Final Project FIN 4000

Paul Pawelec

4/18/2022

Contents

Introduction

The intention of this project is to forecast bitcoin's USD price using daily time series data from financial activities and assets. These financial activities focus on mining, the number of accounts, and various free market supply measurements related to bitcoin. I'm more interested in seeing if the activities around bitcoin can return and predict the price of bitcoin for the next day. Additionally, I add in a market index for the S&P500 and US gold prices to capture effects of markets outside of bitcoin and the blockchain. In earlier years, the correlation between other markets and bitcoin were relatively low and negative, making it a great asset to diversify with, but recently there seems to be some positive correlation growing between these assets, so I thought it would be interesting to try and capture this relationship in our models.

Bitcoin is part of a movement of decentralizing financial markets using blockchain technologies. The biggest differences between bitcoin and more traditional investments is the lack of market regulations. There's nothing stopping bitcoin from crashing to zero, which makes it a very volatile, speculative, and interesting asset to follow. This volatility issue is what I hope to fix with machine learning models, which are able to be much more flexible than traditional benchmarks. I do end up being able to forecast this volatility, but all my models are excessively over/under fitting due to complexity issues.

Initialization

Below are packages needed to run this rmd.

```
#================================================================================
# Packages
#================================================================================
# You will need the below packages to run each chunk.

# List of packages
Packages = c("tidyverse" # for data science tools
             , "openxlsx" # for using xlsx files
             , "zoo" # For more data science tools.
             , "lubridate" # For time formatting
             , "ggplot2" # for plotting
             , "dplyr" # data manipulation tools
             , "glmnet" # ridge and lasso regression models
             , "randomForest" # random forecast models
             , "gbm" # gradient tree boosting models
             , "knitr" # for tables and formatting with pdfs
             , "kableExtra" # Additional table formatting
             , "pls" # pcr and pls regressions
             , "forecast" # for benchmark forecasts
             , "ggpubr" # combine ggplots into one
)

# This will load and install the packages if required
Packages.Check = lapply(
  Packages,
  function(x) {
    if (!require(x, character.only = TRUE)) {
        install.packages(x, dependicies = TRUE)
        library(x, character.only = TRUE)
    }
  }
)
```

```
# Clean up
rm(Packages, Packages.Check)
```

Data

All my data on bitcoin comes from coinmetrics. The original dataset I use contains over 100 variables. A link is provided below if you wish to view and download the data set yourself in its entirety, and additionally there's a link for variable descriptions on each column. Later on in the report, table 2 will show all the predictors and their descriptions used in this report.

On top of the bitcoin data, I grabbed data from yahoo finance for an S&P 500 index, and I took gold prices in USD from the German Federal Bank for the London exchange (it was free and easy to obtain). Since gold prices are relatively the same worldwide it should serve as a reasonably accurate representation of US gold prices from any US institution.

I use the S&P 500 index and US gold prices to get some reflection of the US stock and gold markets which are reasonable investment alternatives for financial investments against or with bitcoin that could explain market changes. For instance, a drop in US stock with a rise in gold could indicate lower market confidence which could also mean higher prices in bitcoin if people are looking to diversify or hedge against losses.

Links to where I downloaded the data can be found below.

- BTC data
  - *https://coinmetrics.io/community-network-data/*
- Gold Prices
  - *https://www.bundesbank.de/en/statistics/time-series-databases/data-basket/748932!submitdataBasket?mode=its&databasketActionDownload=Go+to+download&its_from=2009-01&month=3&year=2022&its_to=2022-04&month=3&year=2022&its_fileFormat=csv&its_csvFormat=en&its_currency=default&its_dateFormat=default&tsId=BBEX3.D.XAU.USD.EA.AC.C05&tsIdToAdd=*
- SPX
  - *https://ca.finance.yahoo.com/quote/%5EGSPC/history?period1=1230768000&period2=1649548800&interval=1d&filter=history&frequency=1d&includeAdjustedClose=true*

For additional information on the coinmetrics data, the below link can be referenced. Later on in the report we will cover the variables used from the coinmetrics dataset.

- https://docs.coinmetrics.io/info/metrics

Data Import

This chunk uses a custom function I wrote to handle loading in multiple files into one variable. It's kind of unnecessary for this project given I'm only importing three files, but should be useful if I wanted to add in more files in the future.

```
#=============================================================================
# Data Import
#=============================================================================
# This section deals with importing our csv files into our environment. To replicate
```

```
# make sure to include these csv files in the same directory as the rmd.

Excel_Import = function(FileDir = FileDir) {
  # This function takes our FileDir list and uses it to import all our csv's

  # Variables

  # Creating a blank list to store our imports
  List = vector(mode = "list", length = length(FileDir))

  # For Loop - Iterates over FileDir and pulls each file from my directory.
  for (i in 1:length(FileDir)) {
    List[[i]] = as.data.frame(read.csv(file = FileDir[[i]]))
  }

  # End of Function
  return(List)

}

# These are the names of the files I import.
FileDir = list(
  btc = "btc.csv",
  gold = "gold.csv",
  spx = "spx.csv"

)

# Call my Excel_Import function and list every file we need.
FileImports = Excel_Import(FileDir)
names(FileImports) = names(FileDir)

# Clean UP
rm(Excel_Import, FileDir)
```

Data Cleaning

The bitcoin, S&P 500, and US gold datasets will now be referred to as; btc, spx, and gold.

The 'Data Cleaning' chunk is responsible for cleaning all our datasets. For our gold data, I replace all our dates with a differently formatted set of dates to match the original btc data set. I then change the column names, and replace any NA values with interpolation. Rather than replace with a zero or a mean average of the column, interpolation allows us to continue any trend from a point before the NA to a point afterwards. This essentially smooths out our data a little bit and possibly lowers variance. It also matches and acts more like our btc data since the decentralized market of bitcoin is 24/7 and isn't missing any dates.

For our spx data, I similarly reformat our dates to match the btc dataset and I replace our NA values with interpolation. However, I also have to remove commas from our prices and reformat them as numeric.

Finally, for our btc dataset, I apply a date format to our time and set the rest of our columns to numeric.

```
#==============================================================================
# Data Cleaning
#==============================================================================
```

```
#    - This section deals with cleaning each data set and filling in missing
#   values.


### Split up imported files into different variables.
btc  = FileImports[[1]]
gold = FileImports[[2]]
spx  = FileImports[[3]]


### Clean Up Gold

# Specific to replace dates for gold
time = data.frame(time=seq(as.Date("2009-01-02"), as.Date("2022-04-7"), by="days"))
time[,1] = time[,1] %>% ymd() %>% data.frame()

colnames(gold) = c("time", "price_per_ounce") # Change column names
gold[grepl("^\\.", gold[,2]),] = NA # replace "." observations with NA
gold[,2] = na.approx(gold[,2]) # Replace NA with interpolation
gold[,1] = time[,1] # replace date with time


### Clean up SPX

# Specific to merge into spx dates
time = data.frame(time=seq(as.Date("2009-01-02"), as.Date("2022-04-8"), by="days"))
time[,1] = time[,1] %>% ymd() %>% data.frame()

spx = select(spx, Date, Adj.Close..) # I just want the adjusted Close

spx[,2] = gsub("\\,", "", spx[,2], perl = TRUE) # remove commas
spx[,2] = spx[,2] %>% as.numeric # convert to numeric

spx[,1] = spx[,1] %>% dmy() # convert spx date column to proper date format
spx = spx %>% arrange(Date)
spx  = merge(spx, time, by.x = "Date", by.y = "time", all.x = T, all.y = T)
spx[,2] = na.approx(spx[,2]) # Replace NA with interpolation


### Clean up btc
btc[,1] = ymd(btc[,1]) # Fix date format

# set all columns to numeric
btc[,2:ncol(btc)] = apply(btc[,2:ncol(btc)], 2, function(x) as.numeric(x))
```

Creating Our Dataset 'CryptoMetrics'

This chunk has us merging our three datasets and removing various NA rows, NA columns, and has further selection and removal of columns based on later findings.

Most NA columns are removed because the interpolation would be excessive for the amount of missing data, and replacing with a mean or zero value would skew our results too much. Any NA rows that are removed are due to NAs being present at the start or end of the dataset due to lagging or bitcoin not being priced

yet. Since bitcoin wasn't price till about 2010, any training of the dataset prior would mislead our model since the only possible result would be a USD price of zero.

Various columns are removed from the start due to them being too correlated or irrelevant to modeling bitcoin's USD price. For instance, the USD value of bitcoin accounts would just be a reflection of the price of bitcoin rather than a predictor of activity in the market that could reflect its price. So, any columns referencing the account balances or the supply account balances are removed. I do keep a couple of the supply account predictors since they indicate the amount of bitcoin in the free market, which indicates bitcoin's liquidity.

After running our linear regressions, I found various columns would not return results, or were incredibly insignificant. These columns tend to have extremely large values and variations. So, it's possible they were too difficult for the model to understand. I also tried scaling these values down and they still returned as insignificant.

Additionally, since I'd like to forecast the next days price, I lag the entire dataset including the USD price by one day against the USD price.

```
#================================================================================
# Create CryptoMetrics dataset
#================================================================================
#   - Combine, format columns, remove columns, organize, and lag everything.

# These merges are for combining all our cleaned data together into one dataset.

# Merge cleaned btc and spx data together (Full Outer Join)
CryptoMetrics = merge(
  btc,
  spx,
  by.x = "time",
  by.y = "Date",
  all.x = TRUE,
  all.y = TRUE
)

# Merge previously merged data set with cleaned gold data set (Full Outer Join)
CryptoMetrics = merge(
  CryptoMetrics,
  gold,
  by.x = "time",
  by.y = "time",
  all.x = TRUE,
  all.y = TRUE
)


### Remove unnecessary columns
#   - I just want predictors that indicate use of bitcoin, rather then just value.
#   - So I'm going to remove variables that just indicate the total balance
#   and value of bitcoin for accounts.

SplyAct1d = CryptoMetrics[,"SplyAct1d"] # Keep this one since it measures number
# of accounts

# Remove account balances
CryptoMetrics = CryptoMetrics[,!grepl("AdrBal", colnames(CryptoMetrics))]
```

```r
# Remove Supply account balances
CryptoMetrics = CryptoMetrics[,!grepl("SplyAct", colnames(CryptoMetrics))]

# Remove Supply address balances
CryptoMetrics = CryptoMetrics[,!grepl("SplyAdrBal", colnames(CryptoMetrics))]

# Put back in number of accounts with supply
CryptoMetrics$SplyAct1d = SplyAct1d
rm(SplyAct1d)

# Removing any additional unnecessary columns, the reasons are found later in
# in the report, but essentially many of these columns failed to return results
# when used in various models.

CryptoMetrics = CryptoMetrics %>%
  select(
    -PriceBTC,
    -ReferenceRateETH,
    -ReferenceRateEUR,
    -ReferenceRateUSD,
    -ReferenceRate,
    -VtyDayRet180d,
    -VtyDayRet30d,
    -ROI1yr,
    -ROI30d,
    -NVTAdjFF90,
    -NVTAdj90,
    -AssetEODCompletionTime,
    -IssContPctDay,
    -IssTotNtv,
    -IssTotUSD,
    -RevHashRateNtv,
    -RevHashUSD,
    -TxCntSec
)


### Remove Unnecessary Rows

# Bitcoin wasn't priced till a couple years later, so i cut everything before that.
CryptoMetrics = CryptoMetrics[CryptoMetrics$time >= as.Date("2010/07/19"),]

# Lag everything so we can predict next days price.
Lag = rbind(NA, head(CryptoMetrics, -1))
colnames(Lag)[which(colnames(CryptoMetrics)=="PriceUSD")] = "PriceUSD_Lag1"
CryptoMetrics = cbind(CryptoMetrics$PriceUSD, Lag)
colnames(CryptoMetrics)[1] = "PriceUSD"

rm(Lag)

# Remove first Row and last two rows with NA values
CryptoMetrics = CryptoMetrics[-c(1, nrow(CryptoMetrics)-1,nrow(CryptoMetrics)),]
```

```
# Reorder columns
CryptoMetrics = CryptoMetrics[,order(colnames(CryptoMetrics))]

# clean up
rm(btc, gold, spx, time, FileImports, time)
```

Creating Correlations

For these predictors, correlations are relatively high; 34 variables are above 0.5 and only one variable below -0.5. I then subset the dataset with only these correlated predictors to try and reduce the complexity of the model. These correlations can be seen in table 1.

```
#=============================================================================
# Creating Correlations
#=============================================================================

# Grab correlations for all our variables
mcor = data.frame(cor(CryptoMetrics %>% select(-time)))

# formato our results
PriceUSDcor = data.frame(mcor["PriceUSD"])
rownames(PriceUSDcor) = rownames(mcor)

# Subset our data for correlations greater than 0.5 or less than -0.5
CryptoMetrics = CryptoMetrics %>%
  select(which(PriceUSDcor > 0.5 | PriceUSDcor < -0.5), time)

# This is how the next table was made, code was embded in the text below using `r ... `.

# kable(PriceUSDcor, "latex", booktabs = TRUE, longtable = TRUE) %>%
#   kable_styling(latex_options =  c("striped", "repeat_header"), font_size = 8)
```

Table 1
Correlations between PriceUSD and other predictors

	mcor....PriceUSD..
Adj.Close..	0.8547473
AdrActCnt	0.6698798
BlkCnt	-0.1705208
BlkSizeMeanByte	0.6012078
BlkWghtMean	0.4598693
BlkWghtTot	0.4441458
CapAct1yrUSD	0.9962147
CapMrktCurUSD	0.9988619
CapMrktFFUSD	0.9990208
CapMVRVCur	0.0611649
CapMVRVFF	-0.0187441
CapRealUSD	0.9548718
DiffLast	0.8518062
DiffMean	0.8517656
FeeByteMeanNtv	-0.1316897
FeeMeanNtv	-0.1139788
FeeMeanUSD	0.5308859
FeeMedNtv	-0.1299120

(continued)

	mcor....PriceUSD..
FeeMedUSD	0.4510677
FeeTotNtv	0.0717127
FeeTotUSD	0.5012368
FlowInExNtv	-0.0367879
FlowInExUSD	0.8589769
FlowOutExNtv	-0.0142702
FlowOutExUSD	0.8507233
FlowTfrFromExCnt	0.4778482
HashRate	0.8515930
HashRate30d	0.8526440
IssContNtv	-0.4913914
IssContPctAnn	-0.3304362
IssContUSD	0.9597880
NDF	-0.3825339
NVTAdj	0.1657595
NVTAdjFF	0.0525484
price_per_ounce	0.5719857
PriceUSD	1.0000000
PriceUSD_Lag1	0.9990592
RevAllTimeUSD	0.8935547
RevHashNtv	-0.0717922
RevHashRateUSD	-0.1154265
RevNtv	-0.4941069
RevUSD	0.9537284
SER	0.6440392
SplyAct1d	0.0973749
SplyAdrTop100	0.2282713
SplyAdrTop10Pct	0.5012235
SplyAdrTop1Pct	0.5442154
SplyCur	0.5310505
SplyExpFut10yr	0.5396882
SplyFF	0.4308313
SplyMiner0HopAllNtv	-0.4416833
SplyMiner0HopAllUSD	0.9990461
SplyMiner1HopAllNtv	-0.5711475
SplyMiner1HopAllUSD	0.9983483
TxCnt	0.4594855
TxTfrCnt	0.5924202
TxTfrValAdjNtv	-0.0031846
TxTfrValAdjUSD	0.9066718
TxTfrValMeanNtv	-0.1489933
TxTfrValMeanUSD	0.6888475
TxTfrValMedNtv	-0.0743131
TxTfrValMedUSD	0.7265165
VelCur1yr	-0.3423244

Creating Data Descriptions

To create a table with all our predictors, I import a table with all possible descriptions from the coinmetrics site, mask over them with our remaining predictors, and then add in the predictors I created from the spx and gold datasets. Table 2 will list the results of these steps.

```
#============================================================================
# Creating Data Descriptions for our predictors
```

```r
#==============================================================================

# This data.frame contains a description of all possible variables used on
# the coin metrics site for any cryptocurrency.
Asset_Metrics = read.xlsx("Asset Metrics.xlsx")


# Using Asset_Metrics, I mask on top all the current predictors in our dataset
# in the CryptoMetrics dataset.
Asset_Metrics = Asset_Metrics[grepl(
  paste0(colnames(CryptoMetrics), collapse = "|"), Asset_Metrics$ID),]


# These are additional predictors that I created for spx and gold
Extra_Predictors = data.frame(
  ID = c("Adj.Close..", "price_per_ounce"),
  Name = c("Adjusted close price for S&P 500 index", "Gold per ounce price in US dollars"),
  Category = c("Financial Asset", "Financial Asset"),
  Subcategory = c("Market Index", "Commodity Asset Price"),
  Frequencies = c("1d*", "1d*")
)


# I then combine all the descriptions together.
Asset_Metrics = rbind(Asset_Metrics, Extra_Predictors)
rownames(Asset_Metrics) = NULL

# clean up
rm(Extra_Predictors)

# This is how the next table was made, code was embded in the text below using `r ... `.

# kable(Asset_Metrics, "latex", booktabs = TRUE, longtable = TRUE) %>%
#   kable_styling(latex_options =  c("striped", "repeat_header"), font_size = 8) %>%
#   column_spec(2, width = "3.5cm") %>%
#   column_spec(3, width = "4cm") %>%
#   column_spec(4, width = "2cm") %>%
#   column_spec(5, width = "2cm") %>%
#   column_spec(6, width = "1cm") %>%
#   footnote(symbol = c("These are daily but with the added interpolation of values that were missing f
```

Table 2
Predictor Descriptions

ID	Name	Category	Subcategory	Frequencies
AdrActCnt	Addresses, active, count	Addresses	Active	1d
BlkSizeMeanByte	Block, size, mean, bytes	Network Usage	Blocks	1d
CapAct1yrUSD	Capitalization, active supply, 1yr, USD	Market	Market Capitalization	1d
CapMrktCurUSD	Capitalization, market, current supply, USD	Market	Market Capitalization	1d
CapMrktFFUSD	Capitalization, market, free float, USD	Market	Market Capitalization	1d
CapRealUSD	Capitalization, realized, USD	Market	Market Capitalization	1d
DiffLast	Difficulty, last	Mining	Difficulty	1d
DiffMean	Difficulty, mean	Mining	Difficulty	1d
FeeMeanUSD	Fees, transaction, mean, USD	Fees and Revenue	Fees	1d
FeeTotUSD	Fees, total, USD	Fees and Revenue	Fees	1d
FlowInExUSD	Flow, in, to exchanges, USD	Exchange	Deposits	1d
FlowOutExUSD	Flow, out, from exchanges, USD	Exchange	Withdrawals	1d
HashRate	Hash rate, mean	Mining	Hash Rate	1d
HashRate30d	Hash rate, mean, 30d	Mining	Hash Rate	1d
HashRate30dOtherHardware	Hash rate, mean, 30d, other hardware	Mining	Hardware Hash Rate	1d
HashRate30dOtherHardwarePct	Hash rate, 30d, other hardware, percent	Mining	Hardware Hash Rate	1d
HashRate30dS7	Hash rate, mean, 30d, S7 line	Mining	Hardware Hash Rate	1d
HashRate30dS7Pct	Hash rate, 30d, S7 line, percent	Mining	Hardware Hash Rate	1d
HashRate30dS9	Hash rate, mean, 30d, S9 line	Mining	Hardware Hash Rate	1d
HashRate30dS9Pct	Hash rate, 30d, S9 line, percent	Mining	Hardware Hash Rate	1d
IssContUSD	Issuance, continuous, USD	Supply	Issuance	1d
PriceUSD	Price, USD	Market	Price	1d
RevAllTimeUSD	Miner revenue, all time, USD	Fees and Revenue	Revenue	1d
RevHashRateNtv	Revenue, daily, per hash unit per second, native units	Mining	Hash Rate	1d
RevHashRateUSD	Revenue, daily, per hash unit per second, USD	Mining	Hash Rate	1d
RevUSD	Miner revenue, USD	Fees and Revenue	Revenue	1d
SER	Supply equality ratio	Supply	Current	1d
SplyAdrTop10Pct	Supply, held by, top 10% addresses	Supply	Current	1d
SplyAdrTop1Pct	Supply, held by, top 1% addresses	Supply	Current	1d
SplyCur	Supply, current	Supply	Current	1d
SplyExpFut10yr	Supply, future expected, next 10yr	Supply	Future Expected	1d
SplyExpFut10yrCMBI	Supply, future expected, next 10yr (CMBI)	Supply	Future Expected	1d
SplyMiner0HopAllUSD	Supply, Miner, held by all mining entities, USD	Mining	Balances	1d

(continued)

ID	Name	Category	Subcategory	Frequencies
SplyMiner1HopAllNtv	Supply, Miner, held by all addresses within one hop of a mining entity, native units	Mining	Balances	1d
SplyMiner1HopAllUSD	Supply, Miner, held by all addresses within one hop of a mining entity, USD	Mining	Balances	1d
TxCnt	Transactions, count	Transactions	Transactions	1d
TxCntSec	Transactions, count, per second	Transactions	Transactions	1d
TxTfrValAdjNtv	Transactions, transfers, value, adjusted, native units	Transactions	Transfer Value	1d
TxTfrValMeanNtv	Transactions, transfers, value, mean, native units	Transactions	Transfer Value	1d
TxTfrValMedNtv	Transactions, transfers, value, median, native units	Transactions	Transfer Value	1d
Adj.Close..	Adjusted close price for S&P 500 index	Financial Asset	Market Index	1d*
price_per_ounce	Gold per ounce price in US dollars	Financial Asset	Commodity Asset Price	1d*

* These are daily but with the added interpolation of values that were missing for weekends and holidays

Creating Measures for Our Predictors

This uses various techniques to quickly apply and combine a dataset of measures for mean, standard deviation, and variance for our predictors.

```
#============================================================================
# Creating Measures for our predictors
#============================================================================


# The measures we wish to apply to each return
Measures = c(mean, sd, var)


# Function to apply other functions with na.rm argument.
ApplyMeasures = function(fun, Column, na_rm = TRUE){
  # This function applys other functions onto a column

  # End of Function
  return(fun(Column,na.rm = na_rm))

}


# nested apply with sapply to return matrix with mean, standard deviation and variance
# for each column selected.
CryptoMetrics.Measures = apply(
  X = CryptoMetrics %>% select(-time),
  MARGIN = 2,
  FUN = function(x) sapply(
```

```
    X = Measures,
    FUN = function(y) ApplyMeasures(fun = y, Column = x)
  )
)

# Transpose to make it long format and rename column names.
CryptoMetrics.Measures = t(CryptoMetrics.Measures)
colnames(CryptoMetrics.Measures) = c("Mean", "Standard Deviation", "Variance")

# Clean up
rm(Measures, ApplyMeasures)


# kable(CryptoMetrics.Measures, "latex", booktabs = TRUE, longtable = TRUE)
```

Table 3
Mean, Standard Deviation, and Variance of Predictors

	Mean	Standard Deviation	Variance
Adj.Close..	2.362041e+03	9.223381e+02	8.507076e+05
AdrActCnt	4.600569e+05	3.653560e+05	1.334850e+11
BlkSizeMeanByte	6.299484e+05	4.660273e+05	2.171815e+11
CapAct1yrUSD	6.284786e+10	1.191968e+11	1.420789e+22
CapMrktCurUSD	1.409333e+11	2.699511e+11	7.287362e+22
CapMrktFFUSD	1.104686e+11	2.089939e+11	4.367843e+22
CapRealUSD	6.915385e+10	1.188080e+11	1.411535e+22
DiffLast	5.046522e+12	7.792676e+12	6.072580e+25
DiffMean	5.042918e+12	7.787991e+12	6.065280e+25
FeeMeanUSD	1.940543e+00	5.222267e+00	2.727207e+01
FeeTotUSD	5.618189e+05	1.591657e+06	2.533374e+12
FlowInExUSD	2.151144e+08	4.082023e+08	1.666291e+17
FlowOutExUSD	2.249397e+08	4.379865e+08	1.918322e+17
HashRate	3.651308e+07	5.642101e+07	3.183330e+15
HashRate30d	3.582438e+07	5.528531e+07	3.056466e+15
IssContUSD	9.167624e+06	1.360308e+07	1.850438e+14
price_per_ounce	1.442351e+03	2.400512e+02	5.762456e+04
PriceUSD	7.691133e+03	1.436123e+04	2.062449e+08
PriceUSD_Lag1	7.681271e+03	1.435200e+04	2.059800e+08
RevAllTimeUSD	7.460391e+09	1.040777e+10	1.083217e+20
RevUSD	9.729394e+06	1.452599e+07	2.110045e+14
SER	4.103090e-02	3.146260e-02	9.899000e-04
SplyAdrTop10Pct	1.378737e+07	4.679745e+06	2.190002e+13
SplyAdrTop1Pct	1.182094e+07	4.808349e+06	2.312022e+13
SplyCur	1.414383e+07	4.240072e+06	1.797821e+13
SplyExpFut10yr	1.979561e+07	7.329752e+05	5.372526e+11
SplyMiner0HopAllUSD	1.384771e+10	2.574758e+10	6.629377e+20
SplyMiner1HopAllNtv	3.084242e+06	3.059112e+05	9.358167e+10
SplyMiner1HopAllUSD	2.118462e+10	3.887122e+10	1.510972e+21
TxCnt	1.689331e+05	1.218252e+05	1.484138e+10

TxTfrValAdjNtv	2.836056e+05	1.563976e+05	2.446019e+10
TxTfrValMeanNtv	2.001476e+01	6.801560e+01	4.626122e+03
TxTfrValMedNtv	6.600681e-01	4.710589e+00	2.218965e+01

In table 3 we can see some extremely high variances which are adjusted for (normalized) in the next chunk.

Methodology

This section covers the various methods, models, and techniques I used to try and forecast bitcoin's USD price. I mostly used regression based models with some trees (randomforest and gbt) to model the relationship between the predictors and USD price.

Creating the Training and Test Sets

For the high variances found in table 3, I normalize everything down so that these predictors don't get too out of hand when used in our models. I do ignore normalizing the priceUSD, spx, and price_per_ounce predictors because I want to see those interactions without any adjustments.

I trim the first 1000 rows of my dataset because the majority of it is bitcoin when it's priced at below 100$. I pretty much want a training set that reflects bitcoin when it became much more popular to rather than when it was mostly unknown. This way we cut out a portion of our data where not much happens and we can focus on the more volatile data later on.

Additionally, I tend to keep a much larger training set at 85% of my data which gave better results overall. Training and Test MSE would increase as I increased the training set. Having the training set be too low would result in good training MSE as well, but widely varied results for my test MSE.

```
#================================================================================
# Creating the Training and Test sets.
#================================================================================

# These are the columns I don't want to normalize
ignored = CryptoMetrics %>%
  select(time, PriceUSD, PriceUSD_Lag1, price_per_ounce, Adj.Close..)

# These are columns I want to normalize
scale = CryptoMetrics %>%
  select(-time, -PriceUSD, -PriceUSD_Lag1, -price_per_ounce, -Adj.Close..)

scaled = apply(scale, 2, function(x) scale(x))

# Combine my two data sets.
CryptoMetrics = data.frame(ignored, scaled)

rm(ignored, scale, scaled)


### Remove lower significant predictors
#   - After various iteration, I opted to remove various columns that deem either
# insignificant or too similar to other predictors to be of use given the complexity
# of the model and the constant overfitting
```

```
CryptoMetrics = CryptoMetrics %>%  select(
  -CapAct1yrUSD,
  -CapMrktFFUSD,
  -CapRealUSD,
  -DiffLast,
  -DiffMean,
  -FeeMeanUSD,
  -RevAllTimeUSD,
  -SER,
  -SplyAdrTop10Pct,
  -SplyAdrTop1Pct,
  -SplyCur,
  -SplyExpFut10yr,
  -SplyMiner1HopAllNtv,
  -TxCnt,
  -TxTfrValAdjNtv,
  -TxTfrValMeanNtv,
  -TxTfrValMedNtv
)


### Seed, train, and test sets are created

# Set seed and create indexes to subset data
set.seed(777)
train = 1000:floor(0.30 * nrow(CryptoMetrics))
test = (tail(train,1)+1):nrow(CryptoMetrics)

# Create training and test sets
train.crypto = CryptoMetrics[train,]
test.crypto = CryptoMetrics[test,]

# Clean up
rm(train, test)
```

Linear Regression

Below I run a linear regression with all my predictors.

```
#=============================================================================
# Multiple Linear Regression.
#=============================================================================

# fit a model to all my remaining predictors.
fit.lm = lm(PriceUSD ~. -time, train.crypto)
```

Ridge and Lasso

Below I run various lambdas and return the lambda with the lowest MSE for each model.

```r
#=============================================================================
# Finding Optimal Shrinkage for Ridge and Lasso
#=============================================================================

# Train and Test Sets for Ridge and Lasso
train.model.x = model.matrix(PriceUSD ~. -time, data = train.crypto)
test.model.x = model.matrix(PriceUSD ~. -time, test.crypto)

# Create Lambdas
power = seq(-2,0,0.1)
Shrinkage = 10^power

# Test for best lambdas for ridge
MSE = sapply(Shrinkage, function(x){
  # Anonymous function to iterate over shrinkage values

  # Fit data
  fit = glmnet(train.model.x, train.crypto$PriceUSD, alpha = 0, lambda = x)

  # Create predictions
  pred = predict(fit, train.model.x)

  # Find MSE
  MSE = mean((pred - train.crypto$PriceUSD)**2)

  # Return MSE
  return(MSE)

})

# Set up data.frame and find lowest MSE
MSEperShrinkage_train = data.frame(MSE, Shrinkage)
best.lambda.ridge = Shrinkage[which(MSEperShrinkage_train[,2] == min(MSEperShrinkage_train[,2]))]


# Test for best lambdas for lasso
MSE = sapply(Shrinkage, function(x){
  # Anonymous function to iterate over shrinkage values

  # Fit data
  fit = glmnet(train.model.x, train.crypto$PriceUSD, alpha = 1, lambda = x)

  # Create predictions
  pred = predict(fit, train.model.x)

  # Find MSE
  MSE = mean((pred - train.crypto$PriceUSD)**2)

  # Return MSE
  return(MSE)
})

# Set up data.frame and find lowest MSE
```

```
MSEperShrinkage_train = data.frame(MSE, Shrinkage)
best.lambda.lasso = Shrinkage[which(MSEperShrinkage_train[,2] == min(MSEperShrinkage_train[,2]))]

# clean up
rm(MSE, MSEperShrinkage_train, power)
```

Below I run ridge and lasso regression with the best lambdas.

```
#=============================================================================
# Ridge and Lasso Regressions
#=============================================================================

# Ridge Regression
fit.ridge = glmnet(train.model.x, train.crypto$PriceUSD, alpha = 0, lambda = best.lambda.ridge)

# Lasso
fit.lasso = glmnet(train.model.x, train.crypto$PriceUSD, alpha = 1, lambda = best.lambda.lasso)
```

Trees

Below I try and find the lowest MSE for frees and lambda. This one takes a while to run, but generally I found lambda as 1 and trees to be 3000 to give the lowest MSE for our training set, while a shrinkage of 0.5 and trees set to 2000 gives us the lowest test MSE. I'll be using the latter results for our optimal GBT model.

```
#=============================================================================
# Finding Optimal Parameters for GBT
#=============================================================================

# time can't be a 'date object' when using gbm, it's not used in the
# model but can't be entered in this format as data

train.crypto.gbt = train.crypto
train.crypto.gbt$time = as.numeric(train.crypto$time)

# I also create a test set with this edit as well.

test.crypto.gbt = test.crypto
test.crypto.gbt$time = as.numeric(test.crypto$time)

# Various values for trees
# n.trees = c(500,1000,1500,2000,2500,3000)


# Test for best lambdas and trees for gbt
#     - To get the best results for our test data, replace 'train.crypto.gbt'
#   with 'test.crypto.gbt' for the second arguments in both he 'Pred' and 'MSE'
#   variables in the anonymous function below.
#     - The below code is commented out due to the time it takes to run making
#   it not feasible to run while knitting and editing the document.

# MSE = sapply(Shrinkage, function(x) sapply(n.trees, function(y){
```

```
#   # Nested anonymous function to iterate over lambda and number of trees.
#
#   # Fit our gbt tree model
#
#   fit = gbm(
#     PriceUSD ~. - time,
#     data = train.crypto.gbt,
#     distribution = "gaussian",
#     n.trees = y,
#     shrinkage = x,
#   )
#
#   # Fit results
#   pred = predict(fit, train.crypto.gbt)
#
#   # Calculate MSE
#   MSE = mean((pred - train.crypto.gbt$PriceUSD)**2)
#
#   # Return MSE
#   return(MSE)
# }))

# To find the lowest MSE, I would just run min() over the entire matrix the
# code would return, and then manually figure out which shrinkage and trees
# are optimal.
```

Below I run randomforest with the square root of our number of predictors and importance set to true (this assesses the importance of predictors for us) to lower our MSE. Furthermore, for our gradient boosted trees model, I base the parameters on the above results.

```
#==============================================================================
# Random Forest and Gradient Boosted Trees
#==============================================================================

# I just use the default settings for random forest.
fit.randfore = randomForest(
  PriceUSD ~. -time
  , data = train.crypto,
  mtry = sqrt(ncol(train.crypto)),
  importance = TRUE
)


# I use the best MSE results from before
fit.gbt = gbm(
  PriceUSD ~. - time,
  data = train.crypto.gbt,
  distribution = "gaussian",
  n.trees = 2000,
  shrinkage = 0.5,
)
```

PCR and PLS

Below I run the pls and pcr regression. I don't want to use the default scale argument since my data is already scaled to my preferences. I also don't use cross validation (or any other method) because I don't want to reduce the effects of any trends or patterns in my time series data.

```
#==============================================================================
# PCR and PLS Regressions
#==============================================================================

# PCR
fit.pcr = pcr(PriceUSD ~. -time, data = train.crypto, scale = FALSE, validation = "none")

# PLS
fit.pls = plsr(PriceUSD~. -time, data = train.crypto, scale = FALSE, validation = "none")
```

Results

Below I calculate the predicted values for our predicted values, residual values, MSE, and R^2 for our training and test data respectively.

```
#==============================================================================
# Predicted Values for Training and Test sets
#==============================================================================

# I create a list of our training predicted values across our various models
predicted.train = list(
    lm = predict(fit.lm, train.crypto)
  , ridge = predict(fit.ridge, train.model.x)
  , lasso = predict(fit.lasso, train.model.x)
  , randfore = predict(fit.randfore, train.crypto)
  , gbt = predict(fit.gbt, train.crypto.gbt)
  , pcr = predict(fit.pcr, train.crypto)
  , pls = predict(fit.pls, train.crypto)
)

# I create a list of our test predicted values across our various models
predicted.test = list(
    lm = predict(fit.lm, test.crypto)
  , ridge = predict(fit.ridge, test.model.x)
  , lasso = predict(fit.lasso, test.model.x)
  , randfore = predict(fit.randfore, test.crypto)
  , gbt = predict(fit.gbt, test.crypto.gbt)
  , pcr = predict(fit.pcr, test.crypto)
  , pls = predict(fit.pls, test.crypto)
)

### Calculate Residuals and MSE for training and test sets.

residuals.train.predicted = lapply(
  predicted.train,
  function(x) x - train.crypto$PriceUSD
)
```

```
MSE.train = lapply(
  residuals.train.predicted,
  function(x) mean(x**2)
)

residuals.test.predicted = lapply(
  predicted.test,
  function(x) x - test.crypto$PriceUSD
)

MSE.test = lapply(
  residuals.test.predicted,
  function(x) mean(x**2)
)


### Calculate R squared

r2 = function(pred, actual) {
  # This function calculates R^2
  top = 1 - sum((pred - actual)**2)
  bot = sum((actual - mean(actual))**2)

  # End of function
  return (top/bot)
}

R2.train = lapply(
  predicted.train,
  function(x) r2(x, train.crypto$PriceUSD)
)

R2.test = lapply(
  predicted.test,
  function(x) r2(x, test.crypto$PriceUSD)
)
```

Residuals Values

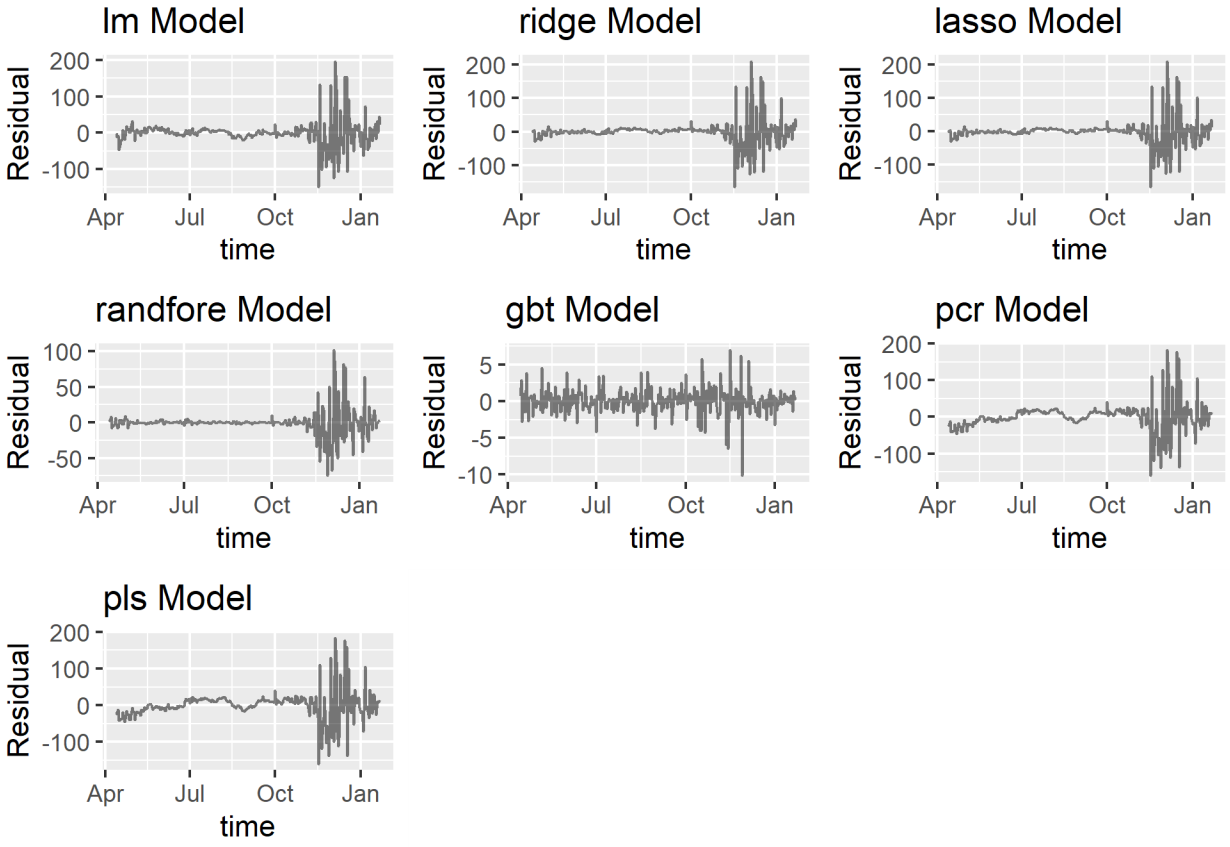
This chunk has us plotting our training residuals.

```
#==============================================================================
# Plot Training Residuals
#==============================================================================

Res.train.plot = mapply(
  function(x,y) {

  # format our plot data for our training residuals.
  plotdata = data.frame(time = train.crypto$time, Residual = unlist(x))
  colnames(plotdata) = c("time", "Residual")
```

```
  # return a ggplot object.
  return(ggplot(plotdata, aes(x = time, y = Residual)) +
    geom_line(alpha = 0.5) +
    ggtitle(paste0(y, " Model")))

  },

  # I pass in vectors for our training residuals and their labels.
  x = residuals.train.predicted,
  y = names(residuals.train.predicted),
  SIMPLIFY = FALSE
  )

# Save our plot to later embed in our document
ggsave(plot = ggarrange(plotlist = Res.train.plot), filename = "plot1.png")
```

Plot 1

Residuals for Training Set Models



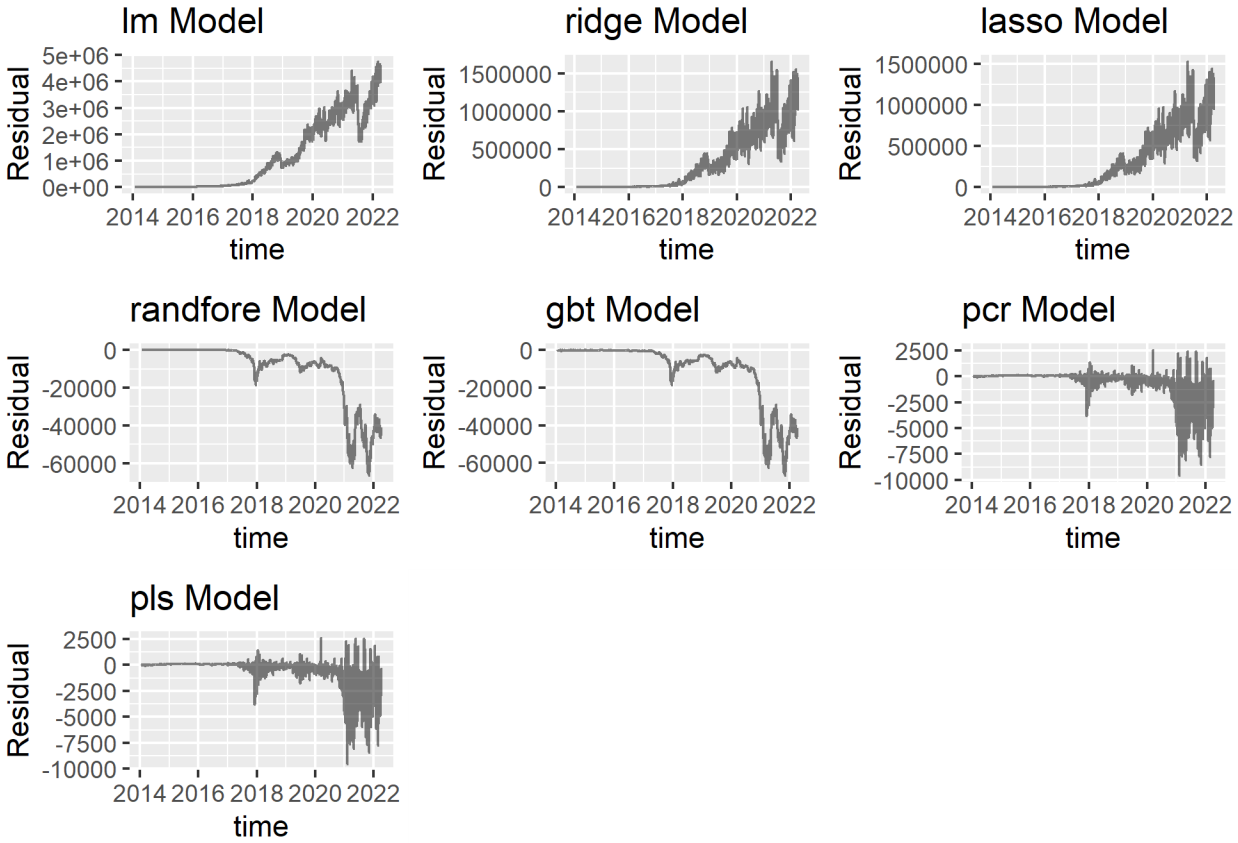
Our training residuals look pretty decent. A lot of the variances have been fitted for early on in the dataset, but by the end we see some major variances in all of models.

```
#==============================================================================
# Plot Test Residuals
#==============================================================================

Res.test.plot = mapply(
```

```
  function(x,y) {

  # format our plot data for our test residuals.
  plotdata = data.frame(time = test.crypto$time, Residual = unlist(x))
  colnames(plotdata) = c("time", "Residual")

  # return a ggplot object.
  return(ggplot(plotdata, aes(x = time, y = Residual)) +
    geom_line(alpha = 0.5) +
    ggtitle(paste0(y, " Model")))

  },

  # I pass in vectors for our test residuals and their labels.
  x = residuals.test.predicted,
  y = names(residuals.test.predicted),
  SIMPLIFY = FALSE
  )

# Save our plot to later embed in our document
ggsave(plot = ggarrange(plotlist = Res.test.plot), filename = "plot2.png")
```

Plot 2

Residuals for Test Set Models



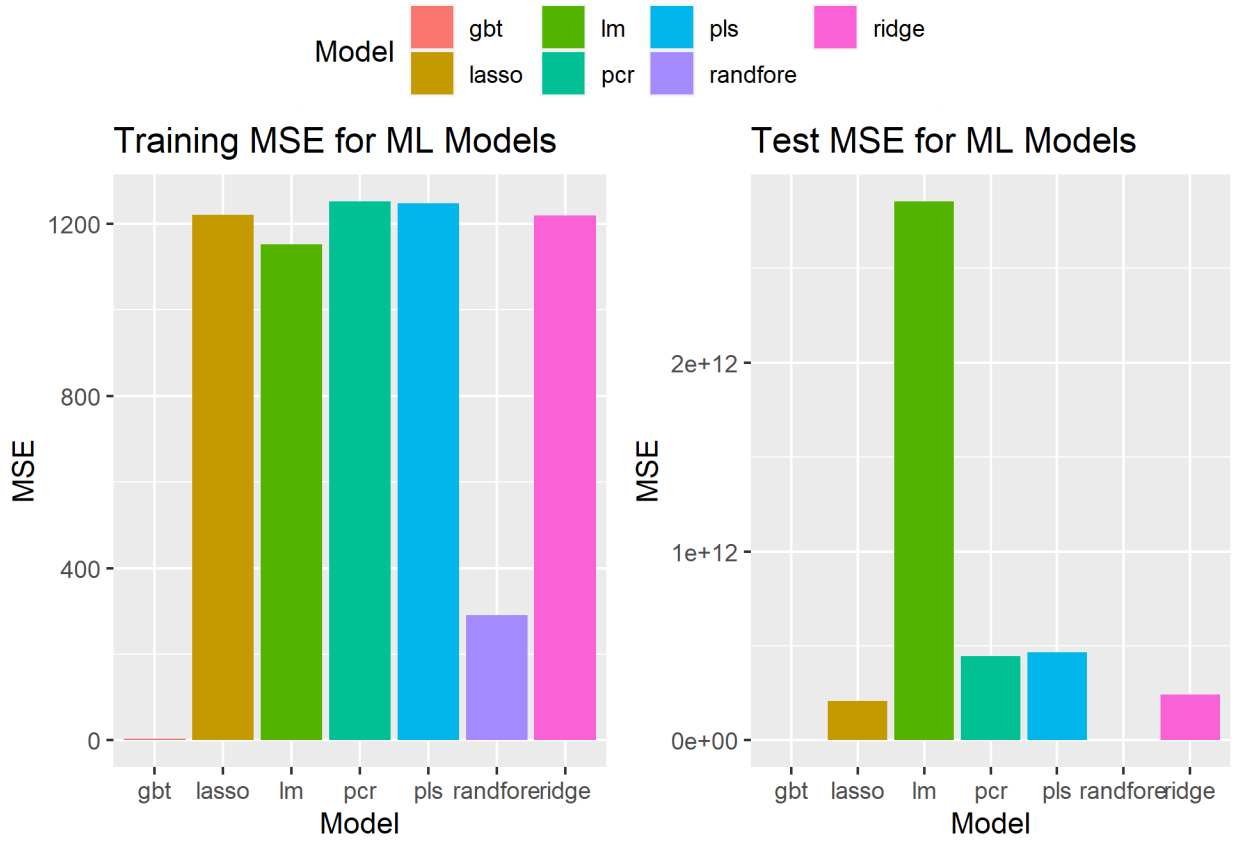
These residuals look okay. lm, ridge, lasso, pcr, and pls, generally look pretty okay at the start, but by the end start to get out of control. There looks like a major issue in our randomforest and gbt models, it's pretty

flat but then dies out completely and under fits our results.

Mean Sqaured Error

```
#==============================================================================
# Plot MSE - Train and Test
#==============================================================================

# Format MSE training data by collapsing list and renaming columns and rows
MSE.train.table = data.frame(do.call(rbind, MSE.train))
MSE.train.table$model = rownames(MSE.train.table)
colnames(MSE.train.table) = c("MSE", "Model")

# Create bar plot using ggplot for training data
MSE.train.plot = ggplot(MSE.train.table, aes(x = Model, y = MSE, fill = Model)) +
  geom_bar(stat = "identity") +
  ggtitle("Training MSE for ML Models")


# Format MSE test data by collapsing list and renaming columns and rows
MSE.test.table = data.frame(do.call(rbind, MSE.test))
MSE.test.table$model = rownames(MSE.test.table)
colnames(MSE.test.table) = c("MSE", "Model")

# Create bar plot using ggplot for test data
MSE.test.plot = ggplot(MSE.test.table, aes(x = Model, y = MSE, fill = Model)) +
  geom_bar(stat = "identity") +
  ggtitle("Test MSE for ML Models")



# Save our plot to later embed in our document
ggsave(plot = ggarrange(MSE.train.plot, MSE.test.plot, common.legend = TRUE), filename = "plot3.png")

# Create MSE table
MSE.table = cbind(MSE.train.table[,1], MSE.test.table)
colnames(MSE.table) = c("MSE - Train", "MSE - Test", "Model")
```

Plot 3
Model MSE for Training and Test Set



MSE looks good overall for our training set, but for our test set it becomes exponentially out of whack with our forest models becoming incredibly useless and our other regression models are all over fitting.

Table 4
MSE values for training and test models

	MSE - Train	MSE - Test	Model
lm	1152.441989	2.854878e+12	lm
ridge	1219.420884	2.409451e+11	ridge
lasso	1221.082218	2.088027e+11	lasso
randfore	291.369338	3.579674e+08	randfore
gbt	3.287365	3.595924e+08	gbt
pcr	1252.326202	4.445473e+11	pcr
pls	1247.045377	4.649569e+11	pls

Looking at our table we can see a pretty substantial increase in our models from training to test sets. This essentially indicates our models are over/under fitting.

R-Squared Error

```
#=============================================================================
# Create table for R2 - Train and Test
#=============================================================================

# Format R2 train data by collapsing list and renaming columns and rows
R2.train.table = data.frame(do.call(rbind, R2.train))
colnames(R2.train.table) = c("R - Train")

# Format R2 test data by collapsing list and renaming columns and rows
R2.test.table = data.frame(do.call(rbind, R2.test))
colnames(R2.test.table) = c("R2 - Test")

# Combine and Add new column for models
R2.table = cbind(R2.train.table, R2.test.table)
R2.table$model = rownames(R2.test.table)
```

Table 5 shows our R2 results and they are are not good. Having them all be negative in our training sets and to have them further drop for our test sets indicates a poor fit across all our models.

Table 5
R2 Values for Training and Test Sets

	R - Train	R2 - Test	model
lm	-0.0126871	-11022.643093	lm
ridge	-0.0134245	-930.285550	ridge
lasso	-0.0134428	-806.183982	lasso
randfore	-0.0032076	-1.382107	randfore
gbt	-0.0000362	-1.388381	gbt
pcr	-0.2068020	-25745.861254	pcr
pls	-0.2059299	-26927.875073	pls

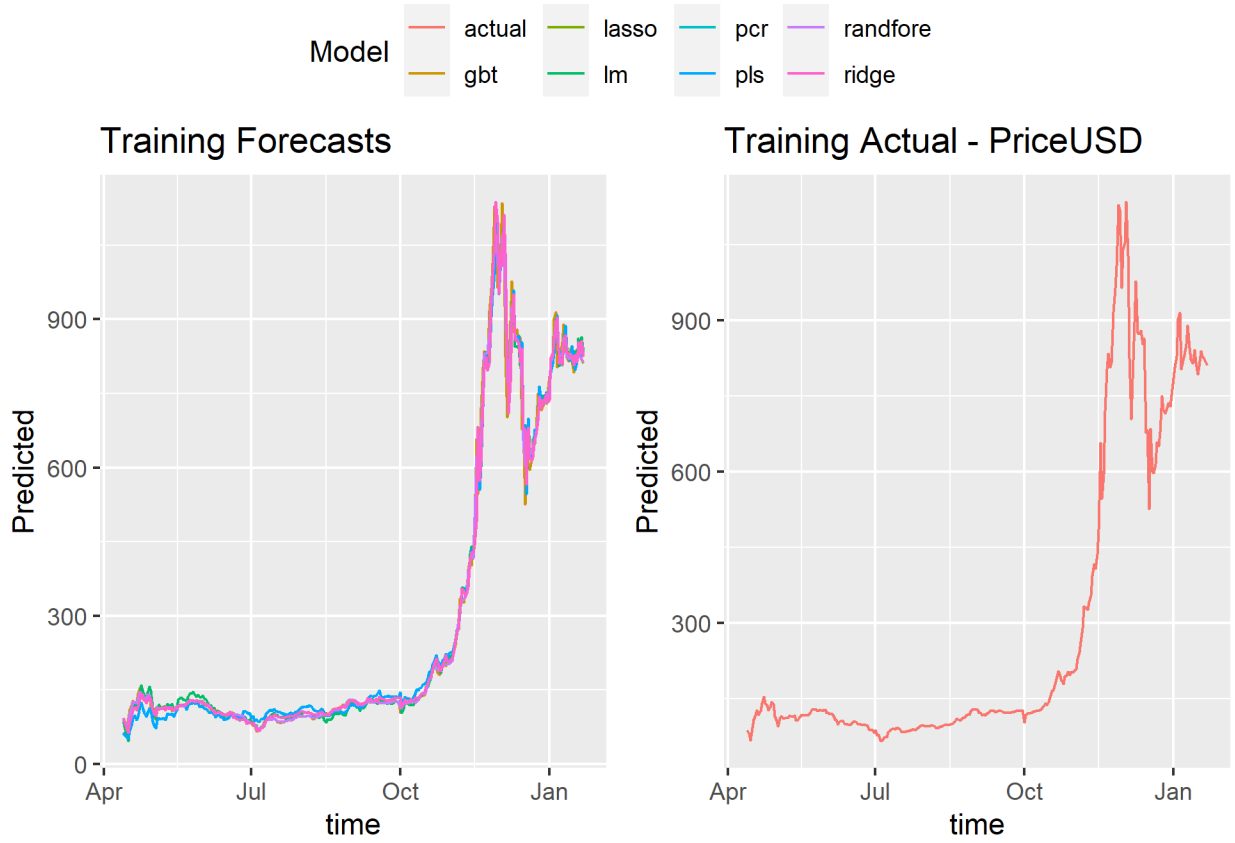
Training Forecasts

```
#=============================================================================
# Plot Training Forecast
#=============================================================================

# Setup our data into a long format to plot multiple models onto one data set.
forecast.train = data.frame(do.call(cbind, predicted.train))
colnames(forecast.train) = names(predicted.train)
forecast.train$actual = train.crypto$PriceUSD
forecast.train$time = train.crypto$time
forecast.train = gather(forecast.train, Model, Predicted, lm:actual)

# Create training forecast plot
fc.train = ggplot(forecast.train, aes(x = time, y = Predicted, color = Model)) +
  geom_line() +
```

```
  ggtitle("Training Forecasts")

# Create training actual plot
fc.train.actual = ggplot(forecast.train[grepl("actual",forecast.train$Model),], aes(x = time, y = Predi
  geom_line() +
  ggtitle("Training Actual - PriceUSD")

# Combine plots and save to embed below.
ggsave(plot = ggarrange(fc.train, fc.train.actual, common.legend = TRUE), "plot4.png")
```

Plot 4

Model Forecast with Training Set



Looking at plot 4, we can also see overfitting. This is probably due to certain predictors being too closely related to our USD price or our model just being too complex.

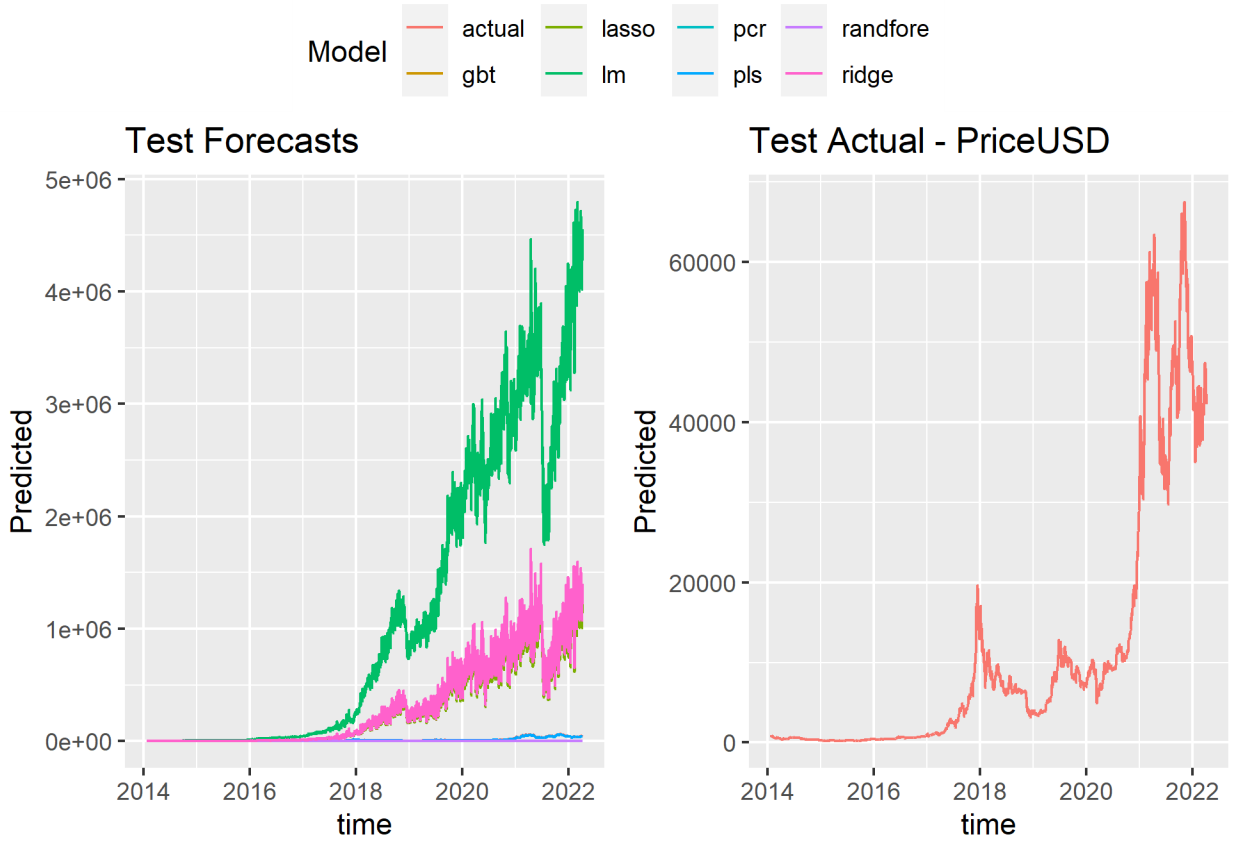
Test Forecasts

```
#================================================================================
# Plot Test Forecast
#================================================================================

# Setup our data into a long format to plot multiple models onto one data set.
forecast.test = data.frame(do.call(cbind, predicted.test))
colnames(forecast.test) = names(predicted.test)
```

```r
forecast.test$actual = test.crypto$PriceUSD
forecast.test$time = test.crypto$time
forecast.test = gather(forecast.test, Model, Predicted, lm:actual)

# Create test forecast plot
fc.test = ggplot(forecast.test, aes(x = time, y = Predicted, color = Model)) +
  geom_line() +
  ggtitle("Test Forecasts")

# Create test actual plot
fc.test.actual =  ggplot(forecast.test[grepl("actual",forecast.test$Model),], aes(x = time, y = Predicte
  geom_line() +
  ggtitle("Test Actual - PriceUSD")

# Combine plots and save to embed below.
ggsave(plot = ggarrange(fc.test, fc.test.actual, common.legend = TRUE), filename = "plot5.png")
```

Plot 5

Model Forecast with Test Set



Again, in our test forecasts, we see our results overfitting to our results again. Most likely due to some predictor being too closely related to our price USD or our model being too complex. However, our tree models are completely useless. I've tried countless variations and there always seems to be some issue in the test set, but if I decrease the size of the training set it seems to do better.

Benchmarks

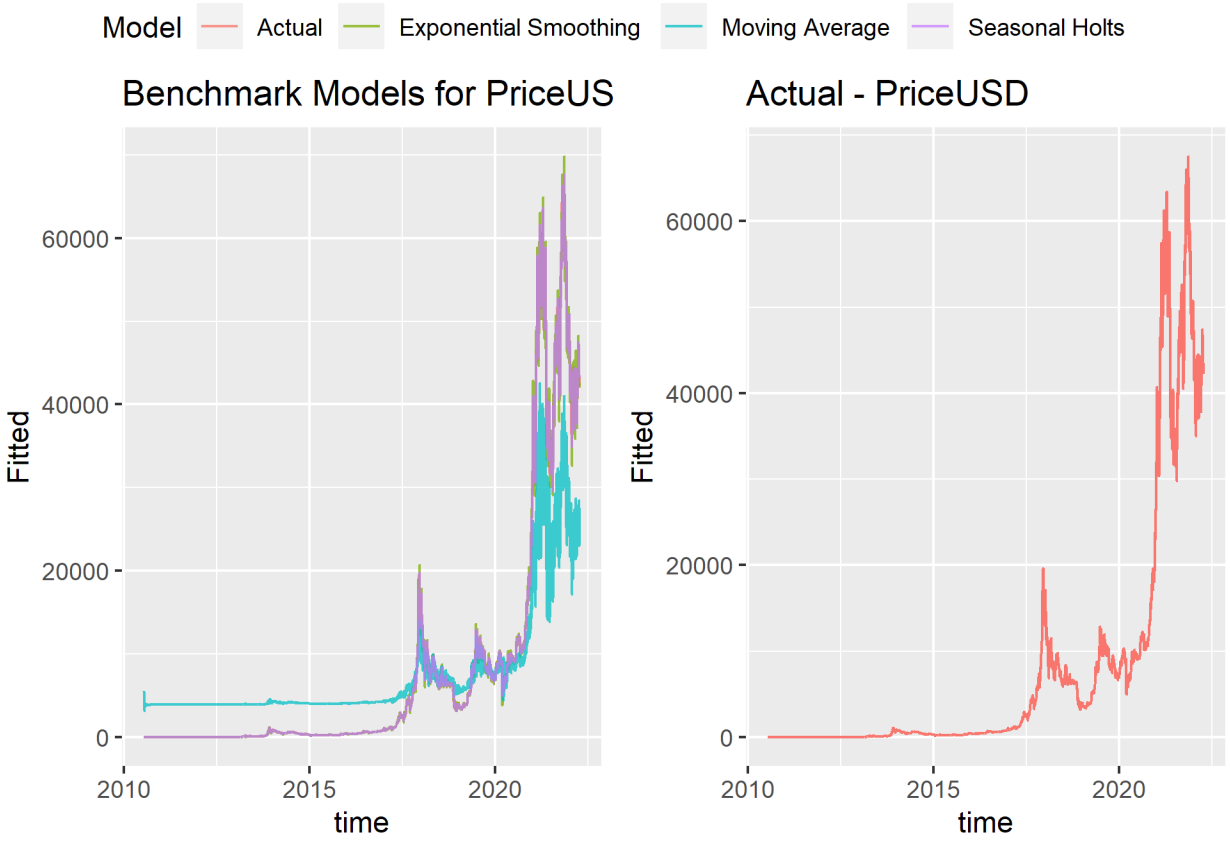
Basic forecast methods or benchmarks would include seasonal methods, moving averages, and exponential smoothing models.

```
#================================================================================
# Benchmarks
#================================================================================

# Create Seasonal Model with Holt
SE.Model = ets(CryptoMetrics$PriceUSD)

# Create Moving Average Model
MA.Model = arima(CryptoMetrics$PriceUSD, order = c(0,0,1))

## Calculate Residuals and fit
res = residuals(MA.Model)
MA.fitted = CryptoMetrics$PriceUSD -res

# Create Exponential Smoothing with Holt
ES.Model = holt(CryptoMetrics$PriceUSD, beta = 0.5, h = 100)

# Combine and convert our benchmark data into long format.
Benchmarks = data.frame(SE.Model$fitted,
                        MA.fitted,
                        ES.Model$fitted,
                        CryptoMetrics$PriceUSD,
                        CryptoMetrics$time
)

colnames(Benchmarks) = c("Seasonal Holts",
                         "Moving Average",
                         "Exponential Smoothing",
                         "Actual",
                         "time"
)

Benchmarks = gather(Benchmarks, Model, Fitted, `Seasonal Holts`:Actual)

# Create Plot with all our benchmarks
Benchmarks.plot = ggplot(Benchmarks, aes(x = time, y = Fitted, color = Model)) +
  geom_line(alpha = 0.75) +
  ggtitle("Benchmark Models for PriceUSD")

# Create Plot with Actuals
Benchmarks.Actual.plot =  ggplot(
  Benchmarks[grepl("Actual",Benchmarks$Model),],
  aes(x = time, y = Fitted, color = Model)) +
  geom_line() +
  ggtitle("Actual - PriceUSD")

# Combine plots and save to embed below.
ggsave(plot = ggarrange(Benchmarks.plot, Benchmarks.Actual.plot, common.legend = TRUE), filename = "plo
```

```
# clean up
rm(res)
```

Plot 6

Benchmarks Against Actual



Plot 6 shows our benchmarks in action, and overall they look similar to our machine learning models. Except these models needed a lot less work and code to return their results. Essentially, it looks like there isn't much of a difference between our benchmarks and machine learning forecasts. The most prominent difference would be our benchmarks performing better than our trees, but against all our regressions our results look the similar. In the end, I would prefer using these benchmarks over the machine learning models I found given their simplicity.

Conclusion

In this report I used the following machine learning methods.

- Multiple Linear Ordinary Least Squares Regression
- Ridge Regression
- Lasso Regression
- Principal Component Regression
- Partial Least Squares regression
- Random Forest
- Gradient Boosted Trees

Using these methods, I don't think I've truly solved the particular problem of volatility in this data set. After various methods of trimming down predictors, adjusting the training and test sets, and finding optimal parameters, I still have issues with massive overfitting/underfitting.

Essentially, the models did not do well and this is due to a multitude of issues. Below are a list of the key issues.

- Poor data quality
- Poor predictor selection
- Poor parameter selection

Poor data quality could mean that I don't have enough data for my training and test sets, the methods of interpolation skewed my results, and removing NA rows/columns weren't done properly or weren't enough for good dataset. A combination of these issues could lead to issues in training the models.

Poor predictor selection could mean I didn't select predictors that could properly reflect the USD price of bitcoin. I thought one issue could be using the lag price of the 'PriceUSD', but most models didn't rely heavily on this except for pcr and pls. I imagine there's one or more predictors on top of the lagged price that I didn't account for which is skewing my results to heavily match the USD price in both my training and test results, since it looks like the forecasts are heavily shadowing my actual prices.

Poor parameter selection could be from not using proper techniques to tune my models. One issue that didn't occur to me until later is how using time series data needs different techniques to tune. Cross validation can be used, but generally you would use a chaining method over a growing training set to make sure to include various trends and patterns throughout the dataset, but due to time constraints I couldn't implement this. Instead I opted to skip cross validation and hope for the best.

Some of the conclusions I can draw from this project is that forecasting bitcoin is even harder than I thought. I already thought it was going to be hard, but this was exceptionally harder than anticipated. Finding good predictors was especially tough, and even after going through various methods of reducing predictors, I still couldn't figure out how to reduce them further without using brute force methods. Additionally, after running the benchmarks, I think I would prefer just using those instead. Simpler models are easier to implement and understand, and given the results I found, they give similar results.

Some of the economics implications we can draw from our finding is that our market index and gold prices seem to correlate well with bitcoin's price, whether or not that helps forecast bitcoin's price is unclear given the results of our models. I could draw some economic implications from other predictors or mention possible use cases with these models, but given the over/fitting I would stay clear from these models for now and be critical of the results.

If I could extend the project I would add these key features.

- Dimension Reduction
- Neural Net Models (LSTM)

- Extensive ARIMA Benchmarks
- Hybrid Forecasting
- Cross Validation Chaining

Reducing and combing some of our predictors could have been a nice way of simplifying our model but still retaining some of the information from other predictors. Neural net models would be cool to add since they can essentially remember past results better than other models. ARIMA models are something I've used in another class, and they would have been interesting to see how they can model the variance of bitcoin. Also, I could have added on various GARCH models as well to forecast the volatility specifically. Furthermore, hybrid forecasting would have been interesting to try and reduce the error present in our models via trimmed mean or OLS techniques. Additionally, I would have tried to use cross validation chaining to further refine our models over each cumulative training set this way I can fine tune our parameters in a way that reflects well on the time-series data.