Problem 3 (15 credits)

HW3

Michael Brucek (mbrucek), Eduardo Chavez (echavezh), Kevin Grady (grady133), Danny Moncada (monca016), Tian Zhang (zhan7003)

March 27, 2020

```
suppressWarnings(suppressPackageStartupMessages({
  library(TSA)
  library(ggplot2)
  library(dplyr)
  library(forecast)
  library(tseries) # Only for the ADF test for testing stationarity
  library(readr) # Use read_csv()
}))
```

Time Series Data of Your Choice

Background

This homework problem will allow you to apply the learned time series analysis and forecasting skills to your own or favorite dataset. This dataset could be your own data (from your interested hobby groups, sports or video game records, previous jobs, past school works, etc). Notice that, we DON'T require data disclosure so please feel free to use your own data if you would like us to help you understand the result. Or if you don't have any time series dataset, please feel free to get one from the Internet. Any topics are welcome!

Hint: If you have trouble finding a good dataset, sources of public time series data include `kaggle.com` where many of the class examples came from, and `Yahoo Finance` which provides rich information about historical prices of nearly every US stock. Or if you are a sport fun or a video game fun, I believe that similar data collections are also available online.

So, please feel free to explore! I would suggest using data from sources other than `Kaggle` and `Yahoo Finance` to avoid similarity.

Importance

Notice that, the questions I provided in this problem are mostly real-world tasks that we encountered in many data scientists' daily job, including those working in financial sectors (such as hedge fund companies). So while finishing the questions as required assignments, please make sure to take a few minutes to understand why these questions are raised, and what's the standard procedures to address them.

The credit will be given based on whether you do everything in standard procedures, as opposed to the results such as whether the forecasting accuracy is good.

General Requirements

However, I do have some very general and mild requirements in order for the analysis to be valid.

1. Please make sure the time series contains at least $T = 500$ time points. The final credits will be prorated if T is less than 500 (`floor` to the nearest hundred, for example, 499 will become 400, and hence $400/500 = 80\%$ credits will be given).

2. Please make sure the data are REAL data, not simulated ones. Given there're plenty of available datasets online, I can't find a reason to simulate data. Only 50% credits will be given if we find out the data are simulated.

3. Also make sure the data are non-trivial (having sufficient data variation and possibly a trend). For example, it is trivial to analyze a series of 500 zeros, denoting something like "the number of spacecrafts I owned in the past 1.5 years". 0 credits will be given if the data are regarded as trivial.

4. If two groups happen to use the same dataset (or one dataset being the subset of another), I reserve the right to place the two homework under scrutiny.

5. Please do not use any datasets (or their subsets) used in the lectures or previous homework. Otherwise, 0 credits will be given to this problem.

Question 1 (1 credit)

Please briefly describe the background of your dataset as I did for the Boston Crime Data in Homework 1 Problem 3, and its source (link) if you are using public data.

```
# The S&P 500 is a stock market index that measure stock performance of 500 large companies
# listed on stock exchanges in the United States.  It is one of the most commonly followed
# equity indices, and many consider it to be one of the best representations of the U.S.
# stock market.[1]

# Right before the subprime mortgage crisis began to spread through the U.S. financial sector,
# the S&P 500 had a record close of $1565.15 on October 9th, 2007.  Less than a month later,
# it dropped to $1,400 and did not reach that level again for FIVE(!) years.

# I selected two years worth of data on the closing price for the S&P 500, from October 9th,
# 2007 (peak) through October 9, 2009; this equals 506 data points.  The data was collected
# using Yahoo! Finance.[2]

# [1] Kenton, W.  (2019, May 18).  S&P 500 Index - Standard & Poor's 500 Index.
# Retreived from https://www.investopedia.com/terms/s/sp500.asp.

# [2] Yahoo! Finance.  (2020, March 16).  S&P500 (^GPSC).
# Retreived from https://finance.yahoo.com/quote/%5EGSPC?p=%5EGSPC.
```

Question 2 (1 credit)

Please plot your data and provide the sample size. Use the first 80% of the data as training, and the last 20% as testing.

```
# Read in data file
sp500_close <- read_csv("GSPC.csv")
```

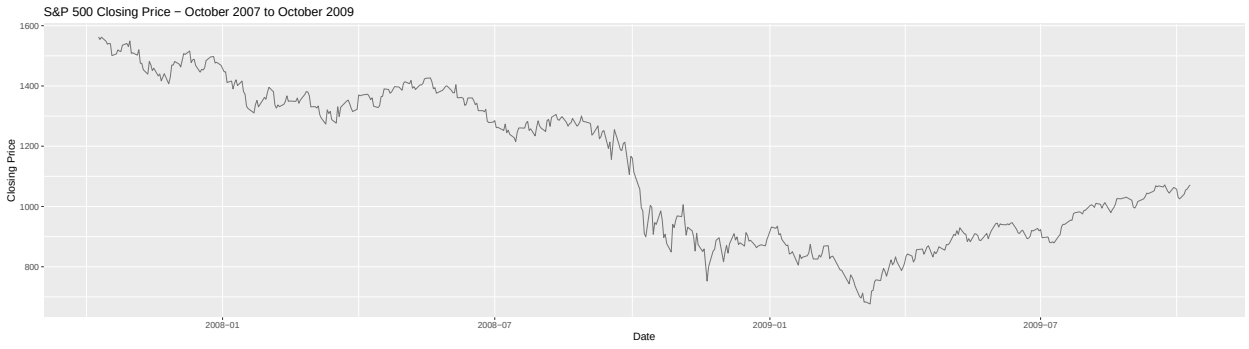
```
## Parsed with column specification:
## cols(
##   Date = col_date(format = ""),
```

```
##   Open = col_double(),
##   High = col_double(),
##   Low = col_double(),
##   Close = col_double(),
##   `Adj Close` = col_double(),
##   Volume = col_double()
## )
```

```
# Examine the first five rows
head(sp500_close)
```

```
## # A tibble: 6 x 7
##   Date        Open  High   Low Close `Adj Close`     Volume
##   <date>     <dbl> <dbl> <dbl> <dbl>       <dbl>      <dbl>
## 1 2007-10-10 1565. 1565. 1555. 1562.       1562. 3044760000
## 2 2007-10-11 1565. 1576. 1547. 1554.       1554. 3911260000
## 3 2007-10-12 1555. 1563. 1554. 1562.       1562. 2788690000
## 4 2007-10-15 1562. 1565. 1541. 1549.       1549. 3139290000
## 5 2007-10-16 1548. 1548. 1536. 1539.       1539. 3234560000
## 6 2007-10-17 1544. 1551. 1526. 1541.       1541. 3638070000
```

```
# Plot the data
p1 <- ggplot(data = sp500_close, aes(x = Date, y = Close)) + geom_line(alpha = 0.5) +
              ggtitle("S&P 500 Closing Price - October 2007 to October 2009") +
                        labs(x = "Date", y = "Closing Price")
p1
```




```
# Split into training and testing data
# The first 404 rows represents 80% of the data, this is what we will use for training
# (calculated manually)
# sp500_close_train = ts(sp500_close[1:404,5], start = 1, end = 404)

# Subset here first - we will probably transform this into a dataframe.
sp500_close_train = ts(sp500_close[1:404,5], start = 1, end = 404)
sp500_close_test = ts(sp500_close[405:506,5], start = 405, end = 506)

# The remaining 102 rows represents 20% of the data to be used for testing
# sp500_close_test = ts(sp500_close[405:506,5], start = 405, end = 506)

# Provide the sample size of training data
dim(sp500_close_train)
```

```
## [1] 404   1
```

```
# Provide the sample size of testing data
dim(sp500_close_test)
```

```
## [1] 102   1
```

Question 3 (2 credits)

On the TRAINING set:

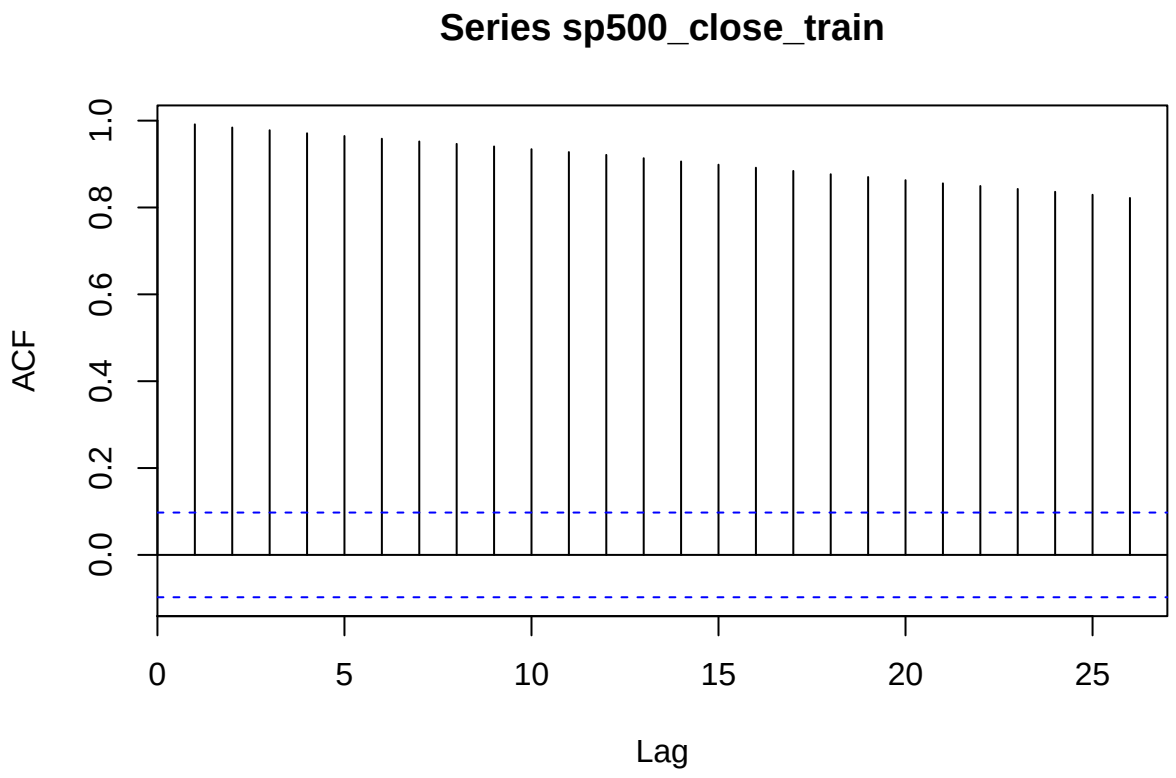
Please (make transformations if necessary, and) use the `ADF test` to check for stationarity. Remove trend if necessary, and check the residuals for spurious regression (proof of random walk)

Check ACF, PACF, and EACF for the order of the ARMA model (after differencing, if it has a random walk). Use AIC or BIC to select a final model from your candidate models. Report the orders.

```
# P-value < 0.05 indicates that TS is stationary
# Use ADF test to check for stationarity
adf.test(sp500_close_train)
```

```
##
##  Augmented Dickey-Fuller Test
##
## data:  sp500_close_train
## Dickey-Fuller = -1.5493, Lag order = 7, p-value = 0.7677
## alternative hypothesis: stationary
```

```
# Initial ACF plot shows HIGH correlation between data points
# This is because we are using the daily closing price of the S&P500, and each data point
# will be highly correlated to the previous day's close.
acf(sp500_close_train)
```




```
# During the exploration of this data set, we determined that the data required differencing
# and that the log of the price would help determine the final model more simply.
# For the sake of brevity, we will not include that code here.

# The steps taken were:
# 1.  Generated a random walk and performed a linear regression using the training set.
# 2.  After confirming spurious regression was indeed happening, we calculated the log of
# the price.
# 3.  We performed another linear regression with the log of the closing price, and encoun-
# tered the same problem.
```

```
# Create a new dataframe that now includes the LOG of the S&P500 closing price.
sp500_close_with_log = sp500_close %>% mutate(log_close = log(Close))

# Get the first 404 rows of the original close price
log_closing_price <- sp500_close_with_log [1:404,]$log_close

# Generate a random walk with the same length as the training data.
N = 404L
e = rnorm(N)
Xt <- cumsum(e)

# Create a linear model to compare with the random walk
lm2 <- lm(Xt ~ log_closing_price)

# Show the summary
```

```r
summary(lm2)
```

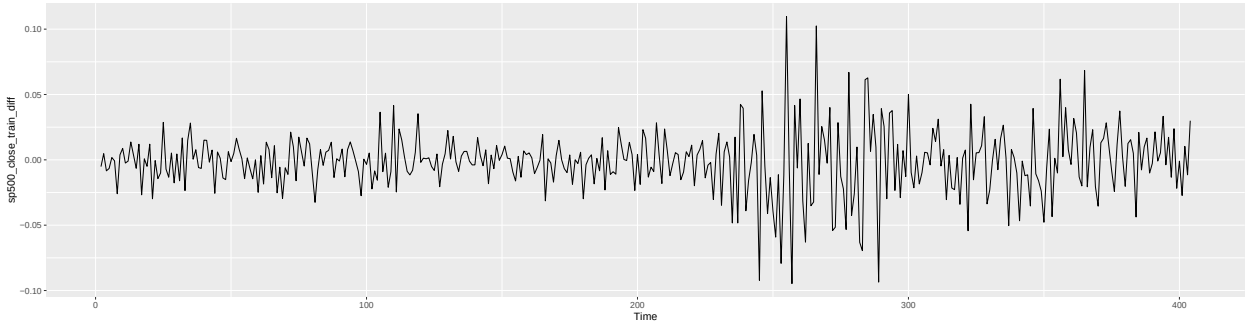
```
##
## Call:
## lm(formula = Xt ~ log_closing_price)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -14.949  -4.905   1.202   5.581  11.070
##
## Coefficients:
##                   Estimate Std. Error t value Pr(>|t|)
## (Intercept)         80.338      9.703   8.280 1.85e-15 ***
## log_closing_price  -11.176      1.378  -8.108 6.29e-15 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 6.555 on 402 degrees of freedom
## Multiple R-squared:  0.1405, Adjusted R-squared:  0.1384
## F-statistic: 65.74 on 1 and 402 DF,  p-value: 6.289e-15
```

```r
# We know the same effect is happening - so now we try differencing to see if that helps
# make this process stationary.
```

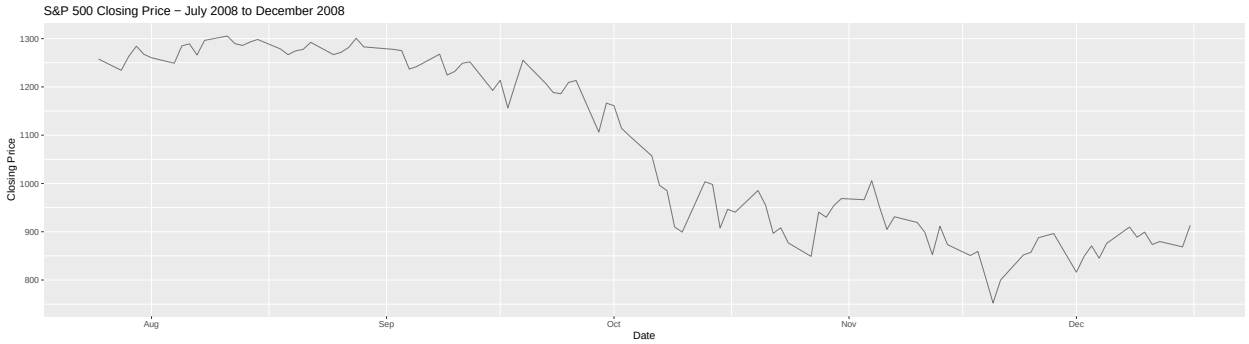
```r
# Since we created the log of price above, I will generate a new set of training/testing
# data, using the new column in place of the old one.

# Geneate a new set of training/testing data with log of price
sp500_close_train = ts(sp500_close_with_log[1:404,8], start = 1, end = 404)
sp500_close_test = ts(sp500_close_with_log[405:506,8], start = 405, end = 506)

# check stationarity again
adf.test(sp500_close_train)
```

```
##
##  Augmented Dickey-Fuller Test
##
## data:  sp500_close_train
## Dickey-Fuller = -1.6477, Lag order = 7, p-value = 0.7261
## alternative hypothesis: stationary
```

```r
# Still no good... let's try differencing.

# Differencing by 1 to see if this helps
sp500_close_train_diff <- diff(sp500_close_train, 1)
autoplot(sp500_close_train_diff)
```




```
# There's still a large amount of variance in Days 200 to 300
# Which equates to July 25th, 2008 to December 15th, 2008.
p2 <- ggplot(data = sp500_close[200:300,], aes(x = Date, y = Close)) + geom_line(alpha = 0.5) +
              ggtitle("S&P 500 Closing Price - July 2008 to December 2008") +
                        labs(x = "Date", y = "Closing Price")
p2
```



Here we see that the closing price really starts to nose dive - that's why there is still variance after differencing.

Now that we know why that's happening, let us test our new training data that has been differenced.

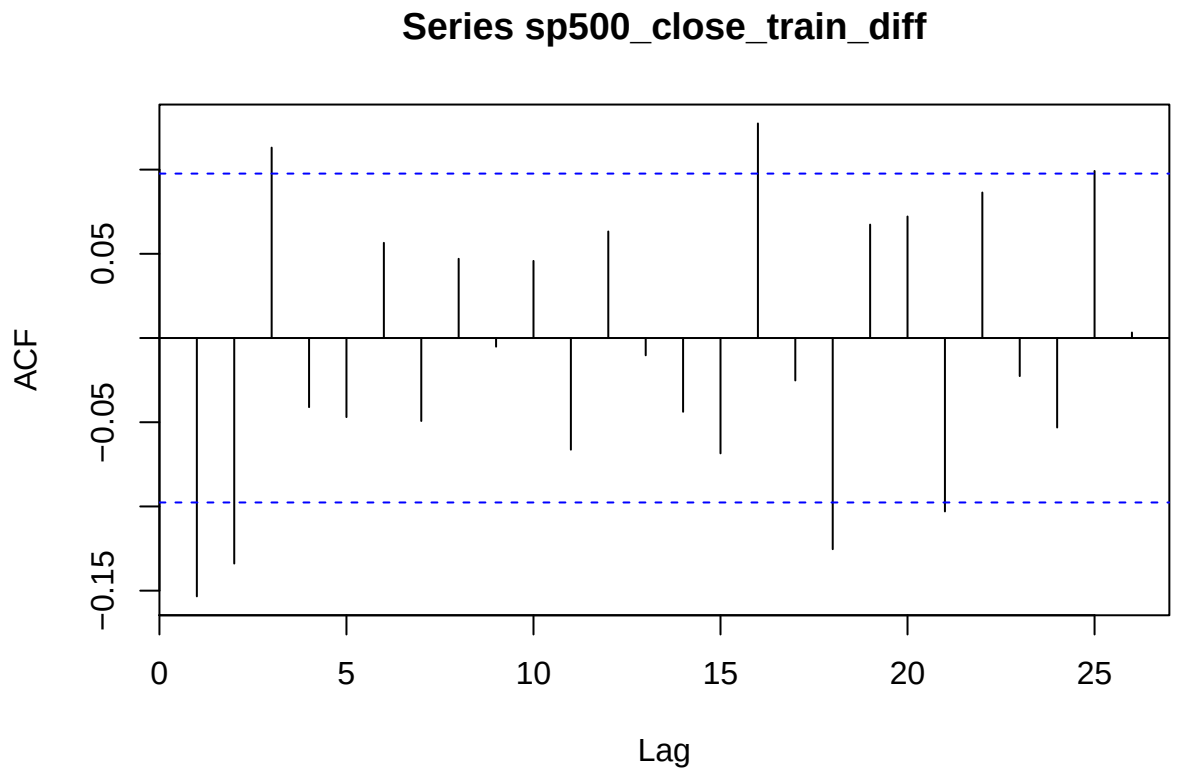
```
# Run adf test on difference training set
adf.test(sp500_close_train_diff)
```

```
## Warning in adf.test(sp500_close_train_diff): p-value smaller than printed
## p-value

##
##  Augmented Dickey-Fuller Test
##
## data:  sp500_close_train_diff
## Dickey-Fuller = -7.1493, Lag order = 7, p-value = 0.01
## alternative hypothesis: stationary
```

```
# Seems to improve it significantly
# Our p-value drops to 0.01 - we should perform the other checks.
```

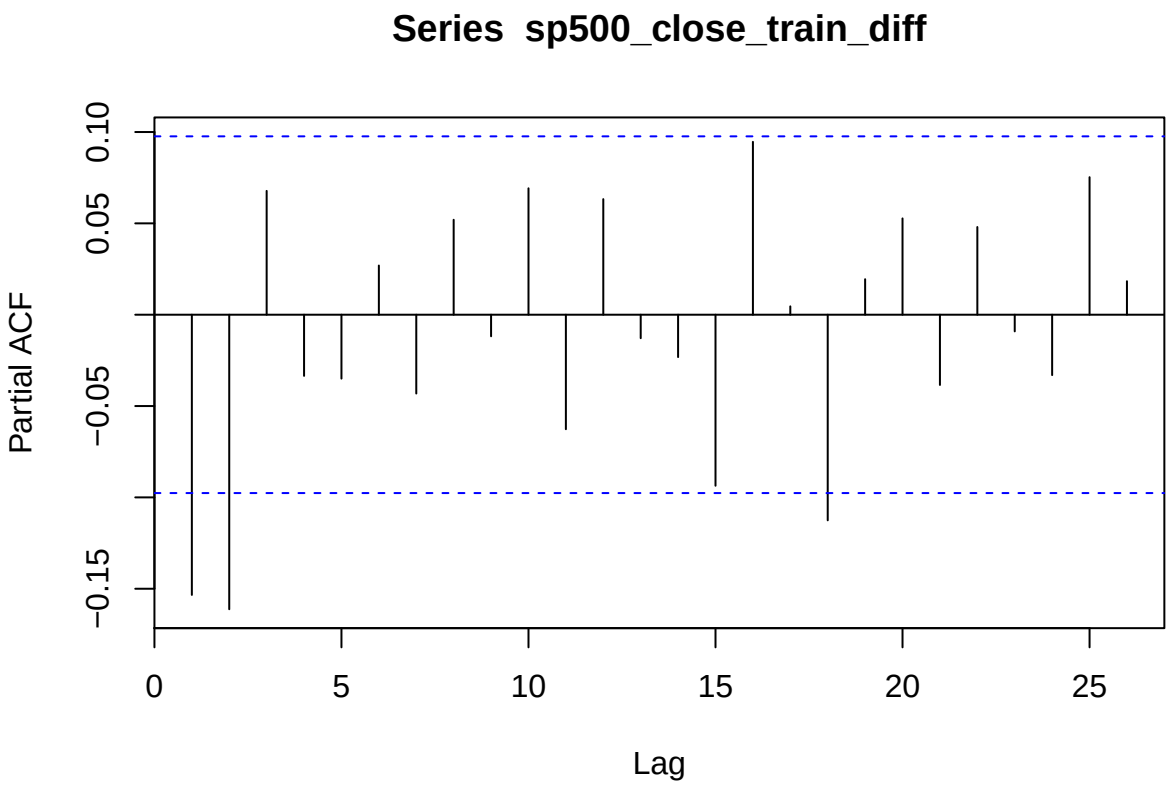
A. Our ADF test confirms that the differenced training data is indeed stationary.

```r
# Check the ACF plot
acf(sp500_close_train_diff)
```



B. Our ACF plot seems to indicate that it is an AR(2) model.

```r
# Check the PACF plot
pacf(sp500_close_train_diff)
```



C. Our PACF plot seems to indicate that it is an MA(2) model.

```
# Check the EACF
eacf(sp500_close_train_diff)
```

```
## AR/MA
##   0 1 2 3 4 5 6 7 8 9 10 11 12 13
## 0 x x x o o o o o o o o  o  o  o
## 1 x x o o o o o o o o o  o  o  o
## 2 x o o o o o o o o o o  o  o  o
## 3 x o x o o o o o o o o  o  o  o
## 4 x x x o x o o o o o o  o  o  o
## 5 x x o x o o o o o o o  o  o  o
## 6 x x o x o o o o o o o  o  o  o
## 7 x x x o x x o o o o o  o  o  o
```

D. Our EACF seems to indicate that our model is an AR(2), MA(1) or an AR(2), MA(2).

```
# Test an ARMA(2, 2) model with our differenced data
Arima(sp500_close_train_diff, c(2,0,2))
```

```
## Series: sp500_close_train_diff
## ARIMA(2,0,2) with non-zero mean
##
## Coefficients:
```

```
##           ar1      ar2     ma1     ma2     mean
##       -0.5705  -0.3506  0.4087  0.1416  -0.0014
## s.e.   0.2298   0.2059  0.2443  0.2153   0.0009
##
## sigma^2 estimated as 0.0005514:  log likelihood=942.51
## AIC=-1873.03   AICc=-1872.81   BIC=-1849.03
```

```
# Call auto arima on the differenced data to see if it has a better outcome
# and matches what we have discovered up above
auto.arima(sp500_close_train_diff)
```

```
## Series: sp500_close_train_diff
## ARIMA(0,0,3) with non-zero mean
##
## Coefficients:
##           ma1      ma2     ma3     mean
##       -0.1634  -0.1254  0.0836  -0.0014
## s.e.   0.0494   0.0511  0.0464   0.0009
##
## sigma^2 estimated as 0.0005522:  log likelihood=941.73
## AIC=-1873.45   AICc=-1873.3   BIC=-1853.46
```

```
# Final Model: ARIMA(2,0,2)(0,0,0)[0]
```

Calling auto arima gives us a much different answer; in this case, I will not be using the output from auto.arima and generating the model manually.

Question 4 (2 credits)

Fit your final model, write down the model (You may write down only the non-seasonal part, if you model contains seasonality).

Report the significance of the model coefficients.

Hints:

- Check Homework 2 - Problem 3 - Question 1(b) and 1(c) for how to write a model and how to define significance.

Answer:

$$(Y_t) = -0.5705 \cdot (Y_{t-1}) - 0.3506 \cdot (Y_{t-2}) + e_t + 0.4087 \cdot e_{t-1} + 0.1416 \cdot e_{t-2}$$

The mean isn't exactly 0 but it is close enough.

```
# Fit your final model
arima_fit = Arima(sp500_close_train_diff, c(2,0,2))
arima_fit
```

```
## Series: sp500_close_train_diff
## ARIMA(2,0,2) with non-zero mean
##
```

```
## Coefficients:
##           ar1      ar2     ma1     ma2     mean
##       -0.5705  -0.3506  0.4087  0.1416  -0.0014
## s.e.   0.2298   0.2059  0.2443  0.2153   0.0009
##
## sigma^2 estimated as 0.0005514:  log likelihood=942.51
## AIC=-1873.03   AICc=-1872.81   BIC=-1849.03
```

```
# Report the significance of the coefficents
# A coefficient is significant if its magnitude is at least twice as large as its standard error.

# Coefficients:
#           ar1      ar2     ma1     ma2     mean
#       -0.5705  -0.3506  0.4087  0.1416  -0.0014
# s.e.   0.2298   0.2059  0.2443  0.2153   0.0009
```

Based on the output above, the only coefficient that is significant is ar1.

Question 5 (3 credits)

Forecast on the testing set. Provide RMSE.

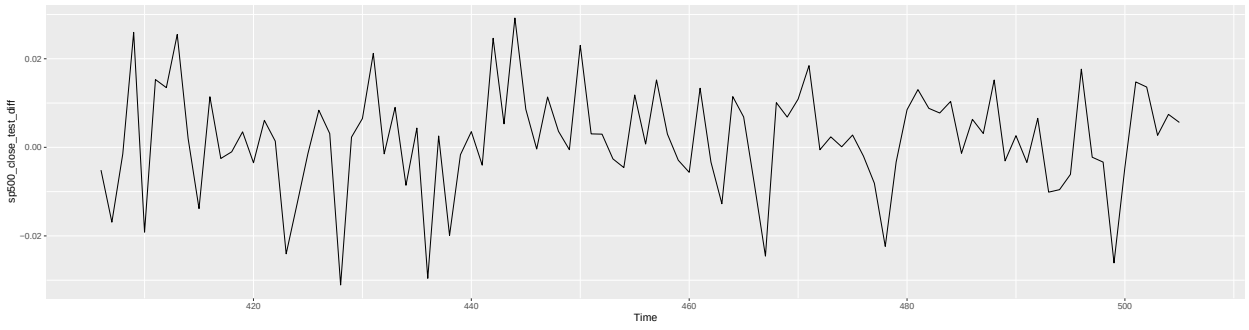
Plot the fitted value, as well as 80% and 95% prediction intervals, superimposed on the raw data.

Explain whether your selected model fit the data well.

Hint:

- Please check the code of Lecture 7, where similar things are done for the US Consumption data, CO2 data and Bitcoin data
- If you made transformations on your training data, please use the same transformation on your testing data as well

```
# Perform the same transformation as we did above - difference the testing data by 1
sp500_close_test_diff <- diff(sp500_close_test, 1)
autoplot(sp500_close_test_diff)
```



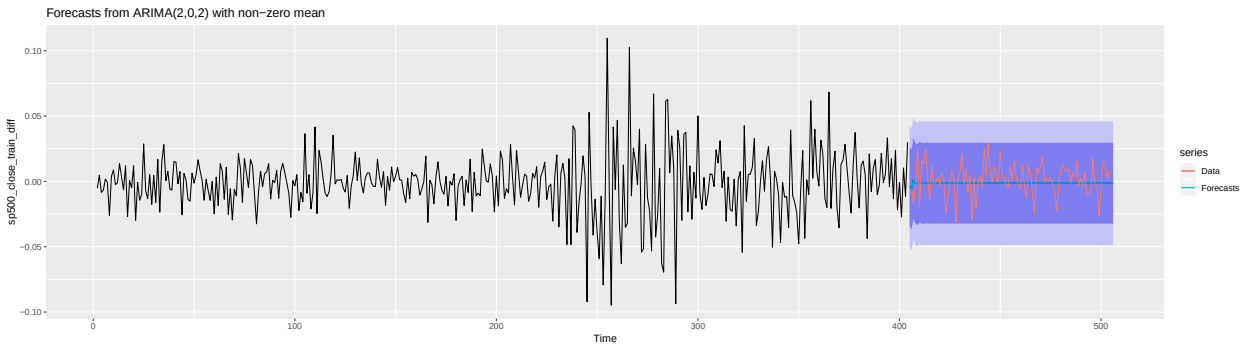

```
# It will be tough to forecast this data - it's not very stationary!

# Forecast on the testing set
```

```r
arima_forecast <- forecast(arima_fit, h = length(sp500_close_test))

# Provide the RMSE - using the new accuracy function
accuracy(arima_forecast, sp500_close_test_diff)[2,2]
```

```
## [1] 0.0122995
```

```r
# Plot the fitted value, as well as $80\%$ and $95\%$ prediction intervals, superimposed
# on the raw data.
autoplot(arima_forecast) +
  autolayer(sp500_close_test_diff, series="Data") +
  autolayer(arima_forecast$mean, series="Forecasts")
```

```
## Warning: Removed 1 rows containing missing values (geom_path).
```



$RMSE$: 0.0122995

Based on the RMSE being just 0.01, this means that the predicted values of the model were not far from the actual values of the testing data. When looking at the plot with the fitted values with the 80% and 95% prediction intervals, it is reasonable to conclude that the model is accurately fitting the data well.

Question 6 (6 credits)

Please do the same forecasting task in Question 5, with XGBoost and LSTM.

Report the RMSE from each method.

For each method, plot the fitted value, superimposed on the raw data (prediction intervals are not required).

Comments on the performance of XGBoost and LSTM compared with ARIMA, in terms of both accuracy and computational speed. Which one is better for your data?

Hint:

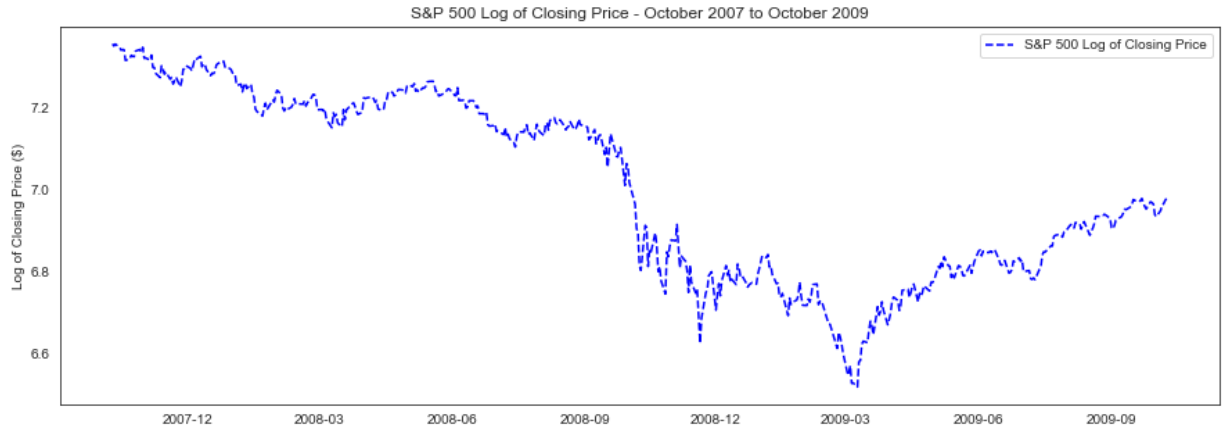
1. Please feel free to use Python or other software to run XGBoost or LSTM if your code has been ready. But please paste all code as comments in the area below for reproducibility reason.
2. Some of my experiences are: ARIMA has the advantages of being fast, and being able to provide prediction intervals as a statistical model. But XGBoost and LSTM may provide better forecasting accuracy.
3. For LSTM, it's OK if you only have time to try a couple of layers with a few neurons.

==========

All of the code below was written in Python 3.6 using a Jupyter notebook & various libraries. All the lines with the double '##' are comments. This notebook can be shared if needed. For both the XGBoost & LSTM models, the data prep and visualization steps were the same.

A. Data Loading, Prep, and Visualization of Data

```
## Load the data from the CSV file
# sp500_df = pd.read_csv('GSPC.csv')

## Convert to date for easier plotting and overall data manip<br>
# sp500_df["Date"] = pd.to_datetime(sp500_df["Date"], format = "%Y-%m-%d")<br>

## To mirror what we did in R, we create a log of the closing price column<br>
# sp500_df["Log Close"] = np.log(sp500_df["Close"])

## Set x variable as the date for sp500
# x = sp500_df["Date"]
## Set y var as the closing price for sp500
# y = sp500_df["Log Close"]

## Reset seaborn to the default background - for better viewing
# sns.set_style("white")
## Generate a new figure object
# plt.figure(1, figsize = (15, 5))
## Plot the first type / subtype combination
# plt.plot(x, y, 'b--', label = "S&P 500 Log of Closing Price")
# plt.ylabel("Log of Closing Price ($)")
# plt.title("S&P 500 Log of Closing Price - October 2007 to October 2009")
# plt.legend()
## Show the plot!
# plt.show()
```



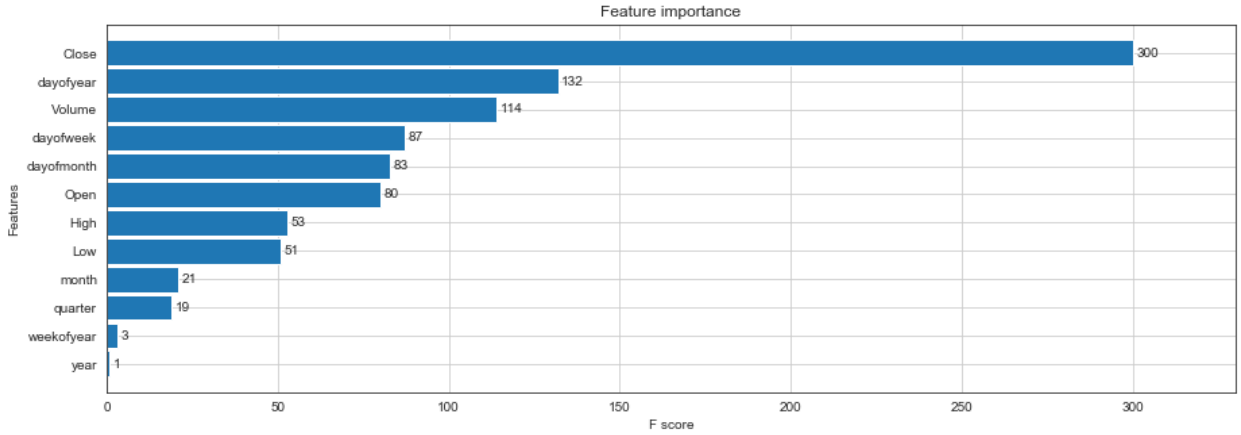
We simply visualize the log of closing price to confirm the data looks like our earlier analysis.

```
## Create a function that creates new features from the date field to help with creating a model
# def create_features(df):
```

```
  #  """
   # Creates time series features from datetime index
    # """
    # df['dayofweek'] = df['Date'].dt.dayofweek
    # df['quarter'] = df['Date'].dt.quarter
    # df['month'] = df['Date'].dt.month
    # df['year'] = df['Date'].dt.year
    # df['dayofyear'] = df['Date'].dt.dayofyear
    # df['dayofmonth'] = df['Date'].dt.day
    # df['weekofyear'] = df['Date'].dt.weekofyear

    # X = df[["dayofweek", "quarter", "month", "year",
      #       "dayofyear","dayofmonth","weekofyear", "Open", # Hard code all of the columns
        #      "High", "Low", "Close", "Volume", "Log Close"]] # To save time

    # return X

## Use the function to create features that we'll need for the model
# sp500_data = create_features(sp500_df)

## Use the first 80% of the data as training

# X_train = sp500_data.head(n = 404).drop("Log Close", axis = 1)
# y_train = sp500_data.head(n = 404)["Log Close"]

## Use the remaining 20% of data to test
# X_test = sp500_data.loc[~sp500_data.index.isin(X_train.index)].drop("Log Close", axis = 1)
# y_test = sp500_data.loc[~sp500_data.index.isin(y_train.index)]["Log Close"]
```

B. XGBoost Model Creation & Evaluation

```
## Create an XGBoost Regressor
# reg = xgb.XGBRegressor(n_estimators=1000)
# reg.fit(X_train, y_train,
  #       eval_set=[(X_train, y_train), (X_test, y_test)],
   #      early_stopping_rounds=100,
    #     verbose=False)

## Reset seaborn to the default background - for better viewing
# sns.set_style("white")
## Generate a new figure object
# fig, ax = plt.subplots(figsize=(15,5))
# xgb.plot_importance(reg, height=0.9, ax = ax)
## Show the plot!
# plt.show()
```



This plot demonstrates what our most important features are. The S&P 500 data is highly correlated to the previous day's stock price. Therefore it makes sense that the Close price is the most important feature.

```
## Generate the predict Log price values using our XGB model
# xgb_predict = reg.predict(X_test)

## Create two new variables for plotting the actual values compared to our predicted values
## Actual values
# y_true = y_test
## Predicted values
# y_pred = xgb_predict

### Reset seaborn to the default background - for better viewing
# sns.set_style("white")
## Generate a new figure object
# plt.figure(1, figsize = (15, 5))
## Plot the first type / subtype combination
# plt.plot(x[404:], y_true, 'b--', label = "Predicted Values of S&P 500 Log Closing Price")
# plt.plot(x[404:], y_pred, 'r--', label = "Actual Values of S&P 500 Log Closing Price")
# plt.ylabel("Log of Closing Price ($)")
# plt.title("S&P 500 Log of Closing Price - May 2009 to October 2009 using XGBoost")
# plt.legend()
## Show the plot!
# plt.show()
```

```
## Report the RMSE
# np.sqrt(mean_squared_error(y_true, y_pred))
```

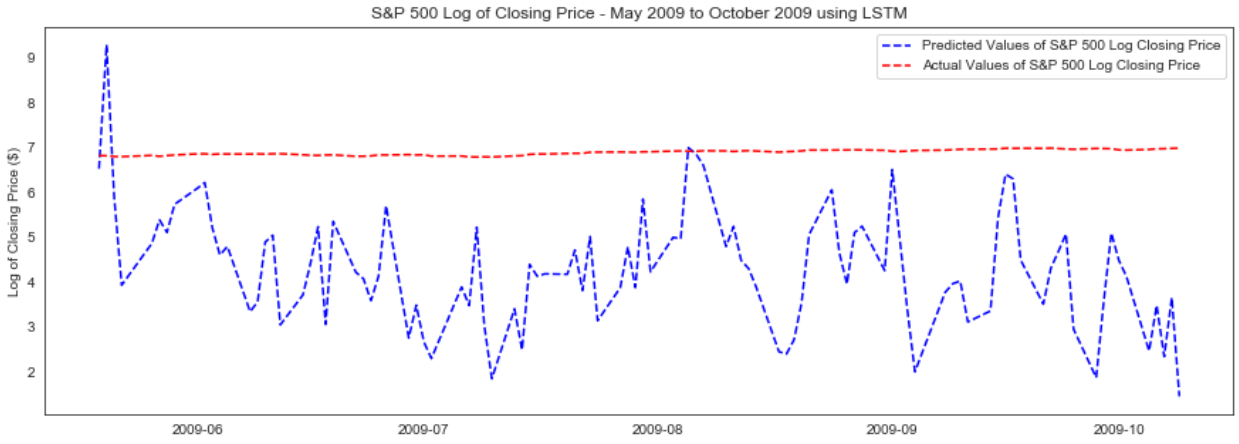
$$RMSE : 0.050678456416386006$$

We are reporting an RMSE of ~0.05 using the XGBoost model. The plot shows us that we have captured the overall model pretty well. The only thing we lack is the 80% and 95% conf. intervals to really tell us what is happening.

C. LSTM Model Creation & Evaluation

```
## Import all the libraries - pretty much everything from Keras & Tensorflow
# from tensorflow.keras.models import Sequential
# from tensorflow.keras.layers import *
# from tensorflow.keras.optimizers import *

# Create a base LSTM model using keras
# def create_lstm_ts_model(activation = "relu", neurons = 1):

    ## Base model
    # model = Sequential()

    ## First and only layer
    # model.add(LSTM(neurons, activation = activation, input_shape = (X_train_arr.shape[1],
    # X_train_arr.shape[2])))

    ## Output layer
    # model.add(Dense(1))
    ##  Compile the completed model
    # model.compile(optimizer='adam', loss='mse')

    # return model

## Create the model using relu as our activation function, 100 neurons, and using the shape
## of the training data
# lstm_reg = create_lstm_ts_model(activation = "relu", neurons = 50)

## Transform our training/testing data into a numpy array because keras cannot handle data-
# frames as input
## And reshape the input to be in 3D
# X_train_arr = np.array(X_train)
# X_train_arr = X_train_arr.reshape(X_train_arr.shape[0], 1, X_train_arr.shape[1])

# X_test_arr = np.array(X_test)
# X_test_arr = X_test_arr.reshape(X_test_arr.shape[0], 1, X_test_arr.shape[1])

## Transform our outcome variables
# y_train_arr = np.array(y_train)
# y_test_arr = np.array(y_test)
```

```
## Fit our data
# history = lstm_reg.fit(X_train_arr, y_train_arr, epochs = 20,
  #                       batch_size = 5, validation_data=(X_test_arr, y_test_arr),
    #                     verbose=2, shuffle = False)

## Save the model!  This one finally produced good results!
# lstm_reg.save("SP500_lstm_reg.h5")

## Generate the predict Log price values using our LSTM model
# lstm_predict = lstm_reg.predict(X_test_arr)

## Check the shape of the test data
# y_true_lstm = y_test

## Reshape the predicted values to match y_test to allow us to plot them together
# lstm_predict = lstm_predict.reshape(101,)

## Reset seaborn to the default background - for better viewing
# sns.set_style("white")

## Generate a new figure object
# plt.figure(1, figsize = (15, 5))
## Plot the first type / subtype combination
# plt.plot(x[404:], lstm_predict, 'b--', label = "Predicted Values of S&P 500 Log Closing Price")
# plt.plot(x[404:], y_test, 'r--', label = "Actual Values of S&P 500 Log Closing Price")
# plt.ylabel("Log of Closing Price ($)")
# plt.title("S&P 500 Log of Closing Price - May 2009 to October 2009 using LSTM")
# plt.legend()
## Show the plot!
# plt.show()
```




```
# Report the RMSE
# np.sqrt(mean_squared_error(y_true_lstm, lstm_predict))
```

$$RMSE : 2.8747696499682753$$

Based on the output from the LSTM model, it is pretty clear that it is overfitting the data significantly. The actual log of closing price hovers around 6 while our predicted values bounce around from 9 to 2. The

RMSE using LSTM is 2.8, which is much worse than our XGBoost model and the output from our Arima analysis. Our final conclusion is that the model fitted using Arima was the strongest for S&P 500 Closing price from October 2007 to 2009.