[thisLabel]

report\_post.addCol(thisLabel, thisCol)

#print(report.to\_string())

report\_post.statsdf.to\_excel("Quality\_Report\_Post\_Prep.xlsx")

#%% Sus out Bad Elements

from utils\_project1 import replace\_missing\_values\_avg

df\_final = df\_prep.copy()

temp\_report\_df = report\_post.statsdf

for element in tqdm(labels\_post):

if temp\_report\_df[element][10] > len(df\_prep)\*0.1: # Weeding out Elements that have more than 10% of missing values

df\_final = df\_final.drop(columns = [element])

print('ELEMENT Dropped:', element)

elif temp\_report\_df[element][10] < len(df\_prep)\*0.1:

if element == 'NEXTDAYPRECIPAMT' or element == 'NEXTDAYPRECIPFLAG':

avg\_value = 0

else:

avg\_value = temp\_report\_df[element][1]

replace\_missing\_values\_avg(df\_final, element, avg\_value)

#%% Run Quality Report and Output Data to Excel

df\_final.to\_excel('Weather\_Data\_Final.xlsx')

labels\_final = df\_final.columns

report\_final = StatsReport()

# Create a simple data set summary for the console

for thisLabel in tqdm(labels\_final): # for each column, report stats

thisCol = df\_final[thisLabel]

report\_final.addCol(thisLabel, thisCol)

#print(report.to\_string())

#report\_final.statsdf.to\_excel("Quality\_Report\_Final.xlsx")

#%% Setting up Training Data

# Data

feature\_names = df\_final.columns.drop(['NEXTDAYPRECIPFLAG','NEXTDAYPRECIPAMT'])

X = df\_final[feature\_names]

# Target

y\_precip\_flag = df\_final.loc[:, ['NEXTDAYPRECIPFLAG']]

labels = y\_precip\_flag['NEXTDAYPRECIPFLAG'].unique()

#%% Create Testing and Training data for Precip Flag

from sklearn.model\_selection import train\_test\_split

import numpy as np

from utils\_project1 import writegraphtofile, get\_true\_positive

from sklearn import tree

X\_train\_flag, X\_test\_flag, y\_train\_flag, y\_test\_flag = train\_test\_split(X, y\_precip\_flag, test\_size=0.3,

train\_size=0.7, random\_state=1996,

shuffle=True, stratify=None)

#%% Create Decision Tree - Entropy

score\_df = pd.DataFrame()

idx = 0

for i in range(3,9):

clf\_entropy = tree.DecisionTreeClassifier(criterion = "entropy", max\_depth = i)

clf\_entropy = clf\_entropy.fit(X\_train\_flag, np.array(y\_train\_flag['NEXTDAYPRECIPFLAG']))

training\_score = clf\_entropy.score(X\_train\_flag, y\_train\_flag['NEXTDAYPRECIPFLAG'])

true\_positive\_entropy, matrix\_df\_entropy = get\_true\_positive(clf\_entropy, X\_test\_flag, y\_test\_flag)

score\_df.loc[idx,'Tree Depth'] = i

score\_df.loc[idx,'Entropy Training Score'] = training\_score

score\_df.loc[idx,'Entropy Test Score'] = true\_positive\_entropy

idx = idx+1

# Measure Performance

print("Entropy Training set score = ", clf\_entropy.score(X\_train\_flag, y\_train\_flag['NEXTDAYPRECIPFLAG']))

print("Entropy Test set score = ", clf\_entropy.score(X\_test\_flag, y\_test\_flag['NEXTDAYPRECIPFLAG']))

print('Entropy True Positive Rate = ', true\_positive\_entropy)

# Create Graphic

path\_name = os.path.join(dir\_name, "Weather\_Data\_DecisionTree\_Entropy\_NextDayPrecipFlag.png")

writegraphtofile(clf\_entropy, feature\_names, (str(labels[0]), str(labels[1])), path\_name)

tree.export\_graphviz(clf\_entropy)

#%% Create Decision Tree - Gini

idx = 0

for i in range(3,9):

clf\_gini = tree.DecisionTreeClassifier(criterion = "gini", max\_depth = i)

clf\_gini = clf\_gini.fit(X\_train\_flag, np.array(y\_train\_flag['NEXTDAYPRECIPFLAG']))

training\_score = clf\_gini.score(X\_train\_flag, y\_train\_flag['NEXTDAYPRECIPFLAG'])

true\_positive\_gini, matrix\_df\_gini = get\_true\_positive(clf\_gini, X\_test\_flag, y\_test\_flag)

score\_df.loc[idx,'Gini Training Score'] = training\_score

score\_df.loc[idx,'Gini Test Score'] = true\_positive\_gini

idx = idx+1

# Measure Performance

print("Gini Training set score = ", clf\_gini.score(X\_train\_flag, y\_train\_flag['NEXTDAYPRECIPFLAG']))

print("Gini Test set score = ", clf\_gini.score(X\_test\_flag, y\_test\_flag['NEXTDAYPRECIPFLAG']))

print('Gini True Positive Rate = ', true\_positive\_gini)

# Create Graphic

path\_name = os.path.join(dir\_name, "Weather\_Data\_DecisionTree\_Gini\_NextDayPrecipFlag.png")

writegraphtofile(clf\_gini, feature\_names, (str(labels[0]), str(labels[1])), path\_name)

tree.export\_graphviz(clf\_gini)

#%% Linear Regression

# Target

y\_precip\_amt = df\_final.loc[:, ['NEXTDAYPRECIPAMT']]

labels\_amt = y\_precip\_amt['NEXTDAYPRECIPAMT'].unique()

# Split training/testing data by precip amt

X\_train\_amt, X\_test\_amt, y\_train\_amt, y\_test\_amt = train\_test\_split(X, y\_precip\_amt, test\_size=0.3,

train\_size=0.7, random\_state=1996,

shuffle=True, stratify=None)

from sklearn.linear\_model import LinearRegression

from utils\_project1 import get\_mse

from sklearn.metrics import mean\_squared\_error

linreg\_model = LinearRegression().fit(X\_train\_amt, np.array(y\_train\_amt['NEXTDAYPRECIPAMT']))

linreg\_model.score(X\_test\_amt, y\_test\_amt)

# Testing score

lin\_model\_pred\_test = linreg\_model.predict(X\_test\_amt)

mean\_squared\_error(y\_test\_amt, lin\_model\_pred\_test)

print('Coefficients:', linreg\_model.coef\_)

print('Intercept:', linreg\_model.intercept\_)

print('Mean squared error (MSE): %.2f'

% mean\_squared\_error(y\_test\_amt, lin\_model\_pred\_test))

#%% RidgeCV

from sklearn.model\_selection import cross\_val\_score, RepeatedKFold

from sklearn.linear\_model import RidgeCV

ridge\_model = RidgeCV().fit(X\_train\_amt, np.array(y\_train\_amt['NEXTDAYPRECIPAMT']))

# Testing score

ridge\_model\_pred\_test = ridge\_model.predict(X\_test\_amt)

mean\_squared\_error(y\_test\_amt, ridge\_model\_pred\_test)

print('Coefficients:', ridge\_model.coef\_)

print('Intercept:', ridge\_model.intercept\_)

print('Mean squared error (MSE): %.2f'

% mean\_squared\_error(y\_test\_amt, ridge\_model\_pred\_test))

#%% Measure Mean Square Error

print("Mean Square Error = ", get\_mse(linreg\_model, X\_test\_amt, y\_test\_amt))

print("Mean Square Error = ", get\_mse(ridge\_model, X\_test\_amt, y\_test\_amt))

#%% Use of Prior Days Data - THIS TAKES SEVERAL MINUTES

from utils\_project1 import create\_prior\_day\_data\_df

feature\_names = df\_final.columns.drop(['NEXTDAYPRECIPFLAG','NEXTDAYPRECIPAMT'])

df\_prior\_day = df\_final.copy()

#Create the DF for prior day

create\_prior\_day\_data\_df(df\_prior\_day, feature\_names)

#%% Setting up Training Data

# Data

feature\_names = df\_prior\_day.columns.drop(['NEXTDAYPRECIPFLAG','NEXTDAYPRECIPAMT'])

X = df\_prior\_day[feature\_names]

# Target

y\_precip\_flag = df\_prior\_day.loc[:, ['NEXTDAYPRECIPFLAG']]

labels = y\_precip\_flag['NEXTDAYPRECIPFLAG'].unique()

#%% Create Testing and Training data for Precip Flag

X\_train\_flag, X\_test\_flag, y\_train\_flag, y\_test\_flag = train\_test\_split(X, y\_precip\_flag, test\_size=0.3,

train\_size=0.7, random\_state=1996,

shuffle=True, stratify=None)

#%% Create Decision Tree - Entropy

idx = 0

for i in range(3,9):

clf\_entropy = tree.DecisionTreeClassifier(criterion = "entropy", max\_depth = i)

clf\_entropy = clf\_entropy.fit(X\_train\_flag, np.array(y\_train\_flag['NEXTDAYPRECIPFLAG']))

training\_score = clf\_entropy.score(X\_train\_flag, y\_train\_flag['NEXTDAYPRECIPFLAG'])

true\_positive\_entropy, matrix\_df\_entropy = get\_true\_positive(clf\_entropy, X\_test\_flag, y\_test\_flag)

score\_df.loc[idx,'Entropy Training Score Prior Day Data'] = training\_score

score\_df.loc[idx,'Entropy Test Score Prior Day Data'] = true\_positive\_entropy

idx = idx+1

# Measure Performance

print("Entropy Training set score = ", clf\_entropy.score(X\_train\_flag, y\_train\_flag['NEXTDAYPRECIPFLAG']))

print("Entropy Test set score = ", clf\_entropy.score(X\_test\_flag, y\_test\_flag['NEXTDAYPRECIPFLAG']))

print('Entropy True Positive Rate = ', true\_positive\_entropy)

# Create Graphic

path\_name = os.path.join(dir\_name, "Weather\_Data\_DecisionTree\_Entropy\_NextDayPrecipFlag\_PriorDay.png")

writegraphtofile(clf\_entropy, feature\_names, (str(labels[0]), str(labels[1])), path\_name)

tree.export\_graphviz(clf\_entropy)

#%% Create Decision Tree - Gini

idx = 0

for i in range(3,9):

clf\_gini = tree.DecisionTreeClassifier(criterion = "gini", max\_depth = i)

clf\_gini = clf\_gini.fit(X\_train\_flag, np.array(y\_train\_flag['NEXTDAYPRECIPFLAG']))

training\_score = clf\_gini.score(X\_train\_flag, y\_train\_flag['NEXTDAYPRECIPFLAG'])

true\_positive\_gini, matrix\_df\_gini = get\_true\_positive(clf\_gini, X\_test\_flag, y\_test\_flag)

score\_df.loc[idx,'Gini Training Score Prior Day Data'] = training\_score

score\_df.loc[idx,'Gini Test Score Prior Day Data'] = true\_positive\_gini

idx = idx+1

# Measure Performance

print("Gini Training set score = ", clf\_gini.score(X\_train\_flag, y\_train\_flag['NEXTDAYPRECIPFLAG']))

print("Gini Test set score = ", clf\_gini.score(X\_test\_flag, y\_test\_flag['NEXTDAYPRECIPFLAG']))

print('Gini True Positive Rate = ', true\_positive\_gini)

# Create Graphic

path\_name = os.path.join(dir\_name, "Weather\_Data\_DecisionTree\_Gini\_NextDayPrecipFlag\_PriorDay.png")

writegraphtofile(clf\_gini, feature\_names, (str(labels[0]), str(labels[1])), path\_name)

tree.export\_graphviz(clf\_gini)

#%% Exporting Score DF

score\_df.to\_excel('Decision Tree Scores.xlsx')

#%% Linear Regression - Using Prior Day Data

# Target

y\_precip\_amt = df\_final.loc[:, ['NEXTDAYPRECIPAMT']]

labels\_amt = y\_precip\_amt['NEXTDAYPRECIPAMT'].unique()

# Split training/testing data by precip amt

X\_train\_amt, X\_test\_amt, y\_train\_amt, y\_test\_amt = train\_test\_split(X, y\_precip\_amt, test\_size=0.3,

train\_size=0.7, random\_state=1996,

shuffle=True, stratify=None)

from sklearn.linear\_model import LinearRegression

from utils\_project1 import get\_mse

linreg\_model = LinearRegression().fit(X\_train\_amt, np.array(y\_train\_amt['NEXTDAYPRECIPAMT']))

linreg\_model.score(X\_test\_amt, y\_test\_amt)

# Testing score

lin\_model\_pred\_test = linreg\_model.predict(X\_test\_amt)

mean\_squared\_error(y\_test\_amt, lin\_model\_pred\_test)

print('Coefficients:', linreg\_model.coef\_)

print('Intercept:', linreg\_model.intercept\_)

print('Mean squared error (MSE): %.2f'

% mean\_squared\_error(y\_test\_amt, lin\_model\_pred\_test))

#%% RidgeCV

from sklearn.model\_selection import cross\_val\_score, RepeatedKFold

from sklearn.linear\_model import RidgeCV

ridge\_model = RidgeCV().fit(X\_train\_amt, np.array(y\_train\_amt['NEXTDAYPRECIPAMT']))

# Testing score

ridge\_model\_pred\_test = ridge\_model.predict(X\_test\_amt)

mean\_squared\_error(y\_test\_amt, ridge\_model\_pred\_test)

print('Coefficients:', ridge\_model.coef\_)

print('Intercept:', ridge\_model.intercept\_)

print('Mean squared error (MSE): %.2f'

% mean\_squared\_error(y\_test\_amt, ridge\_model\_pred\_test))

#%% Measure Mean Square Error

print("Mean Square Error = ", get\_mse(linreg\_model, X\_test\_amt, y\_test\_amt))

print("Mean Square Error = ", get\_mse(ridge\_model, X\_test\_amt, y\_test\_amt))

***B. utils\_project1.py***

import pandas as pd

from sklearn import tree

import pydotplus

import collections

from tqdm import tqdm

from sklearn import metrics

def create\_prior\_day\_data\_df(df, feature\_names):

for feature in feature\_names:

for idx in tqdm(df.index):

if idx == 0:

df.loc[idx, 'PREV\_'+feature] = 0

elif idx > 0:

df.loc[idx, 'PREV\_'+feature] = df.loc[idx-1, feature]

return df

def get\_mse(model, x, y):

model\_pred\_test = model.predict(x)

sum\_ = 0

idx\_pred = 0

for idx\_y in y.index:

a = (y.loc[idx\_y, 'NEXTDAYPRECIPAMT'] - model\_pred\_test[idx\_pred])\*\*2

sum\_ = sum\_ + a

idx\_pred = idx\_pred+1

mse = (1/len(model\_pred\_test)\*sum\_)

return mse

def get\_true\_positive(decision\_tree, x\_test, y\_test):

"""

Returns

-------

true\_positive\_value : float64

Results from the total correctly predicited divided by total predictions.

matrix\_df : DataFrame

Predicted Labels are Columns and True Labels are rows.

"""

test\_pred\_decision\_tree = decision\_tree.predict(x\_test)

confusion\_matrix = metrics.confusion\_matrix(y\_test, test\_pred\_decision\_tree)

#turn this into a dataframe

matrix\_df = pd.DataFrame(confusion\_matrix)

test\_set\_correctly\_pred = matrix\_df[0][0] + matrix\_df[1][1]

true\_positive = test\_set\_correctly\_pred / len(x\_test)

return true\_positive, matrix\_df

def replace\_missing\_values\_avg(df, column\_name, avg\_value):

"""

This function will take in a data frame and replace a missing value with

the average.

"""

missing\_values\_bool = df[column\_name].isna()

for idx in range(len(missing\_values\_bool)):

if missing\_values\_bool[idx] == True:

df.loc[idx, column\_name] = avg\_value

print(f"Value Replaced for {column\_name} at {idx}")

elif missing\_values\_bool[idx] == False:

pass

# for a two-class tree, call this function like this:

# writegraphtofile(clf, ('F', 'T'), dirname+graphfilename)

def writegraphtofile(clf, feature\_labels, classnames, pathname):

dot\_data = tree.export\_graphviz(clf, out\_file=None,

feature\_names=feature\_labels,

class\_names=classnames,

filled=True, rounded=True,

special\_characters=True)

graph = pydotplus.graph\_from\_dot\_data(dot\_data)

colors = ('lightblue', 'green')

edges = collections.defaultdict(list)

for edge in graph.get\_edge\_list():

edges[edge.get\_source()].append(int(edge.get\_destination()))

for edge in edges:

edges[edge].sort()

for i in range(2):

dest = graph.get\_node(str(edges[edge][i]))[0]

dest.set\_fillcolor(colors[i])

graph.write\_png(pathname)

class Weather\_Data\_CSV:

def \_\_init\_\_(self, csv\_path):

self.data\_raw = pd.read\_csv(csv\_path)

self.df\_prep

def get\_unique\_column\_values(df):

"""

Identifying Unique Values of each Column in DF

Output is a Dictionary of each Column

"""

headers\_unique = {}

for label in tqdm(df.columns):

headers\_unique[label] = df[label].unique()

#pbar.close()

return headers\_unique

def prep\_data(df, df\_out, headers\_unique):

"""

Extract Values for Elements and insert into df\_prep

"""

index\_ = 0

for date in tqdm(headers\_unique['DATE']):

date\_idx = df['DATE'] == date

df\_by\_date = df[date\_idx]

df\_out.loc[index\_, 'DATE'] = date

for idx in df\_by\_date['ELEMENT'].index:

df\_out.loc[index\_, df\_by\_date['ELEMENT'][idx]] = df\_by\_date['VALUE1'][idx]

index\_ = index\_+1

def addCol(self, label):

pass

class StatsReport:

def \_\_init\_\_(self):

self.statsdf = pd.DataFrame()

self.statsdf['stat'] = ['cardinality', 'mean', 'median', 'n\_at\_median', 'mode', 'n\_at\_mode', 'stddev', 'min', 'max', 'nzero', 'nmissing']

pass

def addCol(self, label, data):

self.statsdf[label] = [self.cardinality\_(data), self.mean\_(data),

self.median\_(data), self.n\_at\_median(data),

self.mode\_(data), self.n\_at\_mode(data),

self.std\_(data), self.min\_(data),

self.max\_(data), self.nzero\_(data),

self.nmissing\_(data)]

def to\_string(self):

return self.statsdf.to\_string()

def cardinality\_(self, d):

try:

return d.nunique()

except:

return "N/A"

def mean\_(self, d):

try:

return d.mean()

except:

return "N/A"

def median\_(self, d):

try:

return d.median()

except:

return "N/A"

def n\_at\_median(self, d):

try:

n = d == d.median()

return n.sum()

except:

return "N/A"

def mode\_(self, d):

try:

return int(d.mode())

except:

return "N/A"

def n\_at\_mode(self, d):

try:

n = d == int(d.mode())

return n.sum()

except:

return "N/A"

def std\_(self, d):

try:

return d.std()

except:

return "N/A"

def min\_(self, d):

try:

return d.min()

except:

return "N/A"

def max\_(self, d):

try:

return d.max()

except:

return "N/A"

def nzero\_(self, d):

try:

n = d == 0

return n.sum()

except:

return "N/A"

def nmissing\_(self, d):

try:

n = d.isna()

return n.sum()

except:

return "N/A"