Forecasting the Price of Gold

Amin Fesharaki, Ryan Dunn, Kyle Dalope

11/2022

```
library(caret)
```

```
## Warning: package 'caret' was built under R version 4.1.2
```

```
## Loading required package: ggplot2
```

```
## Warning: package 'ggplot2' was built under R version 4.1.2
```

```
## Loading required package: lattice
```

```
library(corrplot)
```

```
## corrplot 0.92 loaded
```

```
library(dplyr)
```

```
## Warning: package 'dplyr' was built under R version 4.1.2
```

```
##
## Attaching package: 'dplyr'
```

```
## The following objects are masked from 'package:stats':
##
##     filter, lag
```

```
## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```
library(DT)
```

```
## Warning: package 'DT' was built under R version 4.1.2
```

```
library(fpp2)
```

```
## Registered S3 method overwritten by 'quantmod':
##   method            from
##   as.zoo.data.frame zoo
```

```
## ── Attaching packages ─────────────────────────────────────── fpp2 2.4 ──
```

```
## ✔ forecast  8.18     ✔ expsmooth 2.3
## ✔ fma       2.4
```

```
## Warning: package 'forecast' was built under R version 4.1.2
```

```
##
```

```
library(ggplot2)
library(gtrendsR)
```

```
## Warning: package 'gtrendsR' was built under R version 4.1.2
```

```
library(imputeTS)
```

```
## Warning: package 'imputeTS' was built under R version 4.1.2
```

```
library(lubridate)
```

```
## Warning: package 'lubridate' was built under R version 4.1.2
```

```
## Loading required package: timechange
```

```
## Warning: package 'timechange' was built under R version 4.1.2
```

```
##
## Attaching package: 'lubridate'
```

```
## The following objects are masked from 'package:base':
##
##     date, intersect, setdiff, union
```

```
library(Metrics)
```

```
##
## Attaching package: 'Metrics'
```

```
## The following object is masked from 'package:forecast':
##
##     accuracy
```

```
## The following objects are masked from 'package:caret':
##
##     precision, recall
```

```
library(plotly)
```

```
## Warning: package 'plotly' was built under R version 4.1.2
```

```
##
## Attaching package: 'plotly'
```

```
## The following object is masked from 'package:ggplot2':
##
##     last_plot
```

```
## The following object is masked from 'package:stats':
##
##     filter
```

```
## The following object is masked from 'package:graphics':
##
##     layout
```

```
library(prophet)
```

```
## Loading required package: Rcpp
```

```
## Warning: package 'Rcpp' was built under R version 4.1.2
```

```
## Loading required package: rlang
```

```
## Warning: package 'rlang' was built under R version 4.1.2
```

```
##
## Attaching package: 'rlang'
```

```
## The following object is masked from 'package:Metrics':
##
##     ll
```

```
library(Quandl)
```

```
## Loading required package: xts
```

```
## Warning: package 'xts' was built under R version 4.1.2
```

```
## Loading required package: zoo
```

```
## Warning: package 'zoo' was built under R version 4.1.2
```

```
##
## Attaching package: 'zoo'
```

```
## The following object is masked from 'package:imputeTS':
##
##     na.locf
```

```
## The following objects are masked from 'package:base':
##
##     as.Date, as.Date.numeric
```

```
##
## Attaching package: 'xts'
```

```
## The following objects are masked from 'package:dplyr':
##
##     first, last
```

```
library(quantmod)
```

```
## Warning: package 'quantmod' was built under R version 4.1.2
```

```
## Loading required package: TTR
```

```
library(readr)
```

```
## Warning: package 'readr' was built under R version 4.1.2
```

```
library(rvest)
```

```
## Warning: package 'rvest' was built under R version 4.1.2
```

```
##
## Attaching package: 'rvest'
```

```
## The following object is masked from 'package:readr':
##
##     guess_encoding
```

```
library(stringr)
```

```
## Warning: package 'stringr' was built under R version 4.1.2
```

```
library(tidymodels)
```

```
## Warning: package 'tidymodels' was built under R version 4.1.2
```

```
## ── Attaching packages ─────────────────────────────────────── tidymodels 1.0.0 ──
```

```
## ✔ broom        1.0.1     ✔ rsample      1.1.0
## ✔ dials        1.1.0     ✔ tibble       3.1.8
## ✔ infer        1.0.3     ✔ tidyr        1.2.1
## ✔ modeldata    1.0.1     ✔ tune         1.0.1
## ✔ parsnip      1.0.3     ✔ workflows    1.1.2
## ✔ purrr        0.3.5     ✔ workflowsets 1.0.0
## ✔ recipes      1.0.3     ✔ yardstick    1.1.0
```

```
## Warning: package 'broom' was built under R version 4.1.2
```

```
## Warning: package 'dials' was built under R version 4.1.2
```

```
## Warning: package 'scales' was built under R version 4.1.2
```

```
## Warning: package 'infer' was built under R version 4.1.2
```

```
## Warning: package 'modeldata' was built under R version 4.1.2
```

```
## Warning: package 'parsnip' was built under R version 4.1.2
```

```
## Warning: package 'purrr' was built under R version 4.1.2
```

```
## Warning: package 'recipes' was built under R version 4.1.2
```

```
## Warning: package 'rsample' was built under R version 4.1.2
```

```
## Warning: package 'tibble' was built under R version 4.1.2
```

```
## Warning: package 'tidyr' was built under R version 4.1.2
```

```
## Warning: package 'tune' was built under R version 4.1.2
```

```
## Warning: package 'workflows' was built under R version 4.1.2
```

```
## Warning: package 'workflowsets' was built under R version 4.1.2
```

```
## Warning: package 'yardstick' was built under R version 4.1.2
```

```
## ── Conflicts ──────────────────────────────────────── tidymodels_conflicts() ──
## ✖ purrr::%@%()             masks rlang::%@%()
## ✖ yardstick::accuracy()    masks Metrics::accuracy(), forecast::accuracy()
## ✖ purrr::as_function()     masks rlang::as_function()
## ✖ purrr::discard()         masks scales::discard()
## ✖ plotly::filter()         masks dplyr::filter(), stats::filter()
## ✖ xts::first()             masks dplyr::first()
## ✖ recipes::fixed()         masks stringr::fixed()
## ✖ purrr::flatten()         masks rlang::flatten()
## ✖ purrr::flatten_chr()     masks rlang::flatten_chr()
## ✖ purrr::flatten_dbl()     masks rlang::flatten_dbl()
## ✖ purrr::flatten_int()     masks rlang::flatten_int()
## ✖ purrr::flatten_lgl()     masks rlang::flatten_lgl()
## ✖ purrr::flatten_raw()     masks rlang::flatten_raw()
## ✖ purrr::invoke()          masks rlang::invoke()
## ✖ dplyr::lag()             masks stats::lag()
## ✖ xts::last()              masks dplyr::last()
## ✖ purrr::lift()            masks caret::lift()
## ✖ rlang::ll()              masks Metrics::ll()
## ✖ yardstick::mae()         masks Metrics::mae()
## ✖ yardstick::mape()        masks Metrics::mape()
## ✖ yardstick::mase()        masks Metrics::mase()
## ✖ dials::momentum()        masks TTR::momentum()
## ✖ rsample::populate()      masks Rcpp::populate()
## ✖ yardstick::precision()   masks Metrics::precision(), caret::precision()
## ✖ yardstick::recall()      masks Metrics::recall(), caret::recall()
## ✖ yardstick::rmse()        masks Metrics::rmse()
## ✖ yardstick::sensitivity() masks caret::sensitivity()
## ✖ yardstick::smape()       masks Metrics::smape()
## ✖ yardstick::spec()        masks readr::spec()
## ✖ yardstick::specificity() masks caret::specificity()
## ✖ purrr::splice()          masks rlang::splice()
## ✖ recipes::step()          masks stats::step()
## • Use suppressPackageStartupMessages() to eliminate package startup messages
```

```
library(tidyr)
library(tidytext)
```

```
## Warning: package 'tidytext' was built under R version 4.1.2
```

```
library(zoo)
```

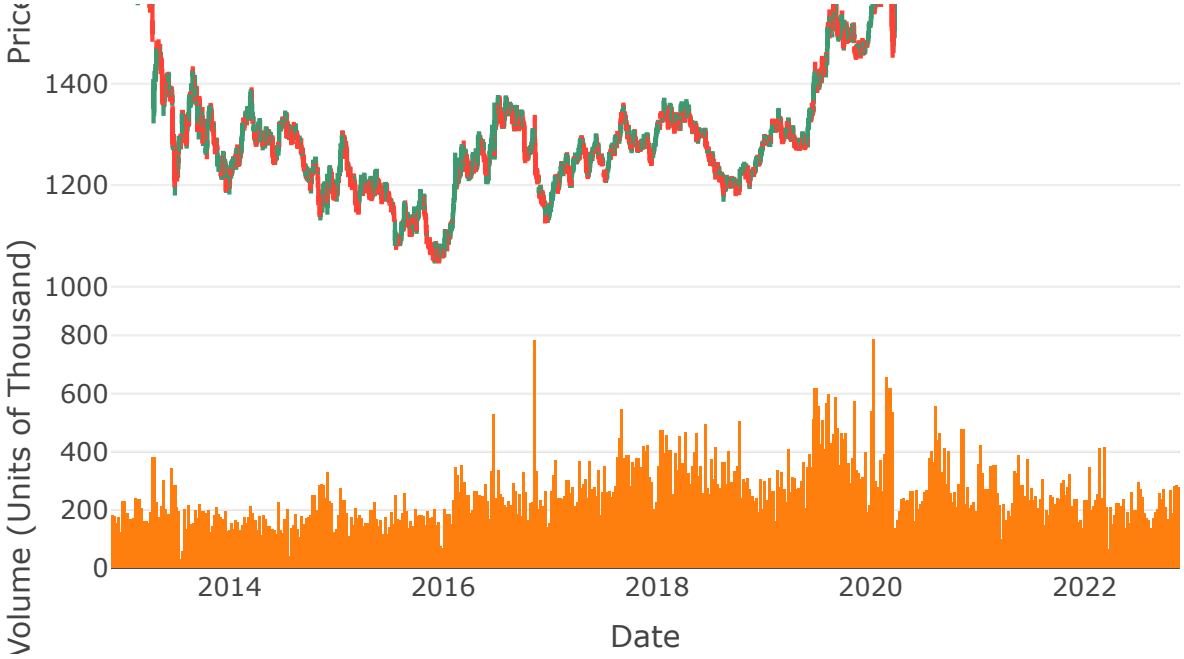
Load Data

```
# Load Data

gold <- read_csv("/Users/datascience/Desktop/Time Series Data Science/Time Series Pro
ject/Update/Gold.csv",
    col_types = cols(Date = col_date(format = "%m/%d/%Y")))

unemp <- read_csv("/Users/datascience/Desktop/Time Series Data Science/Time Series Pr
oject/Update/UNRATE.csv", show_col_types = FALSE)

M2 <- read_csv("/Users/datascience/Desktop/Time Series Data Science/Time Series Proje
ct/Update/WM2NS.csv", show_col_types = FALSE)

DJ <- read_csv("/Users/datascience/Desktop/Time Series Data Science/Time Series Proje
ct/Update/DJIA.csv",  show_col_types = FALSE)

fed_funds <- read_csv("/Users/datascience/Desktop/Time Series Data Science/Time Serie
s Project/Update/DFF.csv", show_col_types = FALSE)

silver <- Quandl('LBMA/SILVER')

dollar_index <- read_csv("/Users/datascience/Desktop/Time Series Data Science/Time Se
ries Project/Update/US Dollar Index Historical Data.csv", col_types = cols(Date = col
_date(format = "%m/%d/%Y")))

Google <- read_csv("/Users/datascience/Desktop/Time Series Data Science/Time Series P
roject/Update/Gold_Trend_hits.csv", show_col_types = FALSE)
```

Create Candlestick/volume graph of gold

```
#edit the column names in data frames
Price_plot <- gold %>%
  plot_ly(x = ~Date,
          type = "candlestick",
          open = ~Open,
          close = ~`Close/Last`,
          high = ~High,
          low = ~Low,
          name = "price") %>%
  layout(
    xaxis = list(
      rangeselector = list(
        buttons = list(
          list(
            count = 3,
            label = "3 mo",
            step = "month",
            stepmode = "backward"),
          list(
```

```
            count = 6,
            label = "6 mo",
            step = "month",
            stepmode = "backward"),
          list(
            count = 1,
            label = "1 yr",
            step = "year",
            stepmode = "backward"),
          list(
            count = 2,
            label = "2 yr",
            step = "year",
            stepmode = "backward"),
          list(
            count = 3,
            label = "3 yr",
            step = "year",
            stepmode = "backward"),
          list(
            count = 5,
            label = "5 yr",
            step = "year",
            stepmode = "backward"),
          list(step = "all"))),

      rangeslider = list(visible = FALSE)),
         yaxis = list(title = "Price ($)",
                      showgrid = TRUE,
                      showticklabels = TRUE))
Volume <- select(gold, Date, Volume)
Volume$Date <- as.Date(Volume$Date , format = "%m/%d/%y")
Volume$Vol <- as.numeric(as.character(Volume$Volume)) / 1000
Volume_plot <- Volume %>%
  plot_ly(x=~Date, y=~Vol, type='bar', name = "Volume") %>%
  layout(yaxis = list(title = "Volume (Units of Thousand)"))
plot <- subplot(Price_plot, Volume_plot, heights = c(0.7,0.3), nrows=2,
             shareX = TRUE, titleY = TRUE) %>%
  layout(title = 'GC:CMX')
plot
```



Adjust data types where needed

```
#change data type from chr to date in the gold data set
gold$Date <- as.Date(gold$Date, format = "%m/%d/%y")
gold$DATE <- gold$Date
#change the datatype to date in the DJIA data set
DJ$DATE <- as.Date(DJ$DATE, format = "%m/%d/%y")
#indicate columns to keep in gold data set
keep_cols <-  c("Date","Close/Last", 'Volume')
gold <- gold[keep_cols]
gold <- gold %>%
  rename(Gold_Close = 'Close/Last', Gold_Volume ='Volume')
```

Create the daily M2 rate data frame (in billions)

```
#create all dates from the date column in the data frame
all_dates <- M2 %>%select(DATE) %>%
  complete(DATE=  seq.Date(min(DATE), max(DATE), by="day"),
           ) %>% mutate(TIME=paste(year(DATE),str_pad(month(DATE), 2, pad = "0"),sep
= "-"))
#join the all_dates object back onto original datafarme
M2_daily <- left_join(all_dates,M2,by="DATE")
#fill the NA values from the join with the first value from the dataset
M2_daily <- na_locf(M2_daily, option = "locf")
keep_cols <- c("DATE", "WM2NS")
M2_daily <- M2_daily[keep_cols]
M2_daily <- M2_daily %>%
  select(c("DATE","WM2NS")) %>%
  rename(Date = DATE)
```

Create the daily unemployment rate data frame

```
#create all dates from the date column in the dataframe
all_dates <- unemp %>%select(DATE) %>%
  complete(DATE=  seq.Date(min(DATE), max(DATE), by="day"),
           ) %>% mutate(TIME=paste(year(DATE),str_pad(month(DATE), 2, pad = "0"),sep
= "-"))
#join the all_dates object back onto original datafarme
unemp_daily <- left_join(all_dates,unemp,by="DATE")
#fill the NA values from the join with the first value from the dataset
unemp_daily <- na_locf(unemp_daily,  option = "locf")
keep_cols <- c("DATE", "UNRATE")
unemp_daily <- unemp_daily[keep_cols]
unemp_daily <- unemp_daily %>%
  select(c("DATE","UNRATE")) %>%
  rename(Date = DATE)
```

Create the daily federal funds data frame

```
#create all dates from the date column in the data frame
all_dates <- fed_funds %>%select(observation_date) %>%
  complete(observation_date =  seq.Date(min(observation_date), max(observation_date),
by="day"),
           ) %>% mutate(TIME=paste(year(observation_date),str_pad(month(observation_d
ate), 2, pad = "0"),sep = "-"))
#join the all_dates object back onto original dataframe
ff_daily <- left_join(all_dates,fed_funds,by="observation_date")
#fill the NA values from the join with the first value from the dataset
ff_daily <- na_locf(ff_daily, option = "locf")
keep_cols <- c("observation_date", "DFF")
ff_daily <- ff_daily[keep_cols]
ff_daily <- ff_daily %>%
  select(c("observation_date","DFF")) %>%
  rename(Date = observation_date)
```

Create the daily Dow Jones IA data frame

```
#create all dates from the date column in the data frame
all_dates <- DJ %>%select(DATE) %>%
  complete(DATE =  seq.Date(min(DATE), max(DATE), by="day"),
           ) %>% mutate(TIME=paste(year(DATE),str_pad(month(DATE), 2, pad = "0"),sep
= "-"))
#join the all_dates object back onto original datafarme
dj_daily <- left_join(all_dates, DJ ,by="DATE")

#fill the NA values from the join with the first value from the dataset
dj_daily <- na_locf(dj_daily,  option = "locf")
keep_cols <- c("DATE", "DJIA")
dj_daily <- dj_daily[keep_cols]
dj_daily <- dj_daily %>%
  select(c("DATE", "DJIA")) %>%
  rename(Date = DATE)
```

Create the daily Silver Price data frame

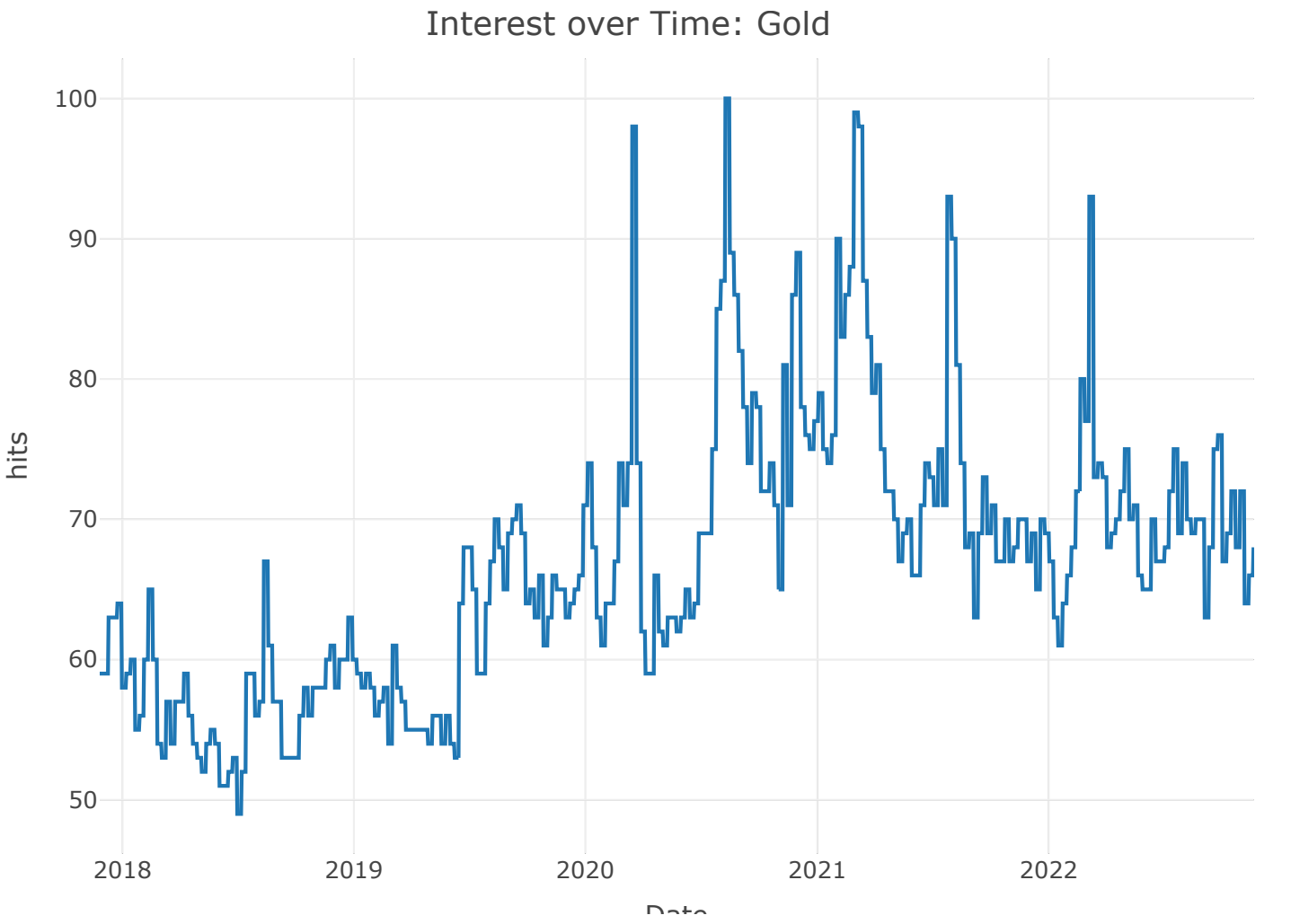
```
# add in Silver
silver <- silver[,c('Date', 'USD')]
silver <- silver %>%
  rename(Silver_Close = 'USD')
# Join Silver and Gold Data
Metals <- inner_join(silver, gold, by="Date")
```

Create the Dollar Index Price data frame

```
# Add in Dollar Index
dollar_index <- dollar_index %>%
  select(c("Date","Price")) %>%
  rename(DXY = Price)
```

Google trends to find hits on Gold worlwide

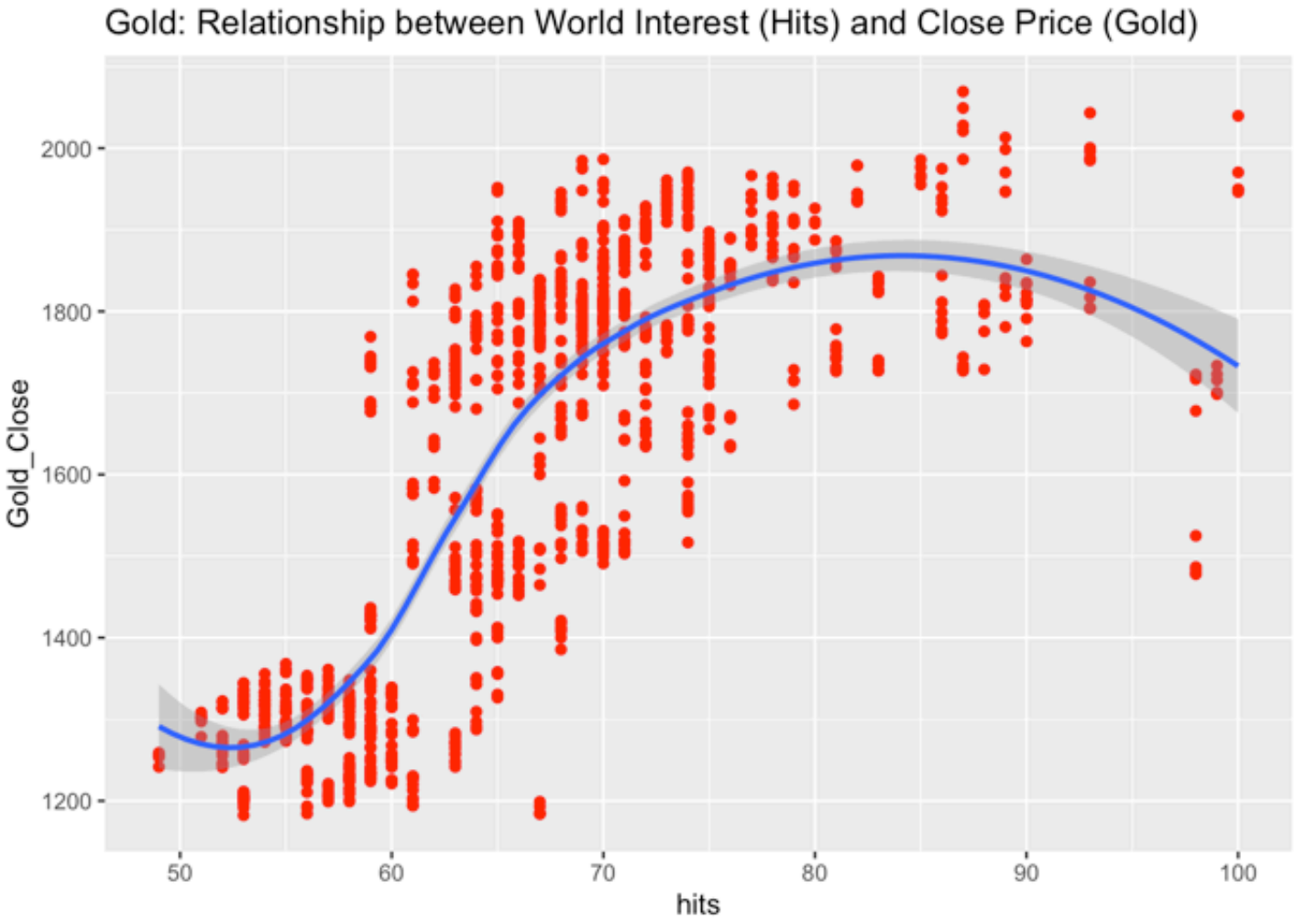
```
#trends <- gtrends(keyword = "Gold", onlyInterest = TRUE, time = "today+5-y")

#create all dates from the date column in the data frame
all_dates <- Google %>%select(Date) %>%
  complete(Date =  seq.Date(min(Date), max(Date), by="day"),
           ) %>% mutate(TIME=paste(year(Date),str_pad(month(Date), 2, pad = "0"),sep
= "-"))
#join the all_dates object back onto original datafarme
Google_daily <- left_join(all_dates, Google ,by="Date")
#fill the NA values from the join with the first value from the dataset
Google_daily <- na_locf(Google_daily,  option = "locf")

# Create Daily Data Fame
keep_cols <- c("Date", "Gold: (Worldwide)")
Google_daily <- Google_daily[keep_cols]
Google_daily <- Google_daily %>%
  select(c("Date", "Gold: (Worldwide)")) %>%
  rename(hits = 'Gold: (Worldwide)')
Google_daily %>%
  plot_ly(type='scatter',x=~Date, y=~hits, mode = 'lines', name = "Google Search Tren
ds") %>%
  layout(title = paste0("Interest over Time: ", "Gold"), yaxis = list(title = "hits")
)
```



Date

```
trends_daily <- Google_daily
# Hits verse Gold
trends_daily %>%
  left_join(gold, by = "Date") %>%
  select(one_of(c("Date", "hits", "Gold_Close"))) %>%
  drop_na() %>%
  ggplot(aes(hits, Gold_Close)) + geom_point(color="red") + geom_smooth(method = 'loe
ss') +
  labs(title =paste0("Gold",": Relationship between World Interest (Hits) and Close P
rice (Gold)"))
```

```
## `geom_smooth()` using formula = 'y ~ x'
```



Merge data frames and create the final dataframe for EDA

```
#merge the dataframes together on dates
full_df <- left_join(Metals, dollar_index, by="Date")
full_df <- left_join(full_df, unemp_daily, by="Date")
full_df <- left_join(full_df, ff_daily, by="Date")
full_df <- left_join(full_df, M2_daily, by="Date")
full_df <- left_join(full_df, Google_daily, by="Date")
full_df <- left_join(full_df, dj_daily, by="Date")
full_df <- left_join(full_df, trends_daily, by="Date")
full_df$Gold_Volume <- na_ma(full_df$Gold_Volume, k=1, weighting = "simple")
keep_cols <- c("Date", "Silver_Close", "Gold_Close","Gold_Volume", "DXY", "UNRATE", "
DFF", "WM2NS" ,"hits.x",     "DJIA")
full_df <- full_df[keep_cols]
full_df <- full_df %>%
  rename(hits = hits.x)
```

```
# Check for any NA Values
cbind(
   lapply(
     lapply(full_df, is.na)
     , sum)
   )
```

```
##              [,1]
## Date         0
## Silver_Close 0
## Gold_Close   0
## Gold_Volume  0
## DXY          0
## UNRATE       1255
## DFF          1255
## WM2NS        1259
## hits         1259
## DJIA         1255
```

Subset the data frame with values from 2018 forward

```
full_df <- full_df %>% arrange(ymd(full_df$Date))
full_df <- full_df[full_df$Date >= "2018-01-01",]
#Replace NA with the most recent value
full_df$WM2NS <- zoo::na.fill(full_df$WM2NS, "extend")
full_df$hits <- zoo::na.fill(full_df$hits, "extend")
full_df$UNRATE <- zoo::na.fill(full_df$UNRATE, "extend")
tail(full_df)
```

	Date	Silver_Close	Gold_Close	Gold_Volume	DXY	UN…	D…	WM2…	hits
	<date>	<dbl>	<dbl>	<chr>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>
2485	2022-11-15	21.940	1774.7	276828	106.40	3.5	3.83	21346.7	66
2486	2022-11-16	21.950	1775.8	191870	106.28	3.5	3.83	21346.7	66
2487	2022-11-17	21.075	1763.0	164553	106.69	3.5	3.83	21346.7	66
2488	2022-11-18	21.095	1769.0	29695	106.97	3.5	3.83	21346.7	66
2489	2022-11-21	20.640	1754.6	64912	107.78	3.5	3.83	21346.7	66
2490	2022-11-22	21.270	1754.8	57764	107.15	3.5	3.83	21346.7	66

6 rows | 1-10 of 11 columns

```
# Check for any NA Values
cbind(
   lapply(
     lapply(full_df, is.na)
     , sum)
   )
```

```
##              [,1]
## Date         0
## Silver_Close 0
## Gold_Close   0
## Gold_Volume  0
## DXY          0
## UNRATE       0
## DFF          0
## WM2NS        0
## hits         0
## DJIA         0
```

Create the correlation matrix for EDA

```
corr_fields <- c("Gold_Close", "Silver_Close","Gold_Volume", "DXY", "UNRATE", "DFF",
"WM2NS" ,"hits", "DJIA")
full_df$Gold_Volume <- as.numeric(full_df$Gold_Volume)
```

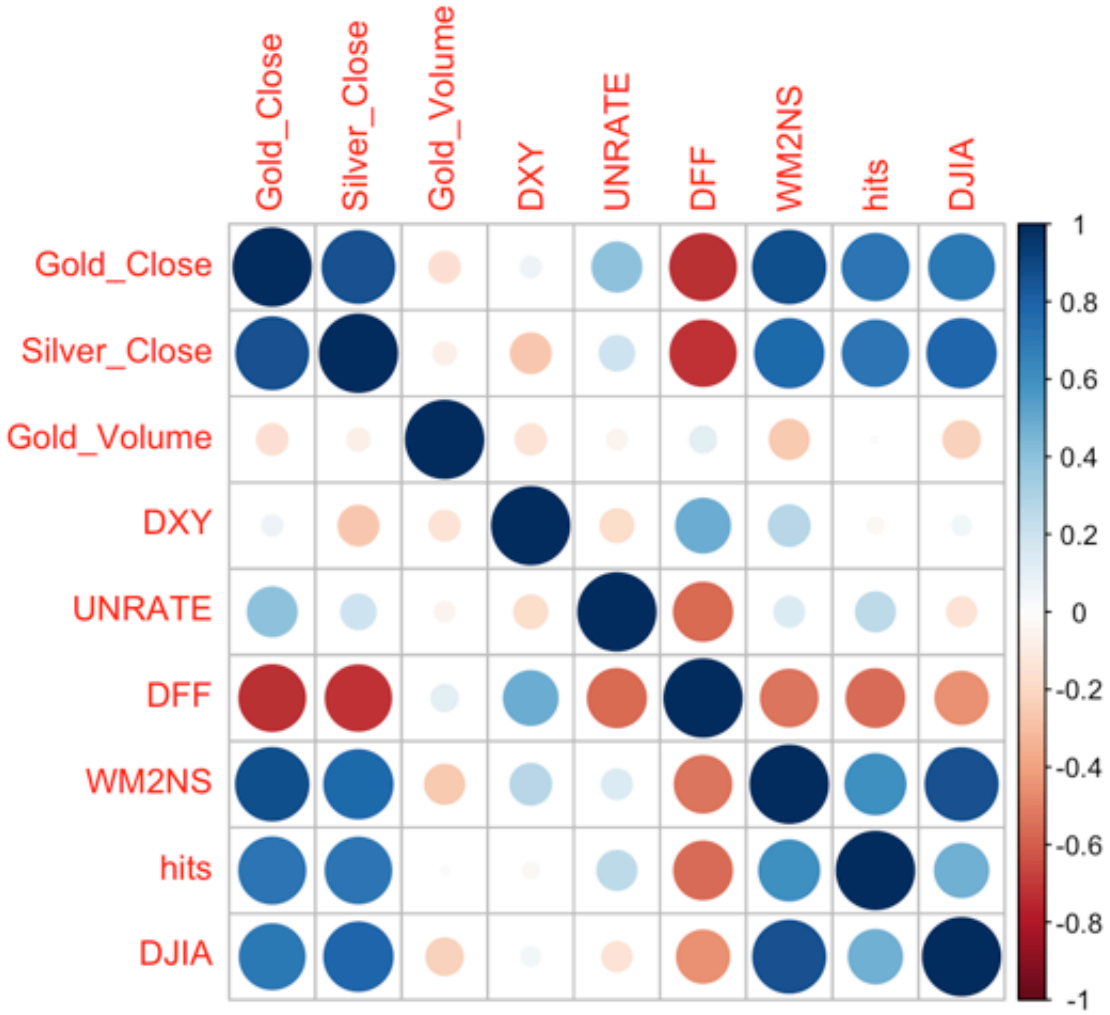
```
## Warning: NAs introduced by coercion
```

```
full_df$WM2NS <- as.numeric(full_df$WM2NS)
full_df$hits <- as.numeric(full_df$hits)

full_df$Gold_Volume <- na_ma(full_df$Gold_Volume, k=1, weighting = "simple")
full_df$WM2NS <- na_ma(full_df$WM2NS, k=1, weighting = "simple")
full_df$hits <- na_ma(full_df$hits, k=1, weighting = "simple")
```

```
corr_df <- full_df[corr_fields]
corr_matrix = cor(corr_df)
#display first row of correlation matrix
corr_matrix[,1]
```

```
##   Gold_Close Silver_Close  Gold_Volume          DXY       UNRATE          DFF
##   1.00000000   0.86040095  -0.15352950   0.06534308   0.39961962  -0.72279326
##        WM2NS         hits         DJIA
##   0.87544073   0.72864839   0.70205860
```

```
#develop the corrplot correlation matrix
corrplot(corr_matrix)
```

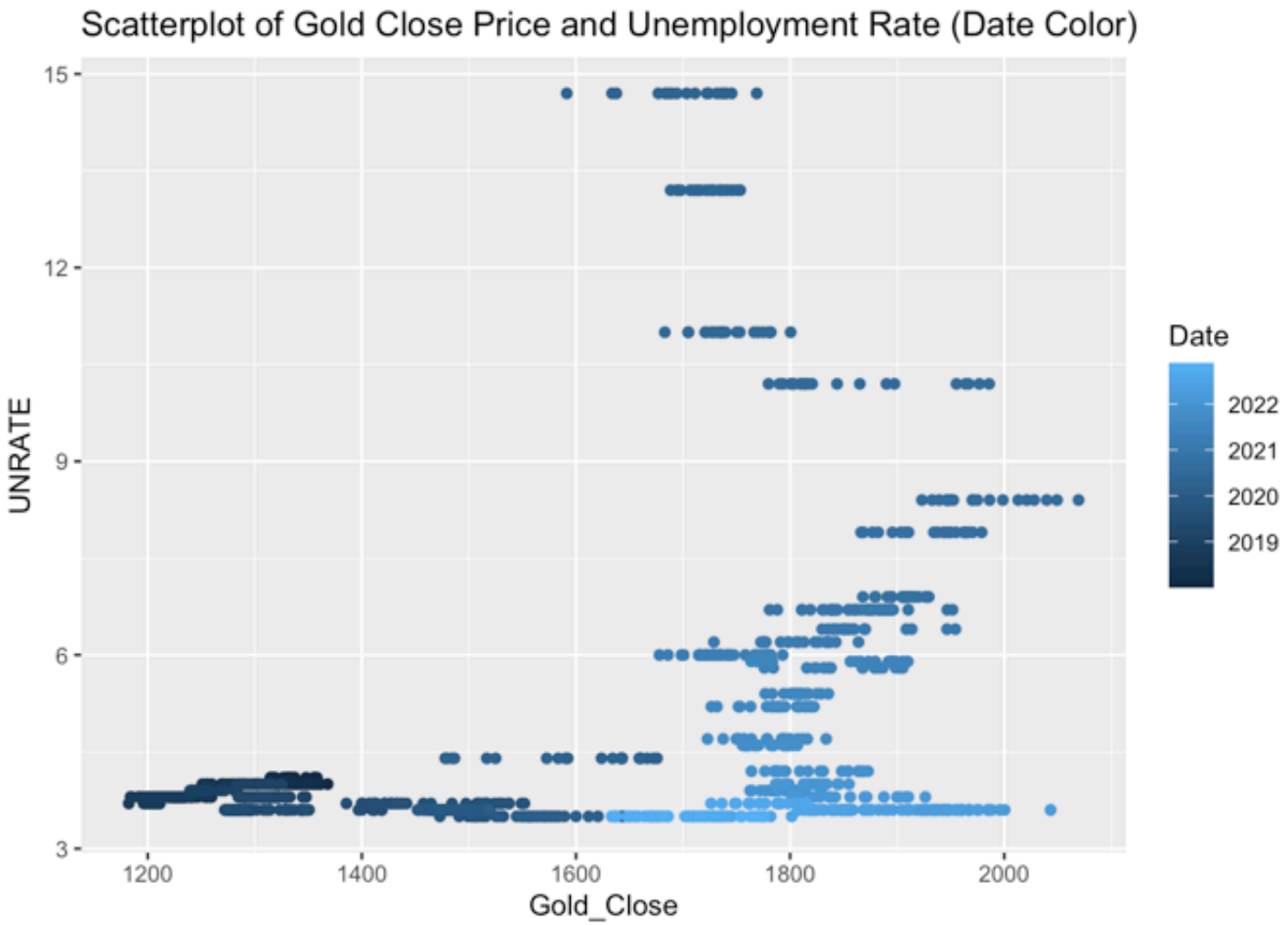


Silver Close, DFF, WM2NS, hits, and DIJA are strongly correlated with gold and therefore will be used as external regressors. ## Normalize variables for EDA

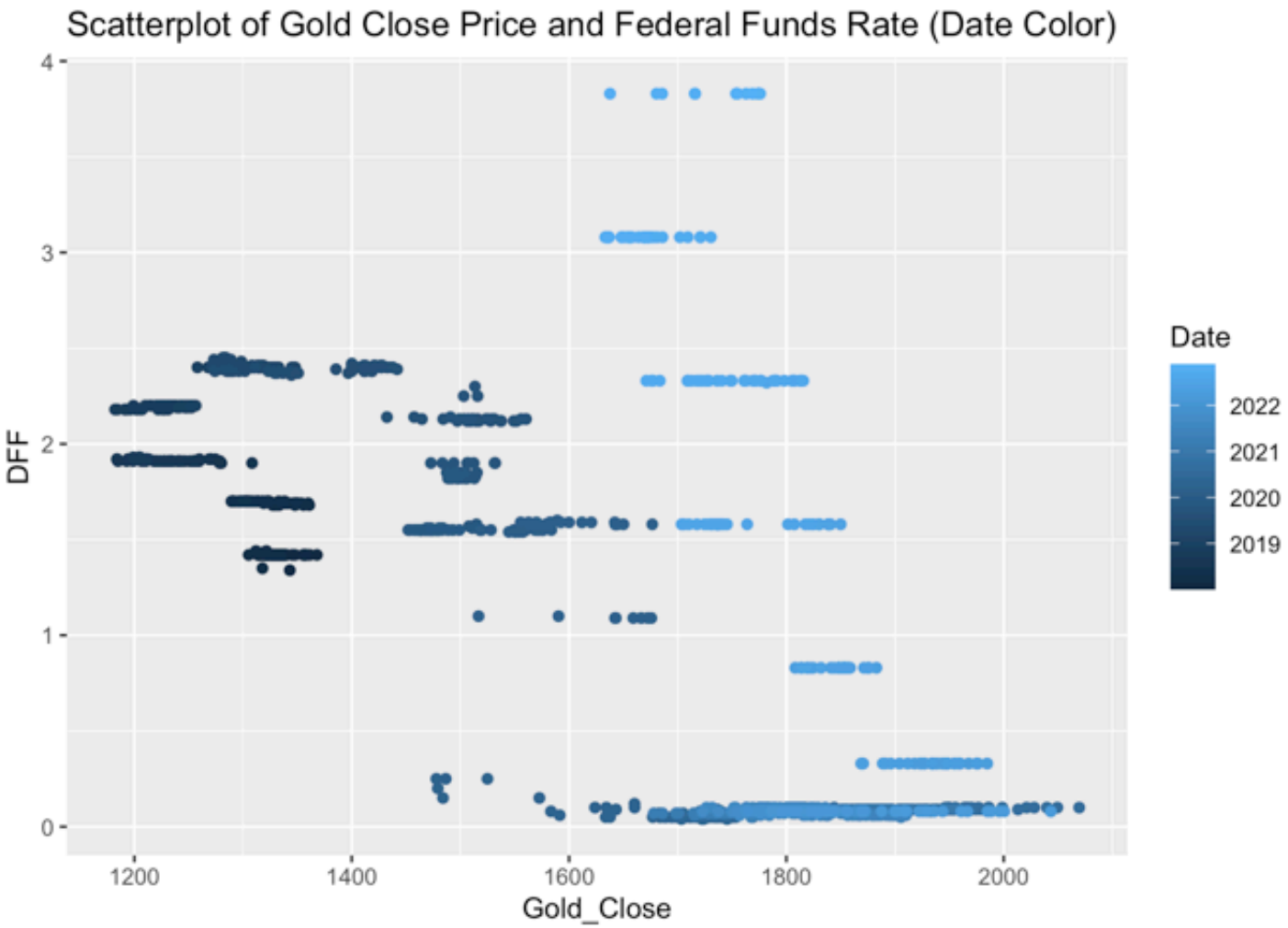
```
#normalize variables to view scaled relationships and add into full_df
full_df_Norm <- full_df
full_df_Norm[-1] <- lapply(full_df_Norm[-1], scale)
```

EDA with non-normalized varialbes

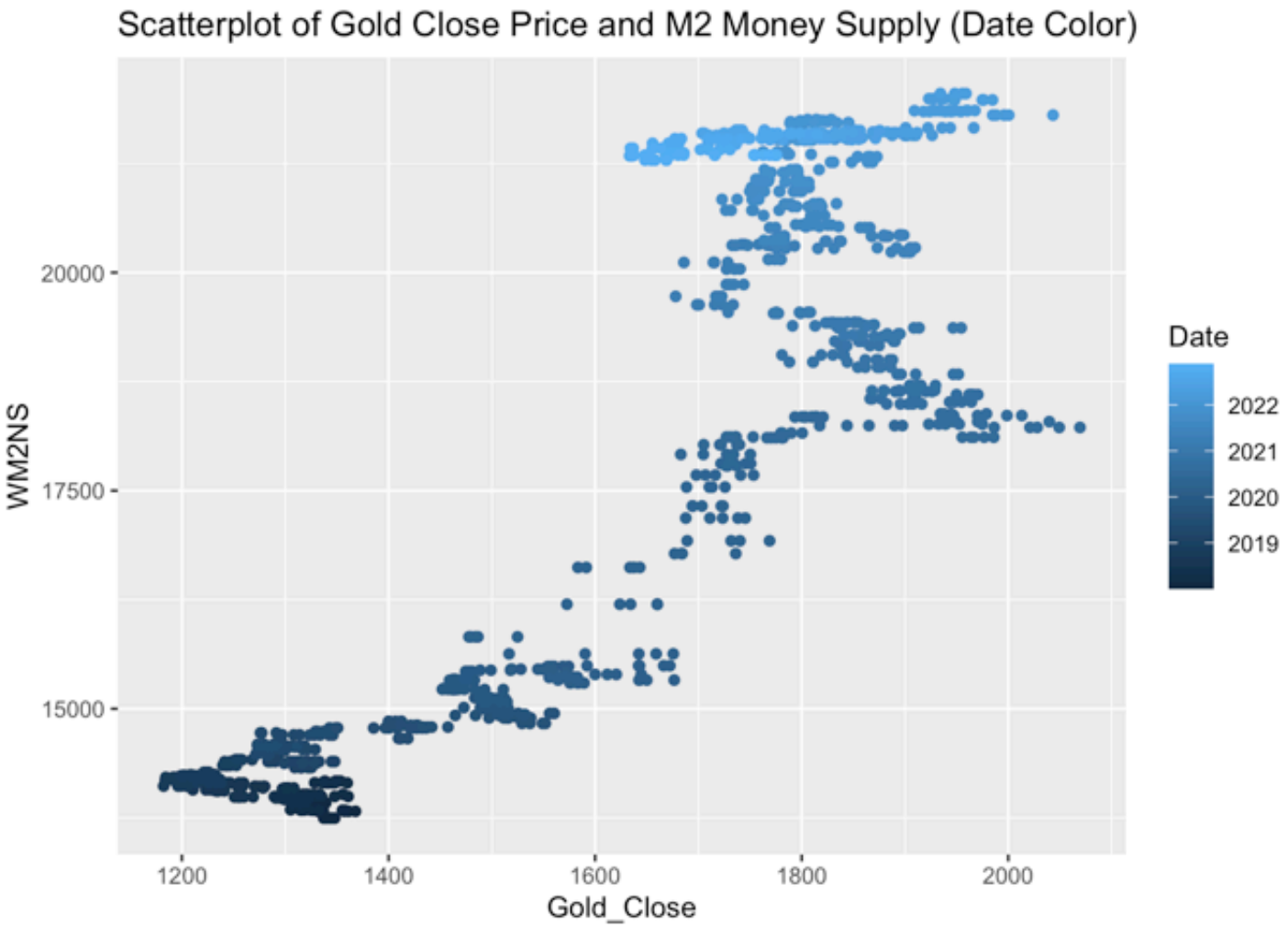
```
#scatterplot of gold & each external variable

ggplot(data = full_df) +
  geom_point(mapping = aes(x = Gold_Close,
                           y = UNRATE, color = Date)) +
  ggtitle("Scatterplot of Gold Close Price and Unemployment Rate (Date Color)")
```



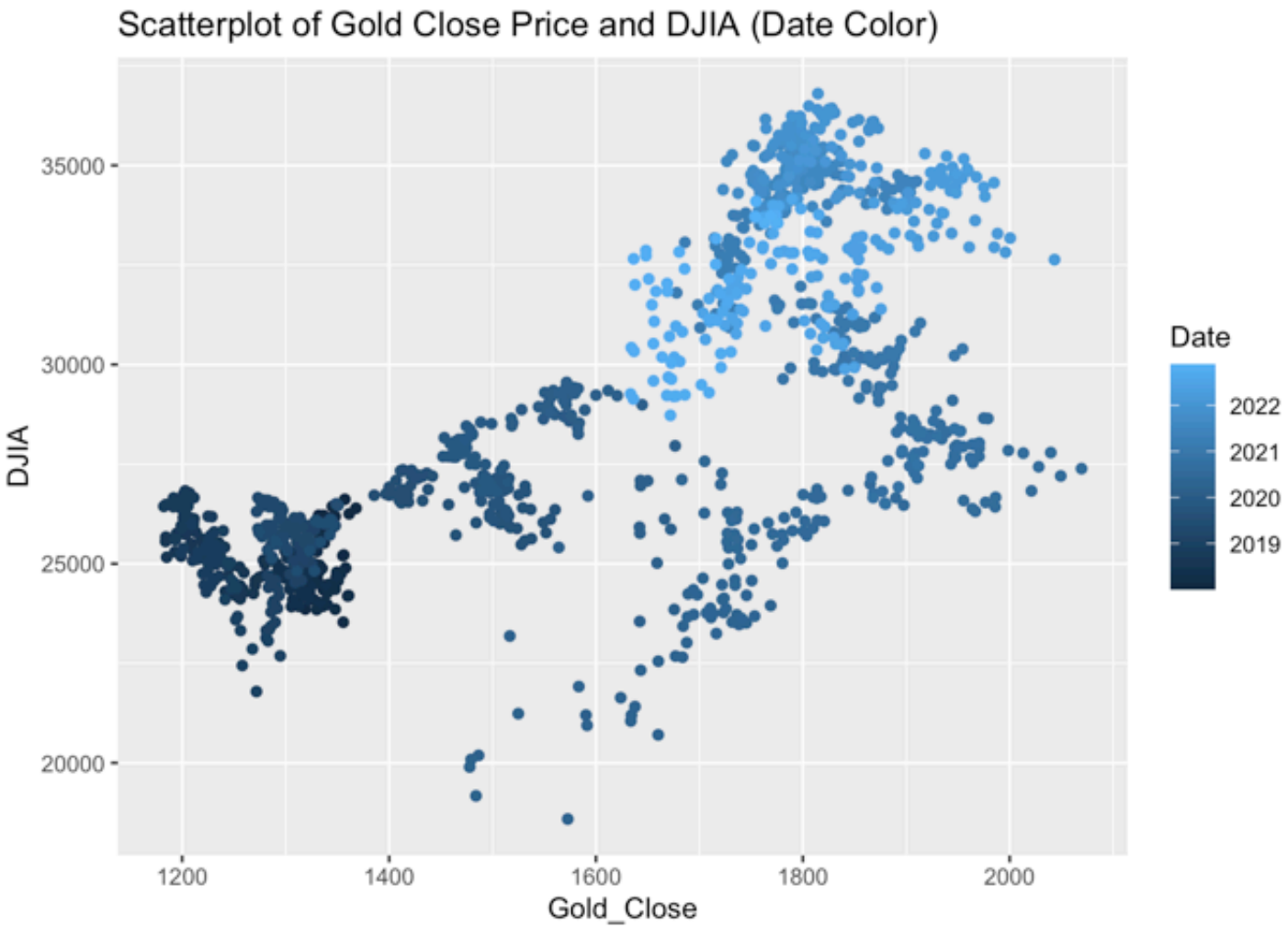

```
ggplot(data = full_df) +
  geom_point(mapping = aes(x = Gold_Close,
                           y = DFF, color = Date)) +
  ggtitle("Scatterplot of Gold Close Price and Federal Funds Rate (Date Color)")
```



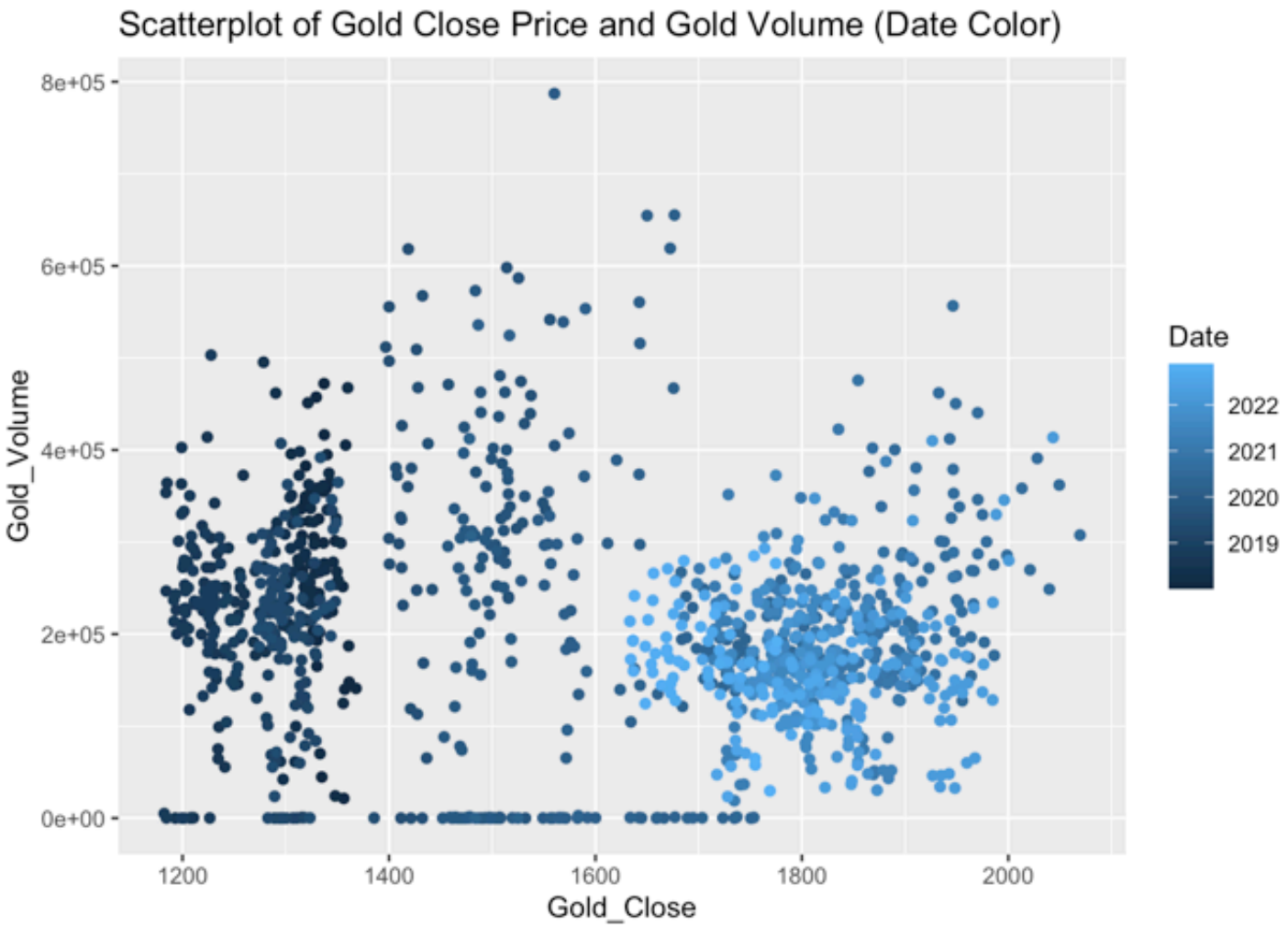

```
ggplot(data = full_df) +
  geom_point(mapping = aes(x = Gold_Close,
                           y = WM2NS, color = Date)) +
  ggtitle("Scatterplot of Gold Close Price and M2 Money Supply (Date Color)")
```



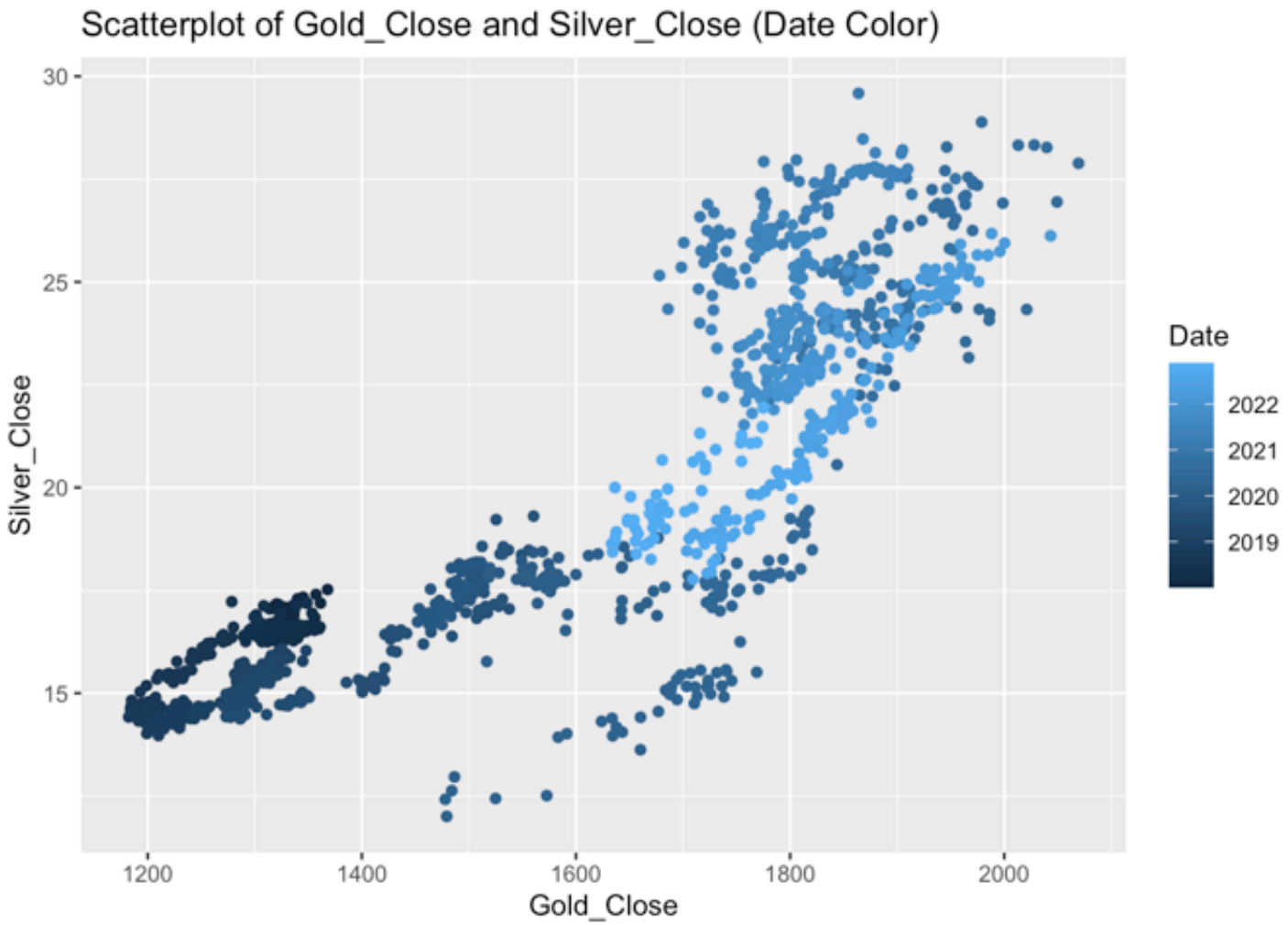

```
ggplot(data = full_df) +
  geom_point(mapping = aes(x = Gold_Close,
                           y = DJIA, color = Date)) +
  ggtitle("Scatterplot of Gold Close Price and DJIA (Date Color)")
```



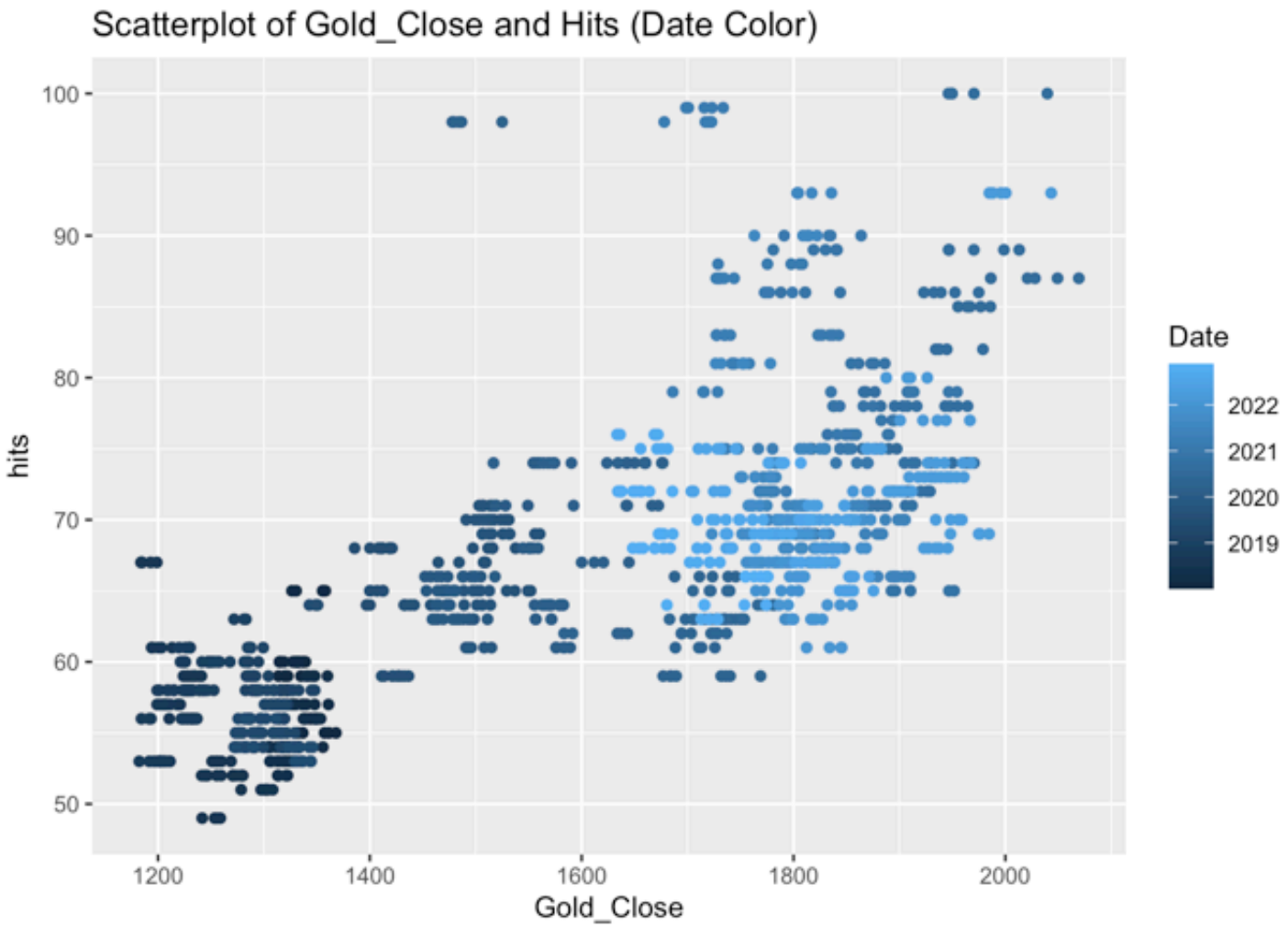

```
ggplot(data = full_df) +
  geom_point(mapping = aes(x = Gold_Close,
                           y = Gold_Volume, color = Date)) +
  ggtitle("Scatterplot of Gold Close Price and Gold Volume (Date Color)")
```




```
ggplot(data = full_df) +
  geom_point(mapping = aes(x = Gold_Close,
                           y = Silver_Close, color = Date)) +
  ggtitle("Scatterplot of Gold_Close and Silver_Close (Date Color)")
```




```
ggplot(data = full_df) +
  geom_point(mapping = aes(x = Gold_Close,
                           y = hits, color = Date)) +
  ggtitle("Scatterplot of Gold_Close and Hits (Date Color)")
```

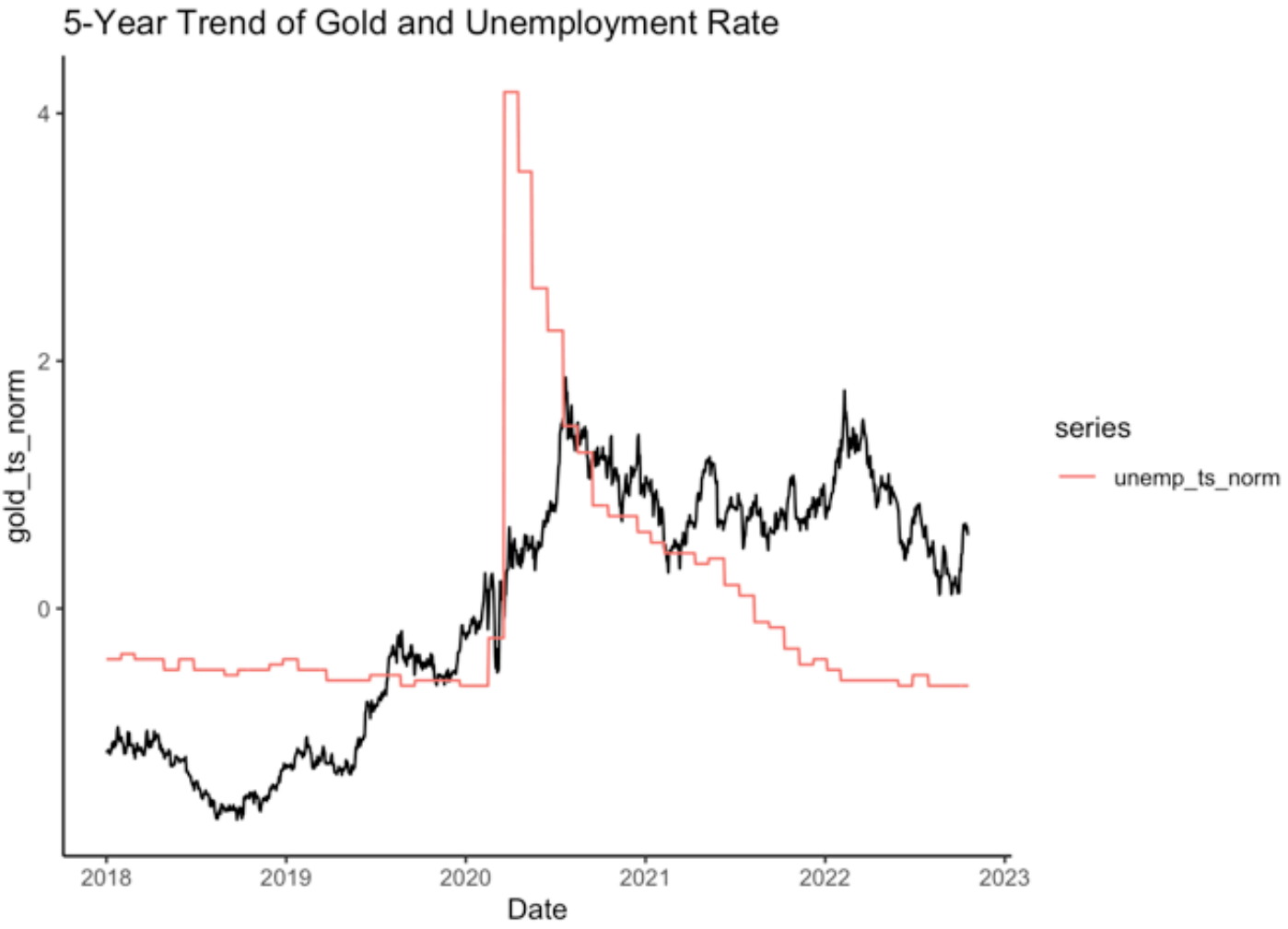


Create dataframe for Normalize EDA (on same axis)

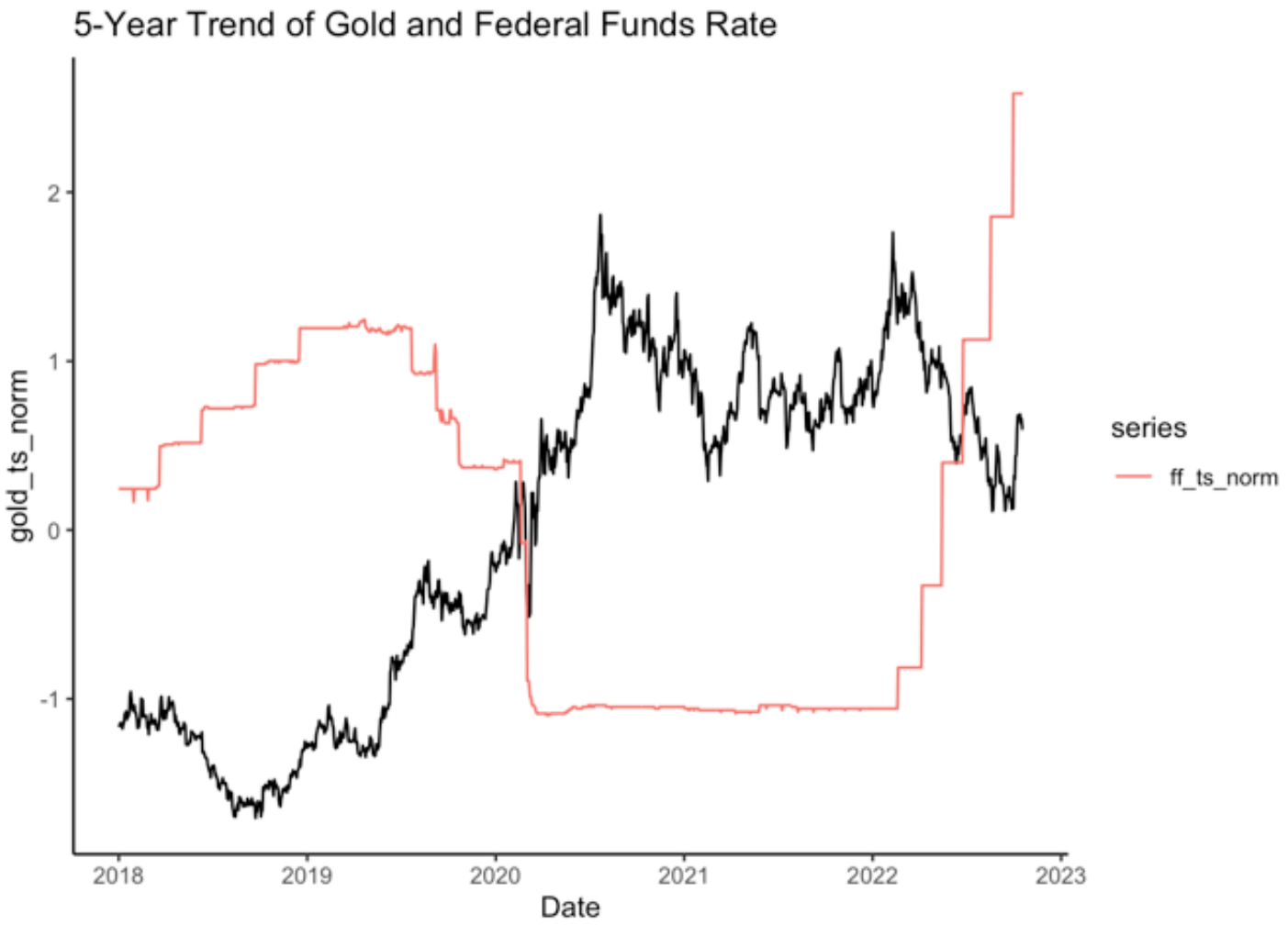
```
#create time series objects for each of the normalized data elements
gold_ts_norm <- ts(full_df_Norm$Gold_Close, start = c(2018, 01), frequency = 252)
unemp_ts_norm <- ts(full_df_Norm$UNRATE, start = c(2018, 01), frequency = 252)
ff_ts_norm <- ts(full_df_Norm$DFF, start = c(2018, 01), frequency = 252)
M2_ts_norm <- ts(full_df_Norm$WM2NS, start = c(2018, 01), frequency = 252)
dj_ts_norm <- ts(full_df_Norm$DJIA, start = c(2018, 01), frequency = 252)
silver_ts_norm <- ts(full_df_Norm$Silver_Close, start = c(2018, 01), frequency = 252)
hits_ts_norm <- ts(full_df_Norm$hits, start = c(2018, 01), frequency = 252)
gold_vol_ts_norm <- ts(full_df_Norm$Gold_Volume, start = c(2018, 01), frequency = 252
)
```

Create time series plots of all factors

```
#line chart of all indicators
autoplot(gold_ts_norm, main = "5-Year Trend of Gold and Unemployment Rate", xlab = "D
ate") +
  autolayer(unemp_ts_norm) + theme_classic()
```

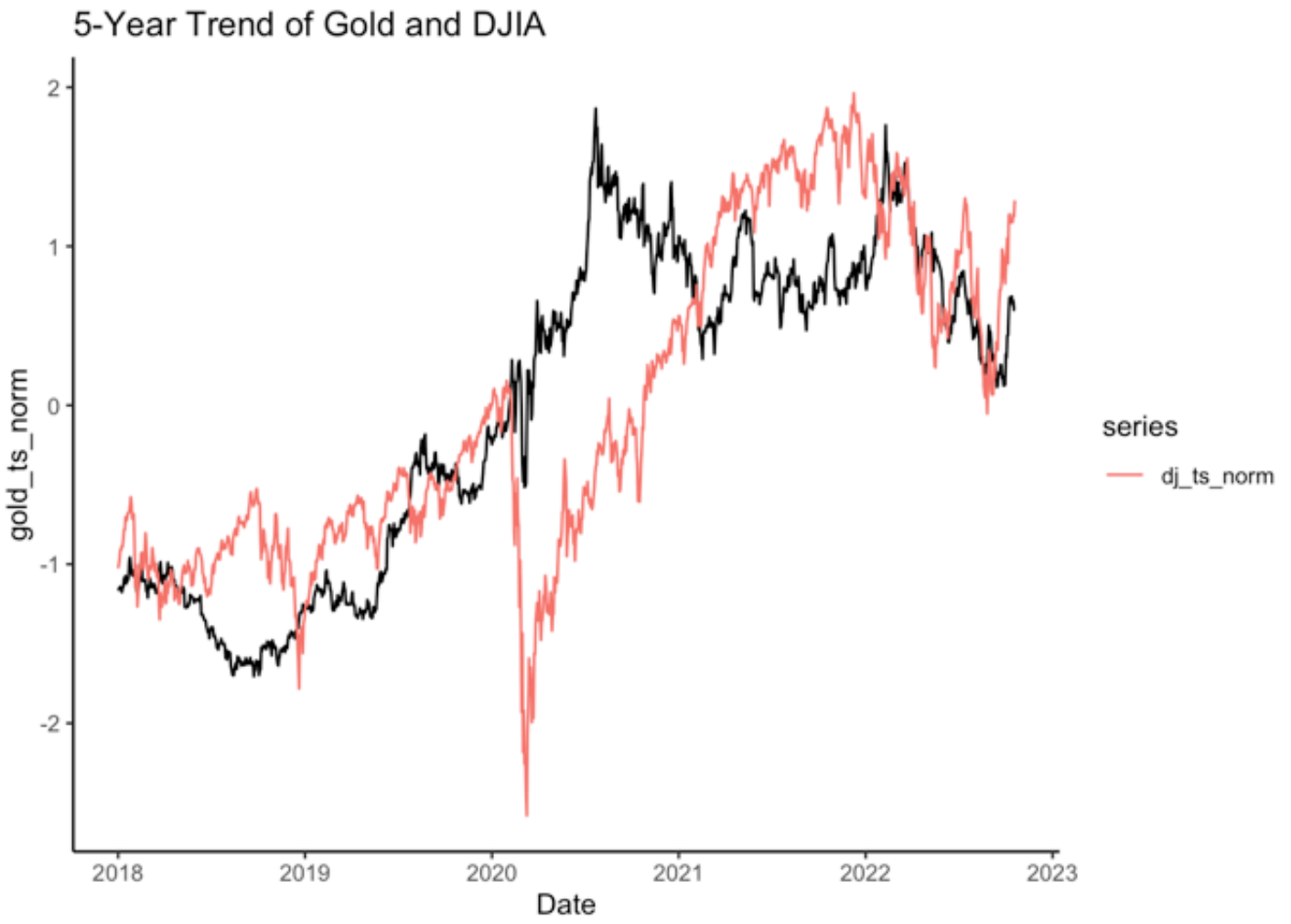



```
autoplot(gold_ts_norm, main = "5-Year Trend of Gold and Federal Funds Rate", xlab = "
Date") +
  autolayer(ff_ts_norm) + theme_classic()
```



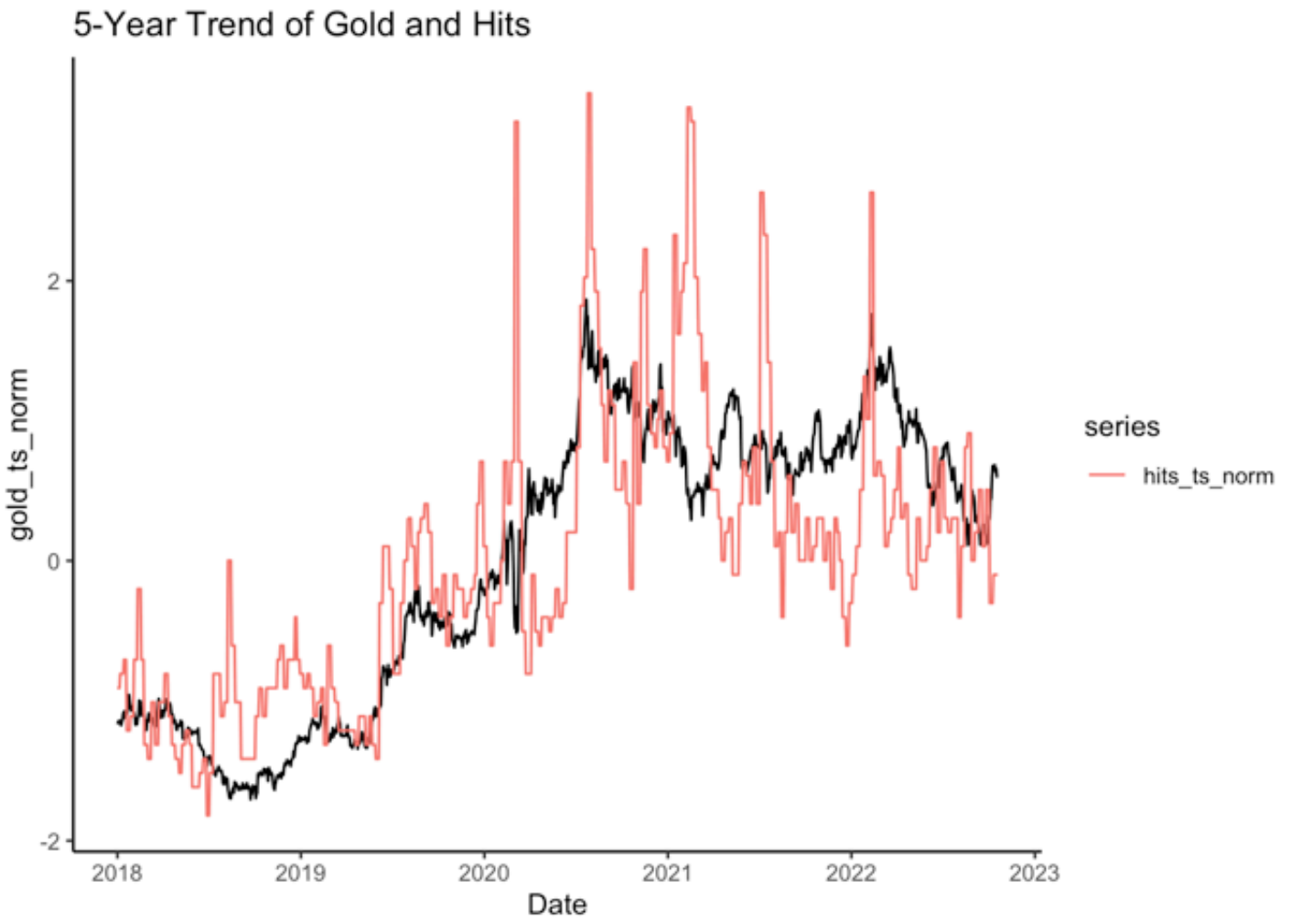

```
autoplot(gold_ts_norm, main = "5-Year Trend of Gold Close and M2", xlab = "Date") +
  autolayer(M2_ts_norm) + theme_classic()
```


```
autoplot(gold_ts_norm, main = "5-Year Trend of Gold and DJIA", xlab = "Date") +
  autolayer(dj_ts_norm) + theme_classic()
```

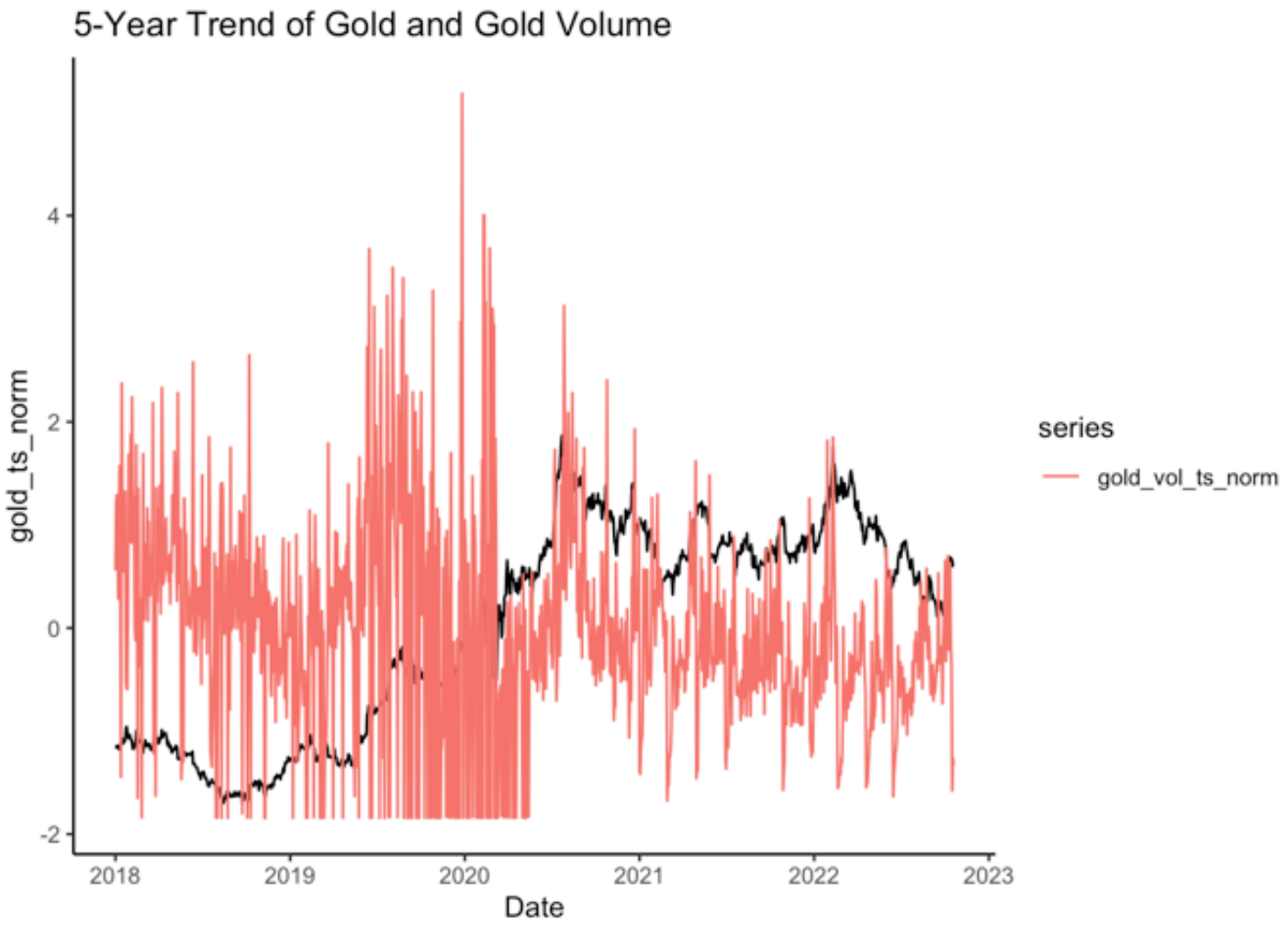



```
autoplot(gold_ts_norm, main = "5-Year Trend of Gold and Silver Close", xlab = "Date")
+
  autolayer(silver_ts_norm) + theme_classic()
```


```
autoplot(gold_ts_norm, main = "5-Year Trend of Gold and Hits", xlab = "Date") +
  autolayer(hits_ts_norm) + theme_classic()
```

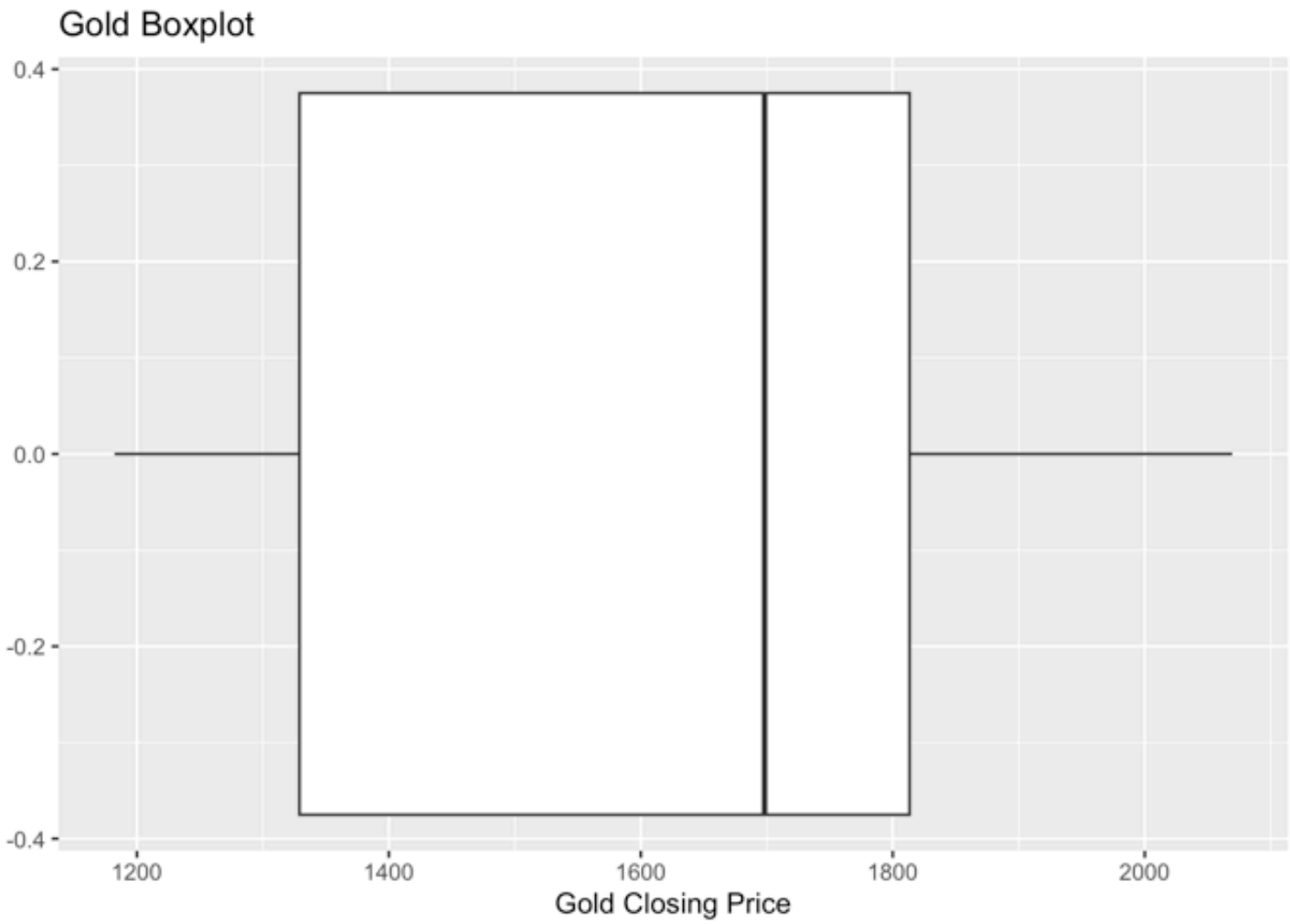



```
autoplot(gold_ts_norm, main = "5-Year Trend of Gold and Gold Volume", xlab = "Date")
+
  autolayer(gold_vol_ts_norm) + theme_classic()
```

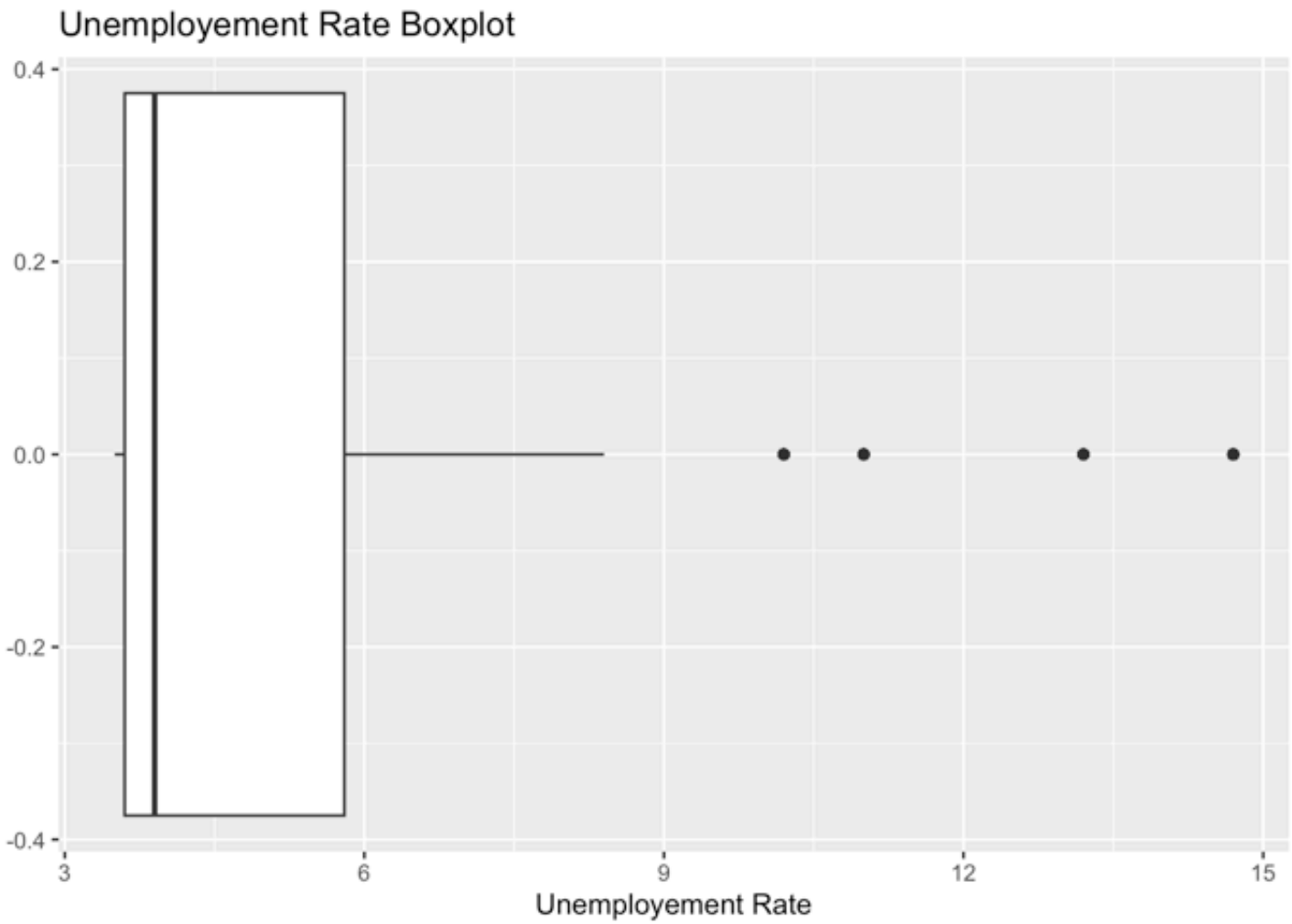


Boxplot to identify outlier variables

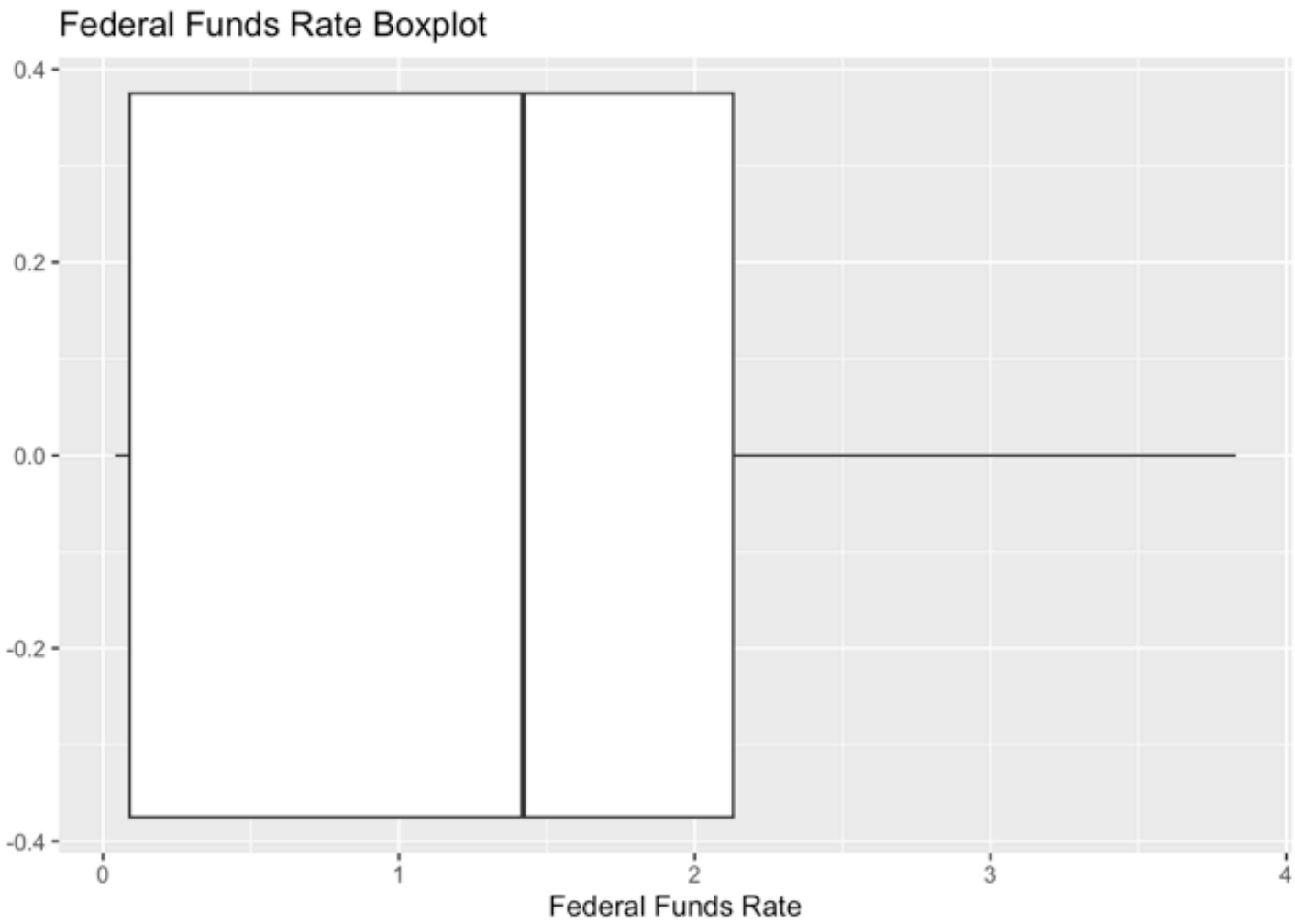
```
#boxplot of all varialbes
ggplot(corr_df, aes(x=Gold_Close)) +
  geom_boxplot() + ggtitle("Gold Boxplot") + xlab("Gold Closing Price")
```



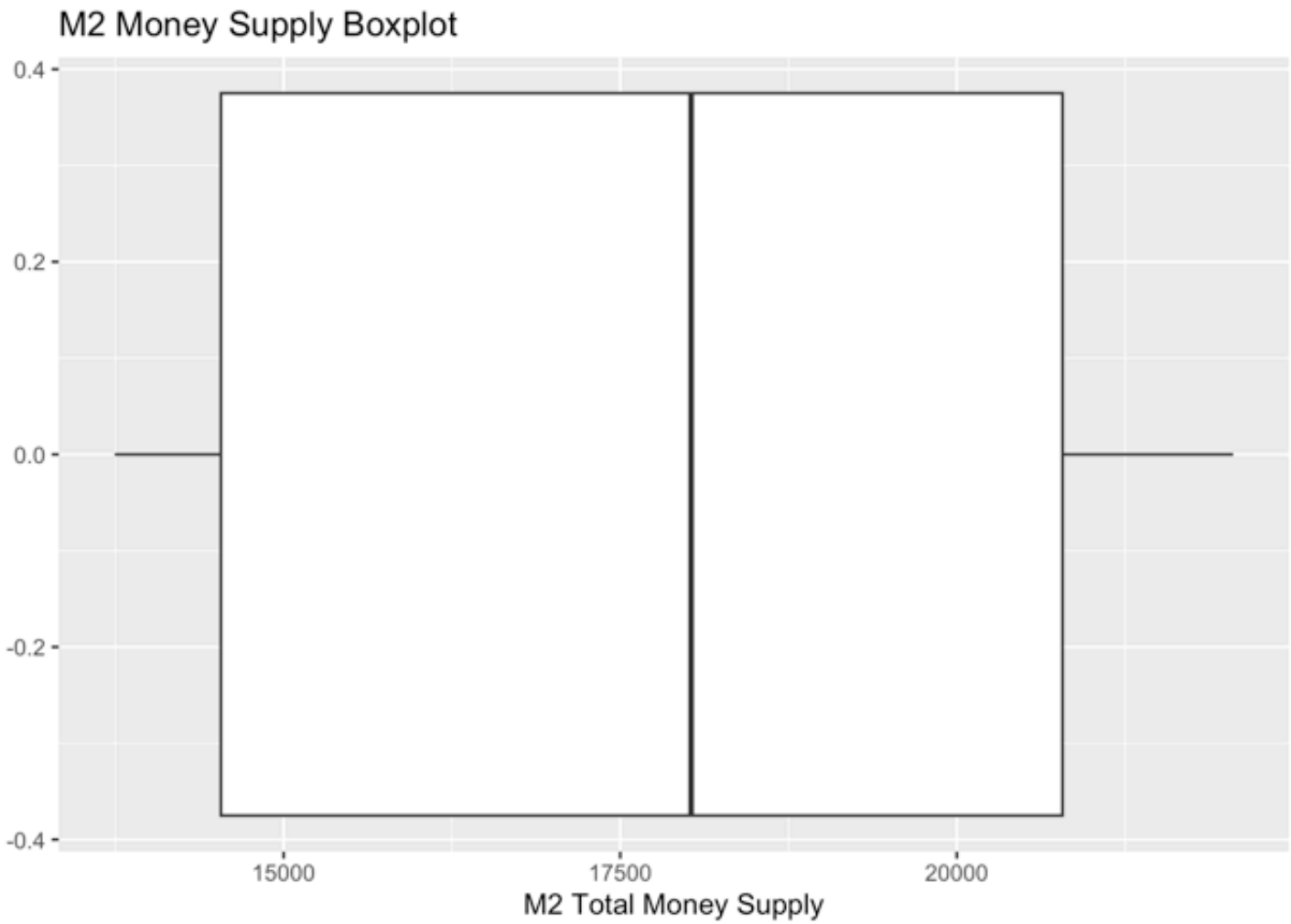

```
ggplot(corr_df, aes(x=UNRATE)) +
  geom_boxplot() + ggtitle("Unemployement Rate Boxplot") + xlab("Unemployement Rate")
```



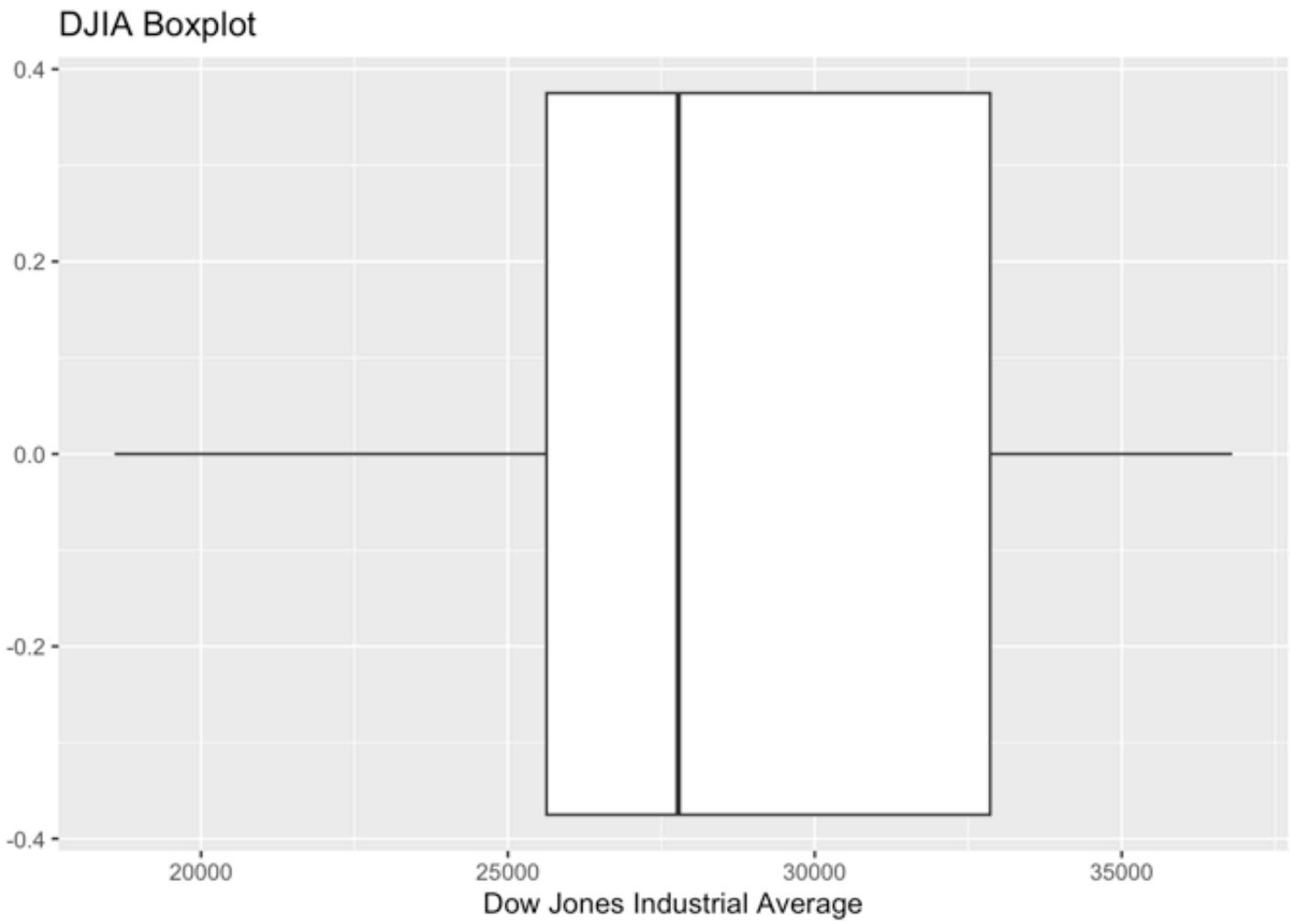

```
ggplot(corr_df, aes(x=DFF)) +
  geom_boxplot() + ggtitle("Federal Funds Rate Boxplot") + xlab("Federal Funds Rate")
```



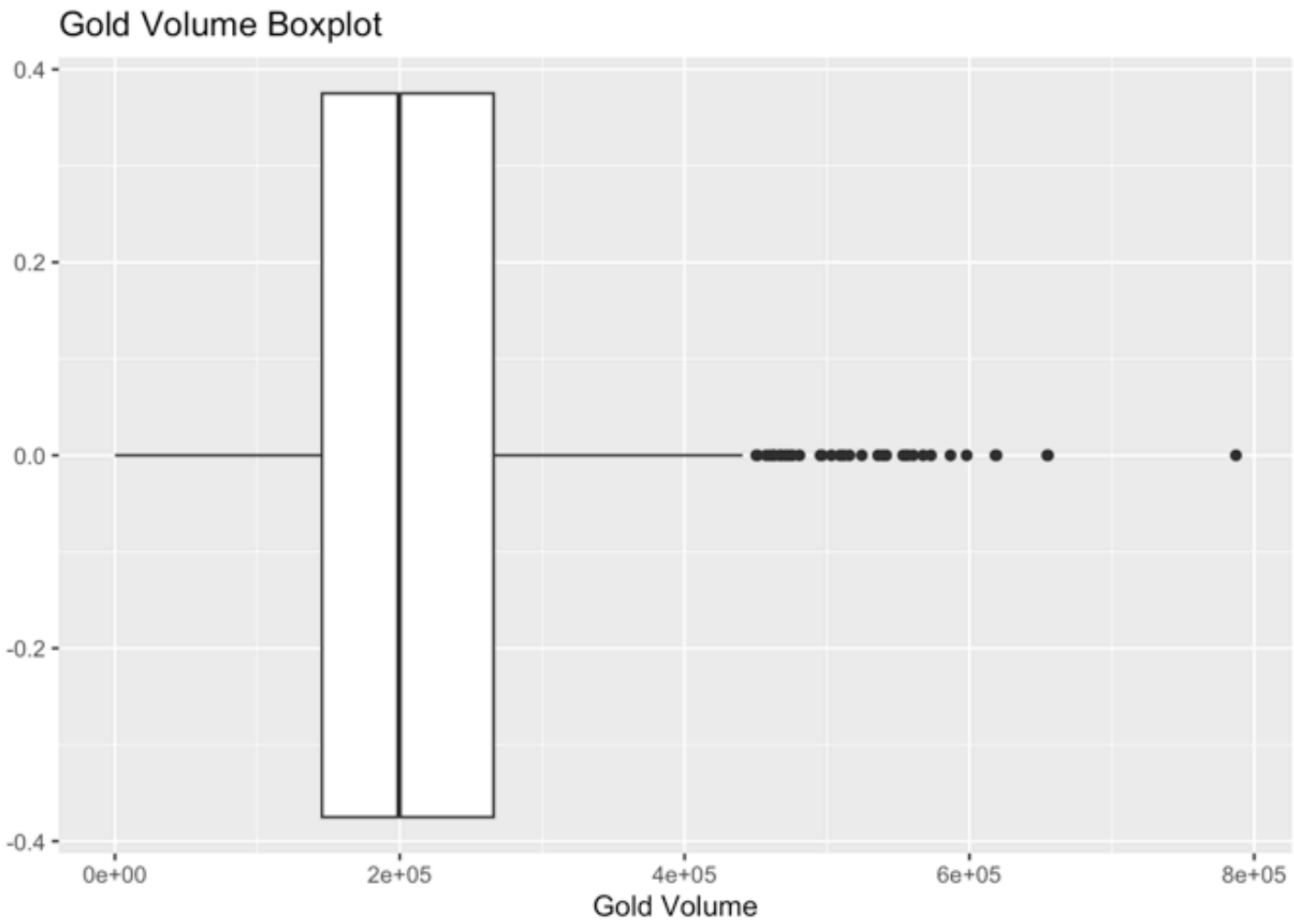

```
ggplot(corr_df, aes(x=WM2NS)) +
  geom_boxplot() + ggtitle("M2 Money Supply Boxplot") + xlab("M2 Total Money Supply")
```



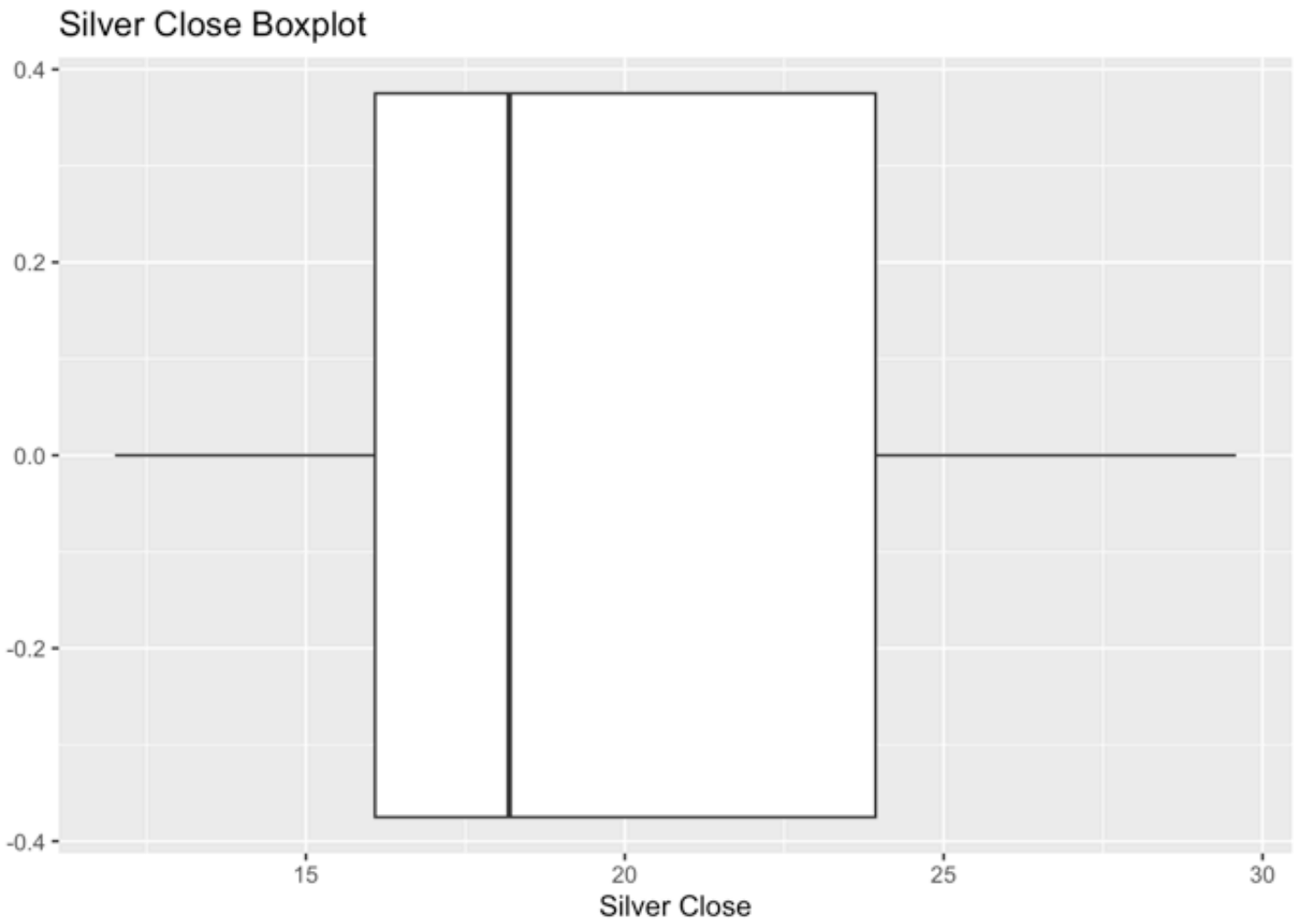

```
ggplot(corr_df, aes(x=DJIA)) +
  geom_boxplot() + ggtitle("DJIA Boxplot") + xlab("Dow Jones Industrial Average")
```



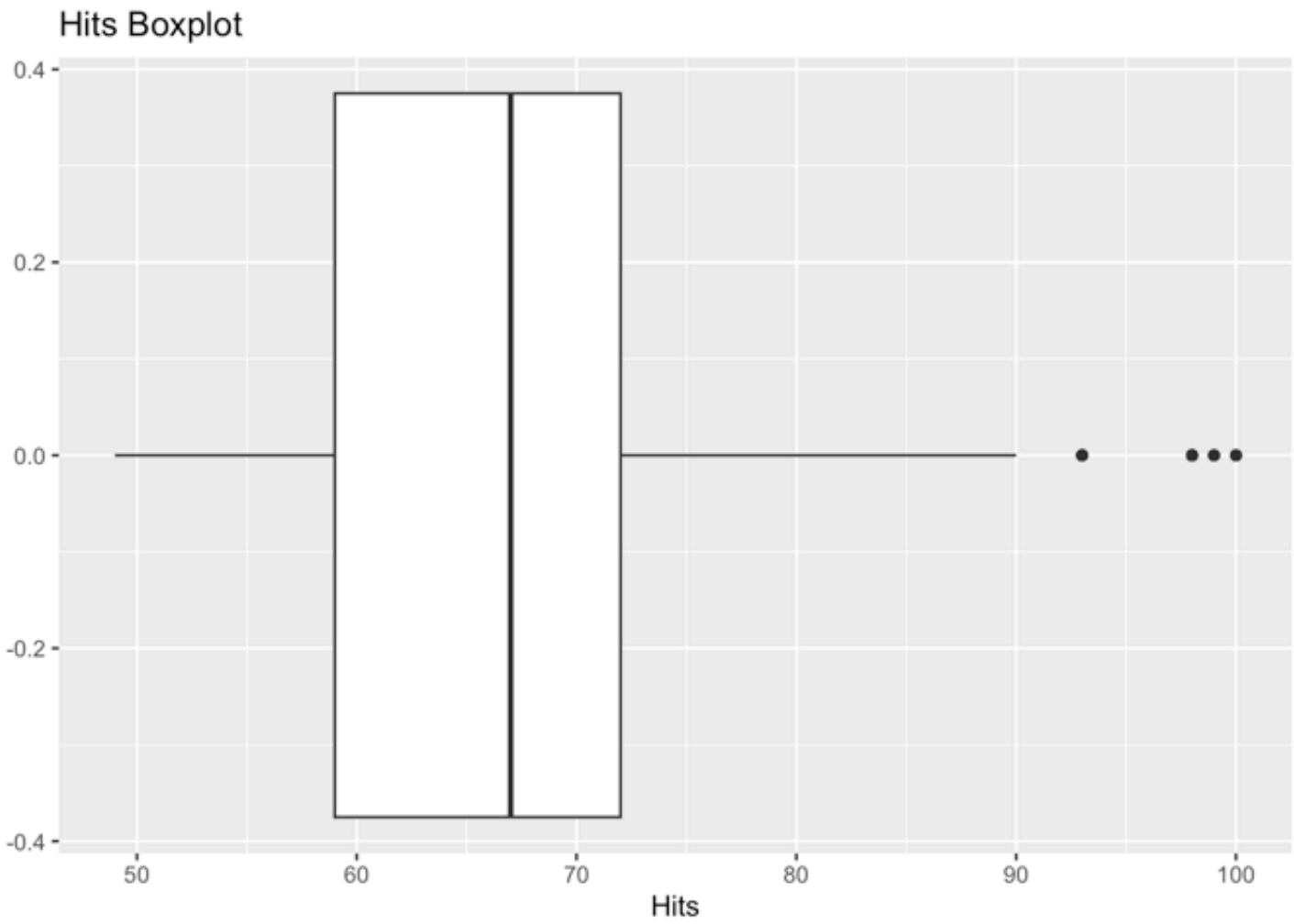

```
ggplot(corr_df, aes(x=Gold_Volume)) +
  geom_boxplot() + ggtitle("Gold Volume Boxplot") + xlab("Gold Volume")
```




```
ggplot(corr_df, aes(x=Silver_Close)) +
  geom_boxplot() + ggtitle("Silver Close Boxplot") + xlab("Silver Close")
```




```
ggplot(corr_df, aes(x=hits)) +
  geom_boxplot() + ggtitle("Hits Boxplot") + xlab("Hits")
```



Full entire Dataset

```
FULL_df  <- left_join(full_df, full_df_Norm, by="Date")
```

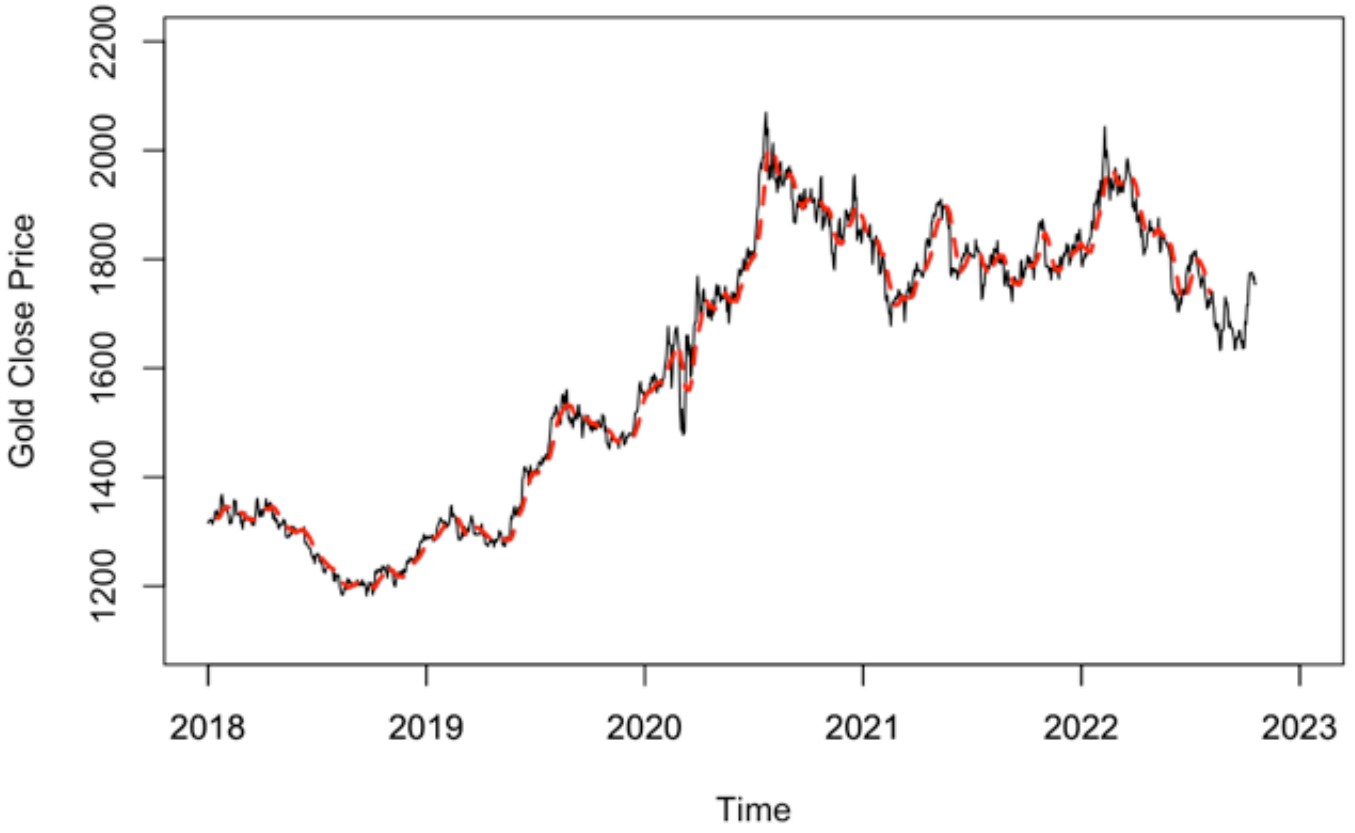
Partition the data into training and testing

```
#create the time series object for the FULL_df
ts_FULL_df <- ts(data=FULL_df$Gold_Close.x, start = c(2018,01), frequency = 252)
train_df <- window(ts_FULL_df, end = c(2022,150)) #161 is end of Aug
test_df <-  window(ts_FULL_df, start = c(2022,151))
#validate the train and test layers connect
autoplot(train_df) + autolayer(test_df) +
  #coord_cartesian(xlim = c(2022,2023)) +
  labs(title = "2022 - 2023 Gold Close Price (Train & Test)",
       x = "Time", y = "Gold Close Price")
```


```
test_df
```

```
## Time Series:
## Start = c(2022, 151)
## End = c(2022, 202)
## Frequency = 252
##  [1] 1728.6 1740.6 1717.4 1709.1 1677.3 1683.5 1671.1 1675.7 1681.1 1655.6
## [11] 1633.4 1636.2 1670.0 1668.6 1672.0 1702.0 1730.5 1720.8 1720.9 1709.3
## [21] 1675.2 1686.0 1677.5 1672.9 1672.9 1664.0 1655.5 1634.2 1636.8 1656.3
## [31] 1654.1 1658.0 1669.2 1668.8 1648.3 1648.3 1636.4 1651.0 1637.7 1685.7
## [41] 1680.5 1716.0 1715.8 1753.7 1774.2 1774.2 1774.7 1775.8 1763.0 1769.0
## [51] 1754.6 1754.8
```

```
#moving average model
nValid <- 12
ma.trailing <- rollmean(train_df, k = 12, align = "right")
last.ma <- tail(ma.trailing, 1)
ma.trailingTEST <- ts(rep(last.ma, nValid), test_df, freq = 12)

plot(train_df, ylim = c(1100, 2200), ylab = "Gold Close Price", xlab = "Time", xlim =
c(2018, 2023), main = "")
axis(1, at = seq(2018, 2023, 1), labels = format(seq(2018, 2023, 1)))
lines(ma.trailing, lwd = 2, col = "red", lty = 2)
lines(ma.trailingTEST, lwd = 2, col = "blue", lty = 2)
lines(test_df)
```




```
summary(ma.trailingTEST)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##    1737    1737    1737    1737    1737    1737
```

```
#Simple Moving Average with QUANTMOD
getSymbols("GOLD", from = "2018-01-02")
```

```
## [1] "GOLD"
```

```
chartSeries(GOLD)
```


```
addSMA(50)
```


```
ma <- SMA(Cl(GOLD), 50) # ma of adjusted close
#accuracy(Cl(GOLD), ma)
```

Full entire Dataset

```
FULL_df  <- left_join(full_df, full_df_Norm, by="Date")
FULL_df %>% arrange(mdy(FULL_df$Date))
```

```
## Warning: All formats failed to parse. No formats found.
```

Date <date>	Silver_Close.x <dbl>	Gold_Close.x <dbl>	Gold_Volume.x <dbl>	DX... <dbl>	UNRAT... <dbl>	DF... <dbl>	WM2... <dbl>
2018-01-02	17.060	1316.1	269072.0	91.87	4.0	1.42	13962.7
2018-01-03	17.125	1318.5	342866.0	92.16	4.0	1.42	13962.7
2018-01-04	17.130	1321.6	350803.0	91.85	4.0	1.42	13962.7
2018-01-05	17.155	1322.3	322422.0	91.95	4.0	1.42	13962.7

2018-01-08	17.170	1320.4	238332.0	92.36	4.0	1.42	13969.3
2018-01-09	17.055	1313.7	321636.0	92.53	4.0	1.42	13969.3
2018-01-10	17.135	1319.3	382605.0	92.33	4.0	1.42	13969.3
2018-01-11	17.010	1322.5	254541.0	91.85	4.0	1.42	13969.3
2018-01-12	17.120	1334.9	44651.0	90.97	4.0	1.42	13969.3
2018-01-16	17.095	1337.1	472155.0	90.39	4.0	1.42	13920.3

1-10 of 1,210 rows | 1-10 of 19 columns Previous **1** 2 3 4 5 6 … 121 Next

```
FULL_df$difference <- c(NA, diff(FULL_df$Gold_Close.x))
FULL_df$diff_boolean <- ifelse(FULL_df$difference >= 0,1,0)
head(FULL_df)
```

	Date <date>	Silver_Close.x <dbl>	Gold_Close.x <dbl>	Gold_Volume.x <dbl>	DX… <dbl>	UNRAT… <dbl>	DF… <dbl>	WM2… <dbl>
1	2018-01-02	17.060	1316.1	269072	91.87	4	1.42	13962.7
2	2018-01-03	17.125	1318.5	342866	92.16	4	1.42	13962.7
3	2018-01-04	17.130	1321.6	350803	91.85	4	1.42	13962.7
4	2018-01-05	17.155	1322.3	322422	91.95	4	1.42	13962.7
5	2018-01-08	17.170	1320.4	238332	92.36	4	1.42	13969.3
6	2018-01-09	17.055	1313.7	321636	92.53	4	1.42	13969.3

6 rows | 1-10 of 22 columns

Partition the data into training and testing

```
#create the time series object for the FULL_df
ts_FULL_df <- ts(data=FULL_df$Gold_Close.x, start = c(2018,01), frequency = 252)

train_df <- window(ts_FULL_df, start = c(2018,01), end = c(2022,186)) #186 is Oct 31
test_df <-  window(ts_FULL_df, start = c(2022,187))

#validate the train and test layers connect
autoplot(train_df) + autolayer(test_df) +
  labs(title = "2022 - 2023 Gold Close Price (Train & Test)",
       x = "Time", y = "Gold Close Price")
```


```
head(train_df)
```

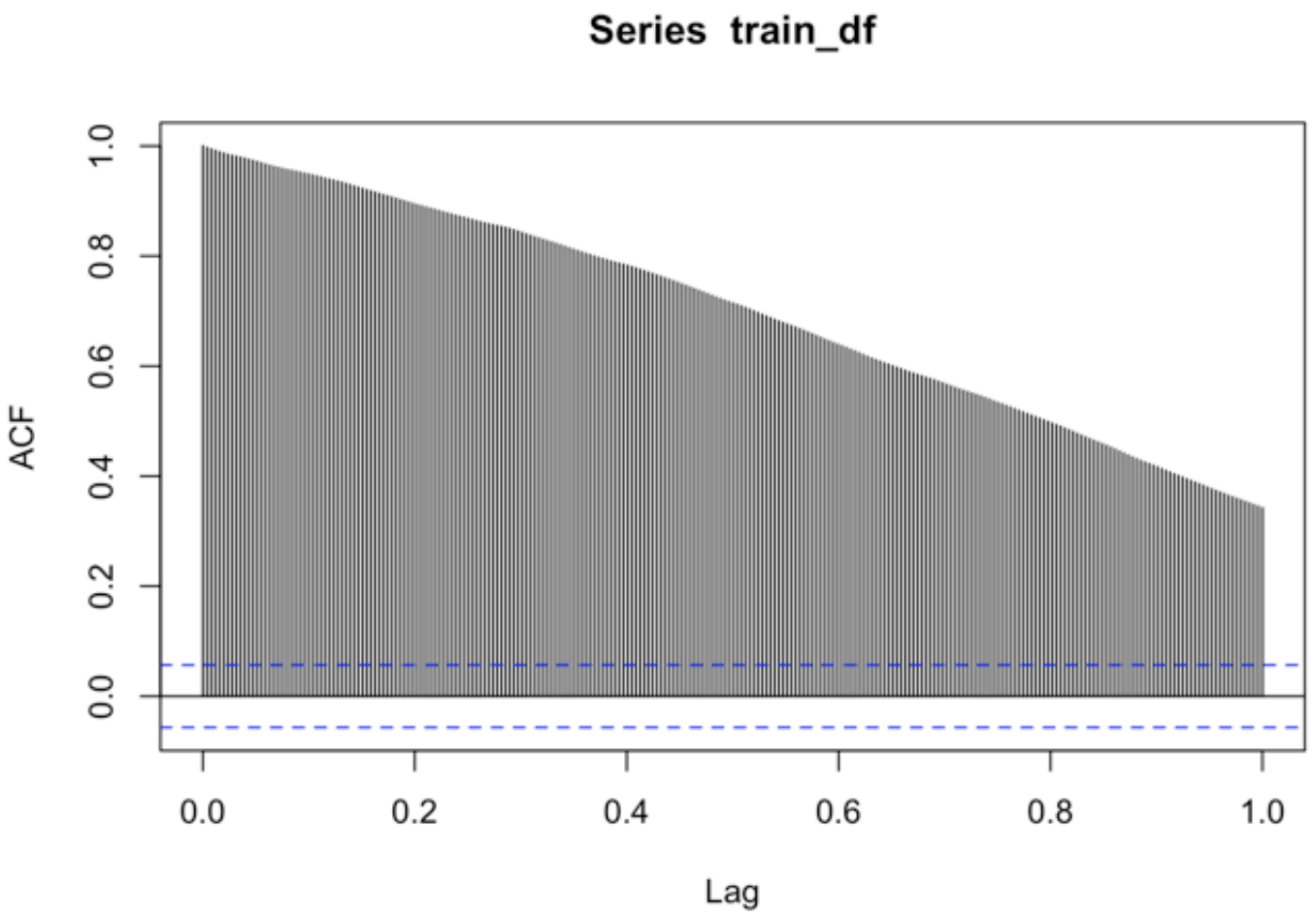
```
## Time Series:
## Start = c(2018, 1)
## End = c(2018, 6)
## Frequency = 252
## [1] 1316.1 1318.5 1321.6 1322.3 1320.4 1313.7
```

Auto ARIMA Model 1 (Gold Close Price Only) - Proof of Random Walk

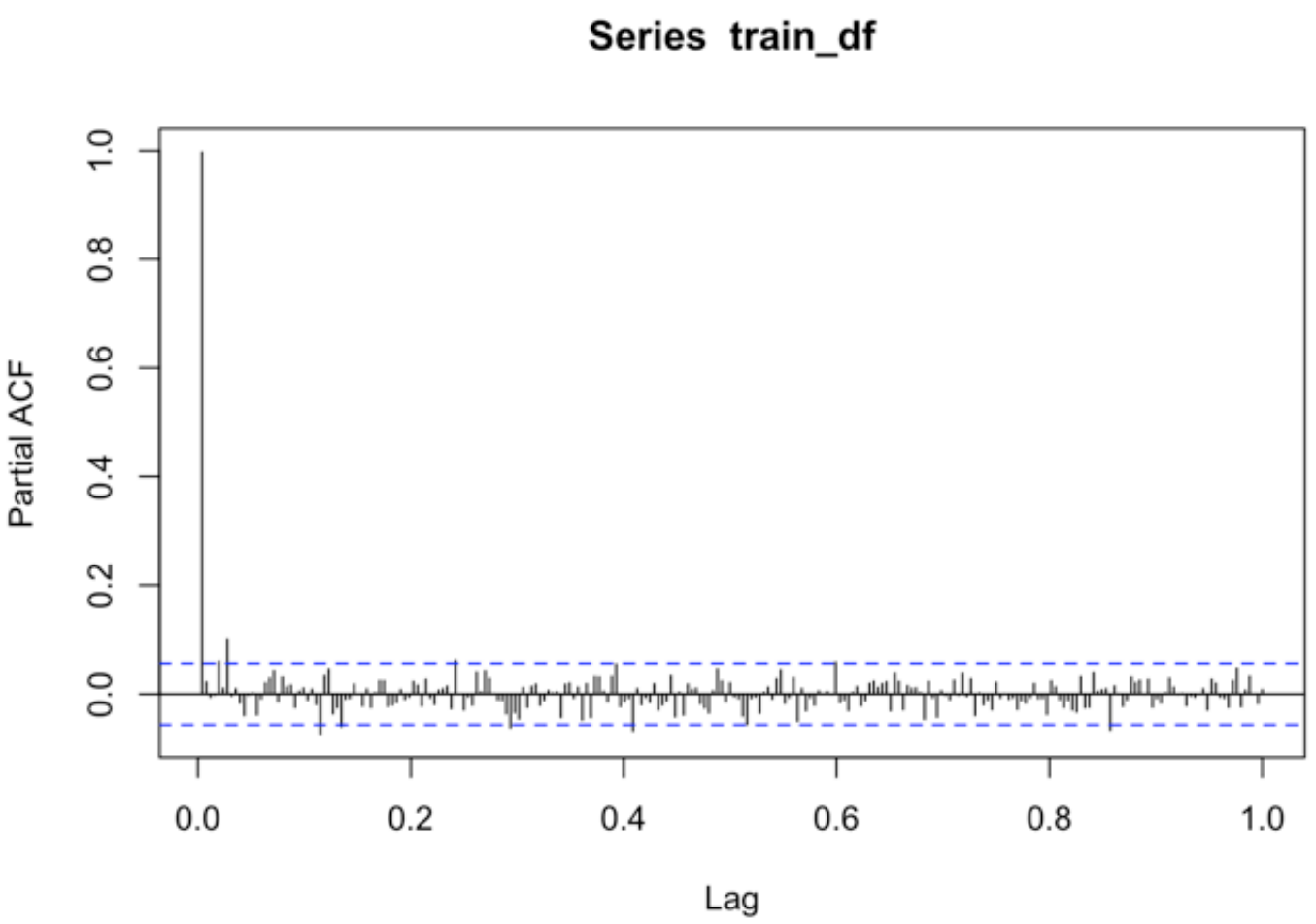
```
#ARIMA Model #1
auto_model <- auto.arima(train_df)
summary(auto_model)
```

```
## Series: train_df
## ARIMA(0,1,0)
##
## sigma^2 = 273.8:  log likelihood = -5040.64
## AIC=10083.28   AICc=10083.28   BIC=10088.36
##
## Training set error measures:
##                      ME    RMSE      MAE        MPE      MAPE       MASE
## Training set 0.2793267 16.5407 11.25923 0.01413727 0.6845585 0.05775767
##                      ACF1
## Training set -0.02760973
```

```
#investigate the potential parameters of the ARIMA model
acf(train_df, lag.max = 252)
```




```
pacf(train_df, lag.max = 252)
```




```
#fit to the test data
sqrt(mean(auto_model$residuals^2))
```

```
## [1] 16.5407
```

```
#accuracy(forecast(auto_model, h = 15), test_df)

gold_pred <- forecast(auto_model, h = 15)
#gold_pred

autoplot(train_df) + autolayer(test_df) + autolayer(gold_pred, alpha = 0.5) +coord_ca
rtesian(c(2022,2023))
```

Logistic Regression Model #1 (All Predictors)

```
#logistic regression model # 1

#columns to keep
keep_cols <- c("Date","Gold_Close.x","Silver_Close.x","DXY.x","UNRATE.x","DFF.x", "hi
ts.x","DJIA.x","WM2NS.x", "diff_boolean")
lr_df <- FULL_df[keep_cols]
head(lr_df)
```

	Date <date>	Gold_Close.x <dbl>	Silver_Close.x <dbl>	DX… <dbl>	UNRAT… <dbl>	DF… <dbl>	hits.x <dbl>	DJIA.x <dbl>	WM2… <dbl>
1	2018-01-02	1316.1	17.060	91.87	4	1.42	58	24824.01	13962.7
2	2018-01-03	1318.5	17.125	92.16	4	1.42	58	24922.68	13962.7
3	2018-01-04	1321.6	17.130	91.85	4	1.42	58	25075.13	13962.7
4	2018-01-05	1322.3	17.155	91.95	4	1.42	58	25295.87	13962.7
5	2018-01-08	1320.4	17.170	92.36	4	1.42	59	25283.00	13969.3
6	2018-01-09	1313.7	17.055	92.53	4	1.42	59	25385.80	13969.3

6 rows | 1-10 of 11 columns

```
#create train and test
lr_train <- lr_df[lr_df$Date <= "2022-10-31",]
lr_actual <- lr_df[lr_df$Date > "2022-10-31",]
lr_actual_gold <- lr_df[lr_df$Date > "2022-10-31",]
```

```
#create the logistic regression model with all predictors
set.seed(100)
gold_lr_model <- glm(diff_boolean ~  Silver_Close.x + DXY.x+ UNRATE.x + DFF.x + hits.
x + DJIA.x + WM2NS.x , lr_train,
                     family = "binomial")

#summarize the model output
summary(gold_lr_model)
```

```
##
## Call:
## glm(formula = diff_boolean ~ Silver_Close.x + DXY.x + UNRATE.x +
##     DFF.x + hits.x + DJIA.x + WM2NS.x, family = "binomial", data = lr_train)
##
## Deviance Residuals:
##    Min      1Q  Median      3Q     Max
## -1.406  -1.226   1.027   1.122   1.259
##
## Coefficients:
##                  Estimate Std. Error z value Pr(>|z|)
## (Intercept)     9.204e-01  2.527e+00   0.364    0.716
## Silver_Close.x -3.914e-02  4.187e-02  -0.935    0.350
## DXY.x          -2.132e-02  3.154e-02  -0.676    0.499
## UNRATE.x        4.921e-02  4.295e-02   1.146    0.252
## DFF.x           4.179e-02  1.644e-01   0.254    0.799
## hits.x          1.145e-02  9.546e-03   1.199    0.230
## DJIA.x          3.725e-05  4.506e-05   0.827    0.408
## WM2NS.x        -4.419e-06  8.841e-05  -0.050    0.960
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 1647.2  on 1192  degrees of freedom
## Residual deviance: 1642.5  on 1185  degrees of freedom
##   (1 observation deleted due to missingness)
## AIC: 1658.5
##
## Number of Fisher Scoring iterations: 4
```

```
gold_lr_model
```

```
##
## Call:  glm(formula = diff_boolean ~ Silver_Close.x + DXY.x + UNRATE.x +
##     DFF.x + hits.x + DJIA.x + WM2NS.x, family = "binomial", data = lr_train)
##
## Coefficients:
##    (Intercept)  Silver_Close.x           DXY.x        UNRATE.x           DFF.x
##      9.204e-01      -3.914e-02      -2.132e-02       4.921e-02       4.179e-02
##         hits.x          DJIA.x         WM2NS.x
##      1.145e-02       3.725e-05      -4.419e-06
##
## Degrees of Freedom: 1192 Total (i.e. Null);  1185 Residual
##   (1 observation deleted due to missingness)
## Null Deviance:       1647
## Residual Deviance: 1643  AIC: 1659
```

```
#generate the predictions of the lr model
gold_pred_prob <- predict(gold_lr_model, newdata = lr_actual, type = "response")
gold_pred_df <- cbind(lr_actual, gold_pred_prob)

#identify columns to keep and create the gold boolean flag
eval_cols <- c("Date","diff_boolean","gold_pred_prob")
eval_df <- gold_pred_df[eval_cols]
eval_df$gold_boolean <- ifelse(eval_df$gold_pred_prob >= .50,1,0)

eval_df <- cbind(eval_df, lr_actual_gold$Gold_Close.x)
eval_df
```

	Date	diff_boolean	gold_pred_prob	gold_boolean	lr_actual_gold$Gold_Clo
	<date>	<dbl>	<dbl>	<dbl>	<
1195	2022-11-01	0	0.5020122	1	16
1196	2022-11-02	1	0.5001512	1	16
1197	2022-11-03	0	0.5066160	1	16
1198	2022-11-04	1	0.5110586	1	16
1199	2022-11-07	0	0.4892661	0	16
1200	2022-11-08	1	0.4941486	0	17
1201	2022-11-09	0	0.4777125	0	17
1202	2022-11-10	1	0.5036066	1	17
1203	2022-11-11	1	0.5103247	1	17
1204	2022-11-14	1	0.5122070	1	17

```
#create the confusion matrix for the LR model with all predictors
confusionMatrix(data = as.factor(eval_df$diff_boolean), reference = as.factor(eval_df
$gold_boolean), positive = "1")
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction 0 1
##          0 2 4
##          1 1 9
##
##                Accuracy : 0.6875
##                  95% CI : (0.4134, 0.8898)
##     No Information Rate : 0.8125
##     P-Value [Acc > NIR] : 0.9373
##
##                   Kappa : 0.2593
##
##  Mcnemar's Test P-Value : 0.3711
##
##             Sensitivity : 0.6923
##             Specificity : 0.6667
##          Pos Pred Value : 0.9000
##          Neg Pred Value : 0.3333
##              Prevalence : 0.8125
##          Detection Rate : 0.5625
##    Detection Prevalence : 0.6250
##       Balanced Accuracy : 0.6795
##
##        'Positive' Class : 1
##
```

Logistic Regresson Model #2 (with highly correlated features)

```
#create the data frame of values with high correlation
high_corr_feat <- c("Date","Silver_Close.x","WM2NS.x","hits.x","DFF.x","diff_boolean"
)
lr_df2 <- FULL_df[high_corr_feat]
#head(lr_df2)
```

```
#create train and test sets
lr_train2 <- lr_df2[lr_df2$Date <= "2022-10-31",]
lr_actual2 <- lr_df2[lr_df2$Date > "2022-10-31",]
```

```
#create the logistic regression model with high only correlated predictors
set.seed(100)
#gold_lr_model2 <- glm(diff_boolean ~  Silver_Close.x  + DFF.x + hits.x + DJIA.x + WM
2NS.x , lr_train2,
#                     family = "binomial")
gold_lr_model2 <- glm(diff_boolean ~  Silver_Close.x   + WM2NS.x , lr_train2,
                    family = "binomial")
#summarize the model output
summary(gold_lr_model2)
```

```
##
## Call:
## glm(formula = diff_boolean ~ Silver_Close.x + WM2NS.x, family = "binomial",
##     data = lr_train2)
##
## Deviance Residuals:
##    Min      1Q  Median      3Q     Max
## -1.292  -1.236   1.081   1.119   1.154
##
## Coefficients:
##                  Estimate Std. Error z value Pr(>|z|)
## (Intercept)     4.055e-02  3.433e-01   0.118    0.906
## Silver_Close.x  1.420e-02  2.150e-02   0.661    0.509
## WM2NS.x        -9.784e-06  3.096e-05  -0.316    0.752
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 1647.2  on 1192  degrees of freedom
## Residual deviance: 1646.7  on 1190  degrees of freedom
##   (1 observation deleted due to missingness)
## AIC: 1652.7
##
## Number of Fisher Scoring iterations: 3
```

```
gold_lr_model2
```

```
##
## Call:  glm(formula = diff_boolean ~ Silver_Close.x + WM2NS.x, family = "binomial",
##     data = lr_train2)
##
## Coefficients:
##    (Intercept)  Silver_Close.x         WM2NS.x
##      4.055e-02       1.420e-02      -9.784e-06
##
## Degrees of Freedom: 1192 Total (i.e. Null);  1190 Residual
##   (1 observation deleted due to missingness)
## Null Deviance:       1647
## Residual Deviance: 1647  AIC: 1653
```

```
#generate the predictions of the lr_model2
gold_pred_prob2 <- predict(gold_lr_model2, newdata = lr_actual2, type = "response")
gold_pred_df2 <- cbind(lr_actual2, gold_pred_prob2)

#identify columns to keep and create the gold boolean flag
eval_cols2 <- c("Date","diff_boolean","gold_pred_prob2")
eval_df2 <- gold_pred_df2[eval_cols2]
eval_df2$gold_boolean <- ifelse(eval_df2$gold_pred_prob >= .50,1,0)

eval_df2
```

	Date	diff_boolean	gold_pred_prob2	gold_boolean
	<date>	<dbl>	<dbl>	<dbl>
1195	2022-11-01	0	0.5289099	1
1196	2022-11-02	1	0.5281313	1
1197	2022-11-03	0	0.5250861	1
1198	2022-11-04	1	0.5287861	1
1199	2022-11-07	0	0.5312804	1
1200	2022-11-08	1	0.5315634	1
1201	2022-11-09	0	0.5335788	1
1202	2022-11-10	1	0.5327657	1
1203	2022-11-11	1	0.5341444	1
1204	2022-11-14	1	0.5341091	1

1-10 of 16 rows Previous **1** 2 Next

The Logistic Regression model with only highly correlated features predicts all positive cases for the gold_boolean flag

```
#return the distinct values of gold Boolean
n_distinct(eval_df2$gold_boolean)
```

```
## [1] 1
```

Function to find local min and max for buy/sell signals

```
 # Locate Local Min and Max for buy/sell signals
locate_xtrem <- function (x, last = FALSE)
{
  # use rle to deal with duplicates
  x_rle <- rle(x)

  # force the first value to be identified as an extrema
  first_value <- x_rle$values[1] - x_rle$values[2]

  #
  # ! NOTE: with this method, last value will be considered as an extrema
  diff_sign_rle <- c(first_value, diff(x_rle$values)) %>% sign() %>% rle()

  # this vector will be used to get the initial positions
  diff_idx <- cumsum(diff_sign_rle$lengths)

  # find min and max
  diff_min <- diff_idx[diff_sign_rle$values < 0]
  diff_max <- diff_idx[diff_sign_rle$values > 0]

  # get the min and max indexes in the original series
  x_idx <- cumsum(x_rle$lengths)
  if (last) {
    min <- x_idx[diff_min]
    max <- x_idx[diff_max]
  } else {
    min <- x_idx[diff_min] - x_rle$lengths[diff_min] + 1
    max <- x_idx[diff_max] - x_rle$lengths[diff_max] + 1
  }
  # just get number of occurences
  min_nb <- x_rle$lengths[diff_min]
  max_nb <- x_rle$lengths[diff_max]

  # format the result as a tibble
  bind_rows(
    tibble(Idx = min, Values = x[min], NB = min_nb, Status = "min"),
    tibble(Idx = max, Values = x[max], NB = max_nb, Status = "max")) %>%
    arrange(.data$Idx) %>%
    mutate(Last = last) %>%
    mutate_at(vars(.data$Idx, .data$NB), as.integer)
}
```

GOLD Prophet FORECAST BASELINE (No External Regressors)

```
# Partition Data frame
prophet.Train <- subset(FULL_df, Date < as.Date("2022-11-08"))
Prophet.Test <- subset(FULL_df, Date >= as.Date("2022-11-17"))

# Set training col for prophet forecast
prophet.Train <- prophet.Train %>%
  select(c("Date","Gold_Close.x")) %>%
  rename(ds = Date, y = Gold_Close.x)

# Prophet Predictions
Prophet <- prophet(prophet.Train, interval.width = 0.95)
```

```
## Disabling daily seasonality. Run prophet with daily.seasonality=TRUE to override t
his.
```

```
future <- make_future_dataframe(Prophet, periods = 7) %>% filter(!wday(ds) %in% c(1,7
)) #account for regular gaps on weekends
Prophet_Forecast_base <- predict(Prophet, future)

# Grab necessary variables
Forecast_subset <- Prophet_Forecast_base %>%
  select(c('ds','yhat','yhat_lower','yhat_upper')) %>%
  rename(Date = ds, ClosePrice = yhat , ClosePrice_lower = yhat_lower, ClosePrice_upp
er = yhat_upper)

# Put into Dataframe
datatable(Forecast_subset[c('Date','ClosePrice','ClosePrice_lower','ClosePrice_upper'
)])
```

Show 10 entries

Search:

	Date	ClosePrice	ClosePrice_lower	ClosePrice_upper
1	2018-01-02T00:00:00Z	1311.63750243046	1247.93996694001	1381.01760672916
2	2018-01-03T00:00:00Z	1312.78369479746	1242.5987268899	1374.00124377786
3	2018-01-04T00:00:00Z	1314.3140227536	1251.07406546917	1383.05403809662
4	2018-01-05T00:00:00Z	1313.50038859346	1249.87168131471	1376.48932533478
5	2018-01-08T00:00:00Z	1315.26168274397	1250.27349150937	1383.04978865276
6	2018-01-09T00:00:00Z	1316.30627861412	1250.90511521725	1379.56533014966
7	2018-01-10T00:00:00Z	1316.63524291449	1249.75806123659	1384.19420117904

8	2018-01-11T00:00:00Z	1317.48061754534	1253.48420655579	1383.56520026427
9	2018-01-12T00:00:00Z	1316.11373602952	1251.79751614467	1378.72919901756
10	2018-01-16T00:00:00Z	1317.89995881398	1253.50883370456	1386.05506926296

Showing 1 to 10 of 1,204 entries Previous 1 2 3 4 5 … 121 Next

```
# Return prediction results from prophet forecast
Prophet_Results_base <- Prophet_Forecast_base %>%
  select(c("ds","yhat")) %>%
  rename(Date = ds, Close_Prediction = yhat)
Prophet_Results_base$Date <- as.Date(Prophet_Results_base$Date , format = "%m/%d/%y")
Prophet_Results_base <- subset(Prophet_Results_base, Date >= as.Date("2022-11-08"))

# Grab actual gold data
Gold_Results <- gold %>%
  select(c("Date","Gold_Close"))  %>%
  rename(Close_Actual = Gold_Close)

# Join prediction and actual results for comparison
results <- left_join(Prophet_Results_base, Gold_Results,  by="Date")
results
```

Date <date>	Close_Prediction <dbl>	Close_Actual <dbl>
2022-11-08	1676.151	1716.0
2022-11-09	1676.111	1715.8
2022-11-10	1676.556	1753.7
2022-11-11	1674.727	1774.2
2022-11-14	1673.528	1774.2

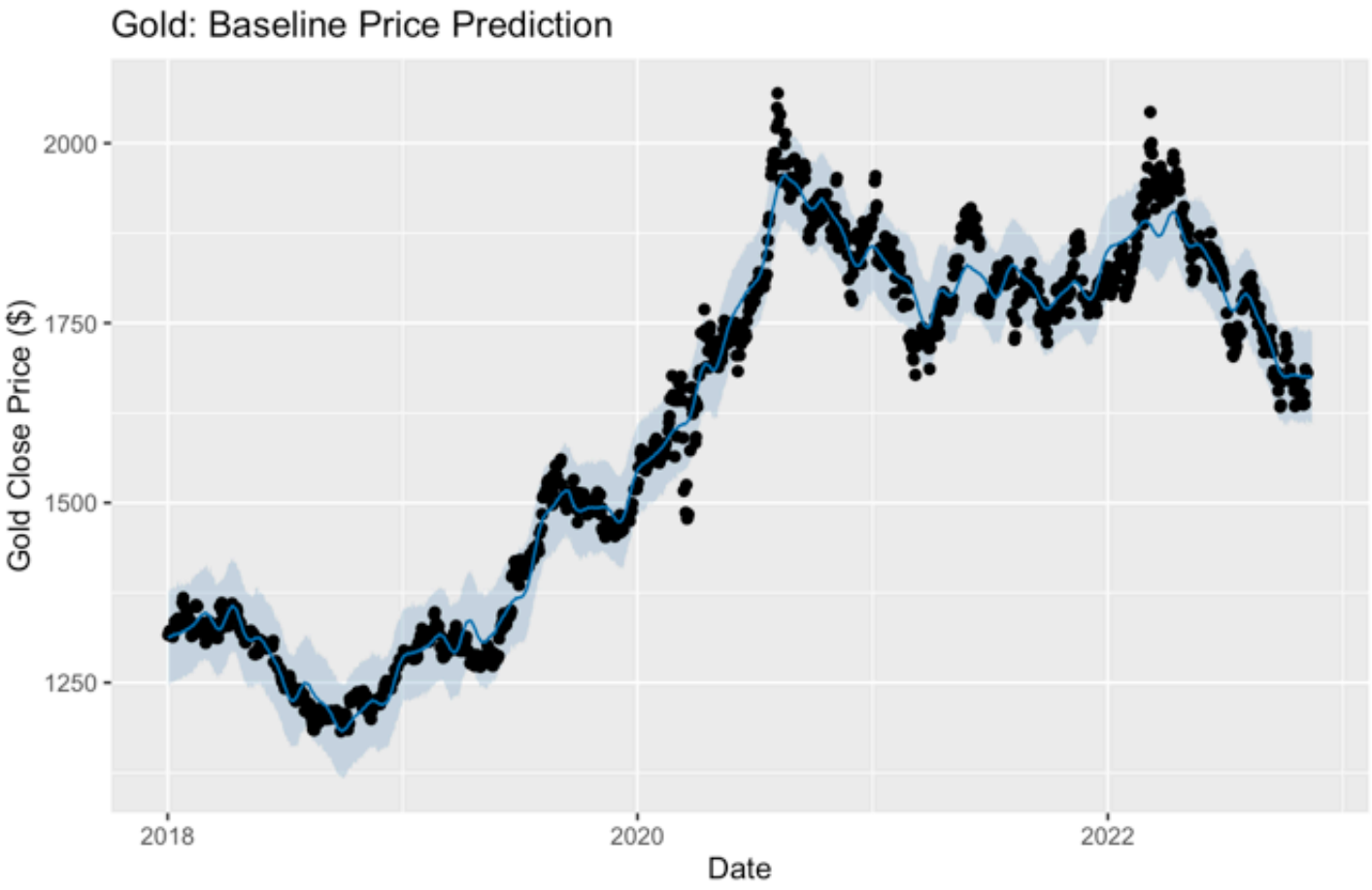
5 rows

```
# Calculate RMSE
RMSE <- sqrt(mean((results$Close_Actual - results$Close_Prediction)^2))
RMSE
```

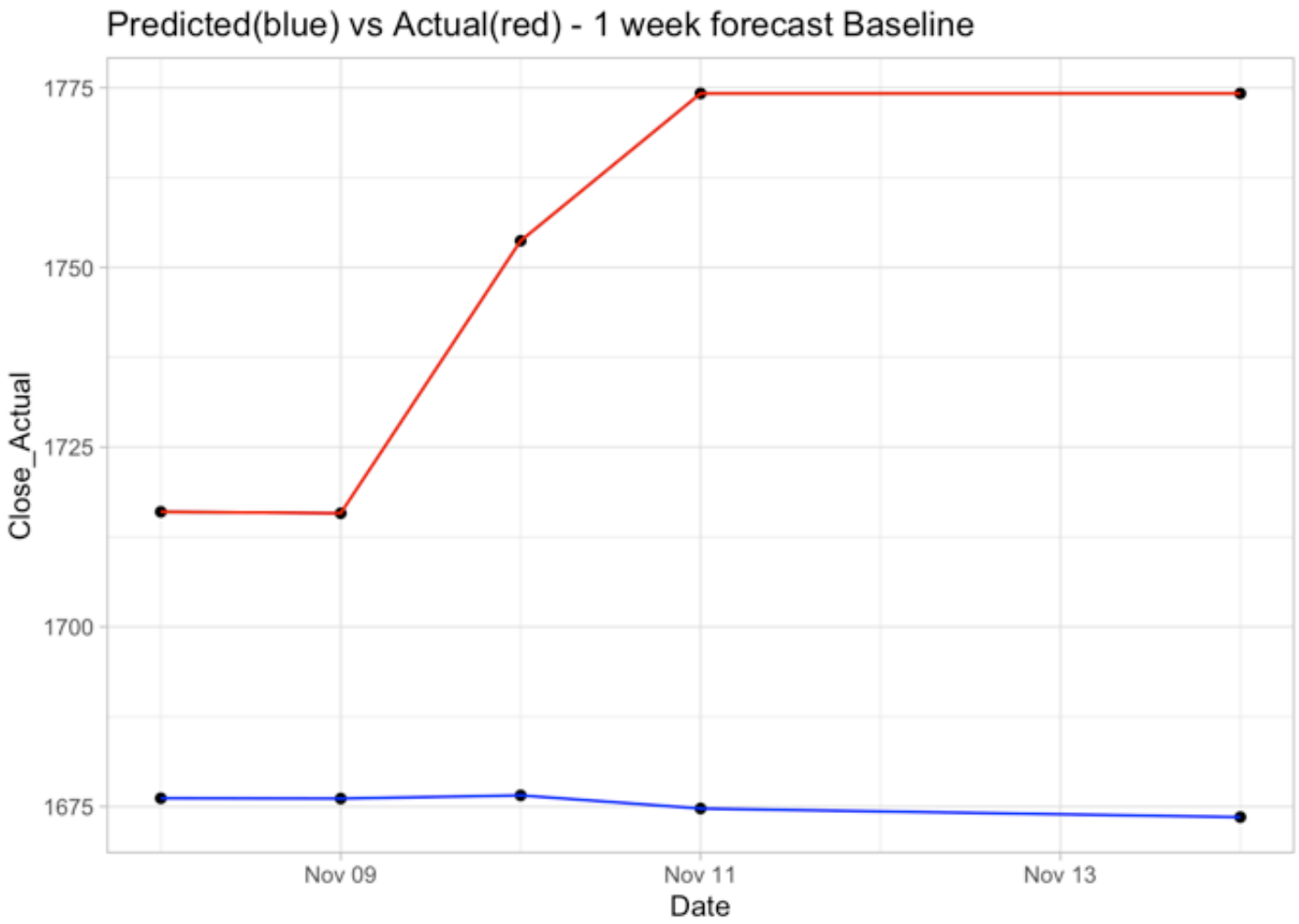
```
## [1] 76.34688
```

Plot Baseline Prophet Forecast

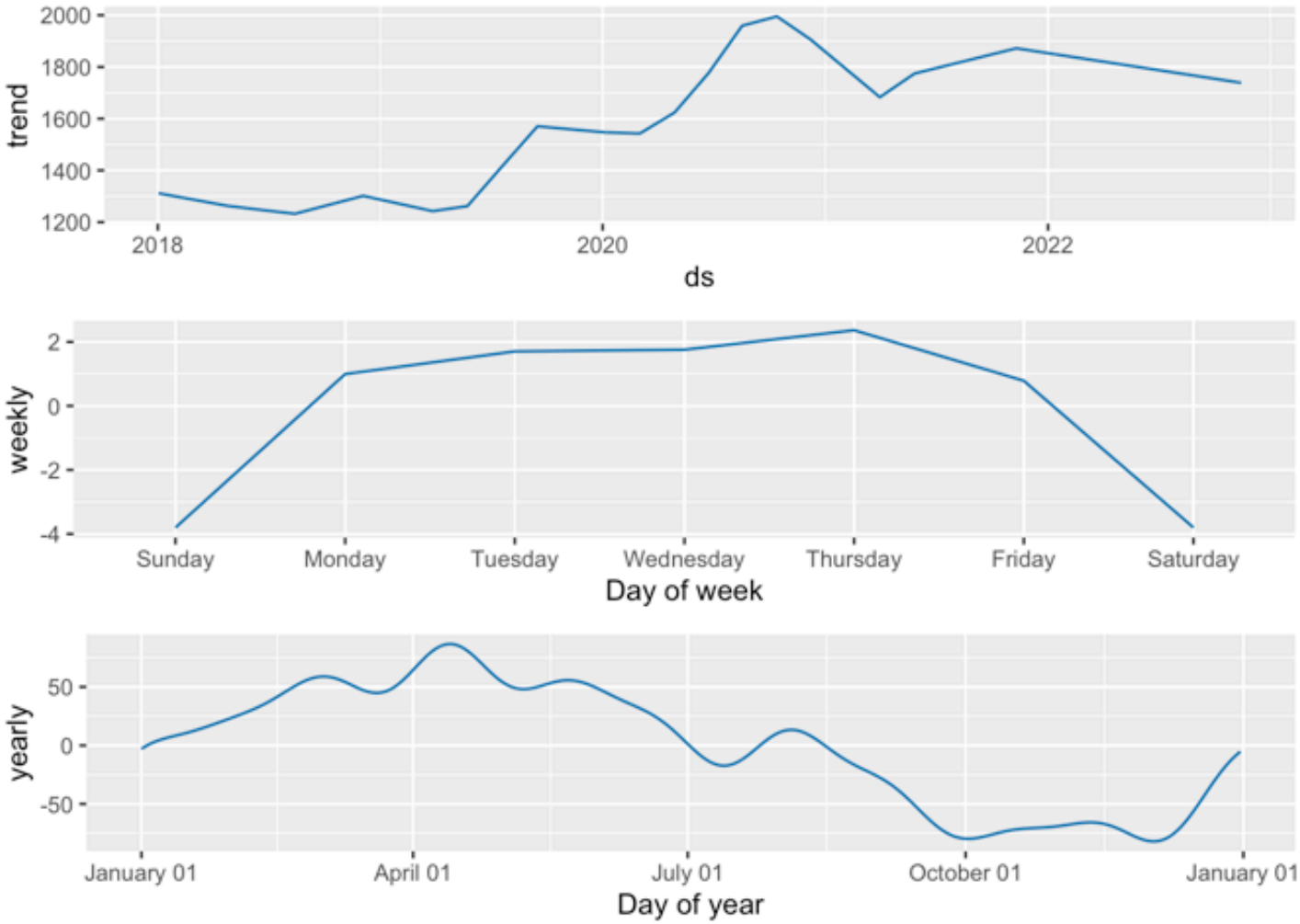
```
# Plot prophet forecast
plot(Prophet, Prophet_Forecast_base, xlabel = "Date", ylabel = "Gold Close Price ($)"
) + ggtitle(paste0("Gold", ": Baseline Price Prediction"))
```




```
# Plot actual
p1 <- ggplot(results, aes(Date, Close_Actual, group=1)) +
    geom_line() +
    theme_light() + ggtitle("Actual 1 Week  Price")

# Plot Predicted
p2 <- ggplot(results, aes(Date, Close_Prediction, group=1)) +
    geom_line() +
    theme_light() + ggtitle("Predicted 1 Week Gold Price")
# Combine Predicted and Actual Plot
p <- p1 +
  geom_point(mapping=p1$mapping) +
  geom_line(color='red') +
  geom_point(mapping=p2$mapping)+
  geom_line(mapping=p2$mapping, color='blue') +
  ggtitle("Predicted(blue) vs Actual(red) - 1 week forecast Baseline")
p
```




```
# Plot prophet components
prophet_plot_components(Prophet, Prophet_Forecast_base)
```



Buy/Sell Baseline Prophet Forecast

```
results$Index <- seq(1, nrow(results), by=1)
vec <- results$Close_Prediction
x <- locate_xtrem(vec)
```

```
## Warning: Use of .data in tidyselect expressions was deprecated in tidyselect 1.2.0
.
## ℹ Please use `"Idx"` instead of `.data$Idx`
```

```
## Warning: Use of .data in tidyselect expressions was deprecated in tidyselect 1.2.0
.
## ℹ Please use `"NB"` instead of `.data$NB`
```

```
x <- x %>%
  select(c('Idx', 'Status')) %>%
  rename(Index = Idx)
results <- left_join(results, x,  by="Index")
results <- results %>%
  select(c('Date', 'Close_Prediction', 'Close_Actual', 'Status'))
results["Status"][results["Status"] == "min"] <- "Buy"
results["Status"][results["Status"] == "max"] <- "Sell"
Profit <- na.omit(results)
Profit <- Profit %>%
  mutate(price_diff = Close_Actual - lag(Close_Actual, default = first(Close_Actual))
) %>%
  filter(Status == 'Sell')
sum(Profit$price_diff) # TWEAK OR USE ONLINE CALC
```

```
## [1] 37.9
```

GOLD FORECAST WEEKLY ROLLOVER

Idea is that if we want to forecast a week ahead, we can use last weeks external values (5 day lag) to forecast gold prices. All data will be known, but continuous forecast require weekly updates. The goal is to predict 1 week in advance, then training data gets expanded 1 week later with actual gold values, retrains model with actual data, and forecasts with 1 week lag predictors.

```
# Create a new full data frame
FULL_df1 <- subset(FULL_df, Date >= as.Date("2018-01-08"))

# Lag External Regressors by 5 (1 week lag)
FULL_df1$Silver_Close.x <- lag(FULL_df1$Silver_Close.x, n=5, default = NA)
FULL_df1$DFF.x <- lag(FULL_df1$DFF.x, n=5, default = NA)
FULL_df1$WM2NS.x <- lag(FULL_df1$WM2NS.x, n=5, default = NA)
FULL_df1$hits.x <- lag(FULL_df1$hits.x, n=5, default = NA)
FULL_df1$DJIA.x <- lag(FULL_df1$DJIA.x, n=5, default = NA)
# Grab data frame to start at the beginning of the week (monday)
FULL_df1 <- subset(FULL_df1, Date >= as.Date("2018-02-12"))

# Store future dataframe (for prophet) with 1 week lag predictors
df_future <- FULL_df1 %>%
  select(c('Date', 'Silver_Close.x', "DFF.x", "WM2NS.x" , "hits.x", "DJIA.x")) %>%
  rename(ds = Date)
```

Gold 1 week rollover prophet forecast

```
# Partition data
prophet.Train <- FULL_df1[1:1160,]
prophet.Test <- FULL_df1[1161:1164,]
```

```
Results <- data.frame()
pred <- data.frame()
# Jumps 4 days in advance
stepsAhead <- 4
# Predicting 4 weeks of forecasts
periods_forecast <- 4
length <- seq(4, stepsAhead * periods_forecast , by=stepsAhead)

# Actual Gold Values
Gold_Results <- gold %>%
  select(c("Date","Gold_Close"))  %>%
  rename(Close_Actual = Gold_Close)

# Create a for loop to retrain model with a new training set
for(i in length) {
  df3 <- prophet.Train %>%
  select(c('Date', 'Gold_Close.x', 'Silver_Close.x', "DFF.x", "WM2NS.x" , "hits.x", "
DJIA.x")) %>%
  rename(ds = Date, y = Gold_Close.x)
  # Add Regressors
  m_ext <- prophet(seasonality.mode = "multiplicative", daily.seasonality = FALSE, in
terval.width = .90)
  m_ext <- add_regressor(m_ext,  "Silver_Close.x", mode = 'multiplicative', standardi
ze = "auto")
  m_ext <- add_regressor(m_ext,  'DFF.x', mode = 'multiplicative', standardize = "aut
o")
  m_ext <- add_regressor(m_ext,  'WM2NS.x', mode = 'multiplicative', standardize = "a
uto")
  m_ext <- add_regressor(m_ext,  'hits.x', mode = 'multiplicative', standardize = "au
to")
  m_ext <- add_regressor(m_ext,  'DJIA.x', mode = 'multiplicative', standardize = "au
to")
  # Fit Model
  m_ext <- fit.prophet(m_ext, df3)
  # Create future dataframe
  future <- make_future_dataframe(m_ext, periods = 6, include_history = TRUE) %>% fil
ter(!wday(ds) %in% c(1,7))
  future <- left_join(future, df_future, by="ds")
  # Forecast with model
  forecast_weekly <- predict(m_ext, future)
  # Return Results and organize data
  Prophet_Results <- forecast_weekly %>%
  select(c("ds","yhat")) %>%
  rename(Date = ds, Close_Prediction = yhat)
  Prophet_Results <- Prophet_Results[(nrow(prophet.Train)+1:(nrow(prophet.Train)+step
sAhead + 1)),]
  Prophet_Results <- na.omit(Prophet_Results)
  pred <- rbind(pred, Prophet_Results )
  # Re-size training and test data
  prophet.Train <- FULL_df1[1:(1160+i),]
```

```
  prophet.Test <- FULL_df1[(1161+i):(1161+i+ stepsAhead),]

}
# Join Predicted and Actual Results
all_results <- left_join(pred, Gold_Results,  by="Date")
all_results
```

Date <dttm>	Close_Prediction <dbl>	Close_Actual <dbl>
2022-10-24	1670.945	1654.1
2022-10-25	1671.979	1658.0
2022-10-26	1669.986	1669.2
2022-10-27	1671.842	1668.8
2022-10-28	1659.312	1648.3
2022-10-31	1668.884	1648.3
2022-11-01	1667.862	1636.4
2022-11-02	1669.719	1651.0
2022-11-03	1656.015	1637.7
2022-11-04	1650.569	1685.7

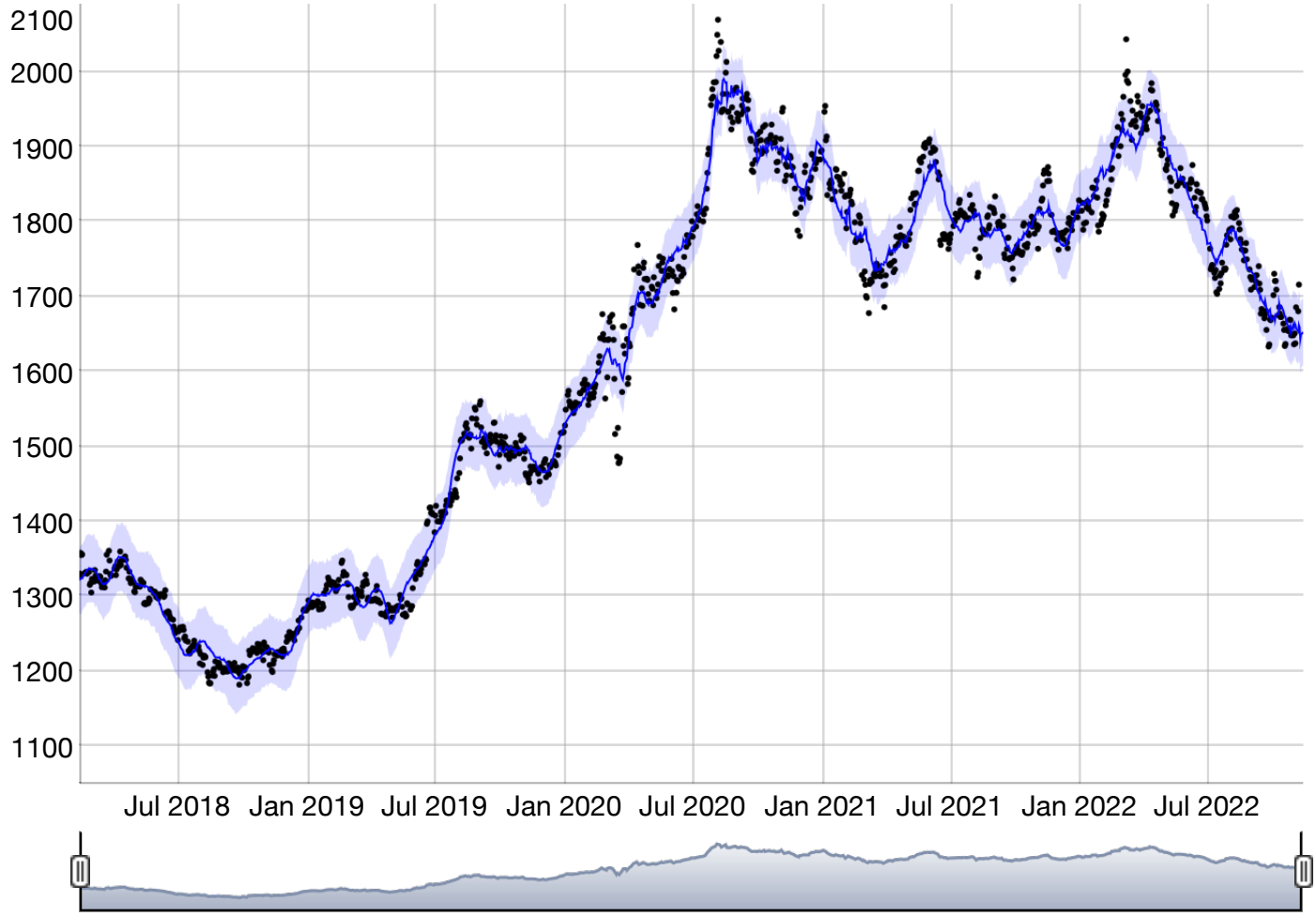

```
# Print RMSE
RMSE <- sqrt(mean((all_results$Close_Actual - all_results$Close_Prediction)^2))
RMSE
```

```
## [1] 58.6095
```

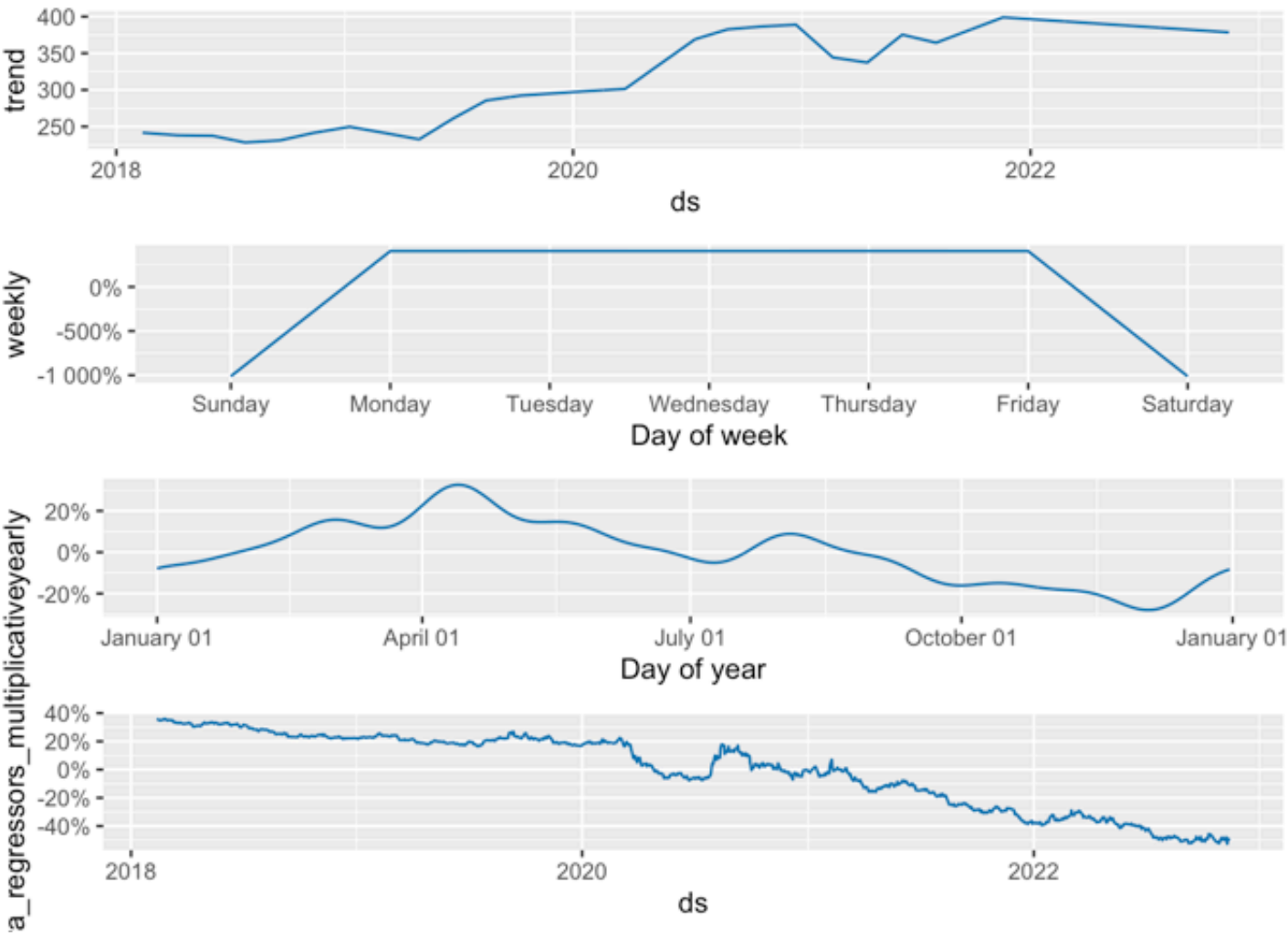
Plot Forecast Values

```
dyplot.prophet(m_ext, forecast_weekly)
```

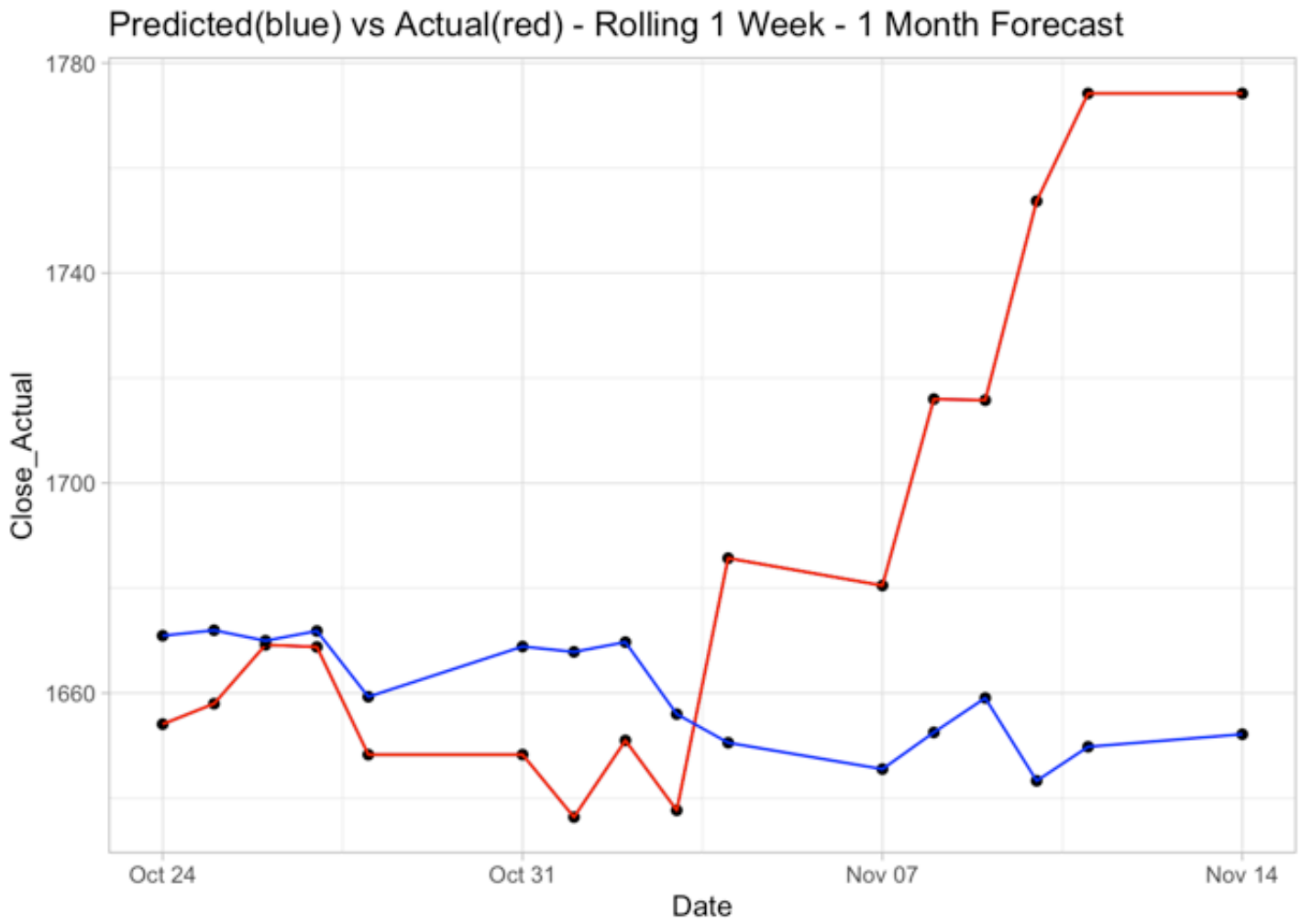
```
## Warning: `select_()` was deprecated in dplyr 0.7.0.
## ℹ Please use `select()` instead.
## ℹ The deprecated feature was likely used in the dplyr package.
##   Please report the issue at <]8;;https://github.com/tidyverse/dplyr/issueshttps://github.com/tidyverse/dplyr/issues]8;;>.
```




```
prophet_plot_components(m_ext, forecast_weekly)
```




```
p1_ext <- ggplot(all_results, aes(Date, Close_Actual, group=1)) +
   geom_line() +
   theme_light() + ggtitle("Actual 1 Week  Price")


 p2_ext <- ggplot(all_results, aes(Date, Close_Prediction, group=1)) +
   geom_line() +
   theme_light() + ggtitle("Predicted 1 Week Gold Price")

 p_ext <- p1_ext +
   geom_point(mapping=p1_ext$mapping) +
   geom_line(color='red') +
   geom_point(mapping=p2_ext$mapping)+
   geom_line(mapping=p2_ext$mapping, color='blue') +
   ggtitle("Predicted(blue) vs Actual(red) - Rolling 1 Week - 1 Month Forecast")
 p_ext
```



Buy/Sell Signal

```
# Find local min and max to create buy/sell signals
all_results$Index <- seq(1, nrow(all_results), by=1)
vec <- all_results$Close_Prediction
x <- locate_xtrem(vec)
x <- x %>%
  select(c('Idx', 'Status')) %>%
  rename(Index = Idx)
all_results <- left_join(all_results, x,  by="Index")
all_results <- all_results %>%
  select(c('Date', 'Close_Prediction', 'Close_Actual', 'Status'))
all_results["Status"][all_results["Status"] == "min"] <- "Buy"
all_results["Status"][all_results["Status"] == "max"] <- "Sell"
Profit <- na.omit(all_results)

# Calculate profit based on buy/sell signals
Profit <- Profit %>%
  mutate(price_diff = Close_Actual - lag(Close_Actual, default = first(Close_Actual))
) %>%
  filter(Status == 'Sell')

sum(Profit$price_diff)  # TWEAK OR USE ONLINE CALC
```

```
## [1] 73.9
```

GOLD FORECAST ROLLOVER 1 Day

Idea is that if we want to forecast 1 day ahead, we can use last weeks external values (1 day lag) to forecast gold prices. All data will be known, but continuous forecast require daily updates. The goal is to predict 1 day in advance, then training data gets expanded 1 day later with actual gold values, retrains model with actual data, and forecasts with 1 day lag predictors.

```
# New dataframe with lagged predictors
FULL_df2 <- subset(FULL_df, Date >= as.Date("2018-01-08"))

# Lag External Regressors by 1 day
FULL_df2$Silver_Close.x <- lag(FULL_df2$Silver_Close.x, n=1, default = NA)
FULL_df2$DFF.x <- lag(FULL_df2$DFF.x, n=1, default = NA)
FULL_df2$WM2NS.x <- lag(FULL_df2$WM2NS.x, n=1, default = NA)
FULL_df2$hits.x <- lag(FULL_df2$hits.x, n=1, default = NA)
FULL_df2$DJIA.x <- lag(FULL_df2$DJIA.x, n=1, default = NA)
FULL_df2 <- subset(FULL_df2, Date >= as.Date("2018-02-12"))
# Store lagged variables in future dataframe for prophet
df_future <- FULL_df2 %>%
  select(c('Date', 'Silver_Close.x', "DFF.x", "WM2NS.x" , "hits.x", "DJIA.x")) %>%
  rename(ds = Date)
```

```
# Partition data
```

```
prophet.Train <- FULL_df2[1:1160,]
prophet.Test <- FULL_df2[1161:1161,]
Results <- data.frame()
pred <- data.frame()
# Jump 1 day in advance
stepsAhead <- 1
# Forecasting for 20 periods aka 20 days
periods_forecast <- 20
length <- seq(1, stepsAhead * periods_forecast , by=stepsAhead)

# Actual gold values
Gold_Results <- gold %>%
  select(c("Date","Gold_Close"))  %>%
  rename(Close_Actual = Gold_Close)

# Create for loop to retrain model with a new training set
for(i in length) {
  df3 <- prophet.Train %>%
  select(c('Date', 'Gold_Close.x', 'Silver_Close.x', "DFF.x", "WM2NS.x" , "hits.x", "
DJIA.x")) %>%
  rename(ds = Date, y = Gold_Close.x)
  # Add regressor
  m_ext <- prophet(seasonality.mode = "multiplicative", daily.seasonality = FALSE, in
terval.width = .90)
  m_ext <- add_regressor(m_ext,  "Silver_Close.x", mode = 'multiplicative', standardi
ze = "auto")
  m_ext <- add_regressor(m_ext,  'DFF.x', mode = 'multiplicative', standardize = "aut
o")
  m_ext <- add_regressor(m_ext,  'WM2NS.x', mode = 'multiplicative', standardize = "a
uto")
  m_ext <- add_regressor(m_ext,  'hits.x', mode = 'multiplicative', standardize = "au
to")
  m_ext <- add_regressor(m_ext,  'DJIA.x', mode = 'multiplicative', standardize = "au
to")
  #Fit Model
  m_ext <- fit.prophet(m_ext, df3)
  # Create future dataframe
  future <- make_future_dataframe(m_ext, periods = 1, include_history = TRUE) %>% fil
ter(!wday(ds) %in% c(1,7))
  future <- left_join(future, df_future, by="ds")
  # Forecast with model
  forecast_weekly <- predict(m_ext, future)

  # Return Results
  Prophet_Results <- forecast_weekly %>%
  select(c("ds","yhat")) %>%
  rename(Date = ds, Close_Prediction = yhat)
  Prophet_Results <- Prophet_Results[(nrow(prophet.Train)+1:(nrow(prophet.Train)+step
sAhead + 1)),]
  Prophet_Results <- na.omit(Prophet_Results)
```

```
  pred <- rbind(pred, Prophet_Results )
  # Resize training data
  prophet.Train <- FULL_df2[1:(1160+i),]
  prophet.Test <- FULL_df2[(1161+i):(1161+i+ stepsAhead),]

}

# Join Predicted and actual Results
all_results <- left_join(pred, Gold_Results,  by="Date")
all_results
```

Date <dttm>	Close_Prediction <dbl>	Close_Actual <dbl>
2022-10-25	1670.467	1658.0
2022-10-26	1657.114	1669.2
2022-10-27	1674.551	1668.8
2022-10-28	1665.178	1648.3
2022-11-01	1652.849	1636.4
2022-11-02	1666.021	1651.0
2022-11-03	1662.231	1637.7
2022-11-04	1613.514	1685.7
2022-11-08	1672.081	1716.0
2022-11-09	1677.143	1715.8

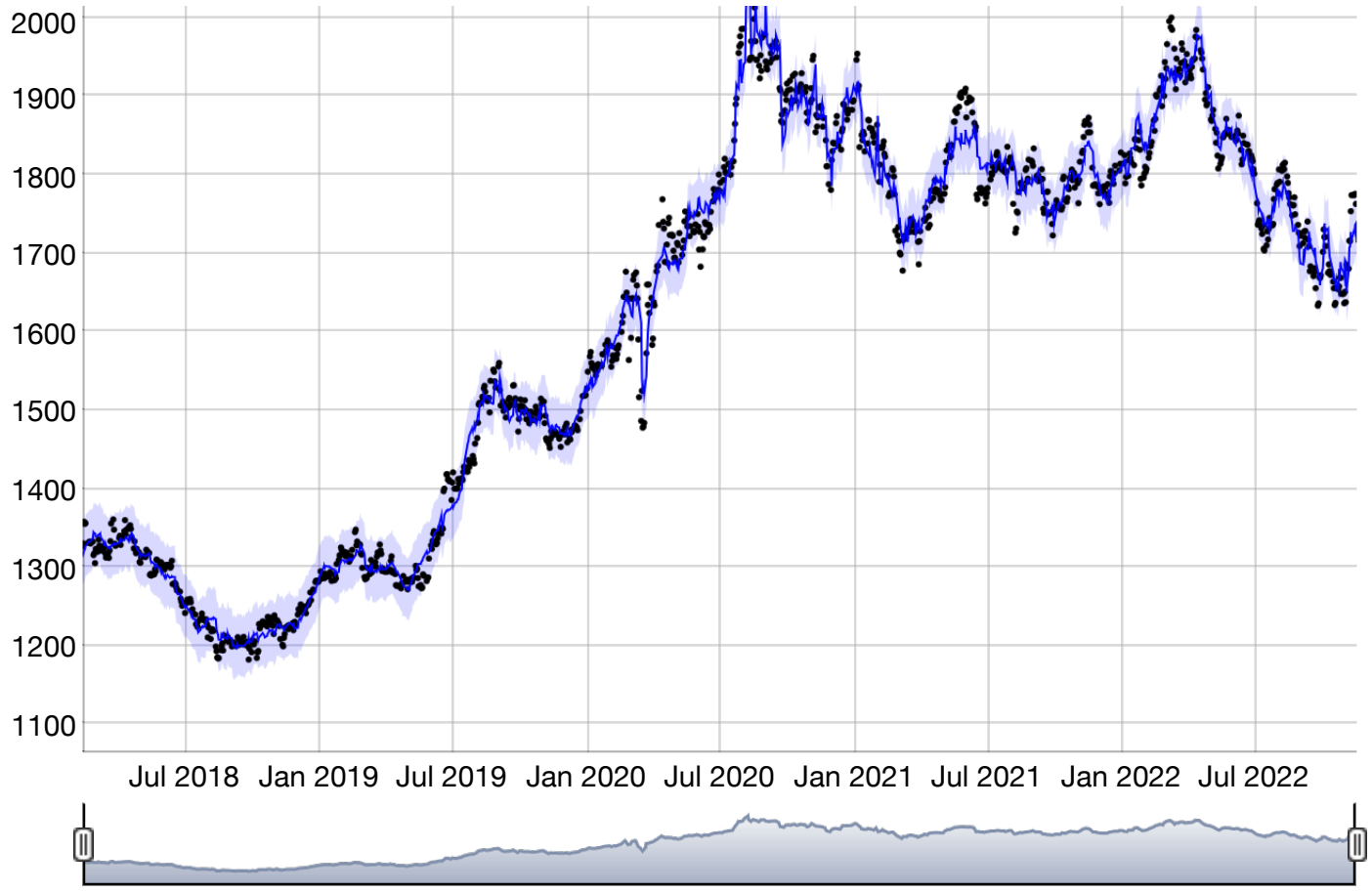
1-10 of 16 rows Previous **1** 2 Next

```
# RMse
RMSE <- sqrt(mean((all_results$Close_Actual - all_results$Close_Prediction)^2))
RMSE
```

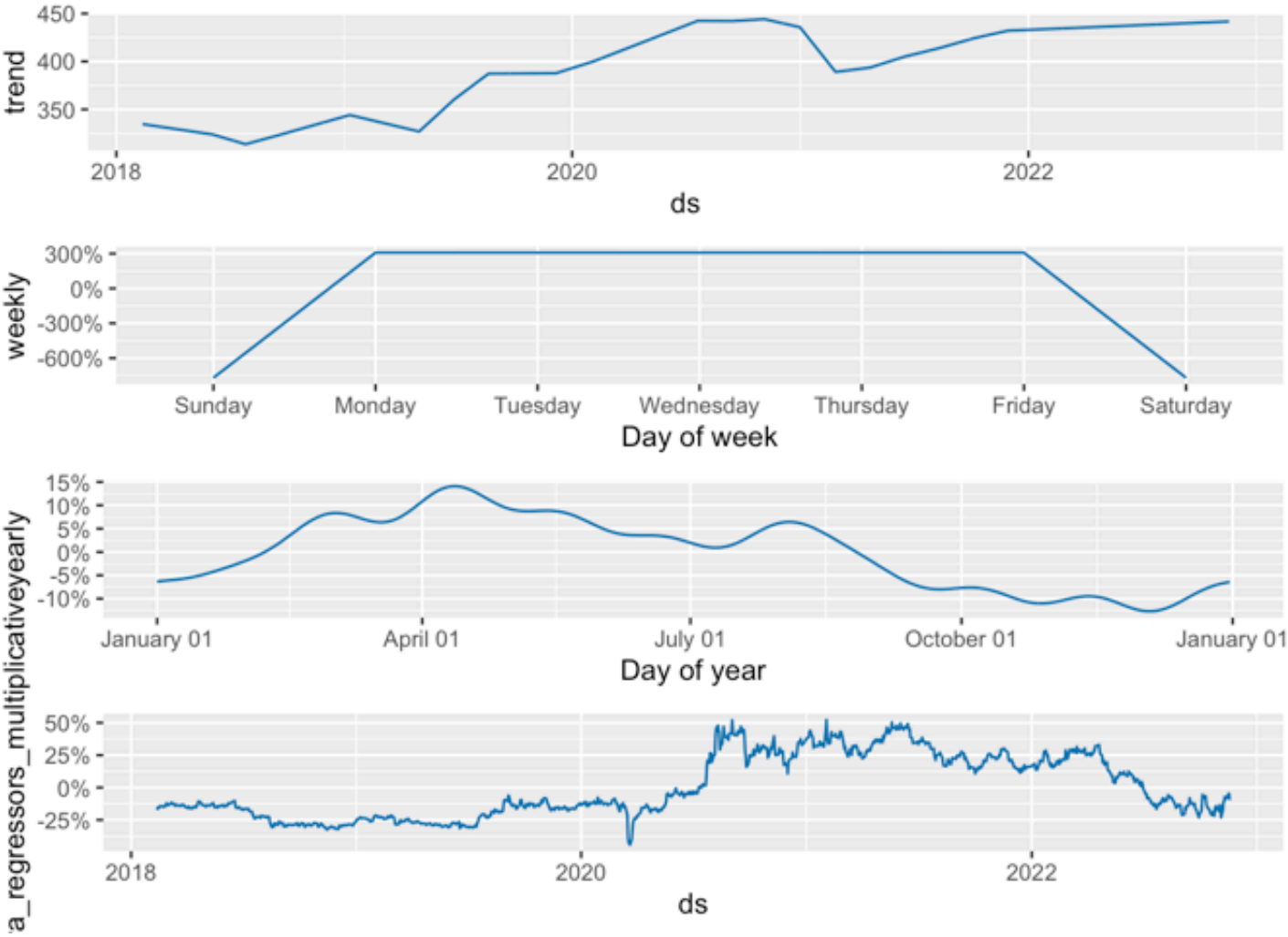
```
## [1] 43.45533
```

Plot Forecast

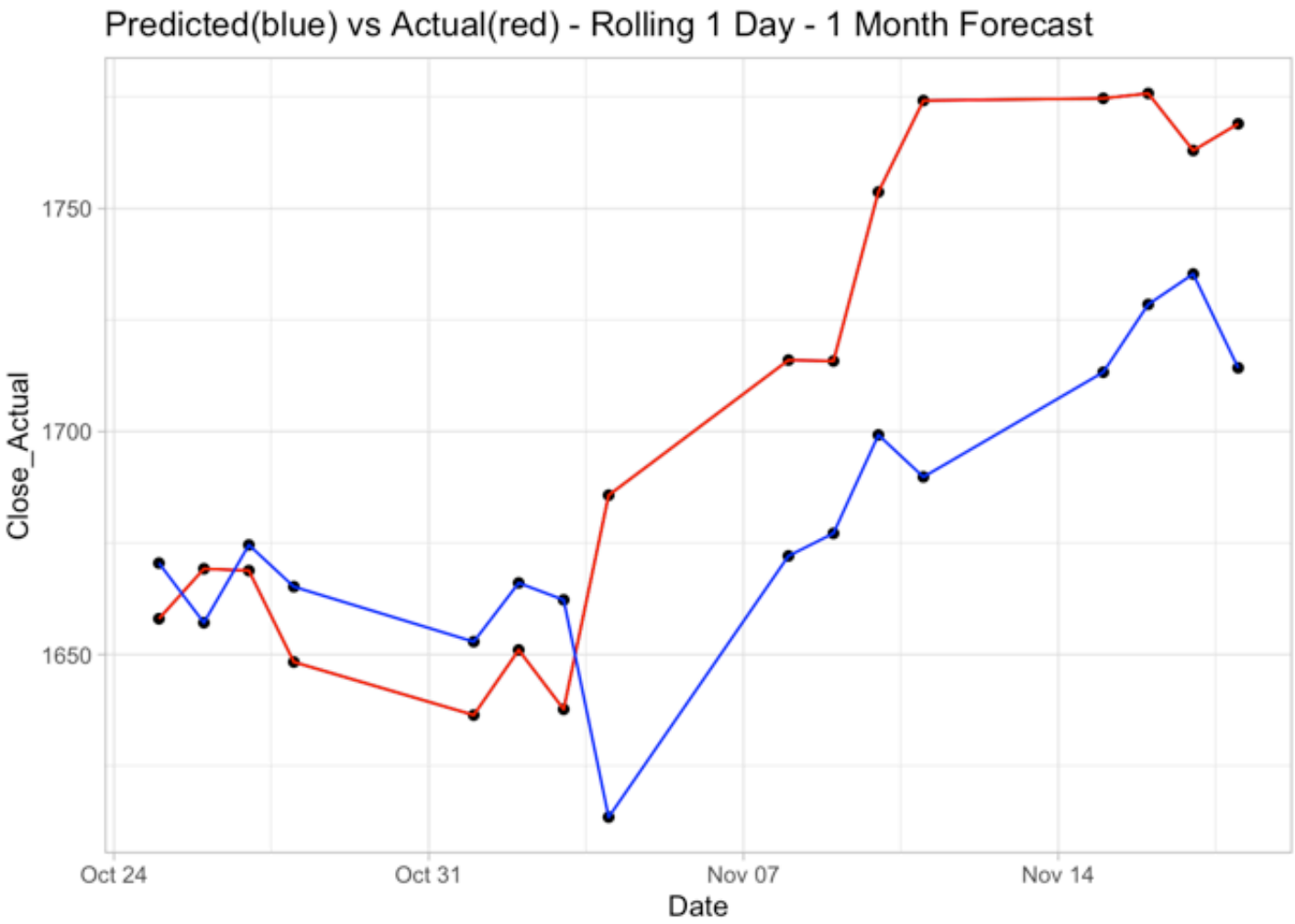
```
# FOrecast plot
dyplot.prophet(m_ext, forecast_weekly)
```




```
# Plot components
prophet_plot_components(m_ext, forecast_weekly)
```




```
# Plot actual and predicted data
 p1_ext <- ggplot(all_results, aes(Date, Close_Actual, group=1)) +
    geom_line() +
    theme_light() + ggtitle("Actual 1 Week  Price")

 p2_ext <- ggplot(all_results, aes(Date, Close_Prediction, group=1)) +
    geom_line() +
    theme_light() + ggtitle("Predicted 1 Week Gold Price")

p_ext <- p1_ext +
    geom_point(mapping=p1_ext$mapping) +
    geom_line(color='red') +
    geom_point(mapping=p2_ext$mapping)+
    geom_line(mapping=p2_ext$mapping, color='blue') +
    ggtitle("Predicted(blue) vs Actual(red) - Rolling 1 Day - 1 Month Forecast")
 p_ext
```



Buy/Sell Signal

```
all_results$Index <- seq(1, nrow(all_results), by=1)
vec <- all_results$Close_Prediction
x <- locate_xtrem(vec)
x <- x %>%
  select(c('Idx', 'Status')) %>%
  rename(Index = Idx)
all_results <- left_join(all_results, x,  by="Index")
all_results <- all_results %>%
  select(c('Date', 'Close_Prediction', 'Close_Actual', 'Status'))
all_results["Status"][all_results["Status"] == "min"] <- "Buy"
all_results["Status"][all_results["Status"] == "max"] <- "Sell"
Profit <- na.omit(all_results)
Profit <- Profit %>%
  mutate(price_diff = Close_Actual - lag(Close_Actual, default = first(Close_Actual))
) %>%
  filter(Status == 'Sell')
sum(Profit$price_diff) # TWEAK OR USE ONLINE CALC
```

```
## [1] 71
```

No Lag on Predictors (Check to see if external regressors help prediction if forecasted perfectly)

```
# New dataframe with lagged predictors
FULL_df2 <- subset(FULL_df, Date >= as.Date("2018-01-08"))

# Lag External Regressors by 1 day
FULL_df2$Silver_Close.x <- FULL_df2$Silver_Close.x
FULL_df2$DFF.x <- FULL_df2$DFF.x
FULL_df2$WM2NS.x <- FULL_df2$WM2NS.x
FULL_df2$hits.x <- FULL_df2$hits.x
FULL_df2$DJIA.x <- FULL_df2$DJIA.x
FULL_df2 <- subset(FULL_df2, Date >= as.Date("2018-02-12"))
# Store lagged variables in future dataframe for prophet
df_future <- FULL_df2 %>%
  select(c('Date', 'Silver_Close.x', "DFF.x", "WM2NS.x" , "hits.x", "DJIA.x")) %>%
  rename(ds = Date)
```

```
# Partition data
prophet.Train <- FULL_df2[1:1160,]
prophet.Test <- FULL_df2[1161:1161,]
Results <- data.frame()
pred <- data.frame()
# Jump 1 day in advance
stepsAhead <- 1
# Forecasting for 20 periods aka 20 days
periods_forecast <- 20
length <- seq(1, stepsAhead * periods_forecast , by=stepsAhead)

# Actual gold values
Gold_Results <- gold %>%
  select(c("Date","Gold_Close"))  %>%
  rename(Close_Actual = Gold_Close)

# Create for loop to retrain model with a new training set
for(i in length) {
  df3 <- prophet.Train %>%
  select(c('Date', 'Gold_Close.x', 'Silver_Close.x', "DFF.x", "WM2NS.x" , "hits.x", "
DJIA.x")) %>%
  rename(ds = Date, y = Gold_Close.x)
  # Add regressor
  m_ext <- prophet(seasonality.mode = "multiplicative", daily.seasonality = FALSE, in
terval.width = .90)
  m_ext <- add_regressor(m_ext,  "Silver_Close.x", mode = 'multiplicative', standardi
ze = "auto")
```

```
  m_ext <- add_regressor(m_ext,  'DFF.x', mode = 'multiplicative', standardize = "aut
o")
  m_ext <- add_regressor(m_ext,  'WM2NS.x', mode = 'multiplicative', standardize = "a
uto")
  m_ext <- add_regressor(m_ext,  'hits.x', mode = 'multiplicative', standardize = "au
to")
  m_ext <- add_regressor(m_ext,  'DJIA.x', mode = 'multiplicative', standardize = "au
to")
  #Fit Model
  m_ext <- fit.prophet(m_ext, df3)
  # Create future dataframe
  future <- make_future_dataframe(m_ext, periods = 1, include_history = TRUE) %>% fil
ter(!wday(ds) %in% c(1,7))
  future <- left_join(future, df_future, by="ds")
  # Forecast with model
  forecast_weekly <- predict(m_ext, future)

  # Return Results
  Prophet_Results <- forecast_weekly %>%
  select(c("ds","yhat")) %>%
  rename(Date = ds, Close_Prediction = yhat)
  Prophet_Results <- Prophet_Results[(nrow(prophet.Train)+1:(nrow(prophet.Train)+step
sAhead + 1)),]
  Prophet_Results <- na.omit(Prophet_Results)
  pred <- rbind(pred, Prophet_Results )
  # Resize training data
  prophet.Train <- FULL_df2[1:(1160+i),]
  prophet.Test <- FULL_df2[(1161+i):(1161+i+ stepsAhead),]

}

# Join Predicted and actual Results
all_results <- left_join(pred, Gold_Results,  by="Date")
all_results
```

Date <dttm>	Close_Prediction <dbl>	Close_Actual <dbl>
2022-10-25	1644.479	1658.0
2022-10-26	1667.214	1669.2
2022-10-27	1659.611	1668.8
2022-10-28	1647.537	1648.3
2022-11-01	1664.852	1636.4
2022-11-02	1656.454	1651.0
2022-11-03	1616.411	1637.7

2022-11-04	1639.876	1685.7
2022-11-08	1674.971	1716.0
2022-11-09	1700.381	1715.8

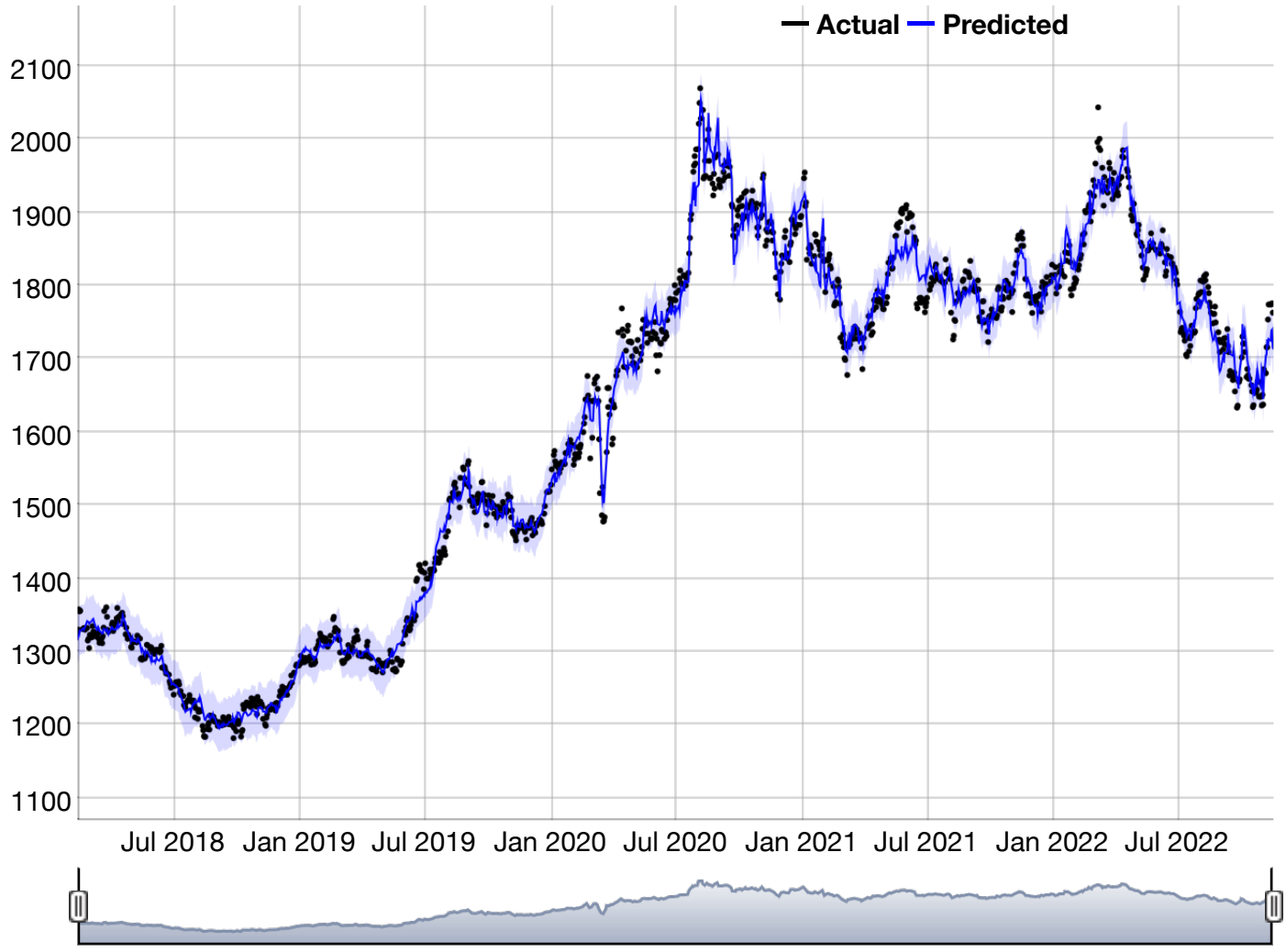
1-10 of 16 rows Previous **1** 2 Next

```
# RMse
RMSE <- sqrt(mean((all_results$Close_Actual - all_results$Close_Prediction)^2))
RMSE
```

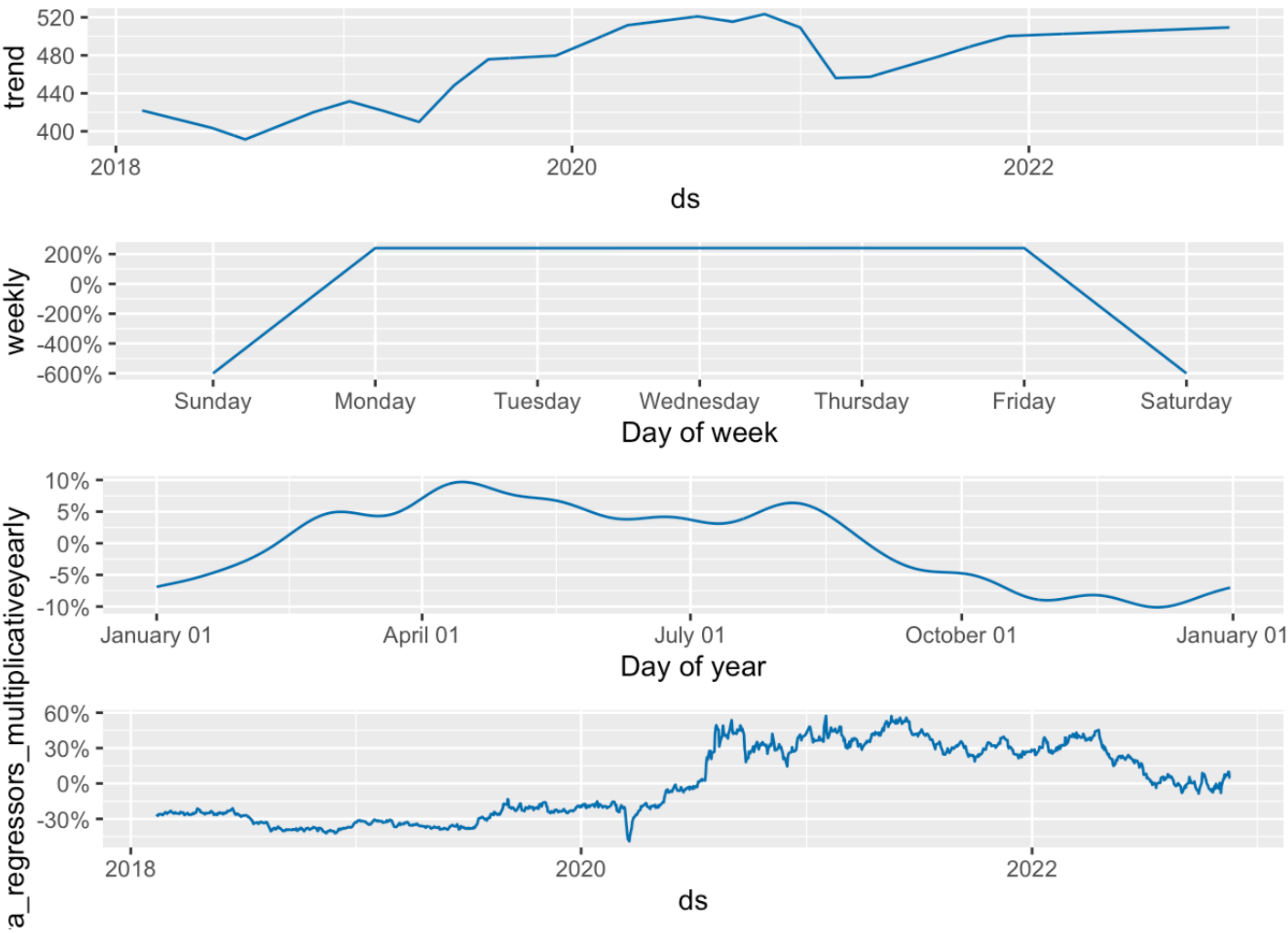
```
## [1] 39.46699
```

Plot Forecast

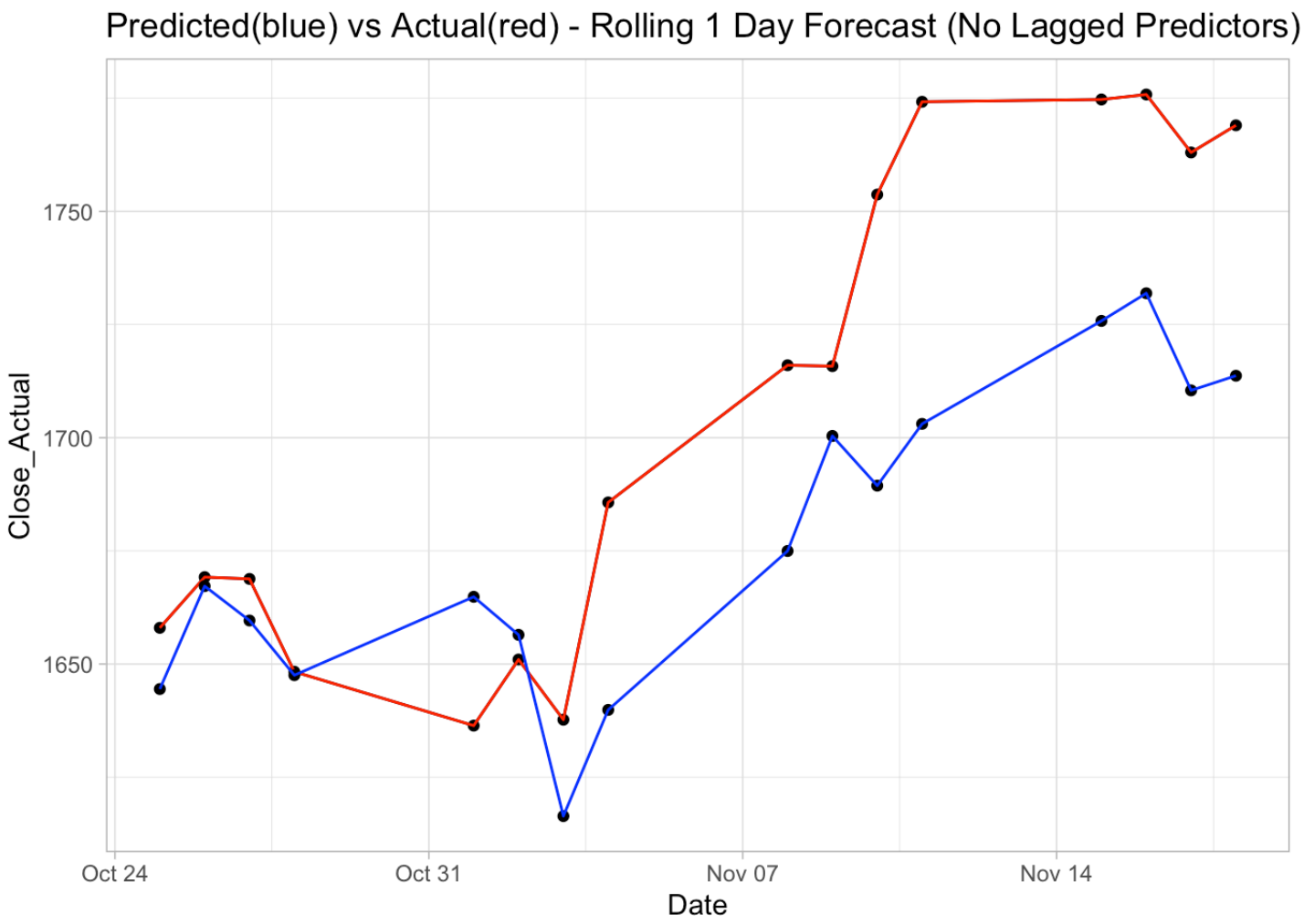
```
# FOrecast plot
dyplot.prophet(m_ext, forecast_weekly)
```



```
# Plot components
prophet_plot_components(m_ext, forecast_weekly)
```




```
# Plot actual and predicted data
 p1_ext <- ggplot(all_results, aes(Date, Close_Actual, group=1)) +
    geom_line() +
    theme_light() + ggtitle("Actual 1 Week  Price")

 p2_ext <- ggplot(all_results, aes(Date, Close_Prediction, group=1)) +
    geom_line() +
    theme_light() + ggtitle("Predicted 1 Week Gold Price")

p_ext <- p1_ext +
    geom_point(mapping=p1_ext$mapping) +
    geom_line(color='red') +
    geom_point(mapping=p2_ext$mapping)+
    geom_line(mapping=p2_ext$mapping, color='blue') +
    ggtitle("Predicted(blue) vs Actual(red) - Rolling 1 Day Forecast (No Lagged Predi
ctors)")
 p_ext
```



Buy/Sell Signal

```
all_results$Index <- seq(1, nrow(all_results), by=1)
vec <- all_results$Close_Prediction
x <- locate_xtrem(vec)
x <- x %>%
  select(c('Idx', 'Status')) %>%
  rename(Index = Idx)
all_results <- left_join(all_results, x,  by="Index")
all_results <- all_results %>%
  select(c('Date', 'Close_Prediction', 'Close_Actual', 'Status'))
all_results["Status"][all_results["Status"] == "min"] <- "Buy"
all_results["Status"][all_results["Status"] == "max"] <- "Sell"
Profit <- na.omit(all_results)
Profit <- Profit %>%
  mutate(price_diff = Close_Actual - lag(Close_Actual, default = first(Close_Actual))
) %>%
  filter(Status == 'Sell')
sum(Profit$price_diff) # TWEAK OR USE ONLINE CALC
```

```
## [1] 105.5
```