Final Machine Learning Project

Differences of Daily Data on Weekly SVB Stock Return Classification

Paul Pawelec

University of Guelph

FIN*6200

Professor Fred Liu

April 24th, 2023

Abstract

The objective of this study is to classify future weekly returns for Silicon Valley Bank using stock data and other important predictors. The methodology used is simply 12 different models ranging from various regressive models with different penalties, gradient boosting, trees, and ensemble techniques applied on daily differenced data for weekly returns. Some of the challenges faced is the high and low correlations, noise, and higher dimensionality in the data. The implications of the paper generally found that it was possible to beat a simple or suboptimal strategy using machine learning techniques even with this kind of data.

Contents

Introduction

Project Focus

This projects focus is to forecast and classify future weekly returns for Silicon Valley Bank (SVB) based on its daily stock data and other factors. I had chosen this stock simply because of the recent near collapse of SVB and thought that it might be interesting to classify its returns based on its stock data and other variables. This is with the hope of capturing some of the endogenous effects surrounding the company's downfall and maybe using that to predict future returns; mainly using the risk-free rate for 3- and 6-month treasury bills and the total value of the mortgage-based securities market. Another aim of the project was to work with imperfect data to test how well machine learning models can work with noisy data, deal with high or low correlations, low amounts of data, and high dimensional data.

Why Use Machine Learning

Classifying returns can be tricky even with good data, but in the cases of poor-quality data it might be more beneficial to use machine learning to capture complex or non-linear relationships to provide better results than typical benchmarks or strategies. By using machine learning over traditional benchmarks, it's easier to identify predictive variables, improve overall predictions, handle noisy data, and to deal with high-dimensional data through the various machine learning techniques and models.

Main Results

The main results show some success with dealing with this kind of data using various machine learning techniques. Mainly, it's shown that models like lasso, PCR, and decision trees were all able to capture similar patterns in the data to outperform the pure strategy benchmark by a decent margin. Essentially, even with data that might not be the most ideal in terms of quality, it can still be better to apply some machine learning techniques to find results that are more predictive than traditional benchmark techniques. However, I'm unable to beat a more strategic benchmark (3-week rolling average) even with more complicated models and ensemble techniques.

Data

Data Sets, Sources, and Consolidation

There are three main data sets used for this project, the stock data, the 3-month and 6-month daily treasury bill rate, and the total value for mortgage-backed securities (MBS) in domestic banks; see Appendix A for graphs. The stock data is simply sourced from yahoo while both the treasury bill rates, and MBS data are sourced from the Federal Reserve Economic Data (FRED) website. After these are downloaded from their respective sources, they undergo some data cleaning to consolidate everything into one data set. The links for this data can be found under the references section.

The stock data is essentially our base that the rest of the data needs to conform to. The stock data only undergoes some formatting for the date/index and a forward fill is applied to replace "nan"; these are simply dates that are missing due to various complications with the market (Holidays, Closed Markets, Trades Halted).

The data for 3-month and 6-month treasury bills are taken as a subset of a larger dataset from FRED, however, due to the incompleteness of the data for longer period bonds, I decided to only keep the 3- and 6-month bills since this would have less issues with cleaning and interpolation. This included converting the date format to the same as the stock data and forward filling for missing data.

The MBS data is in a weekly format, so I had to resample to business weeks. I then just formatted the dates, and everything was ready for consolidation.

The last steps are to consolidate each data set together, trim our data to fit to equal five-day weeks across the data, and then check if all weeks have all their business days.

Initial Data Set and Further Transformations

At this stage our data has 6910 observations of daily data for over twenty-six years and has the following attributes found in Table 1; this table indicates a wide range of values across all our variables, but also that our Open, High, Low, Close and Adjusted Close are all similar in terms of their mean, standard deviation, min, max, and even similar percentile values. Already we see some indications of issues with our data with potential high correlations. Figure 1 shows a heat map pertaining to this data; the high and perfect correlations are then visualized between variables for our stock prices. These are mostly addressed after the further transformations.

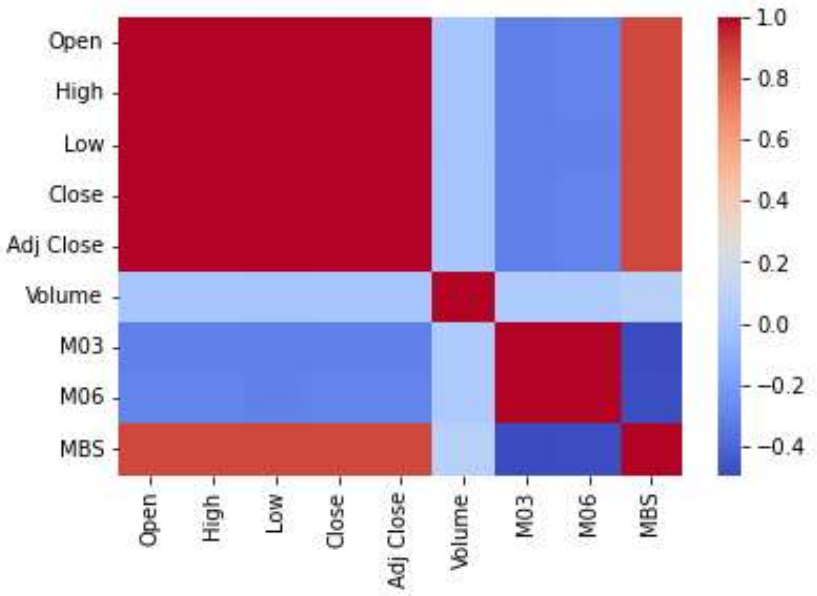
Table 1

Descriptive Statistics for Data Before Additional Transformations

	Open	High	Low	Close	Adj Close	Volume	M03	M06	MBS
Count	6910	6910	6910	6910	6910	6910	6910	6910	6910
Mean	113	115	112	113	113	516358	2	2	861
STD	141	143	139	141	141	1613792	2	2	527
Min	0	1	0	0	0	0	0	0	221
25%	31	32	30	31	31	258552	0	0	429
50%	50	51	50	50	50	394654	1	2	723
75%	133	135	131	133	133	579605	4	4	1192
Max	761	763	745	755	755	84502118	6	6	2131

Figure 1

Heat Map for Data Before Additional Transformations

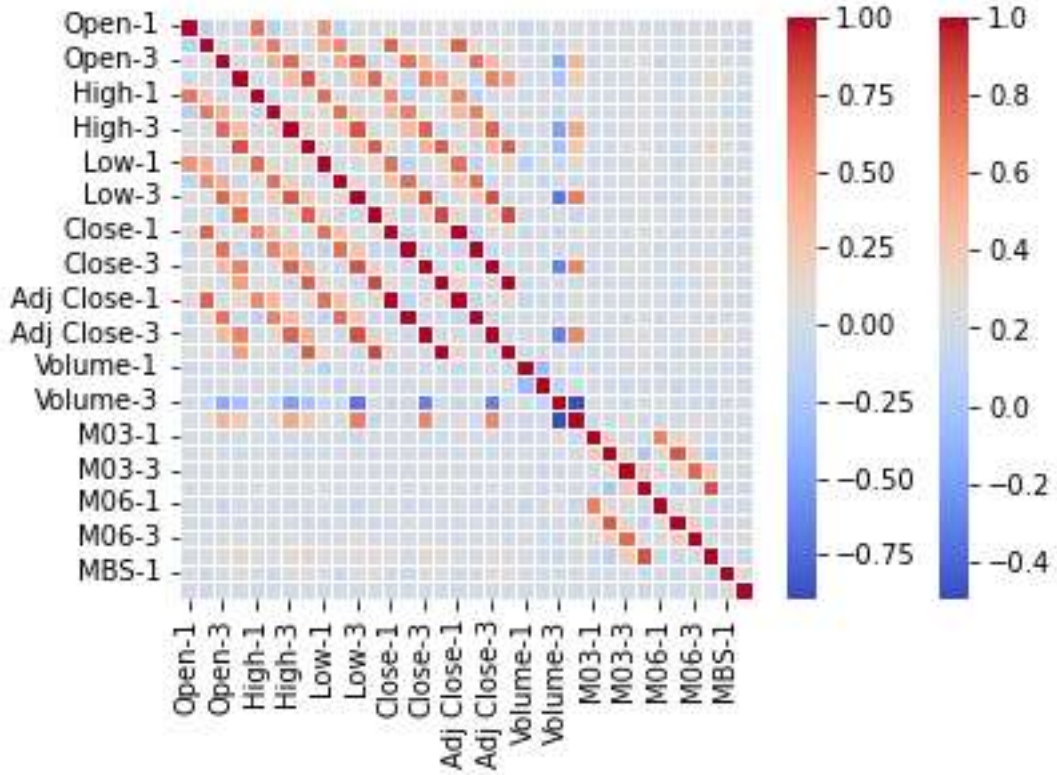


The transformations I apply to our data is simply to take the difference between adjacent time periods for each week of data for all years. I then drop the last observation or day, since I don't want to take the difference of the next day, which would be from next week. This way I have a data set comprised of daily movements and changes isolated for each week. I then transpose the data so that each row contains four daily differences for

each variable. It is at this point that I drop some of the differences for the MBS column since it is weekly data resampled for daily. Which means some differences are zero while one is the actual difference from week to week which is kept. The data's attributes can be found in appendix B under Table A.1, and its respective heatmap in Figure 2. The data looks different now just by looking at the various measures of mean, standard deviations, etc.... We can now also see that most of the perfect correlations have been dealt with.

Figure 2

Heat Map for Data After Additional Transformations



Some of the potential issues now, however, is that although some of the high correlations have been dealt with, there are still some noticeable high ones left, mostly between the close and adjusted close prices, and the data is somewhat noisier now, especially so with it already being stock data at a daily frequency. We also see an issue a fair number of extremely low correlations across volume, the risk-free rates, and the

MBS total values. This could be an issue for several reasons, but mainly it could cause issues with identifying patterns in the data, detecting significant relationships among predictors, and could result in poorer accuracy for our models. However, it could also mean more independent variables which could help with our model's predictability and help them stay more generalized.

Note that at the same time, the dependent binary variable of 1 for positive and 0 for negative is created by taking the weekly returns from the future week and lagging it. This is simply taking the difference of the adjusted close price of the first day of the week with the last day of the week. So, each row contains all the daily differences for our variables and the future weeks return as a binary variable. This way the models associate the prior weeks data with the future weeks return. Additionally, I had also tried using various thresholds for determining whether a return was good enough to be classified as a positive return. This was done using the standard deviation of the weekly return and playing around with lower and lower thresholds until zero. From these results I concluded that this would create an unbalanced numbers of positive and negative classes and would ultimately result in biased models that would generally give pure strategies. I then opted to just keep things simple and split positive and negative returns at zero which would tend to give more balanced groups of data.

Methodology

Test and Train Sets and Sampling

Initially, I used the entire dataset and did various splits of training sets and test sets. I generally found that larger test sets (and smaller training sets) would lead to

some models doing better like trees versus just the logistic models and their offshoots. But it also made tunning parameters harder, as increasing or decreasing the complexity of parameters would give various results. Often models would want to train with higher complexity but then would end up returning poorer out of sample performance. This would then just lead to overfitting and models using pure strategies of only predicting positive or negative outcomes. Which is something I wanted to avoid, since one of our benchmarks would be using this strategy.

I had also done some random sampling and various trims to the data, which would result in poorer model accuracy, but more robust predictions. This led me down various rabbit holes of testing various sizes only to get various disappointing results.

Eventually, I finalized the dataset down to around two years and set the train and test split be at 50%. At this point the models might not have had the same in-sample performance or were accurate as before, but it gave a good starting point to begin tunning our models due to the robustness of its predictions in the out of sample tests. I choose a 50% split after playing around with a few different splits and found 50% to be ideal for predicting the next years returns. Ideally, we could have used more data to mitigate some of the noise and variance. But with forecasting future returns it seems to make more sense to train on less and newer data rather than training on all the data. This is because we can have our model focus on more recent trends and more relevant data to hopefully relate to the data found in our test set.

Originally, I was testing my models with random train and test sampling, this was because I wanted to train these models under harsher conditions where they would have to generalize more and reduce bias in the training and test sets. However, it ended up

making more sense to give the training and test sets temporal order since we were predicting the future based on past data. Overall, using ordered data would result in better models.

Model Tunning

For this project I used 12 models, 11 of these models can be seen in appendix C along with their respective parameters. The 12th model is a voting model that uses all the other 11 predictions to determine an outcome based on majority rules. I had also tried using only the top models with around five or less of the best. This didn't make much of an impact on the accuracy as I had liked, and I found this to because many models shared similar strategies.

These models are tested against two benchmarks. The first being the mean of positive outcomes or a pure strategy of always picking positive, and the second is a rolling 3-week average of positive returns; it averages the past three weeks and if that meets the threshold of 0.5 then it predicts a positive return for that week. I chose these as my benchmarks because they were simple strategies that traders could use. The pure strategy focuses on the market always doing well in the long-run and the rolling average is a way of looking at the momentum of returns.

After a few tests against the benchmarks, it became clear that the predictive nature of our variables is low. Although unfortunate, this made tuning our parameters easier and less computationally intensive. For instance, tree models were kept shallow with lower complexity to keep them more generalized for the test set, and the logistic regressive models had higher penalties to reduce overfitting. I also adjusted the learning

rates to be slower to compensate for the shallower models and to make sure models could somewhat fit to the noisy data. Additionally, I had tried more complex parameters, but this always led to overfitting and pure strategies. Cross validation is also something I adjusted, but it was noticeable that increases to the parameter would not give any discernable improvements to model performance. This could be a cause of concern since it indicates several things, particularly, it could indicate that the model is underfitting, overfitting, has data quality issues, or there's an insufficient amount of data for the model to realistically use cross-validation to improve its results.

Results and Conclusions

Results

I found that in most cases I could reliably construct models to beat the first benchmark, even with various sample, test, and train sizes. But the most robust results were again found with the smaller sample size of two years and 50% split for the training and test set. The robustness and accuracy of each model can be seen in appendix D, Figure D.1, and Graph 1 respectively. For the graph, each bar represents a model with the model accuracy on top along with dotted lines across the graph to indicate the benchmarks. Beating the first benchmark was easy, but there were some cases where I had to adjust some parameters to better fit them. However, the second benchmark was quite challenging to match, and I never managed to get too close. There are a few ideas I could have implemented but I opted to continue using various parameters until I achieved the below results.

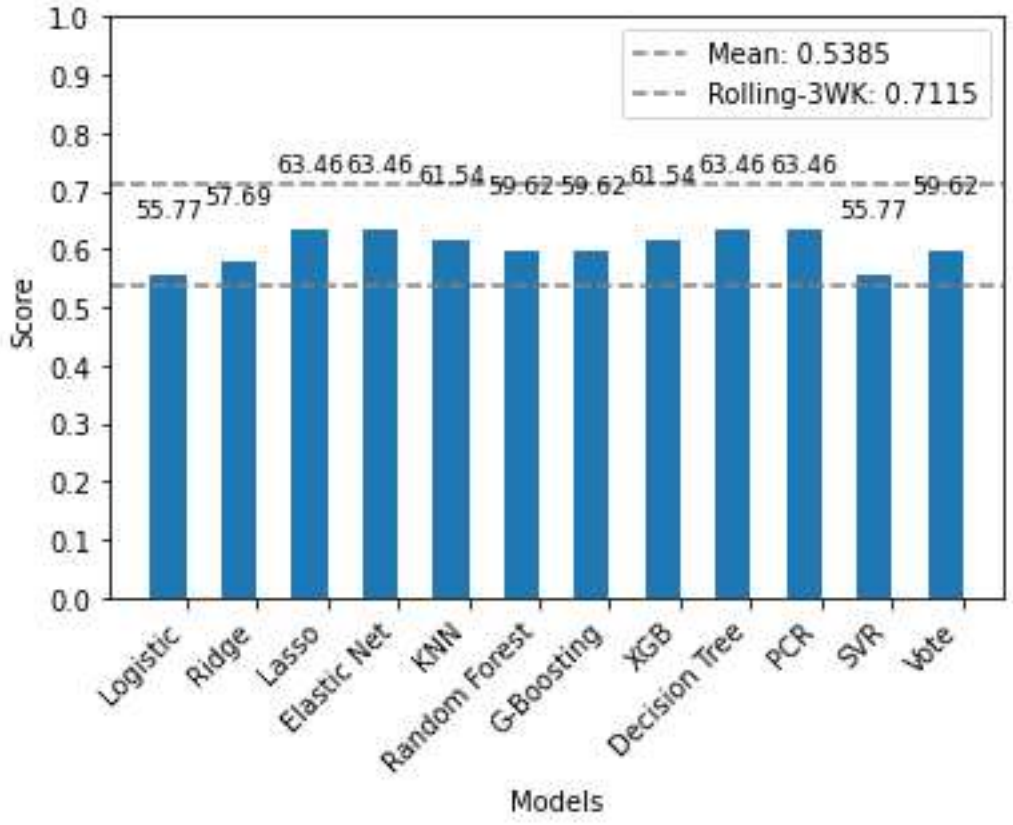
The best models to use were either logistic regression with a lasso penalty, decision trees, or the PCR method. Elastic net also had the same performance, but after tuning the parameter around it seems the best performance for that model was just to use the lasso regression over the ridge and not a mix of both. So, given that these three different models performed almost identically, there are some differences in the confusion tables to note, this tells us that the regularization methods of lasso and PCR don't give that much improvement over a model like decision trees that do not do feature or dimensional reduction. Which means the decision tree model can capture some complex or non-linear patterns in our data that can compete with dimension reduction. This overall could also be an indication that the data set might have been too small for these models to gain substantial returns with their different methods, or the parameters could have been further tuned to outperform each other. Overall, it's good that these models were able to perform similarly since this would suggest that signals in the data are strong enough to be captured by different models.

Furthermore, even though models like logistic regression with no penalty, ridge, and SVR were all sub-optimal compared to the best models, they still outperformed the benchmark while using non-pure strategies.

The economic implications of these findings indicate a solution to using sub-optimal decisions with poor data quality. For instance, in cases where data has high dimensionality with poor predictive powers, and extremely high or low correlations among predictors, it can still be beneficial to use a machine learning approach to discover and use complex patterns to prevent sub-optimal decisions like a simple average or pure strategy.

Graph 1

Accuracy Scores for 12 Models against Benchmarks



Conclusions

Overall, I'm happy with the results and the number of the models I applied, however, I had also hoped to include a computer vision model as an additional model to test against the other machine learning models and the benchmarks. I had started the process of drawing of weekly graphs in a similar format to the paper "(Re-)Imag(in)ing Price Trends", but it didn't seem too feasible to properly create and test these images in a timely manner while also doing the rest of the report. I also had concerns about how computationally intensive it might be given my older hardware and the complexity/demands of these computer vision models.

Furthermore, I would have liked to add in a section that showed the profitability of these strategies over the test set so I could have a final test of their value over the benchmark models. I think this might have been interesting to do because maybe it's the case that they predict weeks with not only positive returns but better positive returns. At the start I was looking for ways to predict not only positive but substantially positive returns, however, the idea of training with unbalanced classes gave me some concerns, things like biased performances, sample bias, and decision biases along with overfitting came to mind. Maybe this could have been mitigated by using more than just two classifications to get more balanced groups. However, this could create further issues with needing to make sure classes are distinct and well-defined.

In conclusion, I'm pleased with my results. Mostly because I managed to beat the first benchmark with this kind of data. I had initially expected that the data might not have been as predictive of future returns, but the results seem to show otherwise. However, I couldn't beat the second benchmark. I had hoped that maybe the voting technique would have been able to push for better accuracy and go above the second benchmark, but it seems that in most cases there was a clear winner or strategy among models, so the technique had few gains if any over other separate models. However, with some of the models, I did get close to the second benchmark, around 6-7%, which I think is pretty good considering I'm only using a weeks' worth of data for each output of y, whereas the rolling average uses 3 weeks. Maybe if I had used a rolling window approach or horizontally stacked my data to use the entire weeks or multiple weeks of data, I could have beaten the rolling average benchmark. This would have potentially avoided the use of transposing multiple columns and creating colinear features and

higher dimensionality in my data which would have reduced overfitting and overcomplicated models.

References

Board of governors of the Federal Reserve System. Federal Reserve Board - H15 - Data Download Program - Download. (n.d.). Retrieved April 24, 2023, from https://www.federalreserve.gov/datadownload/Download.aspx?rel=H15&series=4ab494d223ec49ec01fe09b49c6d17da&filetype=csv&label=include&layout=seriescolumn&from=01%2F01%2F1993&to=04%2F04%2F2023

Treasury and agency securities: Mortgage-backed securities (MBS), large domestically chartered commercial banks. FRED. (2023, April 21). Retrieved April 24, 2023, from https://fred.stlouisfed.org/series/TMBLCBW027SBOG

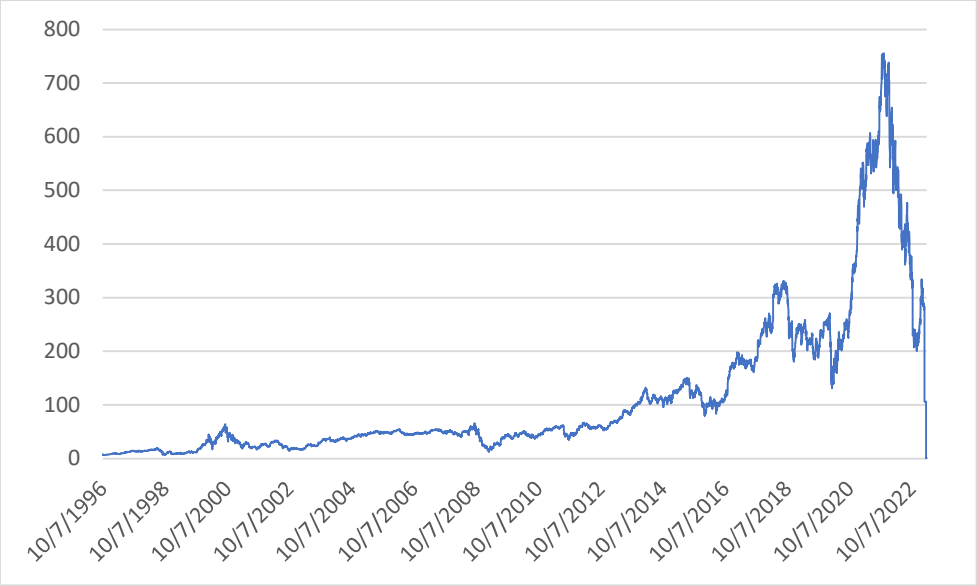
Yahoo! (2023, April 24). *SVB Financial Group (SIVBQ) Stock Historical Prices & Data*. Yahoo! Finance. Retrieved April 24, 2023, from https://finance.yahoo.com/quote/SIVBQ/history?period1=725846400&period2=1680566400&interval=1d&filter=history&frequency=1d&includeAdjustedClose=true

Appendix A

Data Graphs

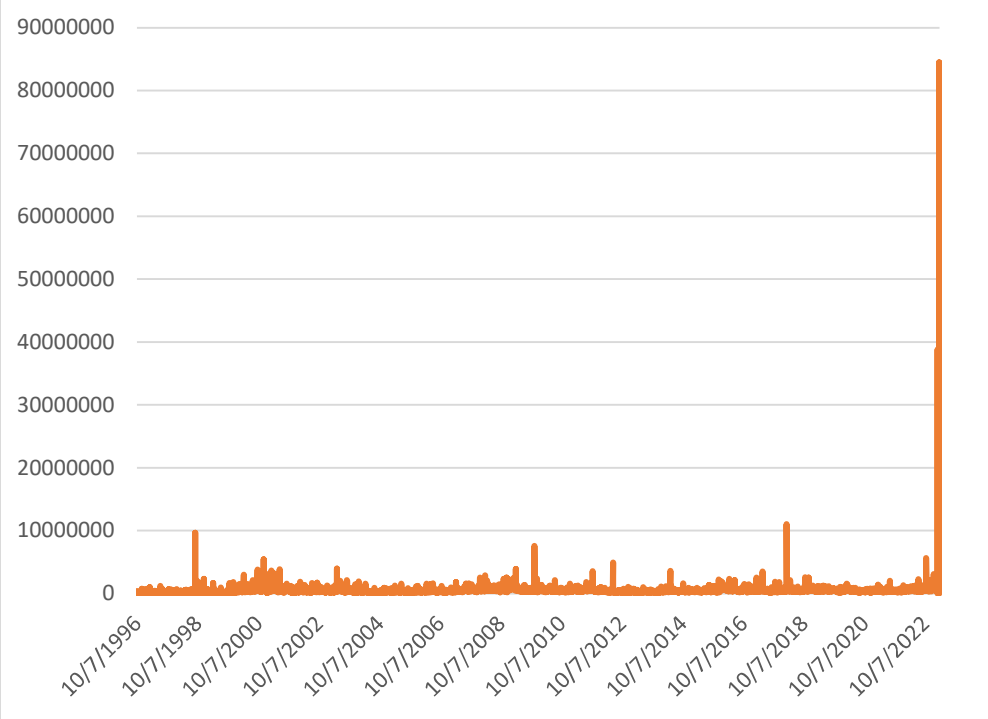
Graph A.1

Adjusted Closing Stock Price for SVB



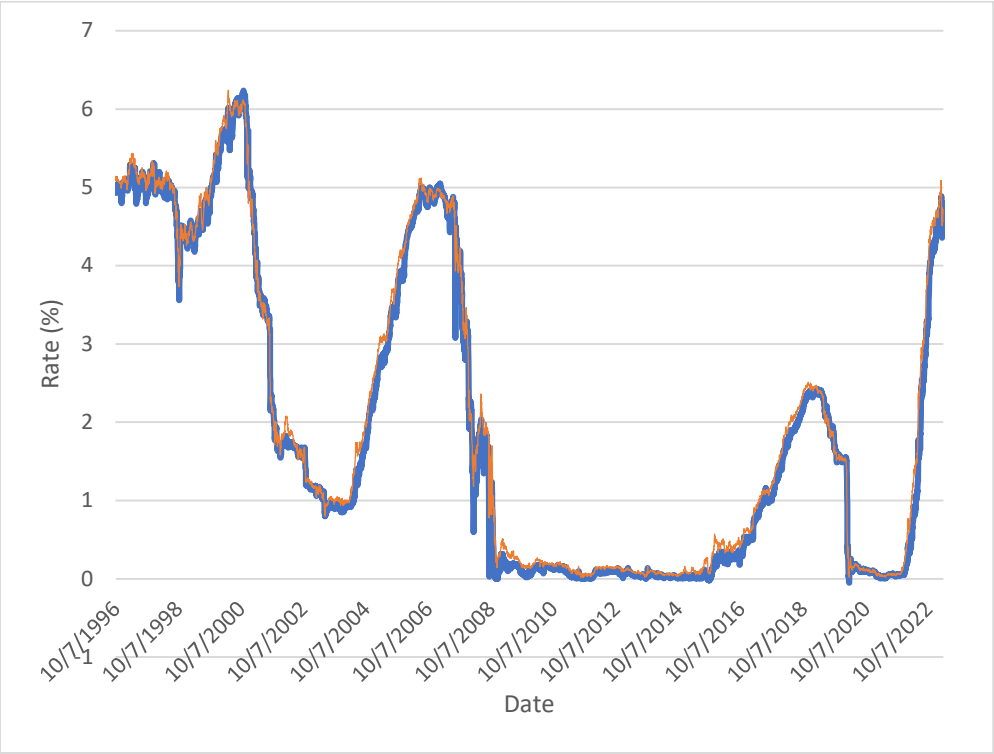
Graph A.2

Stock Volume for SVB



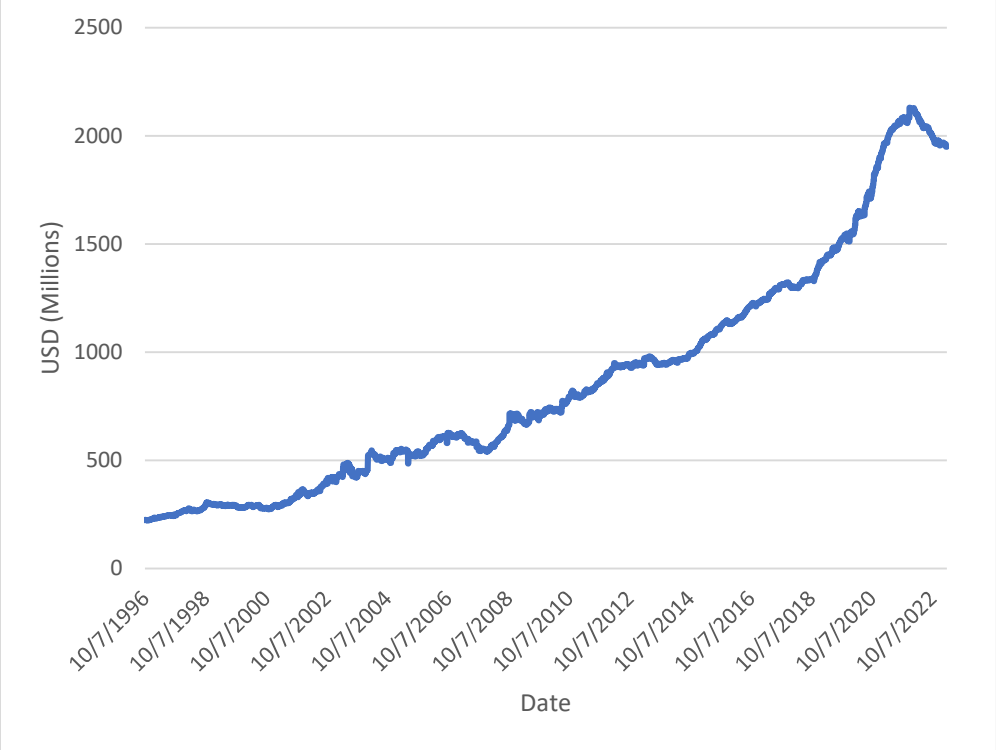
Graph A.3

3- and 6-Month U.S. Treasury Bill Rate



Graph A.4

Total Mortgage Based Securities Value for Domestic Banks



Appendix B

Data Descriptions

Table A.1

Descriptive Statistics for Data After Additional Transformations

	Open-1	Open-2	Open-3	Open-4	High-1	High-2	High-3	High-4
Count	1381.00	1381.00	1381.00	1381.00	1381.00	1381.00	1381.00	1381.00
Mean	0.06	0.13	-0.09	0.10	0.13	0.16	-0.01	0.07
STD	4.00	4.28	5.38	6.26	3.79	3.92	4.75	6.15
Min	-44.76	-36.24	-90.31	-70.51	-36.15	-27.74	-93.26	-71.71
25%	-0.53	-0.53	-0.56	-0.59	-0.47	-0.52	-0.50	-0.53
50%	0.03	0.03	0.00	0.03	0.02	0.00	0.00	0.00
75%	0.66	0.74	0.71	0.76	0.64	0.64	0.58	0.64
Max	25.33	36.44	30.78	46.09	33.33	44.82	30.01	53.52
	Low-1	**Low-2**	**Low-3**	**Low-4**	**Close-1**	**Close-2**	**Close-3**	**Close-4**
Count	1381.00	1381.00	1381.00	1381.00	1381.00	1381.00	1381.00	1381.00
Mean	0.08	0.17	-0.24	-0.01	0.17	0.09	-0.11	0.03
STD	4.01	4.01	6.05	6.12	4.71	4.60	6.28	6.30
Min	-46.04	-33.04	-164.00	-73.36	-48.12	-37.17	-161.79	-74.81
25%	-0.46	-0.47	-0.55	-0.50	-0.59	-0.58	-0.64	-0.56
50%	0.06	0.03	0.00	0.02	0.05	0.03	0.00	0.00
75%	0.61	0.67	0.63	0.65	0.81	0.71	0.70	0.71
Max	25.71	35.98	31.08	45.75	33.38	36.63	51.36	48.24
	Adj Close-1	**Adj Close-2**	**Adj Close-3**	**Adj Close-4**	**Volume-1**	**Volume-2**	**Volume-3**	**Volume-4**
Count	1381	1381	1381	1381	1381	1381	1381	1381
Mean	0.17	0.09	-0.11	0.03	2.76E+04	4.34E+02	2.94E+04	2.51E+04
STD	4.71	4.60	6.28	6.30	2.81E+05	3.19E+05	1.10E+06	1.18E+06
Min	-48.12	-37.17	-161.79	-74.81	-1.46E+06	-4.30E+06	-2.79E+06	-3.87E+07
25%	-0.59	-0.58	-0.64	-0.56	-9.15E+04	-1.07E+05	-1.10E+05	-1.01E+05
50%	0.05	0.03	0.00	0.00	1.55E+04	-6.11E+03	0.00E+00	0.00E+00
75%	0.81	0.71	0.70	0.71	1.22E+05	9.39E+04	1.09E+05	1.38E+05
Max	33.38	36.63	51.36	48.24	3.85E+06	3.38E+06	3.79E+07	1.04E+07
	M03-1	**M03-2**	**M03-3**	**M03-4**	**M06-1**	**M06-2**	**M06-3**	**M06-4**
Count	1381.00	1381.00	1381.00	1381.00	1381.00	1381.00	1381.00	1381.00
Mean	0.00	-0.01	-0.01	0.00	0.00	-0.01	-0.01	0.00
STD	0.04	0.04	0.04	0.04	0.03	0.03	0.03	0.04
Min	-0.49	-0.81	-0.51	-0.49	-0.44	-0.48	-0.45	-0.39
25%	-0.01	-0.02	-0.01	-0.01	-0.01	-0.02	-0.01	-0.01
50%	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00

75%	0.01	0.00	0.01	0.01	0.01	0.00	0.01	0.01
Max	0.48	0.22	0.25	0.74	0.29	0.16	0.26	0.73
	MBS-1	**Return**						
Count	1381.00	1381.00						
Mean	1.25	0.54						
STD	7.76	0.50						
Min	-57.98	0.00						
25%	-2.24	0.00						
50%	0.90	1.00						
75%	4.06	1.00						
Max	68.64	1.00						

Appendix C

Model Parameters

Table C.1

11 Models and their Parameters

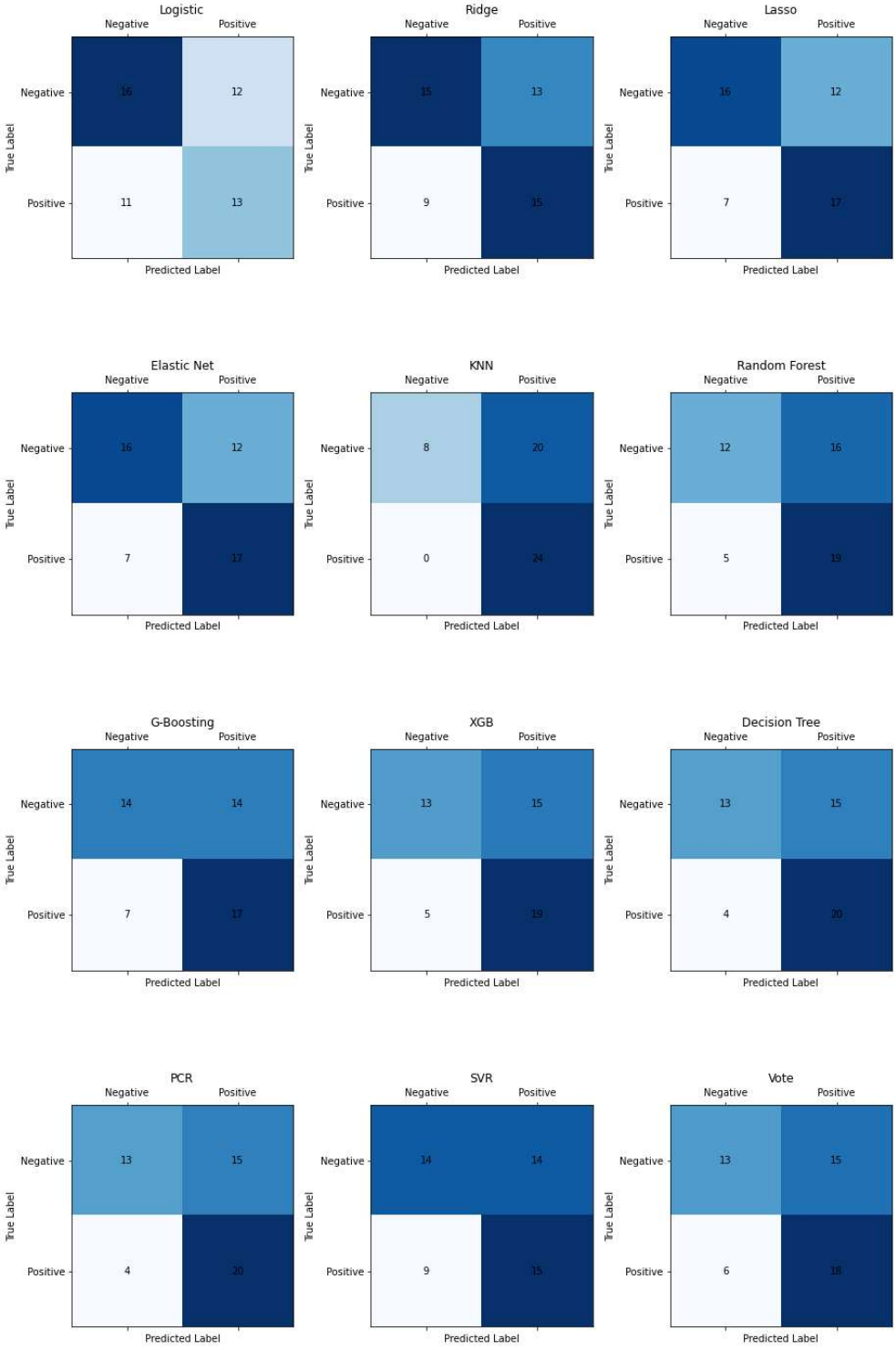
Model	Parameters
LogisticRegression	C=0.5, penalty=None
LogisticRegression - Ridge	C=0.5, solver='saga'
LogisticRegression - Lasso	C=0.5, penalty='l1', solver='saga'
LogisticRegression - Elastic Net	C=0.5, l1_ratio=0.9, penalty='elasticnet', solver='saga'
KNeighborsClassifier	n_neighbors=9
RandomForestRegressor	max_depth=2, max_features='sqrt', min_samples_leaf=3, min_samples_split=3, n_estimators=10, n_jobs=-1, random_state=1
GradientBoostingRegressor	learning_rate=0.05, max_depth=2, max_features='sqrt', min_samples_leaf=4, min_samples_split=10, subsample=0.5
XGBRegressor	learning_rate=0.0085, max_depth=2, ne, n_estimators=100,
DecisionTreeRegressor	max_depth=2, min_samples_leaf=3, min_samples_split=1,
PCR	max_depth=2, min_samples_leaf=3, min_samples_split=1,
SVC	C=0.1, kernel='linear'

Appendix D

Confusion Table Results

Figure D.1

Confusion Table Results for 12 Models



Appendix F

Data Cleaning Code

```
# -*- coding: utf-8 -*-

"""

Created on Tue Apr  4 21:12:01 2023


@author: Paul

"""


# ============================================================================

# Clean Data

# ============================================================================


# Packages ------------------------------------------------------------------

import pandas as pd

import numpy as np

import os

#-----------------------------------------------------------------------------


# Load Our Data -------------------------------------------------------------

# WD

path = "C:\\Users\\Paul\\OneDrive - University of Guelph\\Semester Folders\\Semester 10\\FIN 6200
- Artificial Intelligence in Finace - Financial Econometrics\\Final Project\\Datasets"

os.chdir(path)

# Silicon Valley Bank

SVB = pd.read_csv("SIVBQ.csv")

# US T-Bills

RFR = pd.read_csv("FRB_H15.csv", skiprows=5)

# MBS USD Value (Only Major Banks)

MBS = pd.read_csv("TMBLCBW027SBOG.csv")

#-----------------------------------------------------------------------------


# Clean Data ----------------------------------------------------------------

# SVB

SVB["Date"] = pd.to_datetime(SVB["Date"]) # Set Date format

SVB = SVB.set_index("Date") # Set index

SVB.index = SVB.index.date # Set index to date

SVB = SVB.fillna(method='ffill') # forward fill
```

```
# US T-Bills
RFR = RFR.iloc[:,0:3] # Keep first three columns (4-week and 1-year is missing too much data)
RFR["Time Period"] = pd.to_datetime(RFR["Time Period"])
RFR = RFR.set_index("Time Period") # Set Index
RFR.index = RFR.index.date
RFR = RFR.dropna() # Nothing drops, so no NAs.
RFR = RFR.replace({"ND": np.nan}) # Replace string ND with nan
RFR = RFR.fillna(method='ffill') # forward fill
RFR = RFR.iloc[1:,::] # First value had NA so I dropped it

# MBS
MBS["DATE"] = pd.to_datetime(MBS["DATE"]) # Set Date format
MBS.columns
MBS = MBS.set_index("DATE") # Set index
MBS = MBS.resample("B").ffill() # forward fill
MBS.index = MBS.index.date # Set index to date
#--------------------------------------------------------------------------------

# Combine Datasets ------------------------------------------------------------
df = pd.merge(SVB, RFR, left_index=True, right_index=True, how="outer") # Outer merge
df = pd.merge(df, MBS, left_index=True, right_index=True, how="outer") # Outer merge

df.iloc[:,-1] = (df.iloc[:,-1]).fillna(method='ffill') # Forward fill
df = df.dropna(subset=['TMBLCBW027SBOG']) # Drop any na's found in MBS
df = df.fillna(method='ffill') # Forward fill na values

df = df.iloc[3:,:] # Trim first couple of rows to remove NAs
df = df.iloc[:-1,:] # Remove last report to fit data to full business weeks.
#--------------------------------------------------------------------------------

# Check if Weeks are full -----------------------------------------------------
df.index = pd.to_datetime(df.index)

# resample the data at a weekly frequency and count the number of observations per week
weekly_counts = df.resample('W').count()

# calculate the expected number of observations per week
expected_counts = 5  # since there are 5 business days in a week
```

```
# find the index values of the weeks where the count of observations is less than the expected
count
missing_weeks = weekly_counts[weekly_counts < expected_counts].dropna().index


# print the index values of the missing weeks
if len(missing_weeks) > 0:
    print(f'The following weeks are missing a business day: {missing_weeks}')
else:
    print('All weeks have all business days')
#-------------------------------------------------------------------------------


# Export -----------------------------------------------------------------------


# Move the index to a column named "Date"
df.reset_index(inplace=True)
df.rename(columns={'index': 'Date'}, inplace=True)


df.to_csv("projectdata.csv", index=False)
#-------------------------------------------------------------------------------
```

Appendix G

Model Training Code

```
# ==============================================================================
# Train and Test Mdoels
# ==============================================================================
# Code to train and test various models for final project, and store, plot, and
# Print results.
# ==============================================================================
# Load Packages
# ==============================================================================

# Data Related
import numpy as np
import pandas as pd

# To Change Directories
import os

# Plotting
import matplotlib.pyplot as plt
import math # For deciding the grid size
import seaborn as sns

# Creating and Formating Train and Test Sets
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

# Models and Implementation
from sklearn.model_selection import GridSearchCV
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import RandomForestRegressor
from sklearn.ensemble import GradientBoostingRegressor
import xgboost as xgb
from sklearn.tree import DecisionTreeRegressor
from sklearn import tree
from sklearn.decomposition import PCA
```

```
from sklearn.svm import SVC

from sklearn.ensemble import VotingClassifier

from sklearn.pipeline import make_pipeline, Pipeline

import random


# Accuracy

from sklearn.metrics import accuracy_score, confusion_matrix

from sklearn.metrics import classification_report


# ============================================================================

# Set Seed

# ============================================================================

# Setting the seed to 1

np.random.seed(1)

# ============================================================================

# Pull Data

# ============================================================================

# Data is cleaned in prior script called "CleanData.py".

# - This just combines all our datasets into one uniform DataFrame

#  Contains daily stock data for SVB, Domestic Bank Mortgage Securities, and

#  3-month and 6-month rates


path = "C:\\Users\Paul\\OneDrive - University of Guelph\\Semester Folders\\Semester 10\\FIN 6200
- Artificial Intelligence in Finace - Financial Econometrics\\Final Project\\Datasets"

os.chdir(path)

del(path)


df = pd.read_csv("projectdata.csv")

df = df.rename(columns = {"RIFSGFSM03_N.B":"M03","RIFSGFSM06_N.B":"M06","TMBLCBW027SBOG":"MBS"})


# ============================================================================

# Weekly Returns, Weekly Data Format, Look at Data

# ============================================================================

# Format Date to DateTime and set to index

df['Date'] = pd.to_datetime(df['Date'])

df.set_index('Date', inplace=True)


# Calculate Weekly Return
```

```
weekly_returns = df.groupby(pd.Grouper(freq='5B'))['Adj Close'].last().pct_change()

# ============================================================================
# Categorize Returns
# ============================================================================

# Function to return our dependent variable based on weekly return.
def categorize_return(weekly_return):
    if weekly_return >= 0:
        return 1
    elif weekly_return < 0:
        return 0

weekly_returns_cat = pd.DataFrame(weekly_returns.apply(lambda x: categorize_return(x)))
weekly_returns_cat.columns = ["Return"]

# ============================================================================
# Additional Data Prep
# ============================================================================
# Need to aggregate weekly data and setup a new dataframe

# Create Weekly 1,2,3,4 differences for each column ---------------------------
df2 = pd.DataFrame()

# FOR loop to create differences and transpose rows.
for i in range(0, len(df), 5):

    data = df.iloc[i:i+5]
    data_lag = data.shift(-1)

    data = data.drop(data.index[4])
    data_lag = data_lag.drop(data_lag.index[4])

    diff = data_lag - data

    diff2 = np.concatenate((diff.iloc[:,0].values,
                    diff.iloc[:,1].values,
                    diff.iloc[:,2].values,
```

```
                    diff.iloc[:,3].values,
                    diff.iloc[:,4].values,
                    diff.iloc[:,5].values,
                    diff.iloc[:,6].values,
                    diff.iloc[:,7].values,
                    diff.iloc[:,8].values,
                    ))

    diff2 = pd.DataFrame(diff2).T

    df2 = pd.concat([df2,diff2])

del(i, data, data_lag, diff, diff2)

# original column names
cols = df.columns

# number of lagged columns to generate
n_lags = 4

# generate new column names
new_cols = []
for col in cols:
    for i in range(1, n_lags+1):
        new_col = f"{col}-{i}"
        new_cols.append(new_col)

# print new column names
df2.columns = new_cols

del (i, col, cols, n_lags, new_col)
#-------------------------------------------------------------------------------

# Add Weekly Returns to DF2 ----------------------------------------------------
df2 = df2[:-1] # Drop last row due to differencing
weekly_returns_cat = weekly_returns_cat[1:] # Return only second column
df2['Return'] = weekly_returns_cat.values # Add returns to main data set
#-------------------------------------------------------------------------------
```

```
# Drop Some Columns ------------------------------------------------------------
# The weekly MBS data won't work for doing this kind of differencing.
# So we just drop a couple of them and keep one of the columns that captures
# the weekly change

df2 = df2.drop([new_cols[-1],new_cols[-2],new_cols[-4]], axis=1)
df2 = df2.rename(columns = {"MBS-2":"MBS-1"})
#-------------------------------------------------------------------------------

# ==============================================================================
# Descriptive Statistics
# ==============================================================================

# Summary statistics
summary1 = df.describe()
summary2 = df2.describe()

# Correlation matrix heatmap
corr_matrix = df.corr()
sns.heatmap(corr_matrix, cmap='coolwarm')

corr_matrix2 = df2.corr()
sns.heatmap(corr_matrix2, cmap='coolwarm', annot=False, linewidths=0.5)

# ==============================================================================
# Sampling
# ==============================================================================

# Drop Observations at random
# df2 = df2.sample(frac=0.6, random_state=42)

# # Trim Observations down
years_2 = 52*2
df2 = df2.iloc[-years_2:,:]

# ==============================================================================
# Create Train and Test Sets
# ==============================================================================
```

```
print("Starting: Create Train and Test Sets")

# Split into Training and Test Sets ------------------------------------------
X = df2.drop('Return', axis=1)
y = df2['Return']

# Split in Order
n_test = int(0.50 * len(X))
split_index = len(X) - n_test
X_train, X_test = X[:split_index], X[split_index:]
y_train, y_test = y[:split_index], y[split_index:]

# Split Randomly
# X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.35, random_state=42)

#-------------------------------------------------------------------------------

# Scale Data -----------------------------------------------------------------
sc = StandardScaler()
X_train_scaled = sc.fit_transform(X_train)
X_test_scaled  = sc.transform(X_test)
#-------------------------------------------------------------------------------

print("Finished: Create Train and Test Sets")

# ==============================================================================
# Create and Fit Models
# ==============================================================================

print("Starting: Create and Fit Models")

# Store Models ---------------------------------------------------------------
Fitted_Models = {}
#-------------------------------------------------------------------------------



# Notes ----------------------------------------------------------------------
```

```
# CV Folds
# - Using a higher number of these dosen't seem to improve my results.
#--------------------------------------------------------------------------------



# Parameters ------------------------------------------------------------------
n_cv = 3

# C_grid = [10**10,10**-4, 10**-3,10**-2,10**-1,10**0,10**1,10**2,10**3, 10**4]
C_grid = [0.5] # Gives the same results as 10**4
# C_grid = [10**-4]
param_grid = {'C': C_grid}
#--------------------------------------------------------------------------------



# Logistic Regression ---------------------------------------------------------
print("Logistic Regression")

model = GridSearchCV(LogisticRegression(penalty=None), param_grid=param_grid,cv = n_cv)
model.fit(X_train_scaled, y_train)

Fitted_Models["Logistic"] = model

# y_pred = model.predict(X_test_scaled)
# confu_mat = confusion_matrix(y_test, y_pred)
# model_acc = accuracy_score(y_test, y_pred)
# print(confu_mat)
# print(model_acc)
#--------------------------------------------------------------------------------

# Ridge Regression ------------------------------------------------------------
print("Ridge Regression")
model = GridSearchCV(estimator=LogisticRegression(penalty='l2',solver='saga'),
param_grid=param_grid, cv=2)

model.fit(X_train_scaled,y_train)
```

```
Fitted_Models["Ridge"] = model

#-------------------------------------------------------------------------------


# Lasso Regression ------------------------------------------------------------

print("Lasso Regression")


model = GridSearchCV(estimator=LogisticRegression(penalty='l1', solver='saga'),
param_grid=param_grid, cv=n_cv)

model.fit(X_train_scaled,y_train)


Fitted_Models["Lasso"] = model

#-------------------------------------------------------------------------------




# Elastic Net -----------------------------------------------------------------

print("Elastic Net")


param_grid = {'C': C_grid, 'l1_ratio':[0.90]}


model = GridSearchCV(estimator=LogisticRegression(penalty='elasticnet', solver='saga'),
param_grid=param_grid, cv=n_cv)

model.fit(X_train_scaled,y_train)


Fitted_Models["Elastic Net"] = model

#-------------------------------------------------------------------------------


# KNN -------------------------------------------------------------------------

print("KNN")


# param_grid = {'n_neighbors': list(range(2,25))}

param_grid = {'n_neighbors': [9]}


model = GridSearchCV(estimator=KNeighborsClassifier(), param_grid=param_grid, cv=n_cv, n_jobs=-1)

model.fit(X_train_scaled, y_train)


Fitted_Models["KNN"] = model

#-------------------------------------------------------------------------------
```

```
# Random Forest ------------------------------------------------------------

print("Random Forest")


# param_grid = {'max_depth': [2,3,4], 'n_estimators':[100,200,300], "min_samples_split": [1,2,3],
"min_samples_leaf": [1,2,3]}

# param_grid = {'max_depth': [2,3,4,5,6], 'n_estimators':[100,200,300,400,500],
"min_samples_split": [1,2,3,4,5], "min_samples_leaf": [1,2,3,4,5]}

# param_grid = {'max_depth': [2], 'n_estimators':[300], "min_samples_split": [3],
"min_samples_leaf": [2]}


param_grid = {'max_depth': [2],

              'n_estimators':[10],

              "min_samples_split": [3],

              "min_samples_leaf": [3]

}


model = GridSearchCV(estimator=RandomForestRegressor(max_features='sqrt',random_state=1, n_jobs=-
1),param_grid=param_grid, cv=n_cv)

model.fit(X_train_scaled, y_train)


Fitted_Models["Random Forest"] = model

#------------------------------------------------------------------------------


# Gradient Boosting ---------------------------------------------------------

print("Gradient Boosting")


# param_grid = {"n_estimators": [100,200,300], 'max_depth': [2,3,4], 'learning_rate':
[0.01,0.001], "subsample": [0.10, 0.25, 0.5]}

# param_grid = {"n_estimators": [300,400,500,600], 'max_depth': [2,3,4], 'learning_rate': [0.5,
0.25, 0.1, 0.01], "subsample": [0.10]}

# param_grid = {"n_estimators": [800,1000], 'max_depth': [2], 'learning_rate': [0.01],
"subsample": [0.10,0.25,0.5]}

# param_grid = {"n_estimators": [300], 'max_depth': [2], 'learning_rate': [0.001], "subsample":
[0.10]}

# param_grid = {'learning_rate': [0.01,0.05,0.1],'n_estimators': [100],'max_depth':
[2],'min_samples_split': [2,5,10],'min_samples_leaf': [1,2,4],'max_features':
['sqrt','log2',None]}

# param_grid = {'learning_rate': [0.01,0.05,0.1],'n_estimators': [100],'max_depth':
[2],'min_samples_split': [2,5,10],'min_samples_leaf': [1,2,4],'max_features':
['sqrt','log2',None]}


param_grid = {'learning_rate': [0.05],

              'n_estimators': [100],

              'max_depth': [2],
```

```
                'min_samples_split': [10],
                'min_samples_leaf': [4],
                'max_features': ['sqrt'],
                'subsample':[0.5]
}

model = GridSearchCV(estimator=GradientBoostingRegressor(subsample=0.5,
random_state=1),param_grid=param_grid, cv=n_cv)
model.fit(X_train_scaled, y_train)

Fitted_Models["G-Boosting"] = model
#------------------------------------------------------------------------------

# XGBoost ---------------------------------------------------------------------
print("XGBoost")

# param_grid = {"n_estimators": [100,200,300], 'max_depth': [2,3,4], 'learning_rate':
[0.01,0.001], "subsample": [0.25, 0.5, 0.75]}
# param_grid = {"n_estimators": [100], 'max_depth': [2], 'learning_rate': [0.001], "subsample":
[0.5]}

param_grid = {
    'learning_rate': [0.0085],
    'max_depth': [2],
    'n_estimators': [100],
    'min_child_weight': [1],
    'subsample': [0.5],
}


model = GridSearchCV(estimator=xgb.XGBRegressor(subsample=0.5,
random_state=1),param_grid=param_grid, cv=n_cv)
model.fit(X_train_scaled, y_train)

Fitted_Models["XGB"] = model
#------------------------------------------------------------------------------

# Decision Tree---------------------------------------------------------------
print("Decision Tree")
```

```
# param_grid = {'max_depth': [1,2,3],  "min_samples_split": [1,2,3], "min_samples_leaf": [1,2,3]}

param_grid = {'max_depth': [2],  "min_samples_split": [1], "min_samples_leaf": [3]}


model = GridSearchCV(estimator=DecisionTreeRegressor(random_state=1),param_grid=param_grid,
cv=n_cv)

model.fit(X_train_scaled, y_train)


Fitted_Models["Decision Tree"] = model

#------------------------------------------------------------------------------


# Principal Component Regression ------------------------------------------------

print("PCR")


pipe = Pipeline([("pca", PCA()),

                 ("log", LogisticRegression(penalty=None))])

param_grid = [{"pca__n_components" :
range(1,X_train_scaled.shape[1],int(np.ceil(X_train_scaled.shape[1]/10)))}]


pcr_cv = GridSearchCV(pipe, param_grid=param_grid, cv=5)

pcr_cv.fit(X_train_scaled, y_train)


Fitted_Models["PCR"] = model

#------------------------------------------------------------------------------


# Support Vector Regression -----------------------------------------------------

print("SVR")


# param_grid = {"C": C_grid, "kernel":["linear"]}

# param_grid = {

#     'C': [0.0001, 0.1, 1, 10],  # Penalty parameter C of the error term

#     'kernel': ['linear', 'poly', 'rbf', 'sigmoid'],  # Kernel type to be used in the algorithm

#     'degree': [2, 3, 4],  # Degree of the polynomial kernel function ('poly' only)

# }


param_grid = {"C":[0.1], "kernel":["linear"]}


model = GridSearchCV(estimator=SVC(),param_grid=param_grid, cv=n_cv)

model.fit(X_train_scaled, y_train)
```

```
Fitted_Models["SVR"] = model
#-------------------------------------------------------------------------------

print("Finished: Create and Fit Models")

# ================================================================================
# Make Predictions
# ================================================================================
print("Starting: Make Predictions and Organize Results")

pred_list = {}

# Checks if Values from Predictions are Binary, converts to Binary if not
for key, value in Fitted_Models.items():
    print(key)

    temp = value.predict(X_test_scaled)

    if not all(isinstance(x, int) for x in temp):
        threshold = 0.5
        temp2 = (temp  > threshold).astype(int)
        print(temp2)
        pred_list[key] = temp2
    else:
        pred_list[key] = temp

print("Finished: Make Predictions and Organize Results")

# ================================================================================
# # Voting
# ================================================================================
# Now that I have all my predictions, I'd like to create a model where majoritiy
# rules amoung the models.

print("Voting")

# Combine All Predictions into one DataFrame
poll = pd.DataFrame()
```

```
for value in pred_list.values():
    poll = pd.concat([poll,pd.Series(value)], axis=1)

# Majority Rules: 1 if most models decide 1 and zero otheriwse. Tiebreaker is just 1
# - Weren't very many ties however.
vote = pd.Series([1 if row.sum() >= (len(row) - row.sum()) else 0 for _, row in poll.iterrows()])

# Add to our list of predictions
pred_list["Vote"] = vote

# ============================================================================
# Make Scores
# ============================================================================
scores_list = {}

# Applys predict function to each of my keys
for key, value in pred_list.items():
    y_pred = value

    confu_mat = confusion_matrix(y_test, y_pred)
    model_acc = accuracy_score(y_test, y_pred)

    scores_list[key] = confu_mat, model_acc

# ============================================================================
# Vizualize Confusion Matricies
# ============================================================================
# Define the class labels
class_names = ['Negative', 'Positive']

# Create the plot with the appropriate number of rows and columns
fig, axs = plt.subplots(nrows=4, ncols=3, figsize=(15, 25))

# Loop through each confusion matrix and create a plot
for i, key in enumerate(scores_list):
    # Set Plot to proper index
    row = i // 3
    col = i % 3
```

```
    matrix = scores_list[key][0]
    axs[row, col].matshow(matrix, cmap=plt.cm.Blues)

    # Add proper text values
    for r in range(matrix.shape[0]):
        for c in range(matrix.shape[1]):
            axs[row, col].text(c, r, str(matrix[r, c]), va='center', ha='center')

    # Some additional labeling
    axs[row, col].set_xticklabels([''] + class_names)
    axs[row, col].set_yticklabels([''] + class_names)
    axs[row, col].set_xlabel('Predicted Label')
    axs[row, col].set_ylabel('True Label')
    axs[row, col].set_title(key)

# Hide unused subplots
for i in range(4 * 3 - len(scores_list)):
    axs[3, 2 - i].axis('off')

# Adjust the spacing between subplots
plt.subplots_adjust(hspace=0.35, wspace=0.35)

# Show the plot
plt.show()

# ============================================================================
# Create Rolling Week Benchmark
# ============================================================================
# Need to shift one week up
y_test_shifted = y_test.shift(-1)

# Take a rolling average of 3 weeks
y_pred_rolling_3 = y_test_shifted.rolling(window=3).mean()

# Create a new series with values of 1 or 0 based on the rolling average
y_pred_binary_3 = y_pred_rolling_3.apply(lambda x: 1 if x > 0.5 else 0)

# Replace any missing values with 0
```

```
y_pred_binary_3 = y_pred_binary_3.fillna(0)

# Drop Last Observation
y_pred_binary_3[:-1]

# Accuracy Score
benchmark_rolling_3 = accuracy_score(y_test, y_pred_binary_3)

# ============================================================================
# Vizualize Accuracy with Barplots
# ============================================================================
# Create a figure and axis object
fig, ax = plt.subplots()

# Create a bar plot
bar_width = 0.5
score = [value[1] for value in scores_list.values()]

ax.bar(scores_list.keys(), score, width=bar_width)

for i, (key, value) in enumerate(scores_list.items()):
    # Add value on top of each bar (rounded)
    ax.text(key, value[1] + 0.1, f"{round(value[1]*100, 2)}", ha='center', fontsize=9)

# Set the title and axis labels
ax.set_xlabel('Models')
ax.set_ylabel('Score')

# Add space between bars
ax.set_xticks([i + bar_width/2 for i in range(len(scores_list))])

# Tilt the x-axis labels
ax.set_xticklabels(scores_list.keys(), rotation=45, ha='right')

# Scale the y-axis to be values from 0 to 1
ax.set_ylim([0, 1])

# Set the y-axis ticks to be displayed in increments of 0.1
```

```
ax.set_yticks([i/10 for i in range(11)])

# Add dashed lines for benchmarks
benchmark = y_test.value_counts()[0]/y_test.value_counts().sum()
label = f"Mean: {benchmark:.4f}"
ax.axhline(y=benchmark, color='gray', linestyle='--',label=label)
label2 = f"Rolling-3WK: {benchmark_rolling_3:.4f}"
ax.axhline(y=benchmark_rolling_3, color='gray', linestyle='--',label=label2)
ax.legend()

# Show the plot
plt.show()

# ============================================================================
# Store Best Parameters in a List
# ============================================================================
pd.Series(y_pred).value_counts()
y_test.value_counts()

best_params = {}

for key, values in Fitted_Models.items():
    print(values.best_estimator_)
    best_params[key] = str(values.best_estimator_)

# ============================================================================
# End of Script
# ============================================================================
```