Dow Jones Index Estimator

Prepared by: Josh Levine (jl2108), Harsh Patel (hkp49), Jaini Patel (jp1891), Yifan Liao (yl1463) and Aayush Shah (avs93).

Goal

Our goal is to analyze the moving average index of the Dow Jones using a Time Series and ARIMA model. In our analysis we aim to predict one day or one week in the future of the index. And then analyze our prediction.

The Data

We will be using two data sets, one is index data on a daily basis and the other on a weekly basis. The data starts in 1896 and runs into 2020. Our starting file was the daily index and we wrote a python script to combine in by week. In our analysis we are using the opening data as that days index value.

First few rows of the DAILY index CSV file used

Date	Open	High	Low	Close	Volume
1896-05-27	29.39	29.39	29.39	29.39	
1896-05-28	29.11	29.11	29.11	29.11	
1896-05-29	29.43	29.43	29.43	29.43	
1896-06-01	29.4	29.4	29.4	29.4	
1896-06-02	29	29	29	29	
1896-06-03	28.8	28.8	28.8	28.8	
1896-06-04	28.93	28.93	28.93	28.93	
1896-06-05	29.2	29.2	29.2	29.2	
1896-06-08	28.83	28.83	28.83	28.83	

First few rows of the WEEKLY index CSV file we used

Year	Week Number	Average
1896	22	29.33
1896	23	29.065999999999995
1896	24	28.429999999999996
1896	25	29.074
1896	26	27.666000000000004
1896	27	25.479999999999997
1896	28	25.310000000000002
1896	29	23.796
1896	30	22.742

Python script used to generate the weekly CSV using the daily CSV

```
import csv #import needed to read and write csv files
import datetime #import to handle date time objects

values = [] #initialize the values array

#open the csv file
with open("C:\\Users\\Josh\\Documents\\Grad School\\Statistics\\Final Project\\dji_d.csv") as csvfile:
    reader = csv.reader(csvfile, delimiter=',')#create the reader object
    next(reader)#skip the header
    for row in reader:#iterate over each row to create the values array
        dateString = row[0]#get the string value of the date (i.e 6/20/1978)
        #Turn the date string into a date time object
        #We do this so we can extract week data from the data
        date = datetime.datetime.strptime(dateString, '%Y-%m-%d').date()
        #Add to the values list the year, the week of the year and the index
        values.append([date.year, date.isocalendar()[1], float(row[1])])

weeklyAverage = [] #initialize the weeklyAverage array

                    #Year           Week Number      Index (float)     Count
#Add the first row to weeklyAverages
weeklyAverage.append([values[0][0], values[0][1], float(values[0][2]), 1])
#Iterate over each row of the values array to add to the weeklyAverages array
for i in range(len(values)):
    flag = True #flag if a row in the averages array has been created or not
    print(values[i][0]) #Just making sure the program is running
    for j in range(len(weeklyAverage)): #Iterate over the weeklyAverages array
#If a row in the weekly averages has been created for this year and week,
#add to that row/ If not create a new row for that week of that year
        if(values[i][0] == weeklyAverage[j][0] and values[i][1] == weeklyAverage[j][1]):
            weeklyAverage[j][2] += float(values[i][2])#Add to the index total of that specific week
            weeklyAverage[j][3] += 1 #Add to the index of that week by 1
            flag = False #set the flag to false because a row is already created for this week
    if flag: #If the flag was never set to false, create a row for that week and set the count to 1
        #starting a row for that week of that year
        weeklyAverage.append([values[i][0], values[i][1], values[i][2], 1])

#Now the data has been combined in a weekly manner
#Next is to create the new csvfile for further analysis

#Open a new csv file
file = open("C:\\Users\\Josh\\Documents\\Grad School\\Statistics\\Final Project\\dji_WEEKLY.csv", 'w', n
with file:
    header = ['Year', 'Week Number', 'Average'] #Create the header for our new csv file
    writer = csv.DictWriter(file, fieldnames = header)#Create a writer object
    writer.writeheader() #Writes the header

    for k in range(len(weeklyAverage)): #Iterate over the weekly averages to write to the csv
        #Writes the the three column values to the csv
        #Also calculates averege week by diving total index by count
        writer.writerow({'Year' : weeklyAverage[k][0],
```

```
                        'Week Number': weeklyAverage[k][1],
                        'Average': weeklyAverage[k][2]/weeklyAverage[k][3]})
        #Another check just to make sure everything is running
        print(weeklyAverage[k][2]/weeklyAverage[k][3])
```

The script is fully commented, so you may go through each line to learn how it was combined weekly.

Analysis with ARIMA: Auto Regressive Integrated Moving Average

Arima models are used when we can assume the data we are working with is non-stationary. And this is clearly true when it comes to Dow Jones Index data. We say that time series data sets are stationary when their means, variance and auto-covariance don't change during time. Most economic Time Series are non-stationary. We can apply ARIMA models to the index or any stock price of our choosing. We denote the forecasting model by ARIMA(p, d, q) where:

$Y_t = c + \phi_l y_{dt} - 1 + \phi_p y_{dtp} + ... + \theta_1 e_{t-1} + \theta_q e_{tq} + e_t$

In ARIMA, p denotes the number of auto regressive terms, d denotes the number of time that the set should be differentiated for making it stationary and the last parameter q denotes the number of invertible moving average terms. Creating the model is done by iterative approaches using 4 steps.

1. Identification: Find the best values reproducing the time series variable to forecast.
2. Analysis and Differentiation: Study the time series, in our study we will use different statistical tools such as ACF and PACF tests.
3. Adjust the Arima Model: Extract and determine coefficients and adjust the model.
4. Prediction: Once the best model has been selected, we can make a forecast based on probabilistic future values.

For our approach we will be using the auto.arima function that will automatically return the best ARIMA model according to the data.

First we will conduct an ADF test for both our daily and weekly index values:

```
adf.test(dji.daily.ts)
```

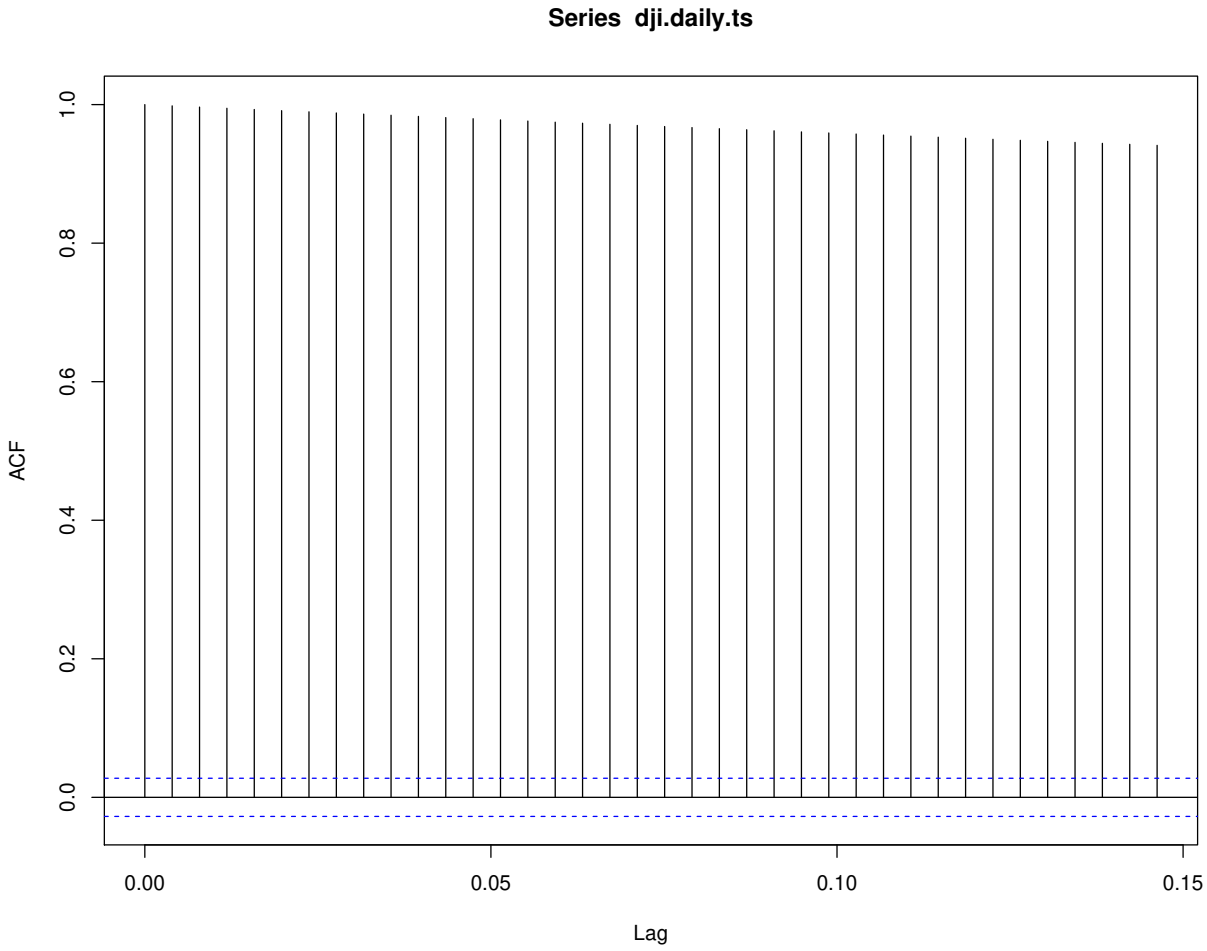
```
##
##  Augmented Dickey-Fuller Test
##
## data:  dji.daily.ts
## Dickey-Fuller = -1.4541, Lag order = 17, p-value = 0.8095
## alternative hypothesis: stationary
```

```
adf.test(dji.weekly.ts)
```

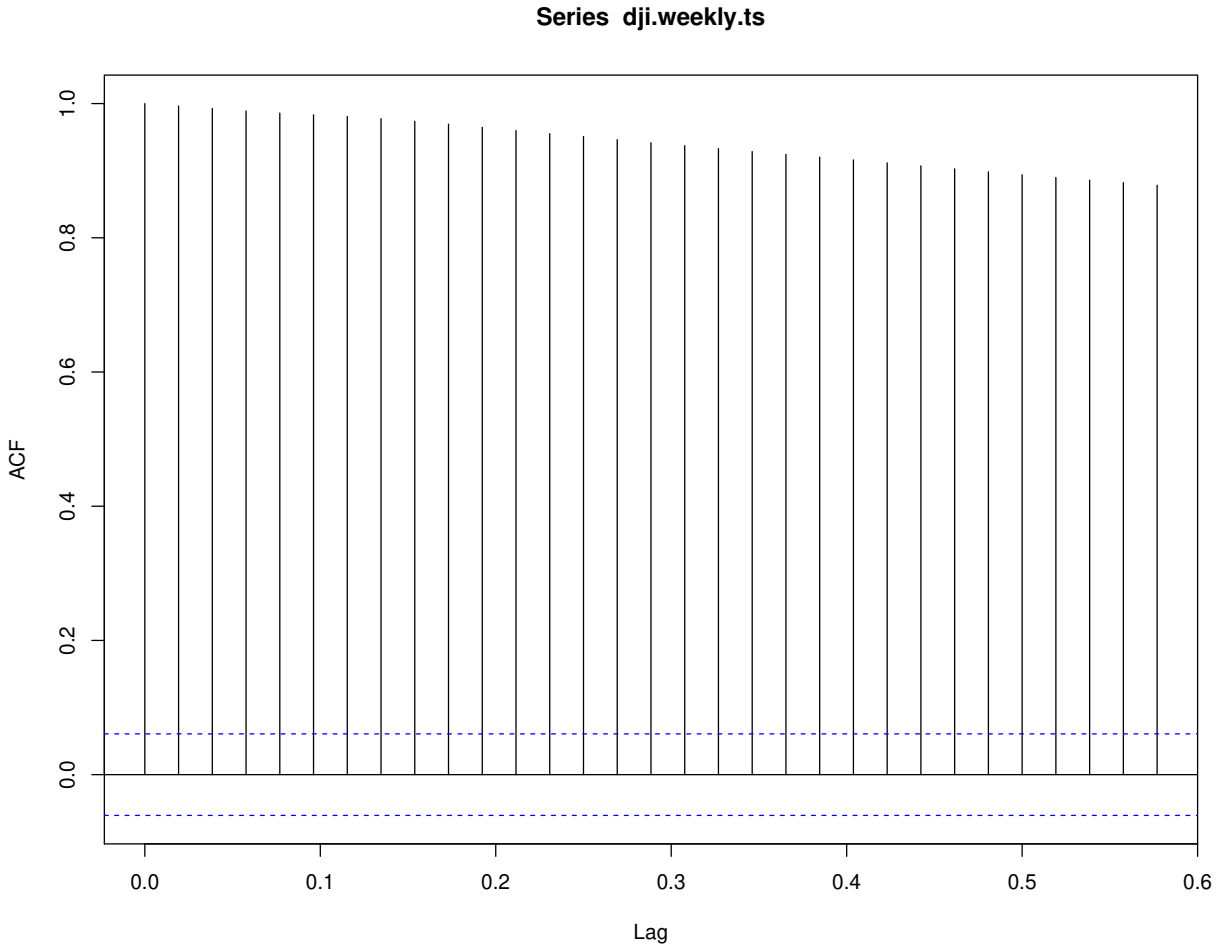
```
##
##  Augmented Dickey-Fuller Test
##
## data:  dji.weekly.ts
## Dickey-Fuller = -1.9791, Lag order = 10, p-value = 0.5871
## alternative hypothesis: stationary
```

After conducting the ADF test we now apply the ACF (Autocorrelation Function) and PACF (Partial Autocorrelation Function) to both the daily and weekly data set.

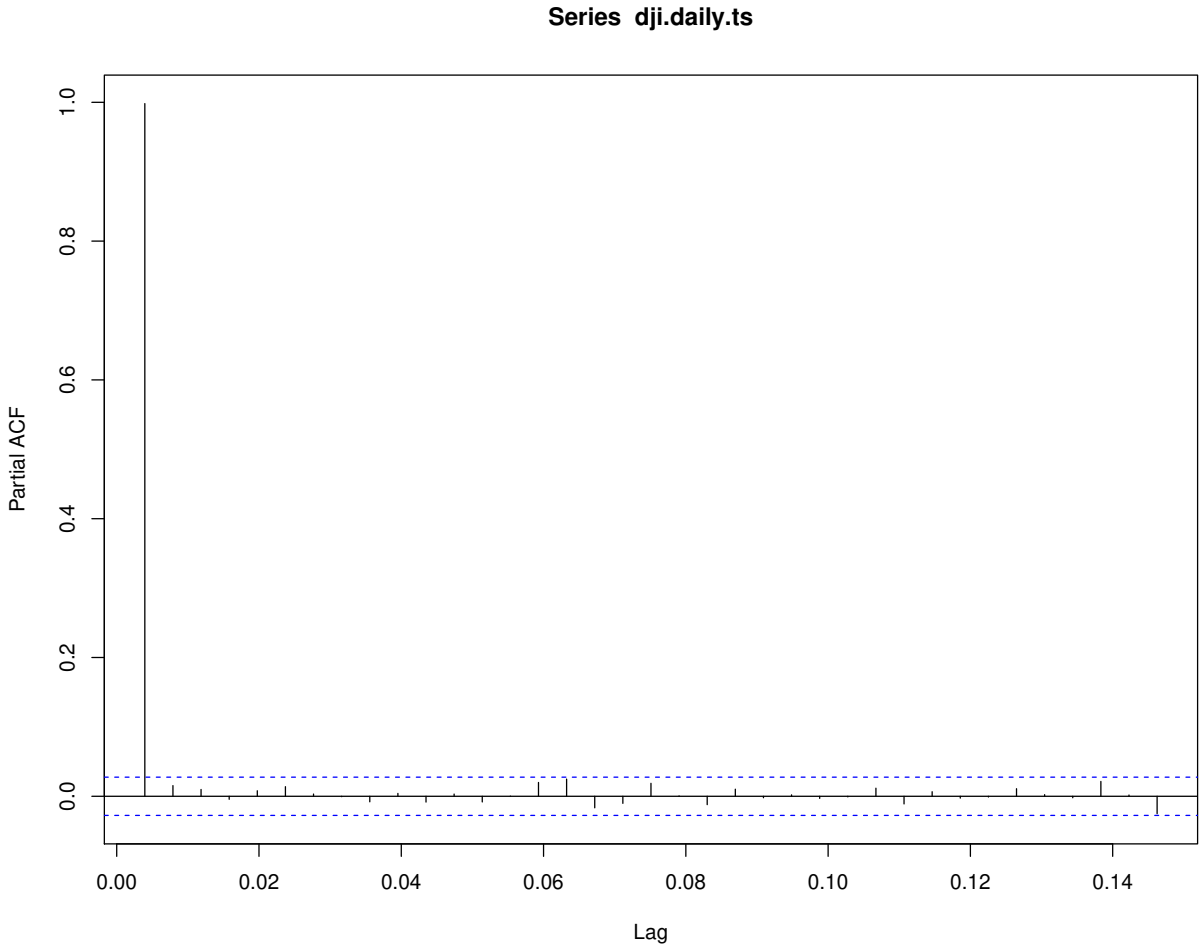
```
acf(dji.daily.ts)
```



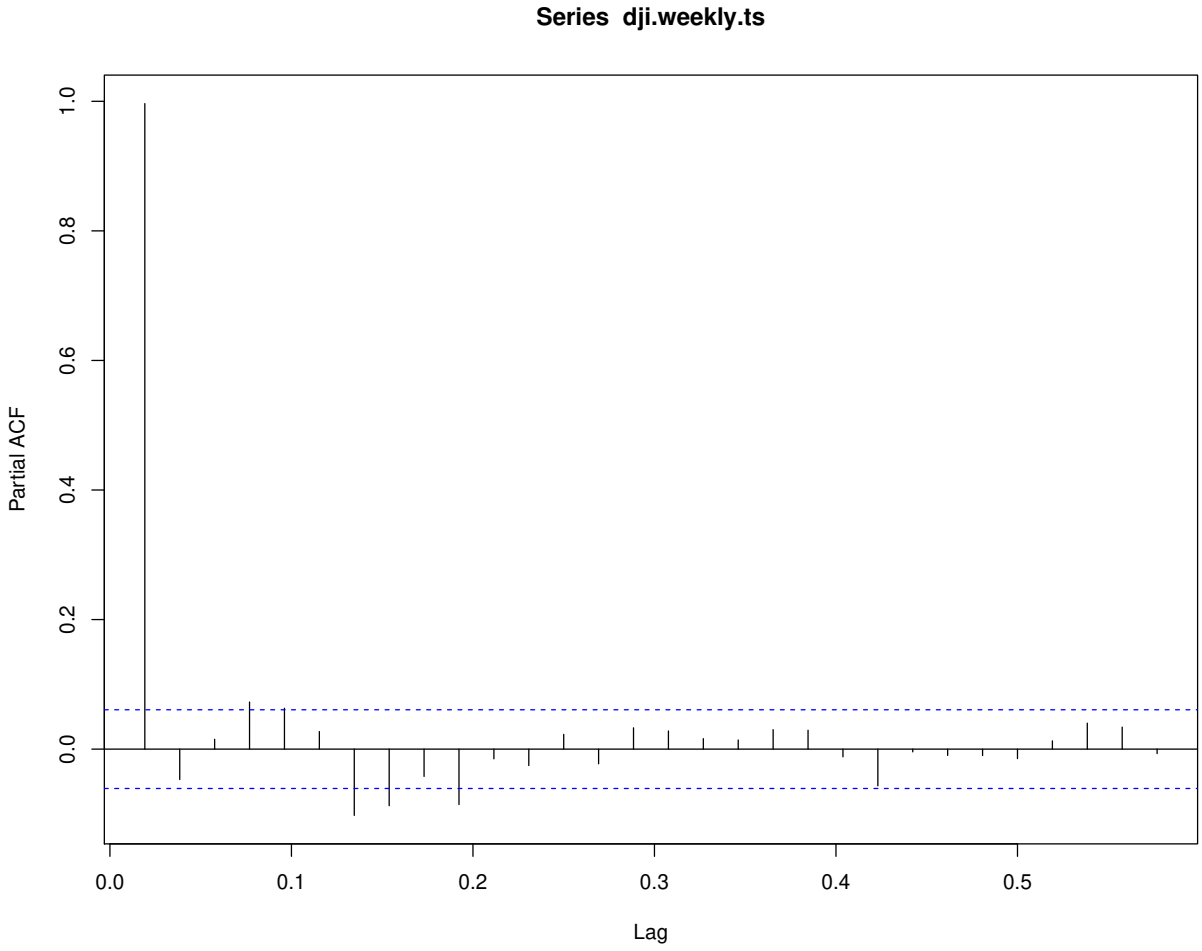
```r
acf(dji.weekly.ts)
```



```
pacf(dji.daily.ts)
```



```
pacf(dji.weekly.ts)
```



Autocorrelation refers to how correlated a time series is with its past data points. As in Autoregressive models, the ACF will decrease exponentially, that can be seen in both the daily and weekly ACF graphed above. The ACF is used to display the plot used to see the correlation between the points, up to and including the lag unit. We observe in both the daily and weekly data sets that the autocorrelations are significant for large number of lags. To identify the (p) order of the autoregressive model we use the PACF plot. For moving average models we use the ACF plot to identify the (q) order. Looking at both the daily and weekly PACF plots, we can see a significant spike ONLY at the first lag, which means that all higher order autocorrelations are effectively explained by the first lag autocorrelation. As we plan to use the auto.arima function which gives us the best approach for our data, we will not dive too deep into the analysis of finding model parameters.

Using the auto.arima function on both the daily and weekly data sets:

```
#Run the auto.arima function
daily.arima.model <- auto.arima(dji.daily.ts, lambda = "auto")
weekly.arima.model <- auto.arima(dji.weekly.ts, lambda = "auto")

#display the arima model for both the daily and weekly data
daily.arima.model
```

```
## Series: dji.daily.ts
## ARIMA(0,1,2) with drift
## Box Cox transformation: lambda= 1.159327
##
## Coefficients:
##           ma1      ma2    drift
##       -0.0461  -0.0280  12.9058
## s.e.   0.0141   0.0143   7.0431
##
## sigma^2 estimated as 292911:  log likelihood=-39025.02
## AIC=78058.04   AICc=78058.05   BIC=78084.16
```

```
weekly.arima.model
```

```
## Series: dji.weekly.ts
## ARIMA(0,1,0)
## Box Cox transformation: lambda= 0.5377399
##
## sigma^2 estimated as 14.16:  log likelihood=-2854.06
## AIC=5710.13   AICc=5710.13   BIC=5715.07
```

The details of the accuracy function are defined as:

1. ME: Mean Error - average of all the errors in a set.
2. RMSE: Root Mean Squared Error - standard deviation of the residuals (prediction errors). Residuals are a measure of how far from the regression line data points are.
3. MAE: Mean Absolute Error - the average of all absolute errors.
4. MPE: Mean Percentage Error - the computed average of percentage errors by which forecasts of a model differ from actual values of the quantity being forecast.
5. MAPE: Mean Absolute Percentage Error - a statistical measure of how accurate a forecast system is. It measures this accuracy as a percentage.
6. MASE: Mean Absolute Scaled Error - a scale-free error metric that gives each error as a ratio compared to a baseline's average error.
7. ACF1: Autocorrelation of errors at lag 1 - the correlation between values that are one time period apart. More generally, a lag k autocorrelation is the correlation between values that are k time periods apart.

```r
#display the accuracy of that model for the daily and weekly data set
accuracy(daily.arima.model)
```

```
##                       ME    RMSE      MAE        MPE      MAPE       MASE
## Training set -0.07332934 121.997 86.68198 -0.0105282 0.7795155 0.06136532
##                      ACF1
## Training set -0.002559159
```

```r
accuracy(weekly.arima.model)
```

```
##                     ME     RMSE      MAE        MPE     MAPE     MASE      ACF1
## Training set 12.34945 329.1197 198.1645 0.04983586 1.451326 0.117382 0.0697966
```

Now that we have our model summary we can check the residuals of the model. The RESIDUALS in a time series model are what is left over after fitting a model. The residuals are equal to the difference between the observations and the corresponding fitted values. Residuals are useful in checking whether a model has adequately captured the information in the data. A good forecasting method will yield residuals with the following properties:

1. The residuals are uncorrelated. If there are correlations between residuals, then there is information left in the residuals which should be used in computing forecasts.
2. The residuals have zero mean. If the residuals have a mean other than zero, then the forecasts are biased

Below is the mean and plot of the residuals for both the daily and weekly data sets. We can see that the daily set has a mean very close to zero while the weekly has a much larger mean. This means the data set for daily data will be better at forecasting than the weekly data set.

```r
mean(daily.arima.model$residuals)
```

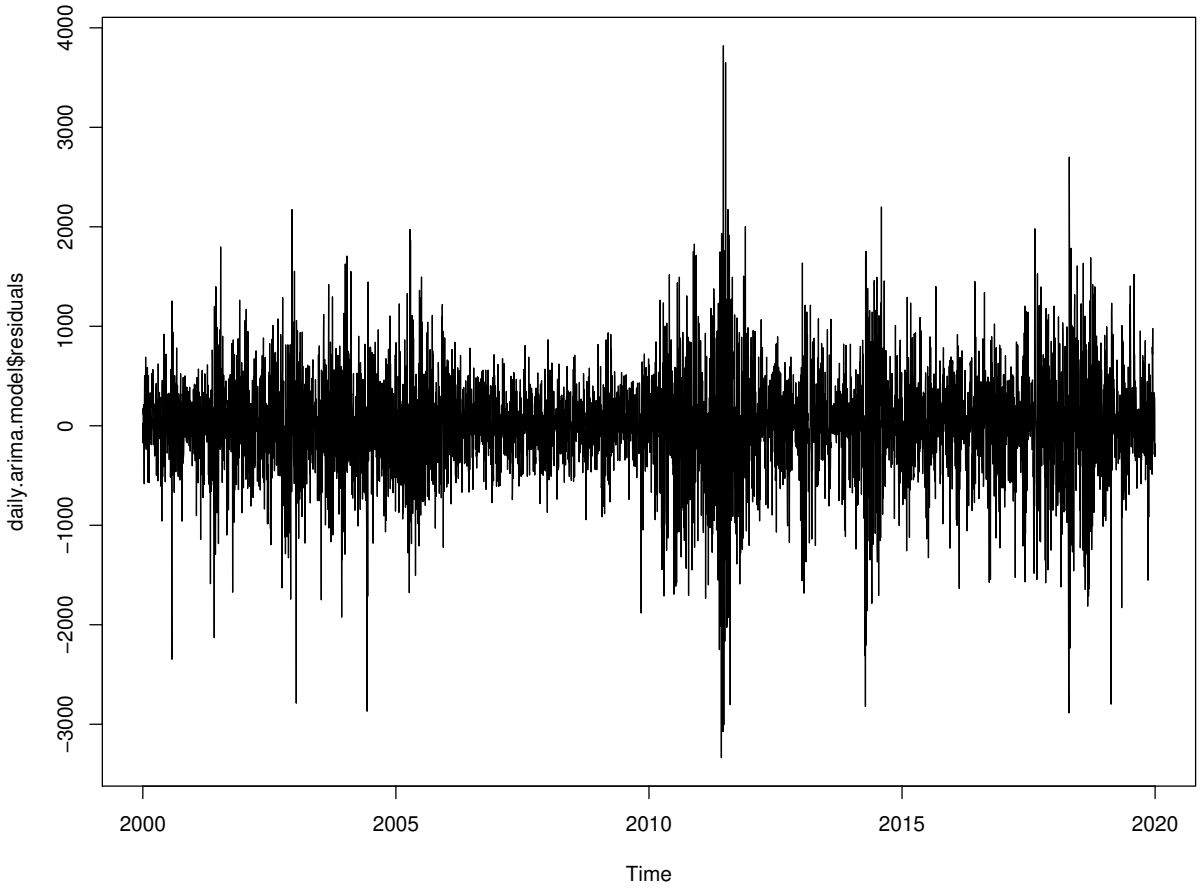
```
## [1] 0.008909637
```

```r
mean(weekly.arima.model$residuals)
```

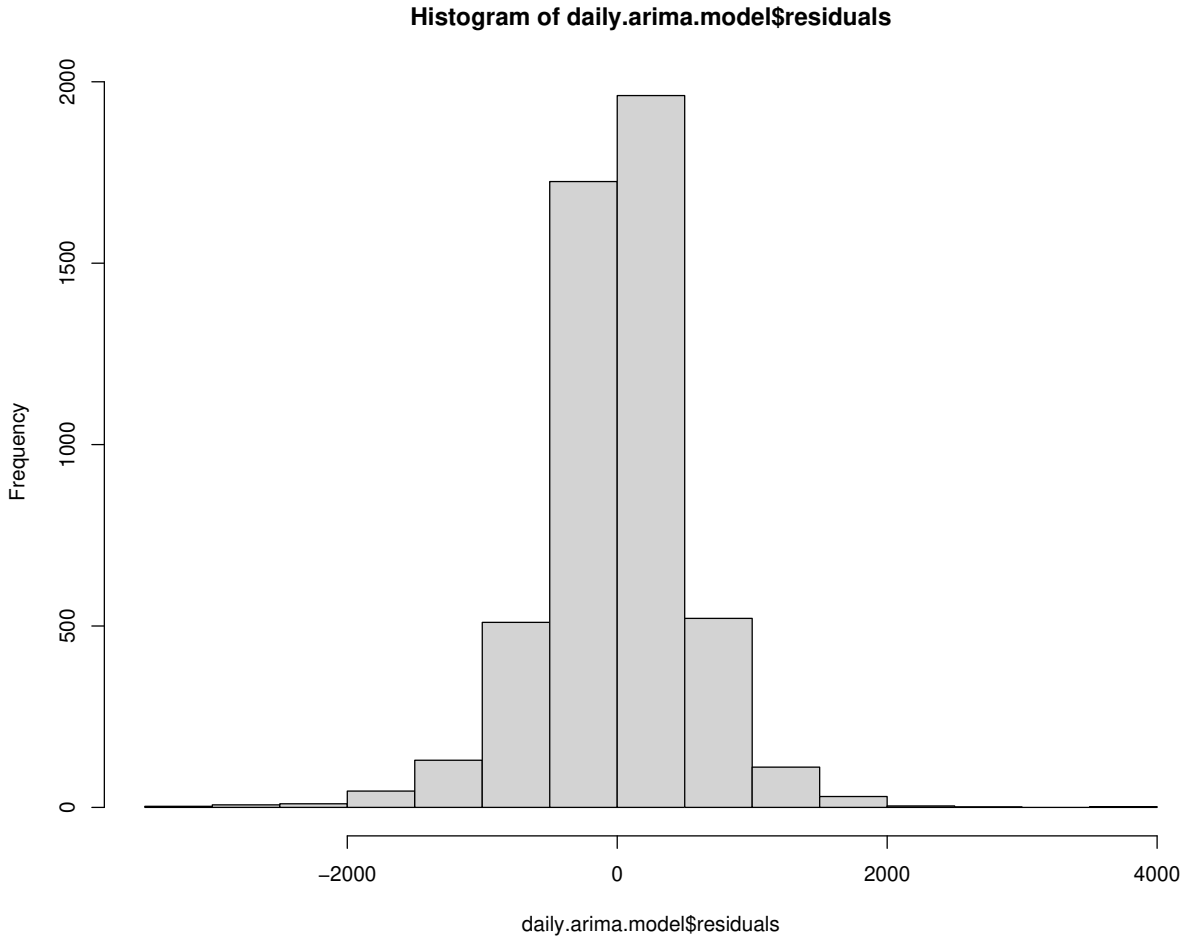
```
## [1] 0.1383966
```

Visualization of the Daily Residuals

```
plot(daily.arima.model$residuals)
```

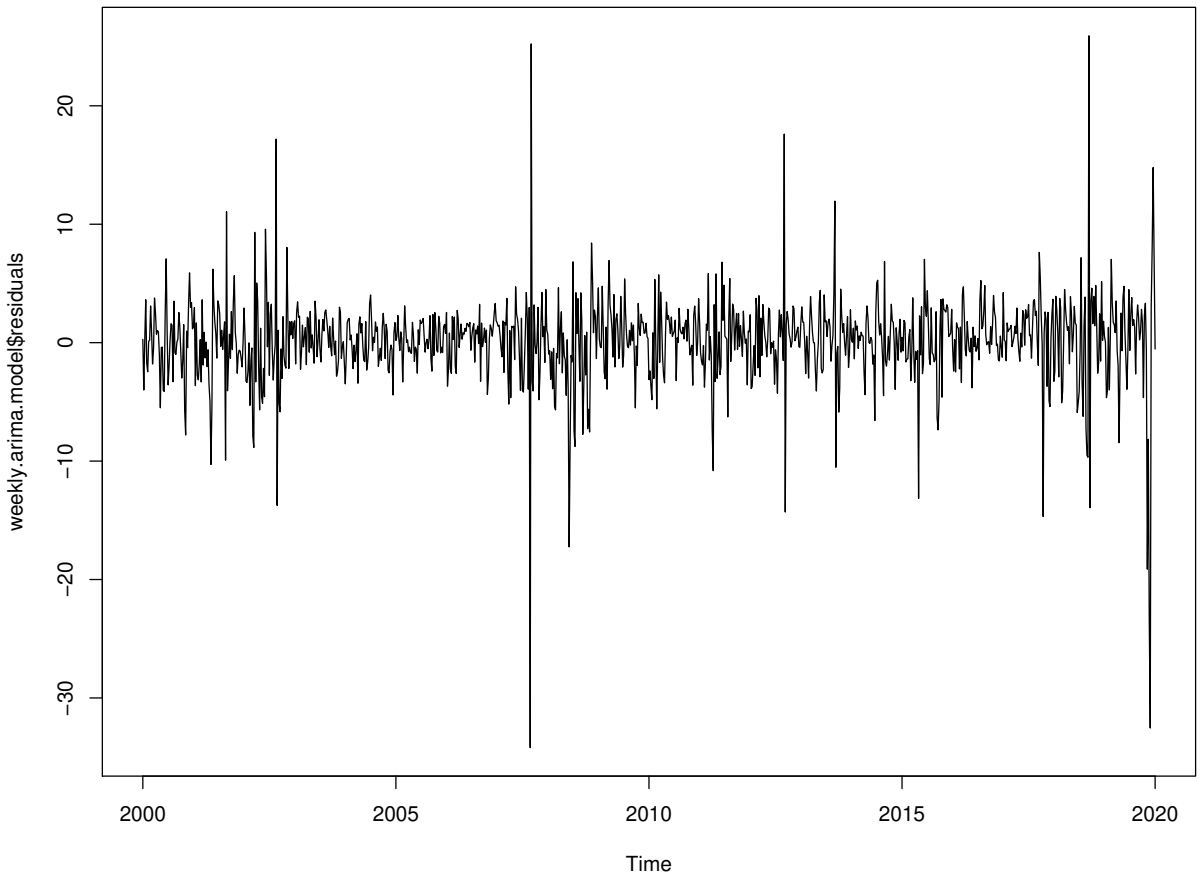


```r
hist(daily.arima.model$residuals)
```

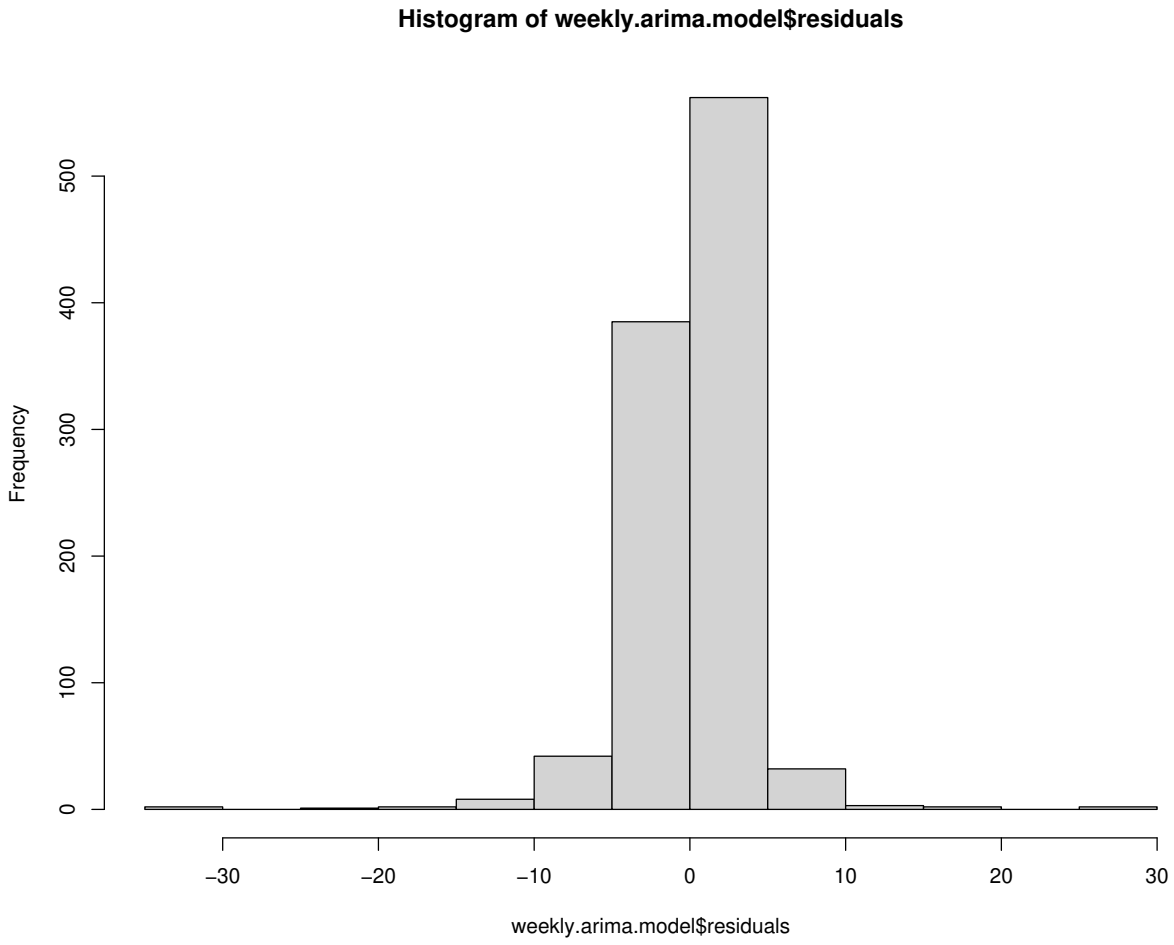


Visualization of the Weekly Residuals

```
plot(weekly.arima.model$residuals)
```



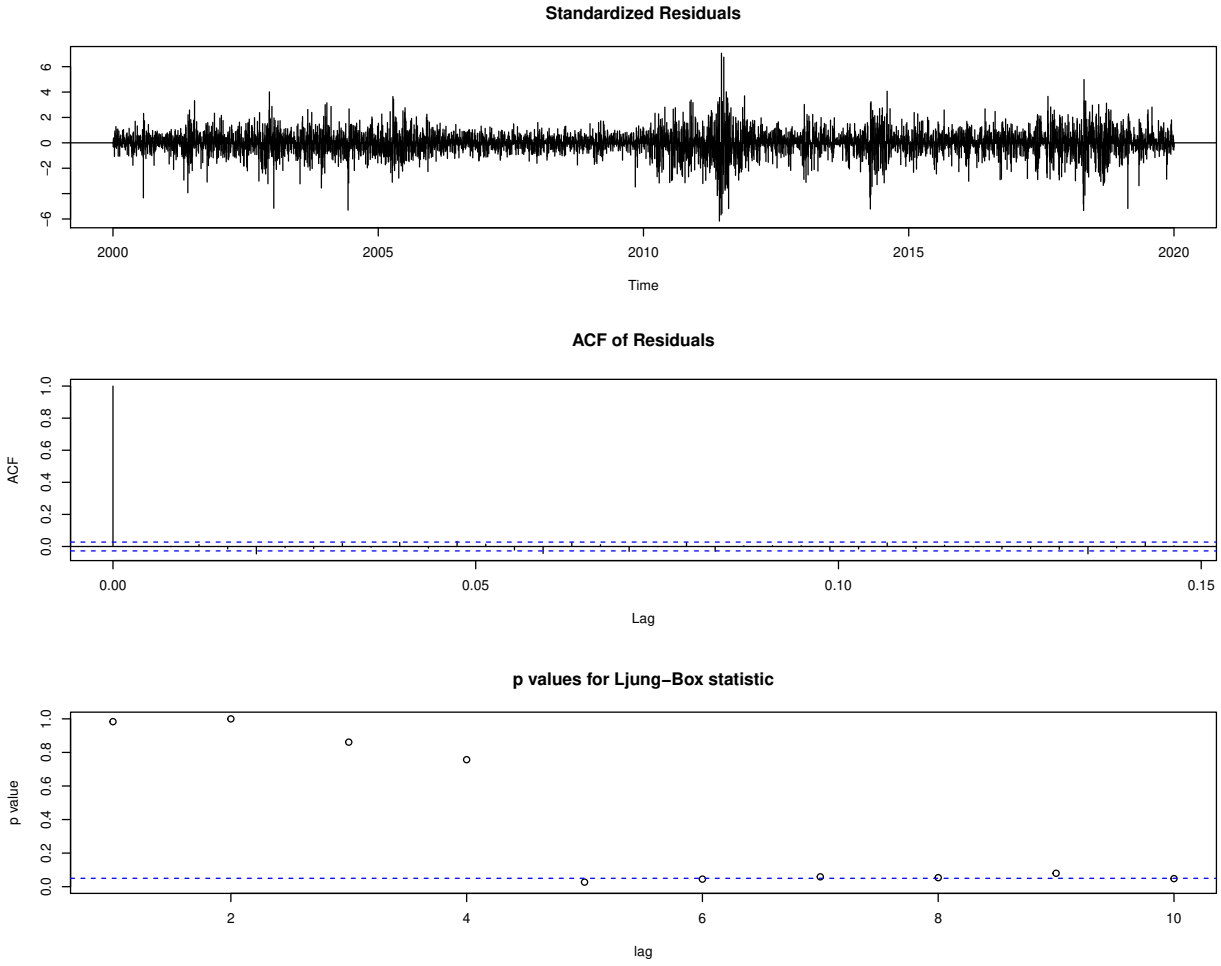
```
hist(weekly.arima.model$residuals)
```



As we can see both data sets do have a lot of residuals that hang around zero but the weekly data set has more outliers than the daily data set. Both do have a mostly normal curve, allowing us to believe that we are good to continue our study. Finally, we will do our last residual plot using the tsdiag function which will give us the Standardized Residuals, ACF of Residuals and the p values for Ljung-Box statistic plots.

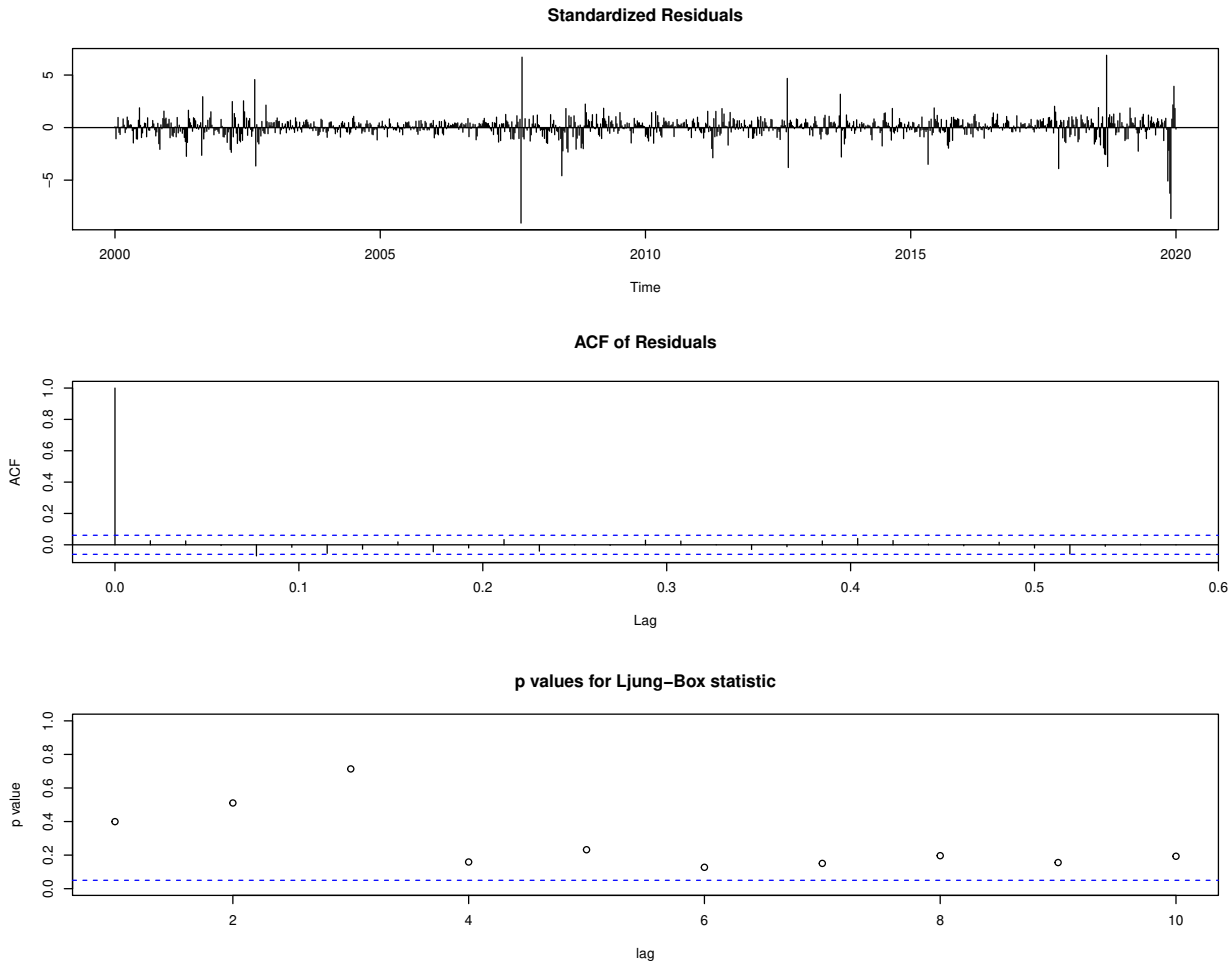
TSDIAG of the Daily data set

```
tsdiag(daily.arima.model)
```



TSDIAG of the Weekly data set

```
tsdiag(weekly.arima.model)
```



We now focus our analysis of the model to the Ljung-Box p-values. Our null hypothesis is as follows: 1. H0: The data set points are independently distributed.

With this as our null hypothesis, a significant p value greater than 0.5 does not reject the fact that the data points are not correlated. (Data points not correlated because we are using Dow Jones Index data). We can now analyze the lag at which the p value is the smallest. For our daily data set that appears to be Lag = 10. And for our Weekly data set that appears to be Lag = 6

```
Box.test(daily.arima.model$residuals, lag= 10, type="Ljung-Box")
```

```
##
##  Box-Ljung test
##
## data:  daily.arima.model$residuals
## X-squared = 18.441, df = 10, p-value = 0.04797
```

```
Box.test(weekly.arima.model$residuals, lag= 6, type="Ljung-Box")
```

```
##
##  Box-Ljung test
##
## data:  weekly.arima.model$residuals
## X-squared = 9.9348, df = 6, p-value = 0.1274
```

We observe both p values are rather large, so this does not bode well for our model. We continue anyway because we would like to see this models prediciton.

```
Box.test(daily.arima.model$residuals, type="Ljung-Box")
```

```
##
##  Box-Ljung test
##
## data:  daily.arima.model$residuals
## X-squared = 0.00043739, df = 1, p-value = 0.9833
```

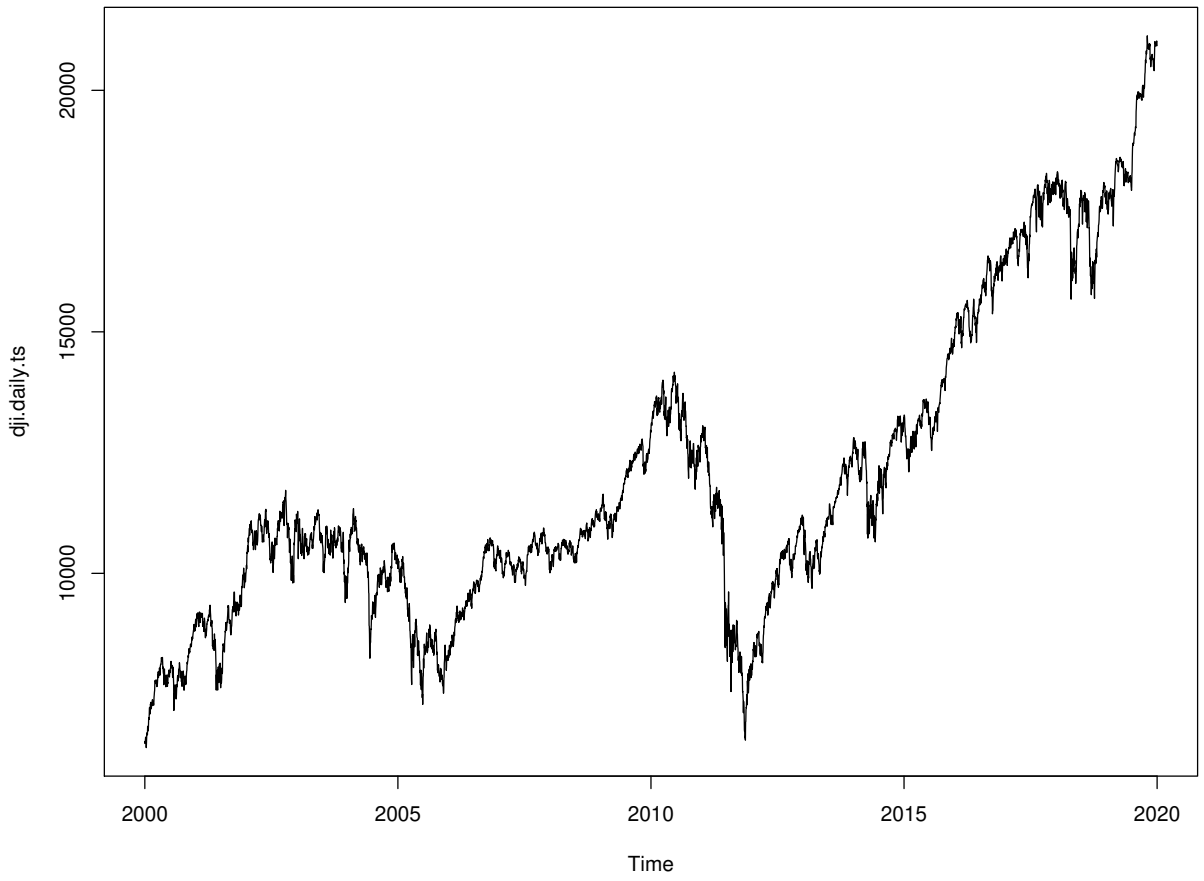
```
Box.test(weekly.arima.model$residuals, type="Ljung-Box")
```

```
##
##  Box-Ljung test
##
## data:  weekly.arima.model$residuals
## X-squared = 0.70987, df = 1, p-value = 0.3995
```

In our generalized box test we can see our null hypothesis is still not rejected, so we continue our study with confidence. Displayed below is a graph of our original time series over 5 years for both the daily and weekly data.

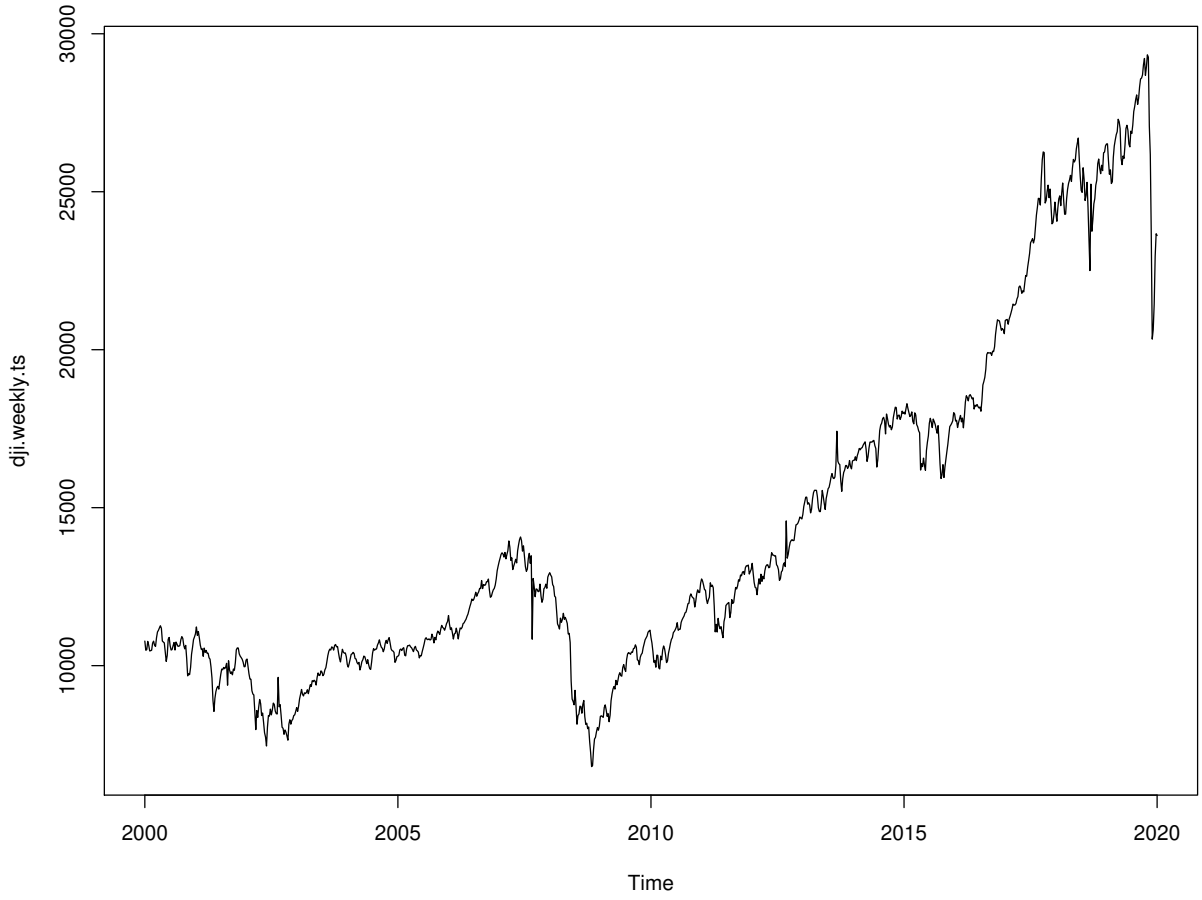
Daily Time Series

```
plot(dji.daily.ts)
```



Weekly Time Series

```
plot(dji.weekly.ts)
```



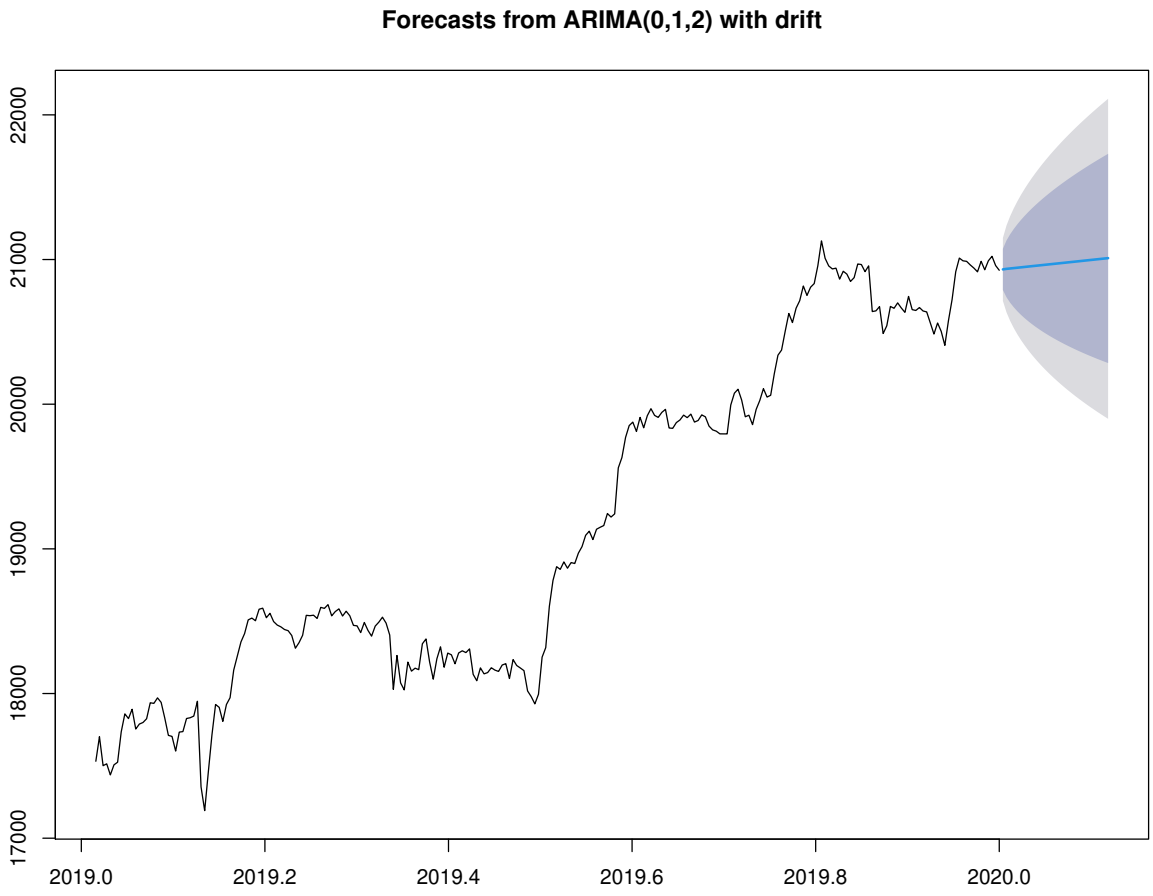
Auto Arima Results

Below is the function that gets the forecasts for both Daily and Weekly data sets. We are predicting 30 days in advanced for the daily data set. We are predicting 4 weeks in advanced for the weekly data set. As seen below h represents the number of prediction we are making.

```
daily.forecast <- forecast(daily.arima.model, h=30)
weekly.forecast <- forecast(weekly.arima.model, h=4)
```

Daily forecast 30 days out

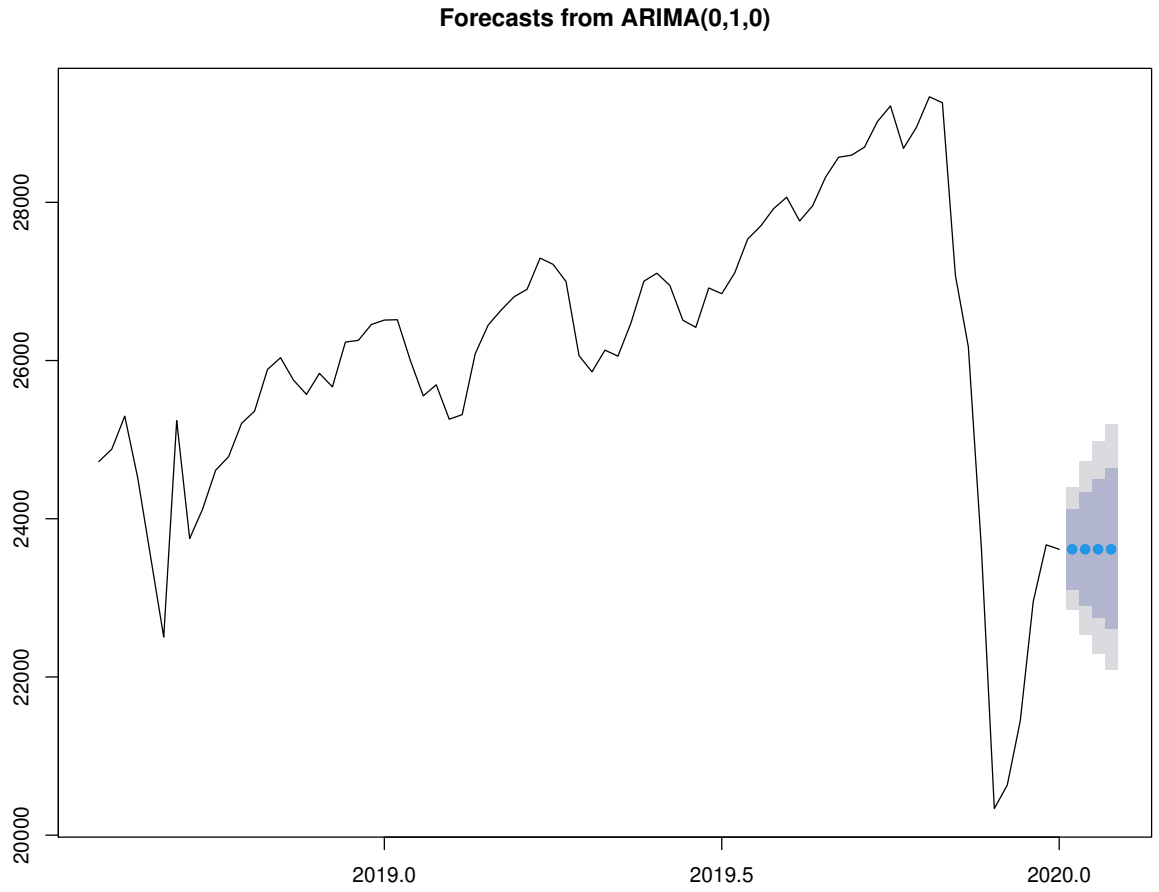
```
plot(daily.forecast, include=250)
```



Note: We are only displaying the prior 250 days because it visualizes better.

Weekly forecast 4 weeks out

```
plot(weekly.forecast, include=75)
```



Note: We are only displaying the prior 75 weeks because it visualizes better.

We can see in both our daily and weekly forecast that the blue line represents our mean predictions. The daily graph shows a slight increase over 30 days. While our weekly graph shows the index staying more or less the same. Now lets dig into some of the predictive statistics.

```
head(daily.forecast$mean)
```

```
## Time Series:
## Start = c(2020, 2)
## End = c(2020, 7)
## Frequency = 253
## [1] 20931.87 20935.56 20938.21 20940.85 20943.50 20946.14
```

```
head(weekly.forecast$mean)
```

```
## Time Series:
## Start = c(2020, 2)
## End = c(2020, 5)
## Frequency = 52
## [1] 23613.8 23613.8 23613.8 23613.8
```

The daily MEAN forecasts shows a slight increase from the start of its predictions! While the weekly shows an exactly same index over time, which certainly is not the case.

Upper Case Scenario:

```
head(daily.forecast$upper)
```

```
## Time Series:
## Start = c(2020, 2)
## End = c(2020, 7)
## Frequency = 253
##               80%      95%
## 2020.004 21073.92 21149.06
## 2020.008 21131.84 21235.62
## 2020.012 21174.42 21299.30
## 2020.016 21211.15 21354.02
## 2020.020 21244.03 21402.85
## 2020.024 21274.12 21447.42
```

```
head(weekly.forecast$upper)
```

```
## Time Series:
## Start = c(2020, 2)
## End = c(2020, 5)
## Frequency = 52
##               80%      95%
## 2020.019 24123.16 24394.82
## 2020.038 24335.61 24721.77
## 2020.058 24499.21 24974.01
## 2020.077 24637.53 25187.58
```

Lower Case Scenario:

```
head(daily.forecast$lower)
```

```
## Time Series:
## Start = c(2020, 2)
## End = c(2020, 7)
## Frequency = 253
##               80%      95%
```

```
## 2020.004 20789.66 20714.32
## 2020.008 20739.00 20634.82
## 2020.012 20701.57 20576.12
## 2020.016 20669.99 20526.38
## 2020.020 20642.27 20482.53
## 2020.024 20617.34 20442.94
```

```
head(weekly.forecast$lower)
```

```
## Time Series:
## Start = c(2020, 2)
## End = c(2020, 5)
## Frequency = 52
##               80%      95%
## 2020.019 23109.48 22844.55
## 2020.038 22902.06 22529.36
## 2020.058 22743.48 22288.88
## 2020.077 22610.19 22087.08
```

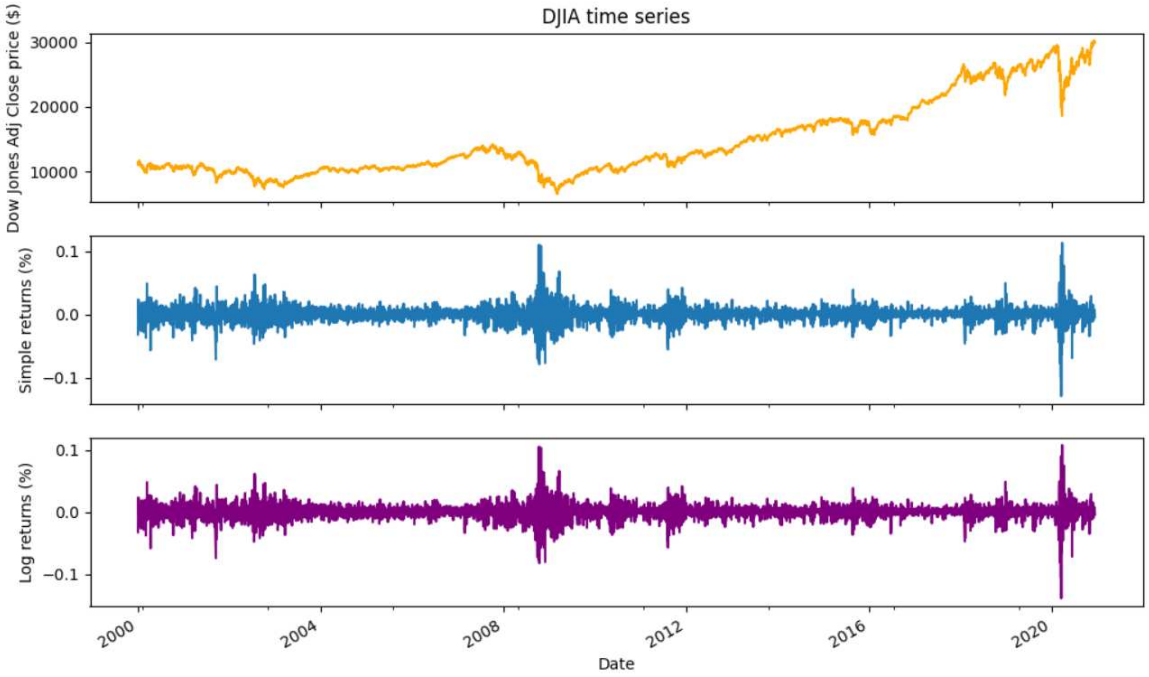
Code

```
import pandas_datareader.data as web
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import statsmodels.api as sm
import scipy.stats as scs
from fbprophet import Prophet
from fbprophet.plot import plot_plotly

import pandas as pd
import os
dji = web.DataReader('^DJI', data_source = 'yahoo', start = '2000-01-
01')
# print(dji.head())
# print('\n')
# print(dji.shape)
dji_series = dji['Adj Close']


#Calculate the simple and log returns using the adj close prices:
dji['simple_rtn'] = dji_series.pct_change()  # Calculating percent chan
ge
# print(dji['simple_rtn'].tail())
dji['log_rtn'] = np.log(dji_series/dji_series.shift(1))  # Calculating
log return

fig, ax = plt.subplots(3, 1, figsize=(14, 10), sharex=True)  # Plotting
 Closing prices, Simple return and Log return
# measuring variability of stock trading schemes
dji_series.plot(ax=ax[0],color = 'orange')
ax[0].set(title = 'DJIA time series', ylabel = 'Dow Jones Adj Close pri
ce ($)')
dji.simple_rtn.plot(ax=ax[1])
ax[1].set(ylabel = 'Simple returns (%)')
dji.log_rtn.plot(ax=ax[2],color = 'purple')
ax[2].set(xlabel = 'Date', ylabel = 'Log returns (%)')
# plt.show()

#Calculate the rolling mean and standard deviation:
df_rolling = dji[['simple_rtn']].rolling(window=21).agg(['mean', 'var']
)


df_rolling.columns = df_rolling.columns.droplevel()


# Histogram of log returns and Q-Q Plot
```

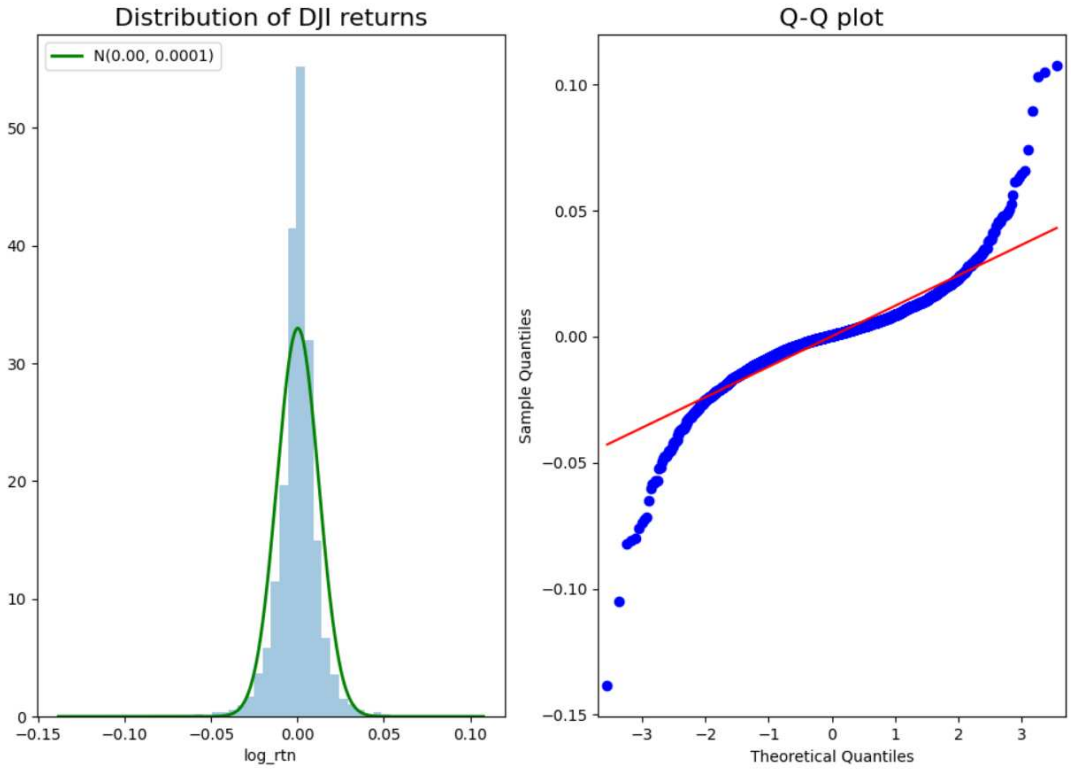
```
r_range = np.linspace(min(dji['log_rtn'].dropna()), max(dji['log_rtn'].
dropna()), num=1000)
mu = dji['log_rtn'].dropna().mean()
sigma = dji['log_rtn'].dropna().std()
norm_pdf = scs.norm.pdf(r_range, loc=mu, scale=sigma)#Plot the histogra
m and the Q-Q plot
fig, ax = plt.subplots(1, 2, figsize=(12, 8))# histogram
sns.distplot(dji['log_rtn'].dropna(), kde=False, norm_hist=True, ax=ax[
0])
ax[0].set_title('Distribution of DJI returns', fontsize=16)
ax[0].plot(r_range, norm_pdf, 'g', lw=2, label=f'N({mu:.2f}, {sigma**2:
.4f})')
ax[0].legend(loc='upper left');# Q-Q plot
qq = sm.qqplot(dji['log_rtn'].dropna().values, line='s', ax=ax[1])
ax[1].set_title('Q-Q plot', fontsize = 16)
plt.show()

N_LAGS = 50
SIGNIFICANCE_LEVEL = 0.05
fig, ax = plt.subplots(2, 1, figsize=(12, 8))
sm.graphics.tsa.plot_acf(dji['log_rtn'].dropna() ** 2, lags=N_LAGS,alph
a=SIGNIFICANCE_LEVEL, ax = ax[0])

ax[0].set(title='Autocorrelation Plots', ylabel='Squared Returns')

sm.graphics.tsa.plot_acf(dji['log_rtn'].dropna(), lags=N_LAGS,alpha=SIG
NIFICANCE_LEVEL, ax = ax[1])

ax[1].set(ylabel='Returns',xlabel='Lag')
plt.show()

prophet = dji_series.reset_index()
prophet.rename(columns={'Date':'ds','Adj Close':'y'},inplace=True)
prophet = prophet[['ds','y']]
train_percent = 0.95

# prepare train and test sets
train_size = int(prophet.shape[0]*train_percent)
train = prophet.iloc[:train_size]
test = prophet.iloc[train_size:]
model_prophet = Prophet(daily_seasonality = True) # build a prophet mod
el
model_prophet.fit(train) # fit the model

# prepare a future dataframe
future_dates = model_prophet.make_future_dataframe(periods=test.shape[0
])
forecast = model_prophet.predict(future_dates) # forecast values
```

```
model_prophet.plot(forecast) # plot for prediction
plt.title('Prediction on Test data')

forecast.set_index('ds', inplace=True, drop=True)
forecast.index = pd.to_datetime(forecast.index)
prophet.set_index('ds', inplace=True, drop=True)
prophet.index = pd.to_datetime(prophet.index)

# Final Plot
plt.plot(forecast['yhat'],c='r',label='Forecast')
plt.plot(forecast.yhat_lower.iloc[train_size:], linestyle='--
',c='b',alpha=0.3, label='Confidence Interval')
plt.plot(forecast.yhat_upper.iloc[train_size:], linestyle='--
',c='b',alpha=0.3)
plt.plot(prophet['y'],c='g',label='True Data')
plt.legend()
plt.title('Prophet Model Forecast vs Actutal Data')
```

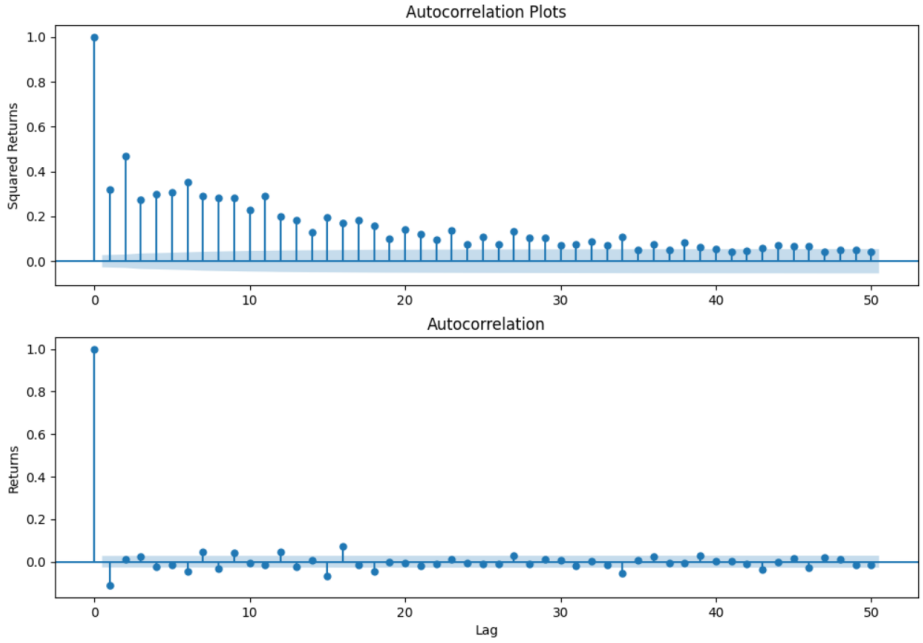
Results



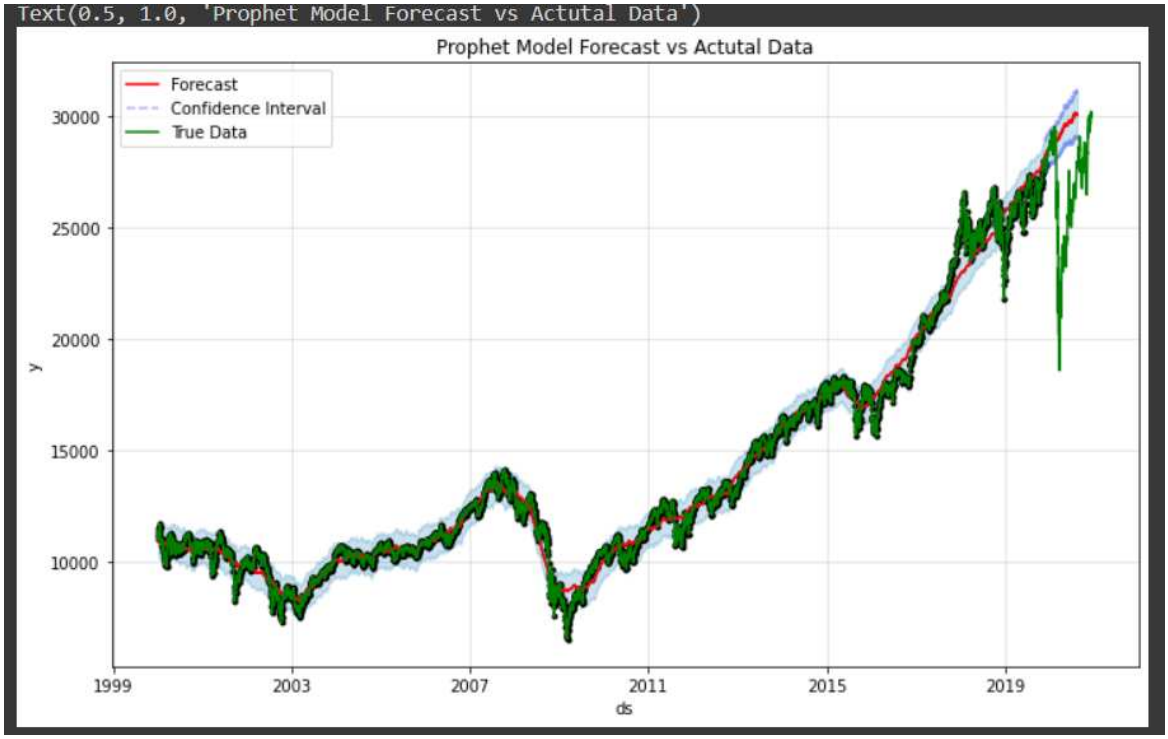
This plot is a combination of 3 values, namely Dow Jones Adj Close price ($), Simple returns (%), Log returns (%).



Here we have showcase Distribution of dJI returns histogram and a Q-Q plot between Sample Quantiles and Theoretical Quantiles.

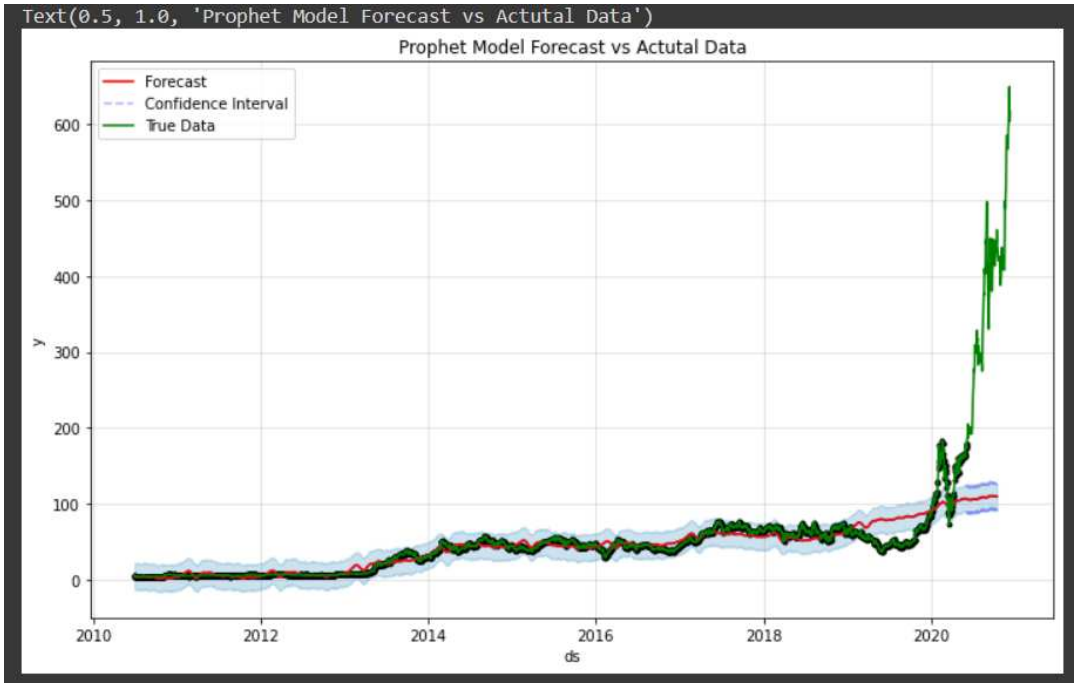


This is a Autocorrelation plot having Squared Returns and Normal Returns with Lags = 50.



This is our Forecast vs actual data using the prophet model for prediction of time series data. Here we can see that the model is not accurately able to discover a crash, but we can see an upward trend in the end of the which is corelating with the actual data.

Now Another Example of a more Volatile Stock, which is Tesla Stock



Here as we can see in Tesla Stock the Actual Trend and Predicted Trend varies drastically because we have not included the market trend such as news, economic scenario, insider updates etc. Hence, we get deviating results in the end. The momentum is still upwards for the stock but differs with actual data.

Here we can also say that the more volatile the market or stock, it becomes difficult to predict the trend.

GARCH

The Generalized Autoregressive Conditional Heteroskedasticity model have a fundation on making "Volatility clustering. This clustering of volatility is based on there are periods with relative calm movements and periods of high volatility. This behavior is very typical in the financial stock market data as we said and GARCH model is a very good approach to minimize the volatility effect.It is a model of order p,q, also known as GARCH(p,q).

GARCH is used extensively within the financial industry as many asset prices are conditional heteroskedastic.

Volatility

The main motivation for studying conditional heteroskedasticity in finance is that of volatility of asset returns. Volatility is an incredibly important concept in finance because it is highly synonymous with risk.

Volatility has a wide range of applications in finance:

- Options Pricing - The Black-Scholes model for options prices is dependent upon the volatility of the underlying instrument
- Risk Management - Volatility plays a role in calculating the VaR(Value at Risk) of a portfolio, the Sharpe Ratio for a trading strategy and in determination of leverage
- Tradeable Securities - Volatility can now be traded directly by the introduction of the CBOE Volatility Index (VIX), and subsequent futures contracts and ETFs

Hence, if we can effectively forecast volatility then we will be able to price options more accurately, create more sophisticated risk management tools for our algorithmic trading portfolios and even come up with new strategies that trade volatility directly.

Heteroskedasticity

In statistics, heteroskedasticity (or heteroscedasticity) happens when the standard deviations of a predicted variable, monitored over different values of an independent variable or as related to prior time periods, are non-constant. With heteroskedasticity, the tell-tale sign upon visual inspection of the residual errors is that they will tend to fan out over time, as depicted in the image below.

Heteroskedasticity often arises in two forms: conditional and unconditional. Conditional heteroskedasticity identifies nonconstant volatility related to prior period's (e.g., daily) volatility. Unconditional heteroskedasticity refers to general structural changes in volatility that are not related to prior period volatility. Unconditional heteroskedasticity is used when future periods of high and low volatility can be identified.

Heteroskedasticity refers to the error variance, or dependence of scattering, within a minimum of one independent variable within a particular sample. These variations can be used to calculate the margin of error between data sets, such as expected results and actual results, as it provides a measure of the deviation of data points from the mean value.

In finance, conditional heteroskedasticity is often seen in the prices of stocks and bonds.A common application of conditional heteroskedasticity is to stock markets, where the volatility today is strongly related to volatility yesterday. This model explains periods of persistent high volatility and low volatility.

Autoregressive Conditional Heteroskedastic Model

Autoregressive Conditional Heteroskedastic Model of Order Unity A time series $\{\epsilon_t\}$ is given at each instance by:

$$\epsilon_t = \sigma_t w_t$$

Where $\{w_t\}$ is discrete white noise, with zero mean and unit variance, and σ_t^2 is given by:

$$\sigma_t^2 = \alpha_0 + \alpha_1 \epsilon_{t-1}^2$$

Where α_0 and α_1 are parameters of the model.

We say that $\{\epsilon_t\}$ is an autoregressive conditional heteroskedastic model of order unity, denoted by ARCH(1). Substituting for σ_t^2, we receive:

$$\epsilon_t = w_t \sqrt{\alpha_0 + \alpha_1 \epsilon_{t-1}^2}$$

It is straightforward to extend ARCH to higher order lags. An ARCH(p) process is given by:

$$\epsilon_t = w_t \sqrt{\alpha_0 + \sum_{i=1}^{p} \alpha_p \epsilon_{t-i}^2}$$

Generalised Autoregressive Conditional Heteroskedastic Models

Generalised Autoregressive Conditional Heteroskedastic Model of Order p, q

A time series $\{\epsilon_t\}$ is given at each instance by:

$$\epsilon_t = \sigma_t w_t$$

Where $\{w_t\}$ is discrete white noise, with zero mean and unit variance, and σ_t^2 is given by:

$$\sigma_t^2 = \alpha_0 + \sum_{i=1}^{q} \alpha_i \epsilon_{t-i}^2 + \sum_{j=1}^{p} \beta_j \sigma_{t-j}^2$$

Where α_i and β_j are parameters of the model.

We say that $\{\epsilon_t\}$ is a generalised autoregressive conditional heteroskedastic model of order p,q, denoted by GARCH(p,q).

Hence this definition is similar to that of ARCH(p), with the exception that we are adding moving average terms, that is the value of σ^2 at t , σ_t^2, is dependent upon previous σ_{t-j}^2 values.

Thus GARCH is the "ARMA equivalent" of ARCH, which only has an autoregressive component.

Below is the Time Series Analysis of Dow Jones Industrial Averages from the 1/1/2000 to 12/10/2020.

We have fetched the DJI data from the quantmod library provided by R using the function getSymbols(). The data comprises of OHLC(Open, High, Low, Close) index value, trade Volume and Adjusted close values.

```
#Fetching Data
suppressMessages(getSymbols("^DJI", from = "2000-01-01", to = "2020-12-10"))
```

```
## [1] "^DJI"
```

```
head(DJI)
```

```
##            DJI.Open DJI.High  DJI.Low DJI.Close DJI.Volume DJI.Adjusted
## 2000-01-03 11501.85 11522.01 11305.69  11357.51  169750000     11357.51
## 2000-01-04 11349.75 11350.06 10986.45  10997.93  178420000     10997.93
## 2000-01-05 10989.37 11215.10 10938.67  11122.65  203190000     11122.65
## 2000-01-06 11113.37 11313.45 11098.45  11253.26  176550000     11253.26
## 2000-01-07 11247.06 11528.14 11239.92  11522.56  184900000     11522.56
## 2000-01-10 11532.48 11638.28 11532.48  11572.20  168180000     11572.20
```

```
tail(DJI)
```

```
##            DJI.Open DJI.High  DJI.Low DJI.Close DJI.Volume DJI.Adjusted
## 2020-12-02 29695.09 29902.51 29599.29  29883.79  385280000     29883.79
## 2020-12-03 29920.83 30110.88 29877.27  29969.52  405680000     29969.52
## 2020-12-04 29989.56 30218.26 29989.56  30218.26  356590000     30218.26
## 2020-12-07 30233.03 30233.03 29967.22  30069.79  365810000     30069.79
## 2020-12-08 29997.95 30246.22 29972.07  30173.88  311190000     30173.88
## 2020-12-09 30229.81 30319.70 29951.85  30068.81  380520000     30068.81
```

With the help of chartSeries function, we can show all the information provided in the data in a single frame.

```
#chartseries
chartSeries(DJI, type = "bars", theme="white")
```

Simple and log returns:

Simple returns are defined as:

$$R_t := \frac{P_t}{P_{t-1}} - 1$$

Log returns are defined as:

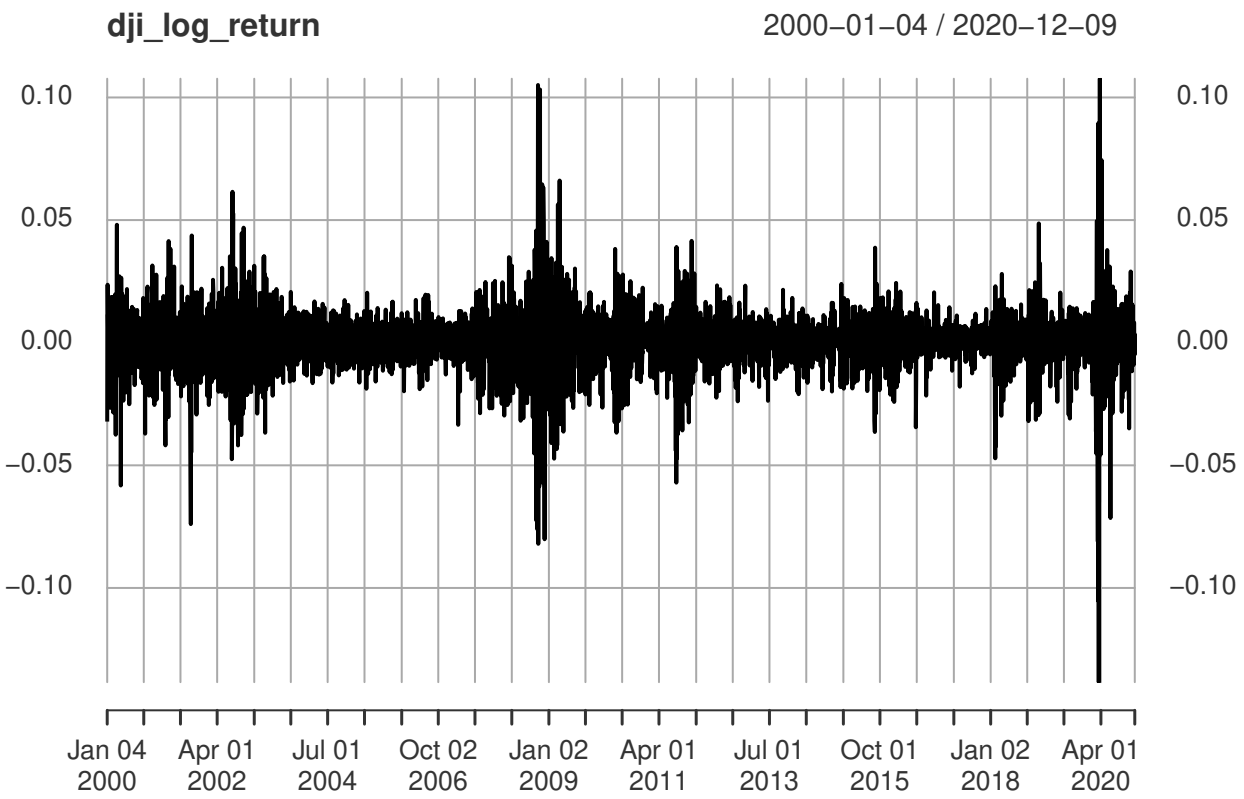
$$r_t := ln\frac{P_t}{P_{t-1}} = ln(1 + R_t)$$

Now, we calculate the log returns of Adjusted Close Values and plot a graph for it.

```
dji_adjusted_close <- DJI[,"DJI.Adjusted"] #Adjusted close value.
dji_log_return <- CalculateReturns(dji_adjusted_close, method = "log") #compute log returns
dji_log_return <- na.omit(dji_log_return) #omit the NA values
head(dji_log_return)
```

```
##            DJI.Adjusted
## 2000-01-04 -0.032172134
## 2000-01-05  0.011276560
## 2000-01-06  0.011674239
## 2000-01-07  0.023648972
## 2000-01-10  0.004298871
## 2000-01-11 -0.005295630
```

```
plot(dji_log_return)
```



Sharp increases and decreases in volatility can be eye-balled.

Now, we will apply a function to convert the time series data to a dataframe with the columns for year and value.

```
dji_return_df <- data.frame(year = factor(year(index(dji_log_return))), value = coredata(dji_log_return
colnames(dji_return_df) <- c( "year", "value")
head(dji_return_df)
```

```
##   year        value
## 1 2000 -0.032172134
## 2 2000  0.011276560
## 3 2000  0.011674239
## 4 2000  0.023648972
## 5 2000  0.004298871
## 6 2000 -0.005295630
```

The function below shows the basic statistics for the data stored as data frame columns. We use the basicStats function to get the row names for basic statictics that we will be showing.

```
basic_stat<- rownames(basicStats(rnorm(10,0,1))) # gathering the basic stats dataframe output row names
result <- with(dji_return_df, tapply(value, year, basicStats))
df_stat <- do.call(cbind, result)
rownames(df_stat) <- basic_stat
dji_stat <- as.data.frame(df_stat)
dji_stat
```

```
##                    2000       2001       2002       2003       2004       2005
## nobs         251.000000 248.000000 252.000000 252.000000 252.000000 252.000000
## NAs            0.000000   0.000000   0.000000   0.000000   0.000000   0.000000
## Minimum       -0.058217  -0.073962  -0.047515  -0.036725  -0.016501  -0.018781
## Maximum        0.048096   0.043719   0.061547   0.035276   0.017379   0.020389
## 1. Quartile   -0.007474  -0.007679  -0.011224  -0.005447  -0.004277  -0.004781
## 3. Quartile    0.007556   0.007216   0.008103   0.006791   0.003984   0.004296
## Mean          -0.000205  -0.000297  -0.000728   0.000896   0.000123  -0.000024
## Median         0.000494  -0.000016  -0.002120   0.001293   0.000198   0.000404
## Sum           -0.051446  -0.073694  -0.183481   0.225718   0.030995  -0.006094
## SE Mean        0.000826   0.000858   0.001010   0.000657   0.000430   0.000409
## LCL Mean      -0.001833  -0.001986  -0.002717  -0.000398  -0.000724  -0.000830
## UCL Mean       0.001423   0.001392   0.001261   0.002190   0.000970   0.000781
## Variance       0.000171   0.000182   0.000257   0.000109   0.000047   0.000042
## Stdev          0.013093   0.013505   0.016035   0.010431   0.006831   0.006492
## Skewness      -0.288213  -0.564552   0.487613   0.111023   0.009722  -0.004091
## Kurtosis       1.656967   3.903018   1.152989   1.054773  -0.159511  -0.007669
##                    2006       2007       2008       2009       2010       2011
## nobs         251.000000 251.000000 253.000000 252.000000 252.000000 252.000000
## NAs            0.000000   0.000000   0.000000   0.000000   0.000000   0.000000
## Minimum       -0.019800  -0.033488  -0.082005  -0.047286  -0.036700  -0.057061
## Maximum        0.019603   0.025223   0.105083   0.066116   0.038247   0.041533
## 1. Quartile   -0.002882  -0.003795  -0.012993  -0.006897  -0.003853  -0.006193
## 3. Quartile    0.004282   0.005179   0.007843   0.008248   0.004457   0.006531
## Mean           0.000601   0.000248  -0.001633   0.000684   0.000415   0.000214
## Median         0.000561   0.001094  -0.000890   0.001082   0.000681   0.000941
```

```
## Sum           0.150898   0.062339  -0.413050   0.172434   0.104565   0.053810
## SE Mean       0.000392   0.000579   0.001497   0.000960   0.000641   0.000837
## LCL Mean     -0.000171  -0.000893  -0.004580  -0.001207  -0.000848  -0.001434
## UCL Mean      0.001374   0.001389   0.001315   0.002575   0.001678   0.001861
## Variance      0.000039   0.000084   0.000567   0.000232   0.000104   0.000176
## Stdev         0.006215   0.009178   0.023808   0.015242   0.010182   0.013283
## Skewness     -0.108524  -0.615912   0.224042   0.070840  -0.174816  -0.526083
## Kurtosis      1.156968   1.543836   3.670796   2.074240   2.055407   2.453822
##                   2012       2013       2014       2015       2016       2017
## nobs        250.000000 252.000000 252.000000 252.000000 252.000000 251.000000
## NAs           0.000000   0.000000   0.000000   0.000000   0.000000   0.000000
## Minimum      -0.023910  -0.023695  -0.020988  -0.036402  -0.034473  -0.017930
## Maximum       0.023376   0.023263   0.023982   0.038755   0.024384   0.014468
## 1. Quartile  -0.003896  -0.002812  -0.002621  -0.005283  -0.002845  -0.001404
## 3. Quartile   0.004924   0.004750   0.004230   0.005801   0.004311   0.003054
## Mean          0.000280   0.000933   0.000288  -0.000090   0.000500   0.000892
## Median       -0.000122   0.001158   0.000728  -0.000211   0.000738   0.000655
## Sum           0.070054   0.235068   0.072498  -0.022586   0.125884   0.223790
## SE Mean       0.000470   0.000403   0.000432   0.000613   0.000501   0.000263
## LCL Mean     -0.000645   0.000139  -0.000564  -0.001298  -0.000487   0.000373
## UCL Mean      0.001206   0.001727   0.001139   0.001118   0.001486   0.001410
## Variance      0.000055   0.000041   0.000047   0.000095   0.000063   0.000017
## Stdev         0.007429   0.006399   0.006861   0.009738   0.007951   0.004172
## Skewness      0.027235  -0.199407  -0.332766  -0.127788  -0.449311  -0.189808
## Kurtosis      0.842890   1.275821   1.073234   1.394268   2.079671   2.244076
##                   2018       2019       2020
## nobs        251.000000 252.000000 238.000000
## NAs           0.000000   0.000000   0.000000
## Minimum      -0.047143  -0.030934  -0.138418
## Maximum       0.048643   0.032394   0.107643
## 1. Quartile  -0.005017  -0.002698  -0.007328
## 3. Quartile   0.005895   0.005344   0.010167
## Mean         -0.000231   0.000800   0.000219
## Median        0.000695   0.001002   0.001469
## Sum          -0.057950   0.201621   0.052237
## SE Mean       0.000714   0.000494   0.001557
## LCL Mean     -0.001637  -0.000174  -0.002849
## UCL Mean      0.001175   0.001774   0.003288
## Variance      0.000128   0.000062   0.000577
## Stdev         0.011313   0.007847   0.024027
## Skewness     -0.522618  -0.660265  -0.794348
## Kurtosis      2.802996   3.211825   8.329006
```

We can see the Mean, Median, Mode, Minimum, Maximum, Sum, Skewness, Kurtosis, Variance , Standard Deviation and Quartile values for every year for for the log return data of Adjusted close values.

filter_dji_stats function helps to filter the statistics value we want, from the basic statistics dataframe.

```r
#Basic statistics dataframe row threshold filtering to return the associated column names.
filter_dji_stats <- function(data_basicstats, metricname, threshold) {
    r <- which(rownames(data_basicstats) ==  metricname)
    colnames(data_basicstats[r, which(data_basicstats[r,] > threshold), drop = FALSE])
```

```
}
filter_dji_stats(dji_stat, "Mean", 0) #Years when Dow Jones daily log-returns have positive mean values
```

```
##  [1] "2003" "2004" "2006" "2007" "2009" "2010" "2011" "2012" "2013" "2014"
## [11] "2016" "2017" "2019" "2020"
```

```
dji_stat["Mean",order(dji_stat["Mean",,])] #All Dow Jones daily log-returns mean values in ascending or
```

```
##           2008      2002      2001      2018      2000   2015     2005     2004
## Mean -0.001633 -0.000728 -0.000297 -0.000231 -0.000205 -9e-05 -2.4e-05 0.000123
##          2011     2020     2007    2012     2014     2010  2016     2006
## Mean 0.000214 0.000219 0.000248 0.00028 0.000288 0.000415 5e-04 0.000601
##          2009  2019     2017     2003     2013
## Mean 0.000684 8e-04 0.000892 0.000896 0.000933
```

```
filter_dji_stats(dji_stat, "Median", 0) #Years when Dow Jones daily log-returns have positive median ar
```

```
##  [1] "2000" "2003" "2004" "2005" "2006" "2007" "2009" "2010" "2011" "2013"
## [11] "2014" "2016" "2017" "2018" "2019" "2020"
```

```
dji_stat["Median",order(dji_stat["Median",,])] #for median
```

```
##            2002     2008      2015      2012     2001     2004     2005
## Median -0.00212 -0.00089 -0.000211 -0.000122 -1.6e-05 0.000198 0.000404
##            2000     2006     2017     2010     2018     2014     2016     2011
## Median 0.000494 0.000561 0.000655 0.000681 0.000695 0.000728 0.000738 0.000941
##            2019     2009     2007     2013     2003     2020
## Median 0.001002 0.001082 0.001094 0.001158 0.001293 0.001469
```

```
filter_dji_stats(dji_stat, "Skewness", 0)#Years when Dow Jones daily log-returns have positive skewness
```

```
## [1] "2002" "2003" "2004" "2008" "2009" "2012"
```

```
dji_stat["Skewness",order(dji_stat["Skewness",,])]
```

```
##               2020      2019      2007      2001      2011      2018      2016
## Skewness -0.794348 -0.660265 -0.615912 -0.564552 -0.526083 -0.522618 -0.449311
##               2014      2000      2013      2017      2010      2015      2006
## Skewness -0.332766 -0.288213 -0.199407 -0.189808 -0.174816 -0.127788 -0.108524
##               2005     2004     2012    2009     2003     2008     2002
## Skewness -0.004091 0.009722 0.027235 0.07084 0.111023 0.224042 0.487613
```

```
filter_dji_stats(dji_stat, "Kurtosis", 0) #Years when Dow Jones daily log-returns have positive excess
```

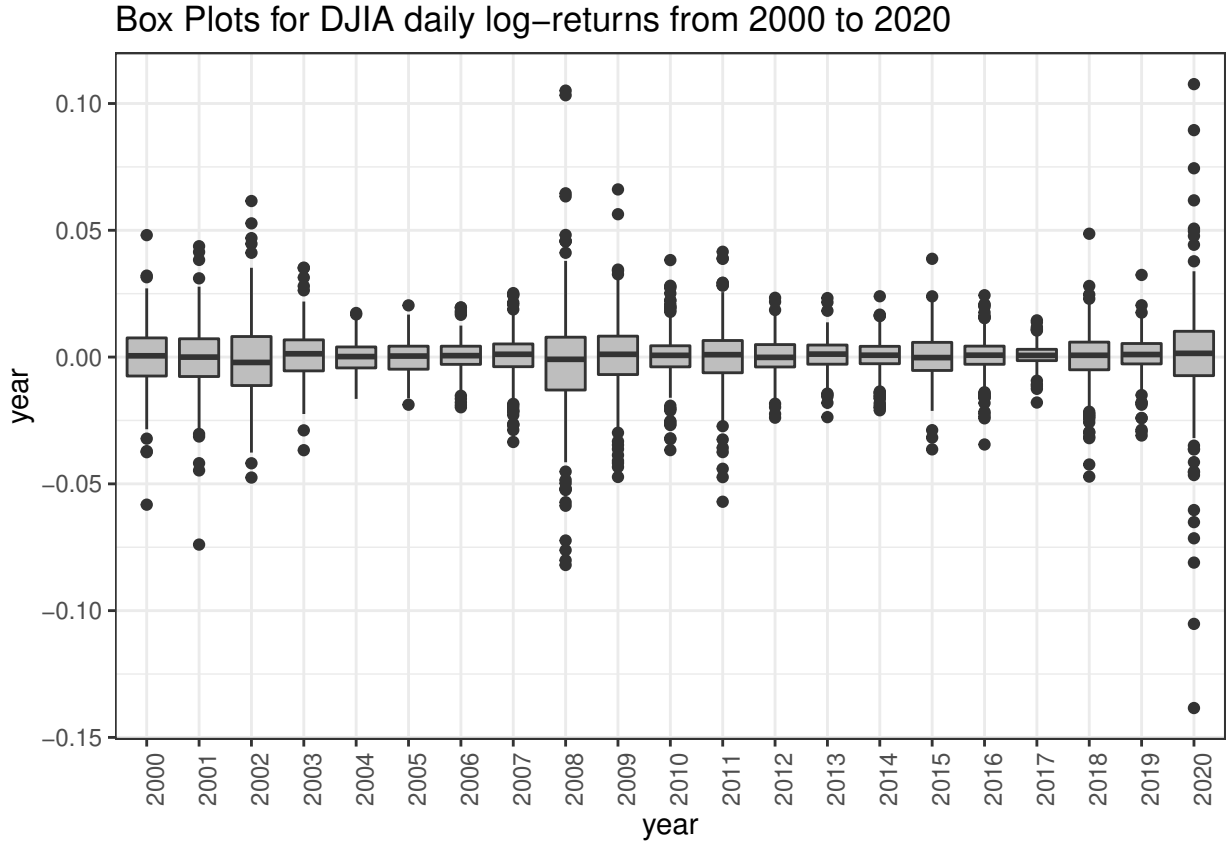
```
##  [1] "2000" "2001" "2002" "2003" "2006" "2007" "2008" "2009" "2010" "2011"
## [11] "2012" "2013" "2014" "2015" "2016" "2017" "2018" "2019" "2020"
```

```
dji_stat["Kurtosis",order(dji_stat["Kurtosis",,])]
```

```
##               2004      2005     2012     2003     2014     2002     2006
## Kurtosis -0.159511 -0.007669 0.84289 1.054773 1.073234 1.152989 1.156968
##              2013     2015     2007     2000     2010    2009     2016     2017
## Kurtosis 1.275821 1.394268 1.543836 1.656967 2.055407 2.07424 2.079671 2.244076
##              2011     2018     2019     2008     2001     2020
## Kurtosis 2.453822 2.802996 3.211825 3.670796 3.903018 8.329006
```

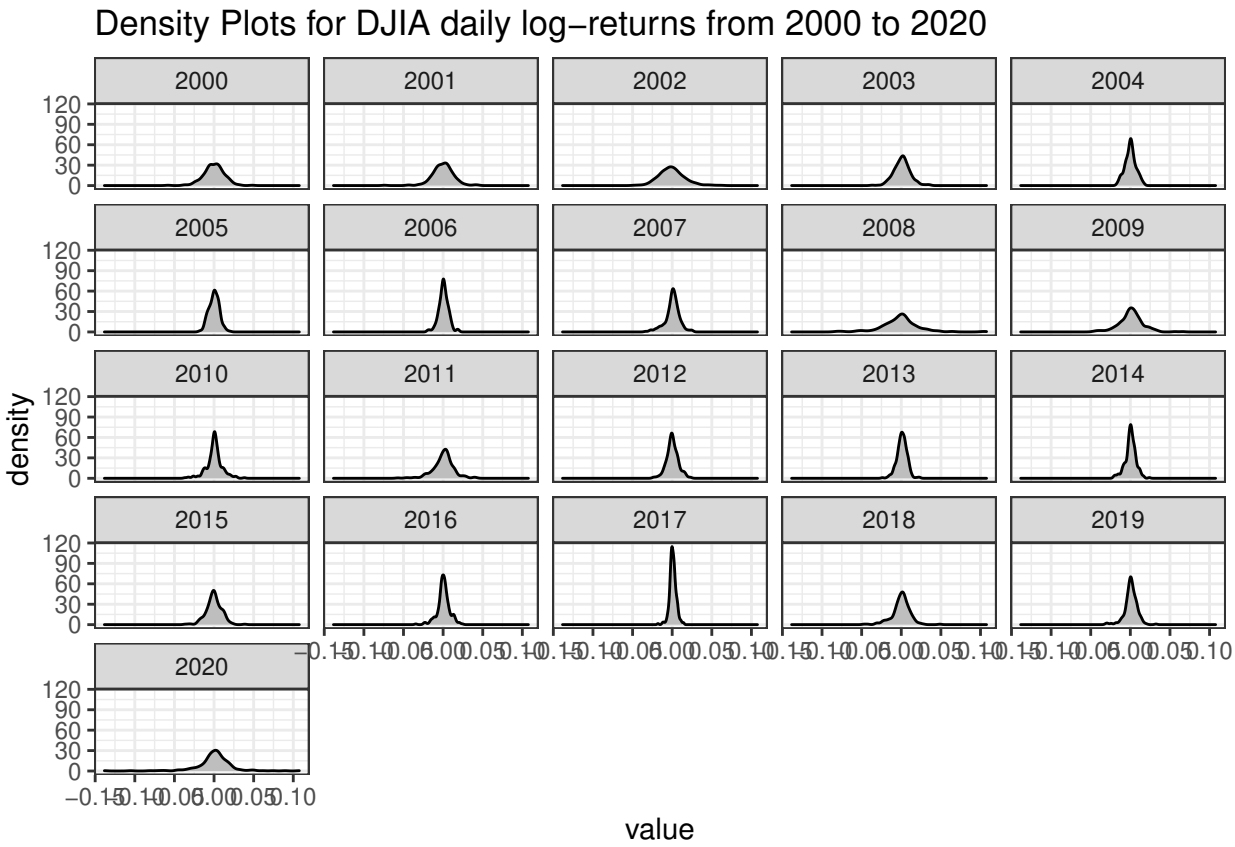
Now, we plot the boxplot of daily log-returns based on years.

```
boxplot <- ggplot(data = dji_return_df, aes(x = year, y = value)) + theme_bw() + theme(legend.position =

boxplot
```



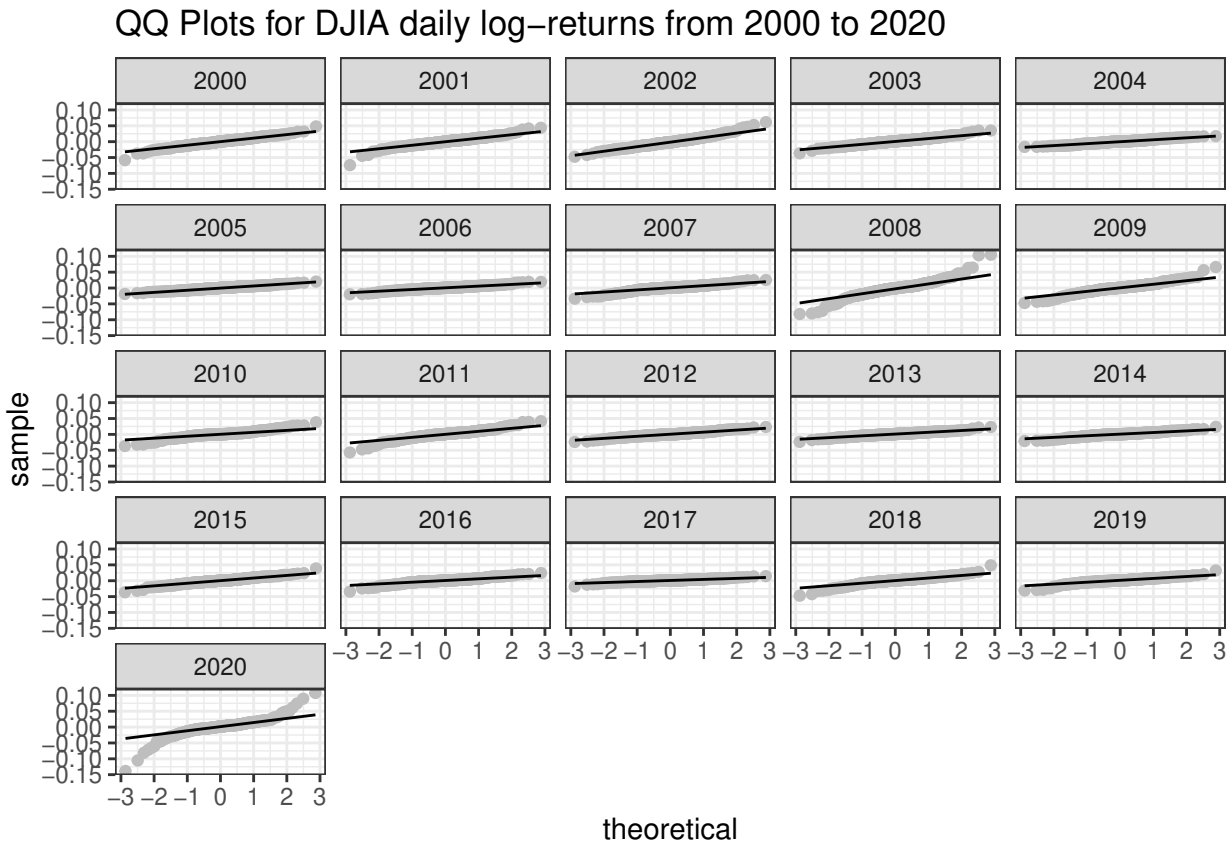
Now, we plot the Density plot of daily log-returns based on years.

```
densityplot <- ggplot(data = dji_return_df, aes(x = value)) + geom_density(fill = "gray") + facet_wrap(
densityplot
```



Now, we plot the Quantile Quantile plot (QQ plot) of daily log-returns based on years.

```
qqplot <- ggplot(data = dji_return_df, aes(sample = value)) + stat_qq(colour = "gray") + stat_qq_line()

qqplot
```



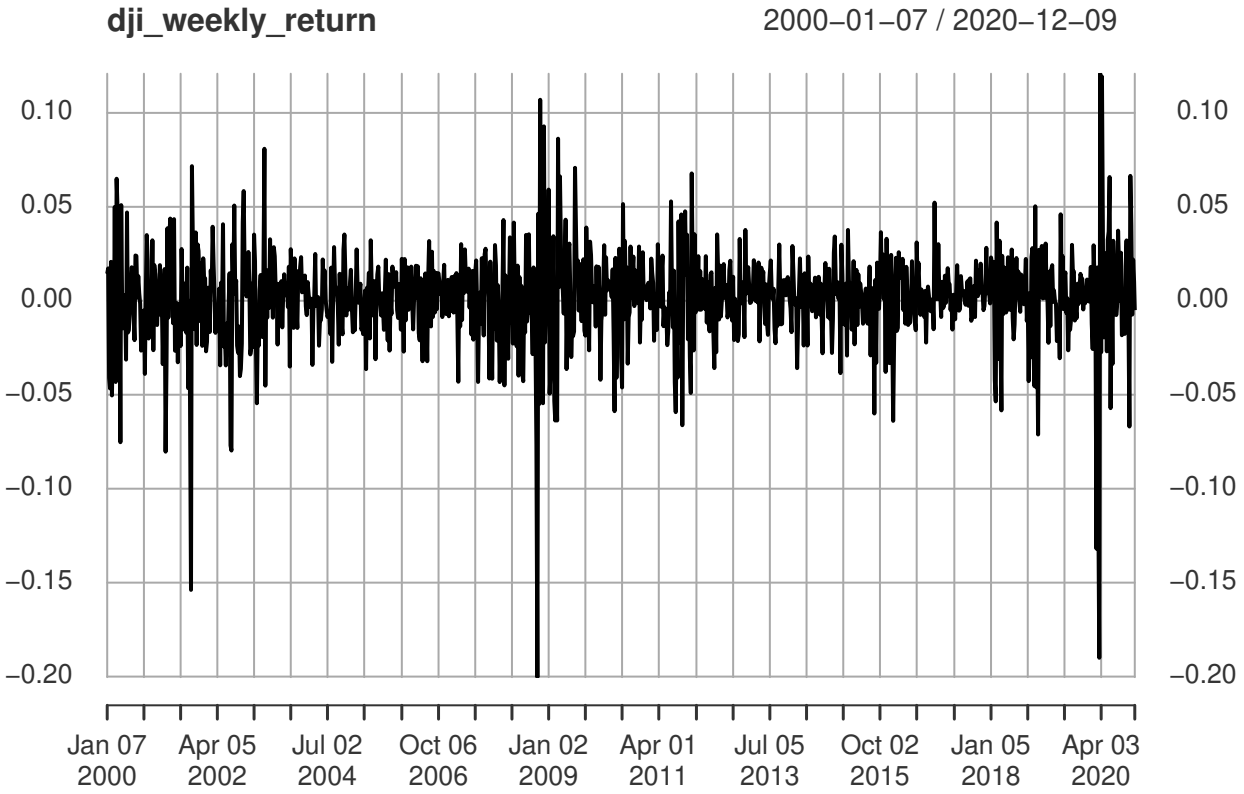
Weekly Log Returns Analysis:

The weekly log returns can be computed starting from the daily ones. Let us suppose to analyse the trading week on days {t-4, t-3, t-2, t-1, t} and to know closing price at day t-5 (last day of the previous week). We define the weekly log-return as:

$$r_t^w := ln\frac{P_t}{P_{t-5}}$$

We convert the data available into weekly data and compute the basic statistics functions to

```
dji_weekly_return <- apply.weekly(dji_log_return, sum)
plot(dji_weekly_return)
```




```
dji_weekly_return_df <- data.frame(year = factor(year(index(dji_weekly_return))), value = coredata(dji_
colnames(dji_weekly_return_df) <- c( "year", "value")
dim(dji_weekly_return_df) #shows the total number of weeks in the data
```

```
## [1] 1093    2
```

```
head(dji_weekly_return_df)
```

```
##   year       value
## 1 2000  0.01442764
## 2 2000  0.01724424
## 3 2000 -0.04103094
## 4 2000 -0.04665023
## 5 2000  0.02072904
## 6 2000 -0.05037201
```

```
basic_stat<- rownames(basicStats(rnorm(10,0,1))) # gathering the basic stats dataframe output row names
result <- with(dji_weekly_return_df, tapply(value, year, basicStats))
df_stat <- do.call(cbind, result)
rownames(df_stat) <- basic_stat
dji_stats <- as.data.frame(df_stat)
dji_stats
```

```
##                 2000      2001      2002      2003      2004      2005
## nobs        52.000000 52.000000 52.000000 52.000000 53.000000 52.000000
## NAs          0.000000  0.000000  0.000000  0.000000  0.000000  0.000000
## Minimum     -0.075275 -0.153845 -0.079697 -0.054534 -0.034125 -0.036389
## Maximum      0.064962  0.071650  0.058400  0.080898  0.035279  0.032197
## 1. Quartile -0.018689 -0.015136 -0.023628 -0.008957 -0.007217 -0.008149
## 3. Quartile  0.017394  0.020298  0.010421  0.015895  0.008360  0.008991
## Mean        -0.000989 -0.001197 -0.003836  0.004189  0.000820 -0.000117
## Median       0.000016  0.000834  0.000574  0.006556  0.000951  0.000839
## Sum         -0.051446 -0.062242 -0.199480  0.217825  0.043436 -0.006094
## SE Mean      0.004003  0.004773  0.003835  0.003003  0.001983  0.002007
## LCL Mean    -0.009026 -0.010779 -0.011536 -0.001840 -0.003159 -0.004145
## UCL Mean     0.007048  0.008385  0.003864  0.010218  0.004798  0.003911
## Variance     0.000833  0.001185  0.000765  0.000469  0.000208  0.000209
## Stdev        0.028868  0.034419  0.027658  0.021656  0.014434  0.014469
## Skewness    -0.003740 -1.621168 -0.255978  0.146644 -0.044612 -0.266755
## Kurtosis    -0.198923  5.759875  0.525494  2.275154  0.111503 -0.213954
##                 2006      2007      2008      2009      2010      2011
## nobs        52.000000 52.000000 52.000000 53.000000 52.000000 52.000000
## NAs          0.000000  0.000000  0.000000  0.000000  0.000000  0.000000
## Minimum     -0.032190 -0.043199 -0.200298 -0.063736 -0.058755 -0.066235
## Maximum      0.031814  0.030143  0.106977  0.086263  0.051463  0.067788
## 1. Quartile -0.005592 -0.009638 -0.031765 -0.015911 -0.007761 -0.015485
## 3. Quartile  0.014549  0.014808  0.012682  0.022115  0.016971  0.014309
## Mean         0.002902  0.001345 -0.008669  0.003823  0.002011  0.001035
## Median       0.002666  0.004244 -0.006811  0.004633  0.004529  0.001757
## Sum          0.150898  0.069928 -0.450811  0.202605  0.104565  0.053810
## SE Mean      0.002008  0.002612  0.006164  0.004454  0.003031  0.003836
## LCL Mean    -0.001129 -0.003899 -0.021043 -0.005115 -0.004074 -0.006666
## UCL Mean     0.006933  0.006589  0.003704  0.012760  0.008096  0.008736
## Variance     0.000210  0.000355  0.001975  0.001051  0.000478  0.000765
## Stdev        0.014479  0.018836  0.044446  0.032424  0.021856  0.027662
## Skewness    -0.465373 -0.683647 -0.985740  0.121331 -0.601407 -0.076579
## Kurtosis    -0.230910 -0.079408  5.446623 -0.033398  0.357708  0.052429
##                 2012      2013      2014      2015      2016      2017
## nobs        52.000000 52.000000 52.000000 53.000000 52.000000 52.000000
## NAs          0.000000  0.000000  0.000000  0.000000  0.000000  0.000000
## Minimum     -0.035829 -0.022556 -0.038482 -0.059991 -0.063897 -0.015317
## Maximum      0.035316  0.037702  0.034224  0.037693  0.052243  0.028192
## 1. Quartile -0.010096 -0.001738 -0.006378 -0.012141 -0.007746 -0.002251
## 3. Quartile  0.011887  0.011432  0.010244  0.009620  0.012791  0.009891
## Mean         0.001102  0.004651  0.001756 -0.000669  0.002421  0.004304
## Median       0.001166  0.006360  0.003961  0.000954  0.001947  0.004080
## Sum          0.057303  0.241874  0.091300 -0.035444  0.125884  0.223790
## SE Mean      0.002133  0.001828  0.002151  0.002609  0.002436  0.001232
## LCL Mean    -0.003181  0.000981 -0.002563 -0.005904 -0.002470  0.001830
## UCL Mean     0.005384  0.008322  0.006075  0.004567  0.007312  0.006778
## Variance     0.000237  0.000174  0.000241  0.000361  0.000309  0.000079
## Stdev        0.015382  0.013185  0.015514  0.018995  0.017568  0.008886
## Skewness    -0.027302 -0.035175 -0.534403 -0.494963 -0.467158  0.266281
## Kurtosis    -0.461228 -0.200282  0.282354  0.665460  2.908942 -0.124341
##                 2018      2019      2020
## nobs        52.000000 52.000000 50.000000
## NAs          0.000000  0.000000  0.000000
```

```
## Minimum     -0.071149 -0.030583 -0.189978
## Maximum      0.050288  0.046029  0.120840
## 1. Quartile -0.012090 -0.005102 -0.016046
## 3. Quartile  0.020065  0.014458  0.024624
## Mean        -0.001334  0.004169  0.000970
## Median       0.001541  0.004172  0.003627
## Sum         -0.069378  0.216784  0.048500
## SE Mean      0.003654  0.002096  0.007248
## LCL Mean    -0.008670 -0.000040 -0.013596
## UCL Mean     0.006002  0.008378  0.015536
## Variance     0.000694  0.000229  0.002627
## Stdev        0.026349  0.015118  0.051253
## Skewness    -0.631938  0.107595 -1.042777
## Kurtosis    -0.057607  0.148103  3.594221
```

```
filter_dji_stats(dji_stats, "Mean", 0)
```

```
##  [1] "2003" "2004" "2006" "2007" "2009" "2010" "2011" "2012" "2013" "2014"
## [11] "2016" "2017" "2019" "2020"
```

```
dji_stats["Mean",order(dji_stats["Mean",,])]
```

```
##           2008      2002      2018      2001      2000      2015      2005
## Mean -0.008669 -0.003836 -0.001334 -0.001197 -0.000989 -0.000669 -0.000117
##         2004    2020     2011     2012     2007     2014     2010     2016
## Mean 0.00082 0.00097 0.001035 0.001102 0.001345 0.001756 0.002011 0.002421
##          2006     2009     2019     2003     2017     2013
## Mean 0.002902 0.003823 0.004169 0.004189 0.004304 0.004651
```

```
filter_dji_stats(dji_stats, "Median", 0)
```

```
##  [1] "2000" "2001" "2002" "2003" "2004" "2005" "2006" "2007" "2009" "2010"
## [11] "2011" "2012" "2013" "2014" "2015" "2016" "2017" "2018" "2019" "2020"
```

```
dji_stats["Median",order(dji_stats["Median",,])]
```

```
##             2008    2000     2002     2001     2005     2004     2015     2012
## Median -0.006811 1.6e-05 0.000574 0.000834 0.000839 0.000951 0.000954 0.001166
##            2018     2011     2016     2006     2020     2014    2017     2019
## Median 0.001541 0.001757 0.001947 0.002666 0.003627 0.003961 0.00408 0.004172
##            2007     2010     2009    2013     2003
## Median 0.004244 0.004529 0.004633 0.00636 0.006556
```

```
filter_dji_stats(dji_stats, "Skewness", 0)
```

```
## [1] "2003" "2009" "2017" "2019"
```

```
dji_stats["Skewness",order(dji_stats["Skewness",,])]
```

```
##                2001      2020     2008      2007      2018      2010      2014
## Skewness -1.621168 -1.042777 -0.98574 -0.683647 -0.631938 -0.601407 -0.534403
##                2015      2016      2006      2005      2002      2011      2004
## Skewness -0.494963 -0.467158 -0.465373 -0.266755 -0.255978 -0.076579 -0.044612
##                2013      2012      2000     2019     2009     2003     2017
## Skewness -0.035175 -0.027302 -0.00374 0.107595 0.121331 0.146644 0.266281
```

```
filter_dji_stats(dji_stats, "Kurtosis", 0)
```

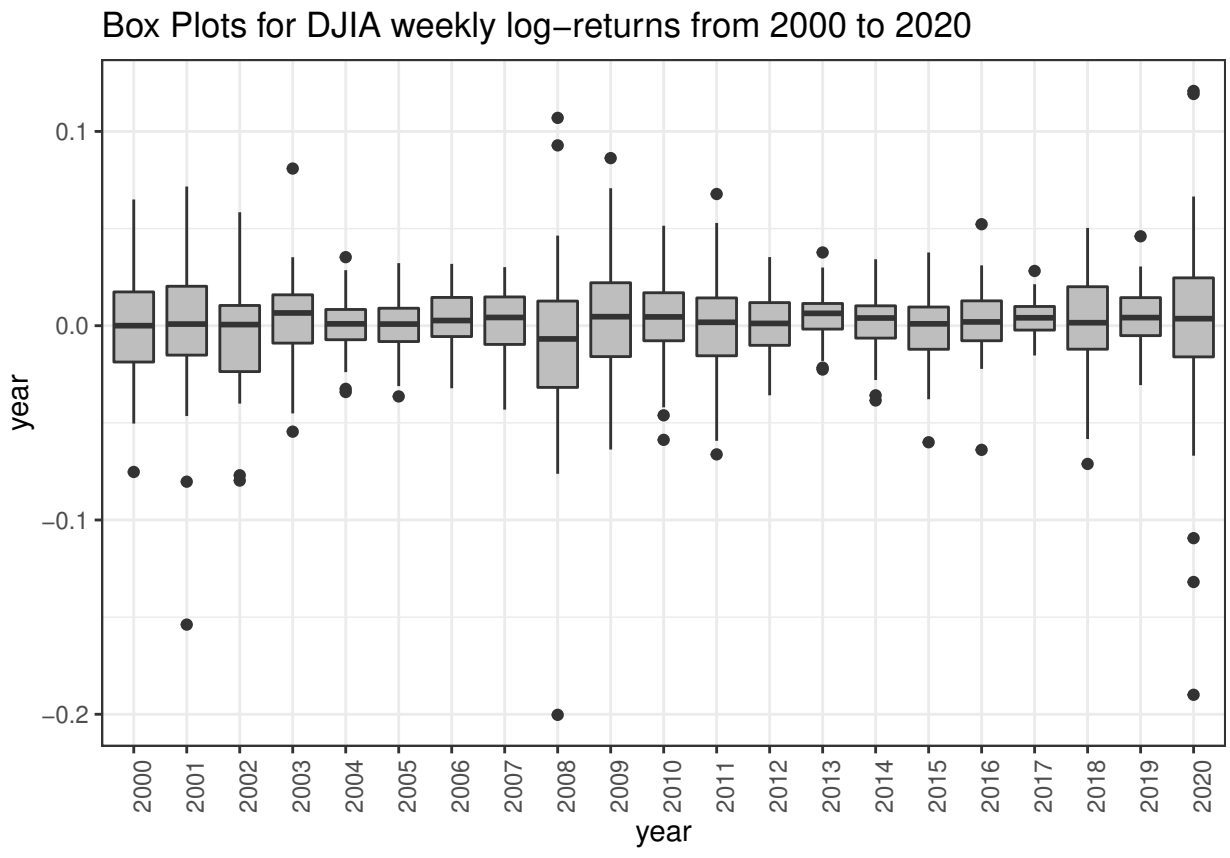
```
##  [1] "2001" "2002" "2003" "2004" "2008" "2010" "2011" "2014" "2015" "2016"
## [11] "2019" "2020"
```

```
dji_stats["Kurtosis",order(dji_stats["Kurtosis",,])]
```

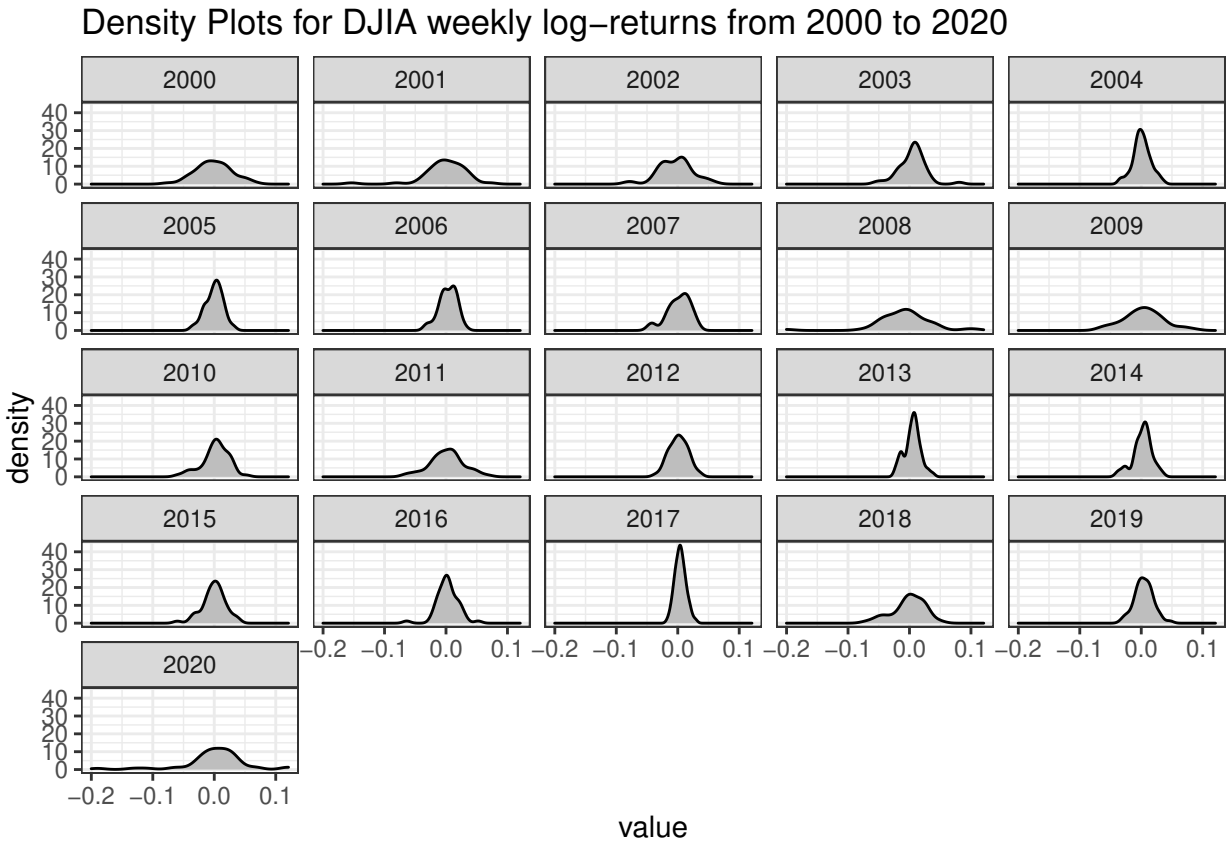
```
##                2012     2006      2005      2013      2000      2017      2007
## Kurtosis -0.461228 -0.23091 -0.213954 -0.200282 -0.198923 -0.124341 -0.079408
##                2018      2009     2011     2004     2019     2014     2010
## Kurtosis -0.057607 -0.033398 0.052429 0.111503 0.148103 0.282354 0.357708
##               2002    2015     2003     2016     2020     2008     2001
## Kurtosis 0.525494 0.66546 2.275154 2.908942 3.594221 5.446623 5.759875
```

Plotting different plots for Weekly log-return data.

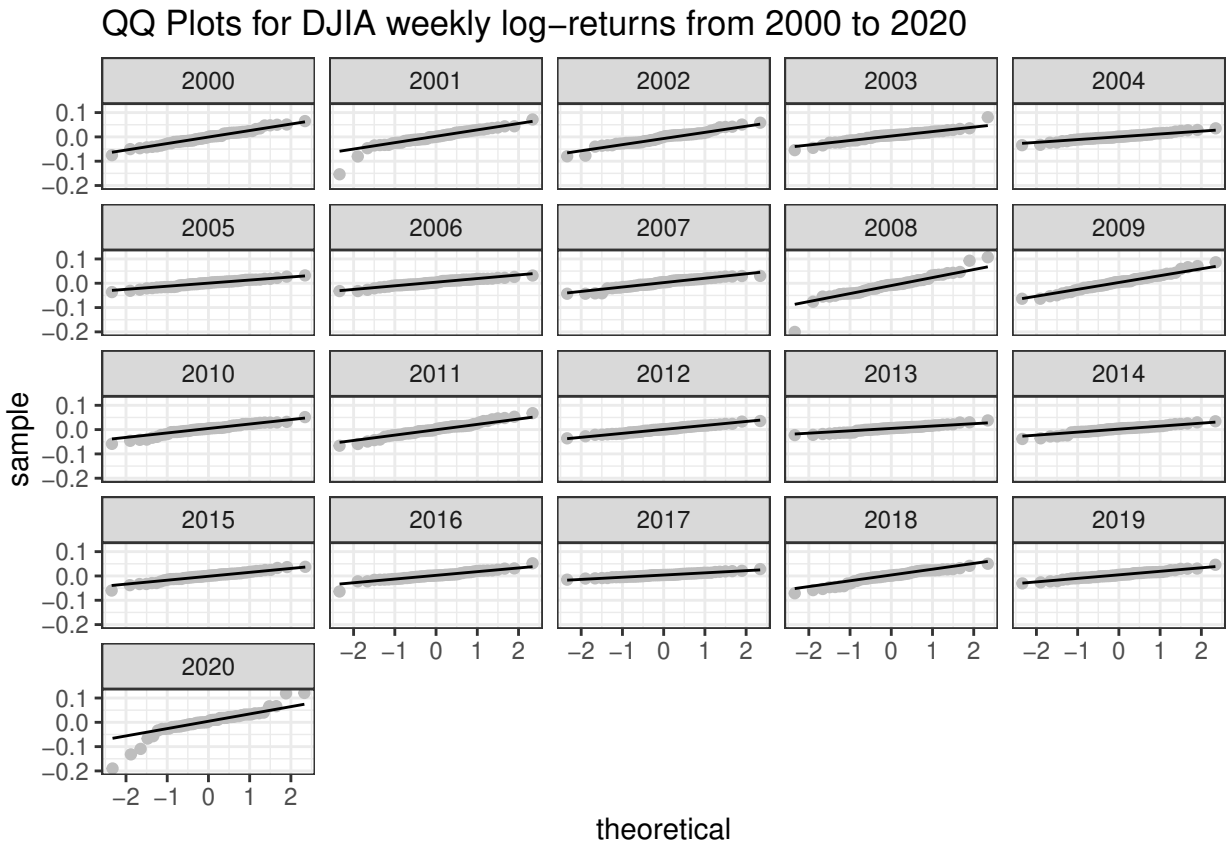
```
boxplot <- ggplot(data = dji_weekly_return_df, aes(x = year, y = value)) + theme_bw() + theme(legend.pos

boxplot
```



```
densityplot <- ggplot(data = dji_weekly_return_df, aes(x = value)) + geom_density(fill = "gray") + facet

densityplot
```




```
qqplot <- ggplot(data = dji_weekly_return_df, aes(sample = value)) + stat_qq(colour = "gray") + stat_qq

qqplot
```



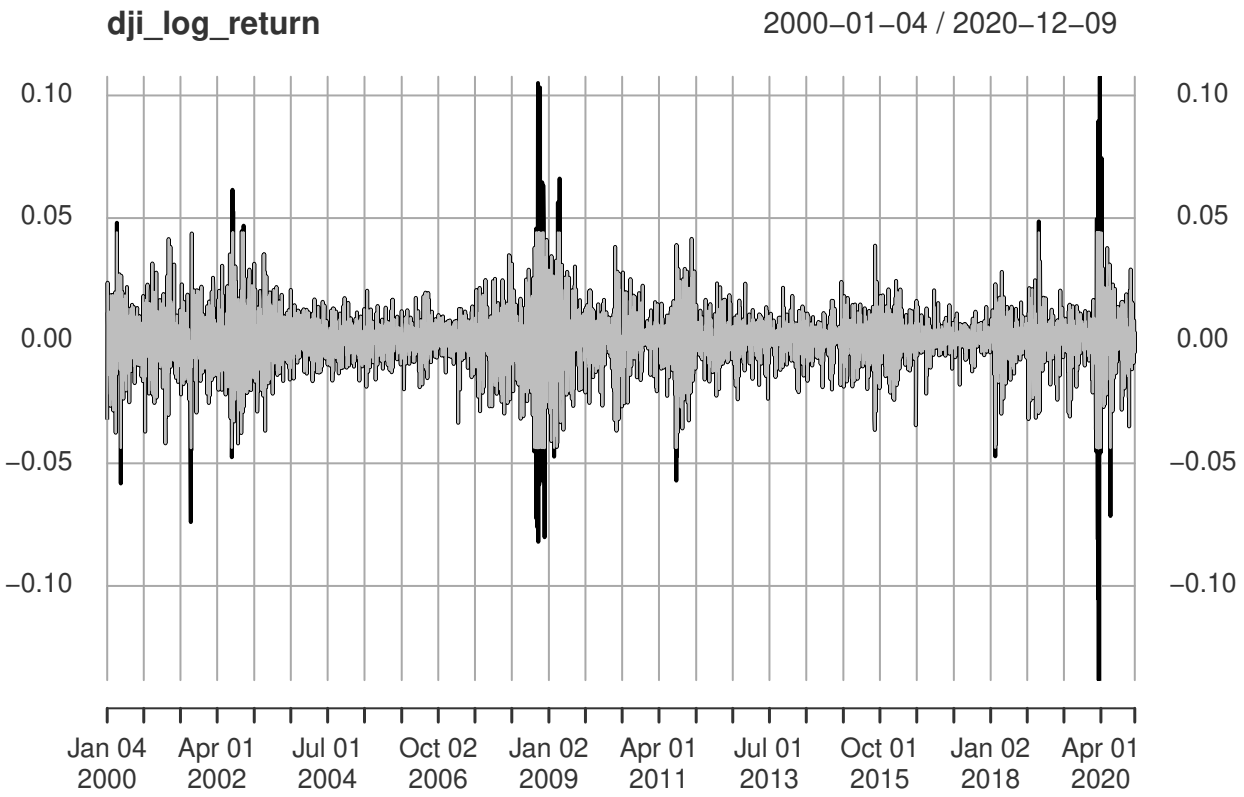
GARCH Model

Now we will build a GARCH model for the daily returns of DJIA.

Outlier Detection

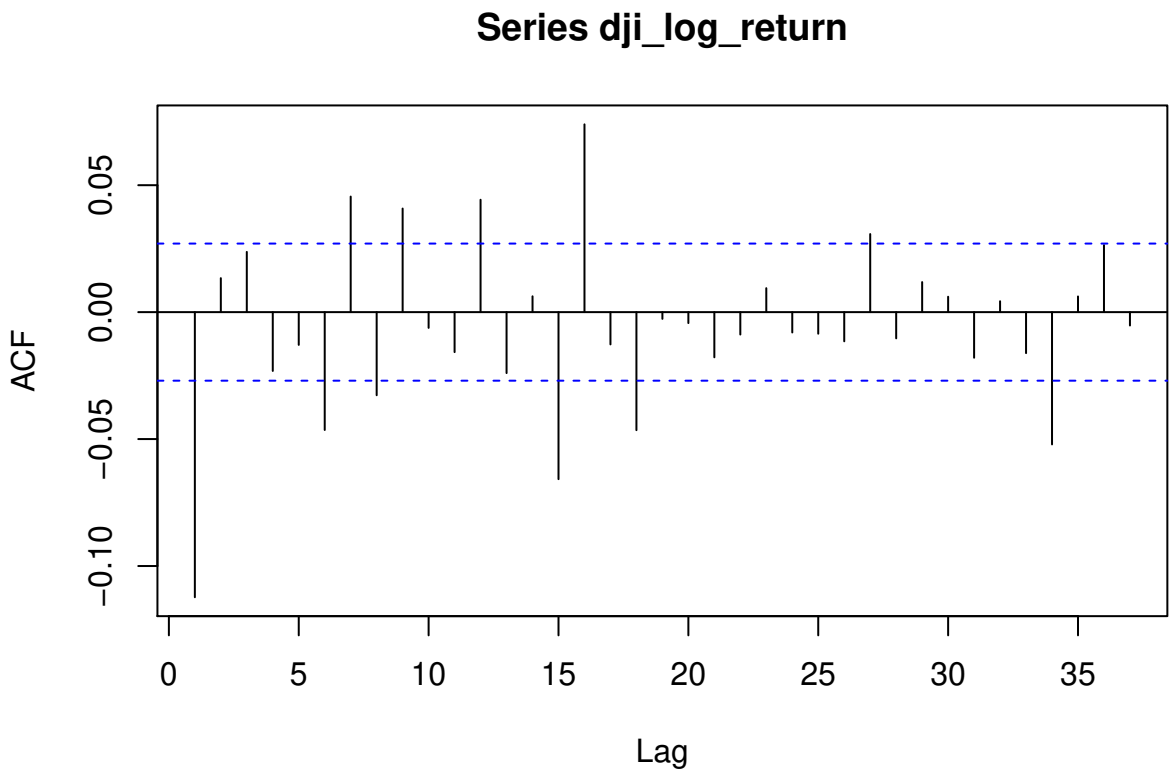
The Return.clean function within Performance Analytics package is able to clean return time series from outliers. Here below we compare the original time series with the outliers adjusted one.

```
#Outlier Detection by log-returns values.
dji_return_outliersadj <- Return.clean(dji_log_return, "boudt")
p <- plot(dji_log_return)
p <- addSeries(dji_return_outliersadj, col = 'gray', on = 1)
p
```



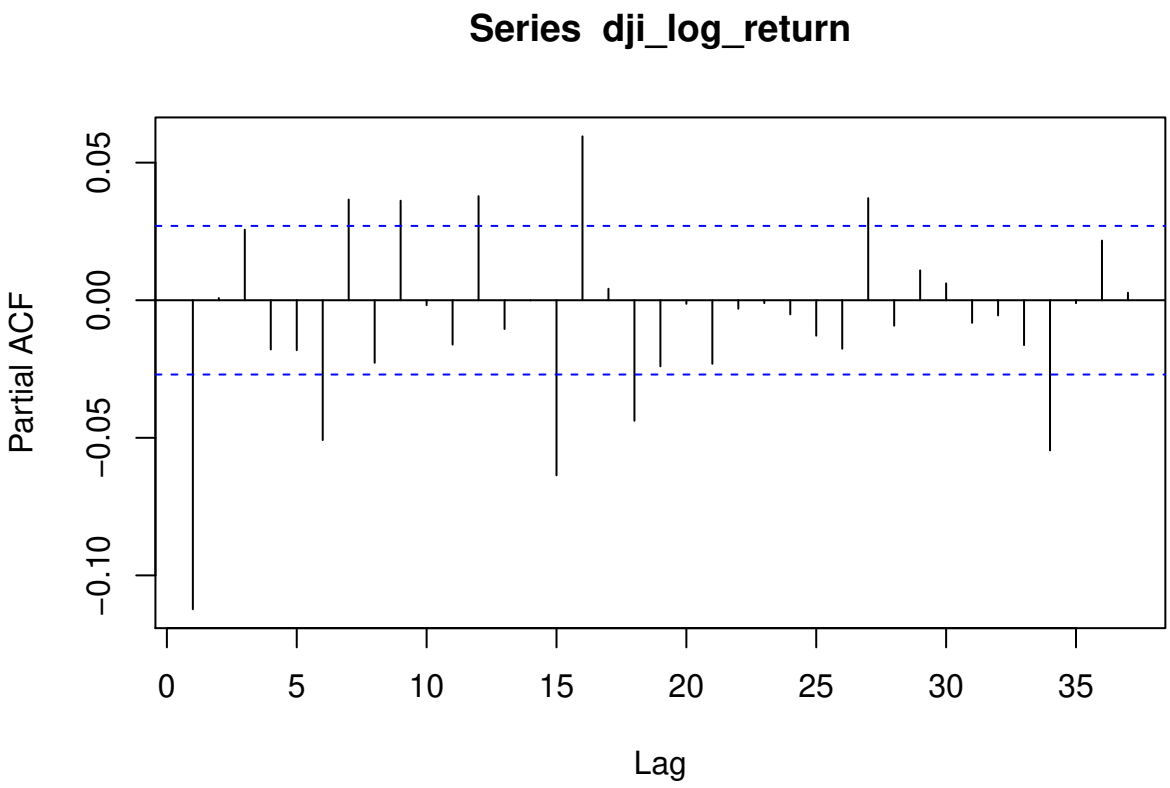
ACF plot:

```
#ACF
acf(dji_log_return)
```



PACF plot:

```
#PACF
pacf(dji_log_return)
```



Above correlation plots suggest some ARMA(p,q) model with p and q > 0. That will be verified within the prosecution of the present analysis.

Now, we run the Augmented Dickey-Fuller unit root test using the urca package. The Augmented Dickey Fuller Test (ADF) is unit root test for stationarity. Unit roots can cause unpredictable results in your time series analysis.

```
#unit root tests
(urdftest_lag = floor(12* (nrow(dji_log_return)/100)^0.25)) #number of lags with unit roots.
```

```
## [1] 32
```

```
summary(ur.df(dji_log_return, type = "none", lags = urdftest_lag, selectlags="BIC"))
```

```
##
## ###############################################
## # Augmented Dickey-Fuller Test Unit Root Test #
## ###############################################
##
## Test regression none
##
##
## Call:
## lm(formula = z.diff ~ z.lag.1 - 1 + z.diff.lag)
##
## Residuals:
```

```
##       Min        1Q    Median        3Q       Max
## -0.128242 -0.004590  0.000660  0.005598  0.104194
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## z.lag.1    -1.112233   0.020611 -53.962   <2e-16 ***
## z.diff.lag -0.000674   0.013816  -0.049    0.961
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.012 on 5233 degrees of freedom
## Multiple R-squared:  0.5568, Adjusted R-squared:  0.5566
## F-statistic:  3287 on 2 and 5233 DF,  p-value: < 2.2e-16
##
##
## Value of test-statistic is: -53.9619
##
## Critical values for test statistics:
##       1pct  5pct 10pct
## tau1 -2.58 -1.95 -1.62
```

Based on reported test statistics compared with critical values, we reject the null hypothesis of unit root presence.

Now, we will be using ARMA(2,2) + GARCH(1,1) model in order to forecast our predictions. Here ARMA model is the mean model and GARCH is the variance model.

(We select the values of p,q for the above models based on the statistical significance of all the coefficients calculated by these models using different values of p and q.)

```
#ARMA-GARCH: ARMA(2,2) + GARCH(1,1)

garchspec <- ugarchspec(mean.model = list(armaOrder = c(2,2)), variance.model = list( garchOrder = c(1,

(garchfit <- ugarchfit(data = dji_adjusted_close, spec = garchspec, out.sample = 40)) #estimate model f
```

```
##
## *---------------------------------*
## *          GARCH Model Fit        *
## *---------------------------------*
##
## Conditional Variance Dynamics
## -----------------------------------
## GARCH Model  : sGARCH(1,1)
## Mean Model   : ARFIMA(2,0,2)
## Distribution : sstd
##
## Optimal Parameters
## ------------------------------------
##           Estimate  Std. Error    t value Pr(>|t|)
## mu      1.4497e+04  1.2960e+03   11.18596 0.000000
## ar1     9.0333e-01  8.7500e-04 1032.49976 0.000000
## ar2     9.7955e-02  8.9700e-04  109.20900 0.000000
## ma1     2.9120e-02  1.3952e-02    2.08711 0.036878
```

```
## ma2    -1.1331e-02  1.4514e-02   -0.78073 0.434964
## omega   3.1643e+04  2.7146e+03   11.65672 0.000000
## alpha1  6.1020e-01  8.1248e-02    7.51029 0.000000
## beta1   3.8881e-01  8.4932e-02    4.57781 0.000005
## skew    9.1346e-01  8.8810e-03  102.85658 0.000000
## shape   2.2520e+00  5.6243e-02   40.03947 0.000000
##
## Robust Standard Errors:
##           Estimate  Std. Error   t value Pr(>|t|)
## mu      1.4497e+04  4.6886e+03   3.09207 0.001988
## ar1     9.0333e-01  9.8000e-04 921.39801 0.000000
## ar2     9.7955e-02  4.6500e-04 210.74462 0.000000
## ma1     2.9120e-02  1.4378e-02   2.02527 0.042839
## ma2    -1.1331e-02  1.7879e-02  -0.63378 0.526227
## omega   3.1643e+04  6.4256e+03   4.92455 0.000001
## alpha1  6.1020e-01  2.6183e-01   2.33053 0.019778
## beta1   3.8881e-01  2.5399e-01   1.53077 0.125827
## skew    9.1346e-01  3.1237e-02  29.24307 0.000000
## shape   2.2520e+00  2.0862e-01  10.79458 0.000000
##
## LogLikelihood : -32956.16
##
## Information Criteria
## ------------------------------------
##
## Akaike       12.609
## Bayes        12.622
## Shibata      12.609
## Hannan-Quinn 12.613
##
## Weighted Ljung-Box Test on Standardized Residuals
## ------------------------------------
##                            statistic p-value
## Lag[1]                         133.8       0
## Lag[2*(p+q)+(p+q)-1][11]       137.4       0
## Lag[4*(p+q)+(p+q)-1][19]       141.9       0
## d.o.f=4
## H0 : No serial correlation
##
## Weighted Ljung-Box Test on Standardized Squared Residuals
## ------------------------------------
##                           statistic p-value
## Lag[1]                         1213       0
## Lag[2*(p+q)+(p+q)-1][5]        1213       0
## Lag[4*(p+q)+(p+q)-1][9]        1213       0
## d.o.f=2
##
## Weighted ARCH LM Tests
## ------------------------------------
##             Statistic Shape Scale P-Value
## ARCH Lag[3]  0.003789 0.500 2.000  0.9509
## ARCH Lag[5]  0.009699 1.440 1.667  0.9995
## ARCH Lag[7]  0.013808 2.315 1.543  1.0000
##
```

```
## Nyblom stability test
## ------------------------------------
## Joint Statistic:  7.4355
## Individual Statistics:
## mu      4.30834
## ar1     0.30760
## ar2     0.31037
## ma1     0.04385
## ma2     0.37918
## omega  22.30776
## alpha1  2.01095
## beta1   5.62893
## skew    0.31971
## shape   9.41176
##
## Asymptotic Critical Values (10% 5% 1%)
## Joint Statistic:          2.29 2.54 3.05
## Individual Statistic:     0.35 0.47 0.75
##
## Sign Bias Test
## ------------------------------------
##                     t-value       prob sig
## Sign Bias            7.9832  1.739e-15 ***
## Negative Sign Bias  25.5977 2.990e-136 ***
## Positive Sign Bias   0.3431  7.315e-01
## Joint Effect       658.5504 2.039e-142 ***
##
##
## Adjusted Pearson Goodness-of-Fit Test:
## ------------------------------------
##   group statistic p-value(g-1)
## 1    20     61.25    2.453e-06
## 2    30     87.59    8.524e-08
## 3    40    111.46    6.716e-09
## 4    50    134.50    6.693e-10
##
##
## Elapsed time : 2.212353
```

```
#garchfit

garchforecast <- ugarchforecast(garchfit, n.ahead = 40, n.roll = 30)
garchforecast #forecast values
```

```
##
## *----------------------------------*
## *       GARCH Model Forecast       *
## *----------------------------------*
## Model: sGARCH
## Horizon: 40
## Roll Steps: 30
## Out of Sample: 40
##
## 0-roll forecast [T0=2020-10-13]:
```

```
##      Series  Sigma
## T+1   28706  297.6
## T+2   28724  346.6
## T+3   28740  389.4
## T+4   28757  428.0
## T+5   28774  463.3
## T+6   28790  496.0
## T+7   28807  526.7
## T+8   28824  555.7
## T+9   28841  583.2
## T+10  28857  609.5
## T+11  28874  634.6
## T+12  28891  658.7
## T+13  28908  682.0
## T+14  28925  704.5
## T+15  28942  726.3
## T+16  28959  747.4
## T+17  28975  767.9
## T+18  28992  787.9
## T+19  29009  807.3
## T+20  29026  826.3
## T+21  29043  844.8
## T+22  29060  862.9
## T+23  29077  880.6
## T+24  29095  898.0
## T+25  29112  915.0
## T+26  29129  931.7
## T+27  29146  948.1
## T+28  29163  964.1
## T+29  29180  979.9
## T+30  29197  995.5
## T+31  29215 1010.7
## T+32  29232 1025.8
## T+33  29249 1040.6
## T+34  29266 1055.2
## T+35  29284 1069.5
## T+36  29301 1083.7
## T+37  29318 1097.7
## T+38  29336 1111.4
## T+39  29353 1125.0
## T+40  29370 1138.5
```

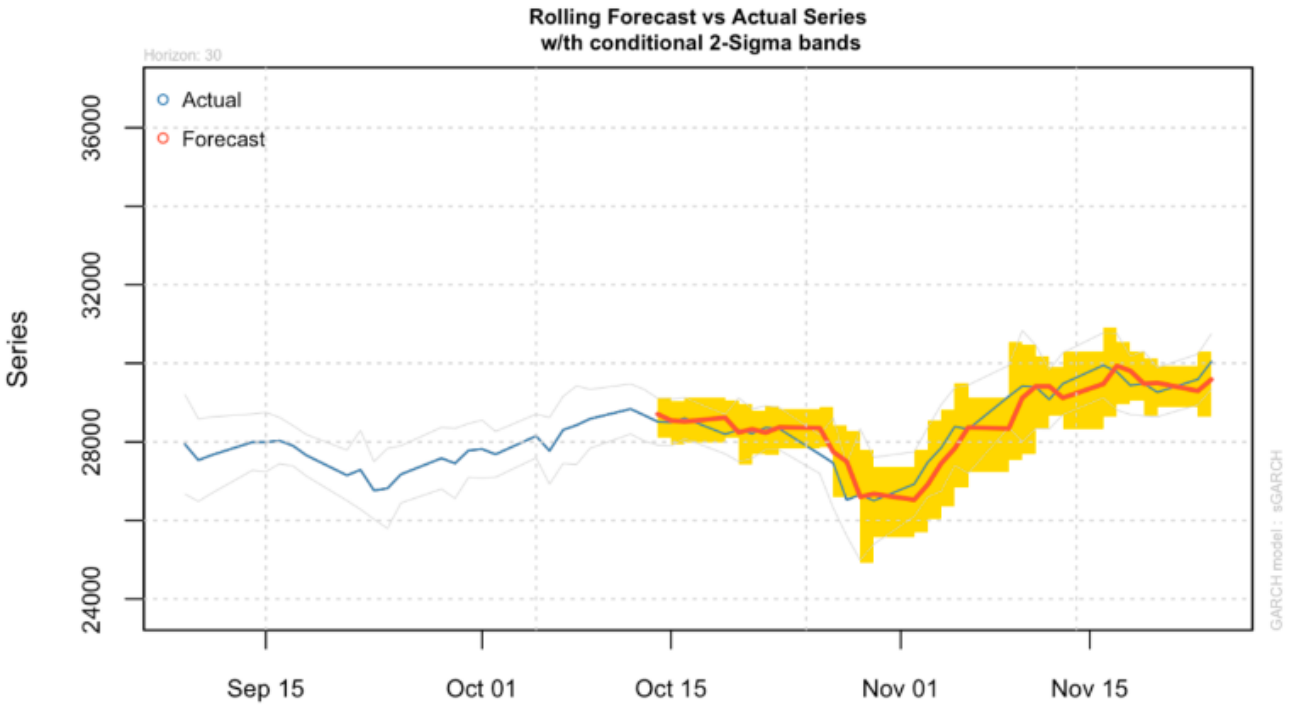
```
#plot(garchforecast) #plot forecasted values
```

We will now calculate the normal residuals and then square them using the log-return data.By doing this residuals plots, any volatile values will visually appear. We try to apply a standard GARCH(1,1) model over ARMA(2,2), looking if we have improved our accuracy and model parameters.

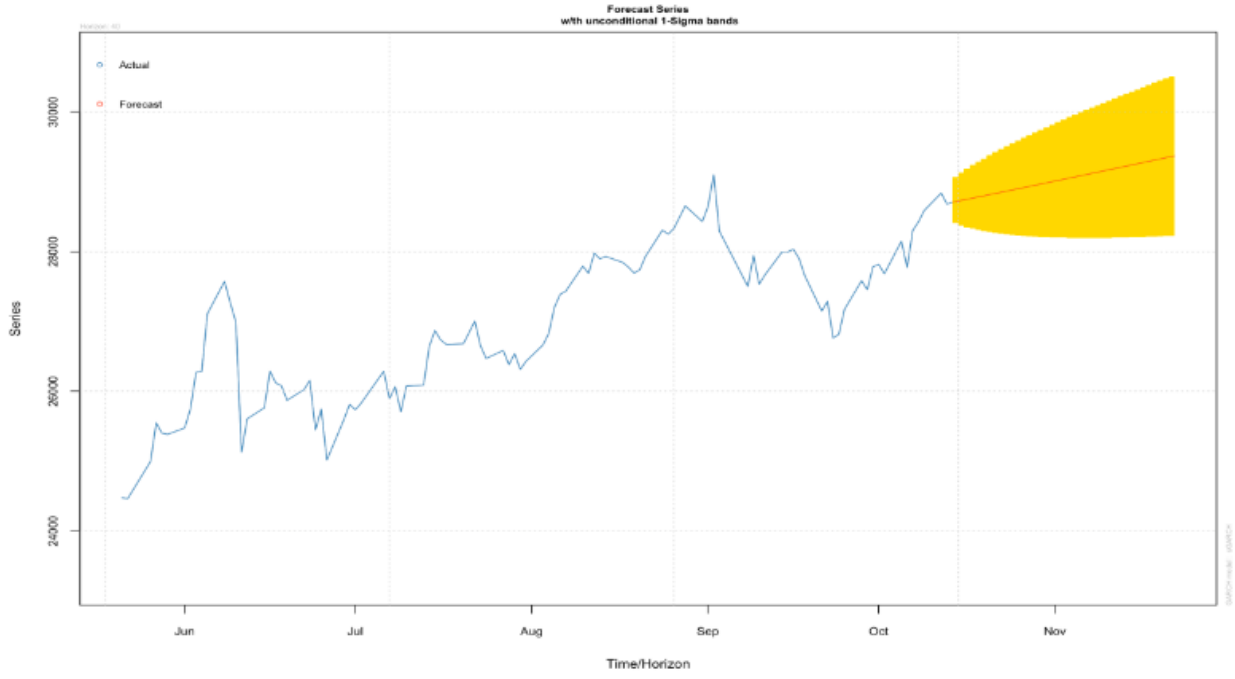
```
garchfit2=ugarchfit(spec=garchspec, data=dji_log_return,  out.sample = 40)
```

Note that the volatility is the square root of the conditional variance.

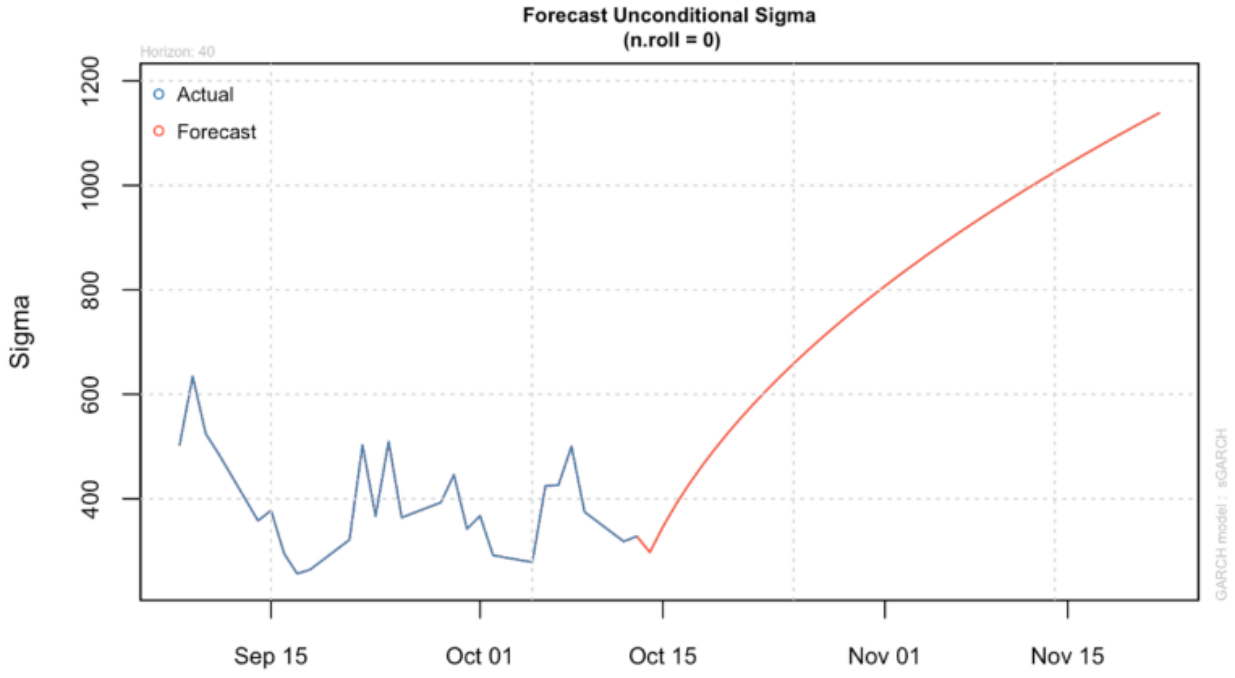

R Console
data.frame
4 x 3


Rolling Forecast vs Actual Series
w/th conditional 2-Sigma bands
Horizon: 30
Actual
Forecast
Series
36000
32000
28000
24000
Sep 15
Oct 01
Oct 15
Nov 01
Nov 15
GARCH model : sGARCH

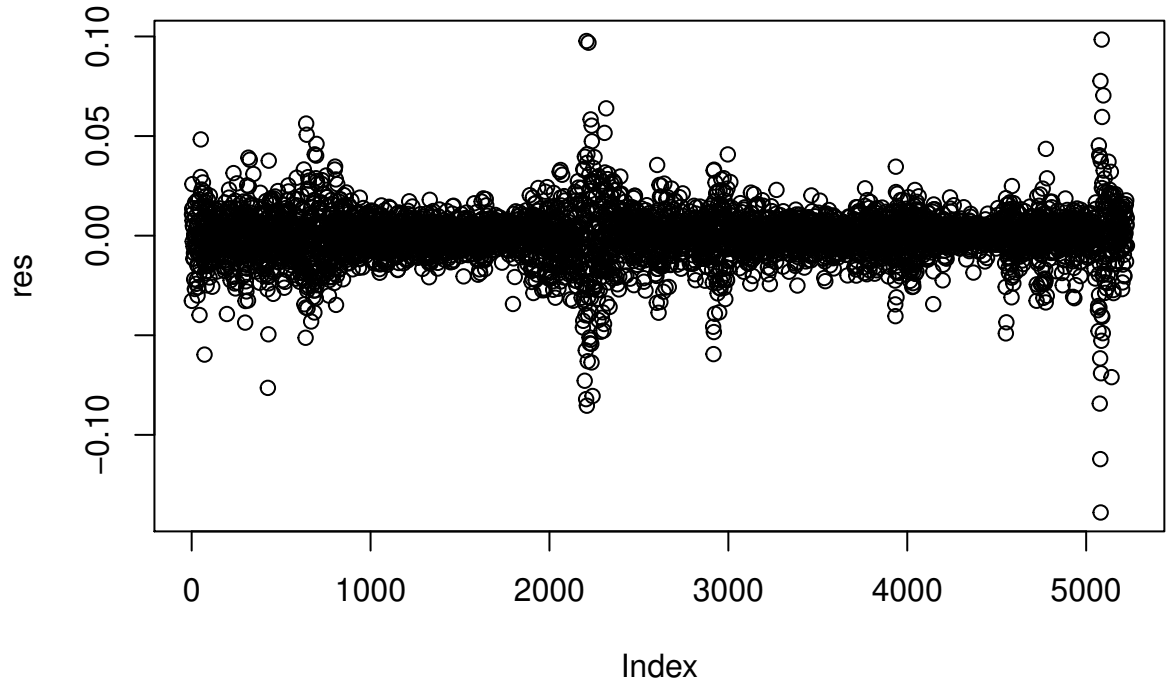

R Console
data.frame
4 x 3


Forecast Series
w/th unconditional 1-Sigma bands
Actual
Forecast
Series
30000
28000
26000
24000
Jun
Jul
Aug
Sep
Oct
Nov
Time/Horizon

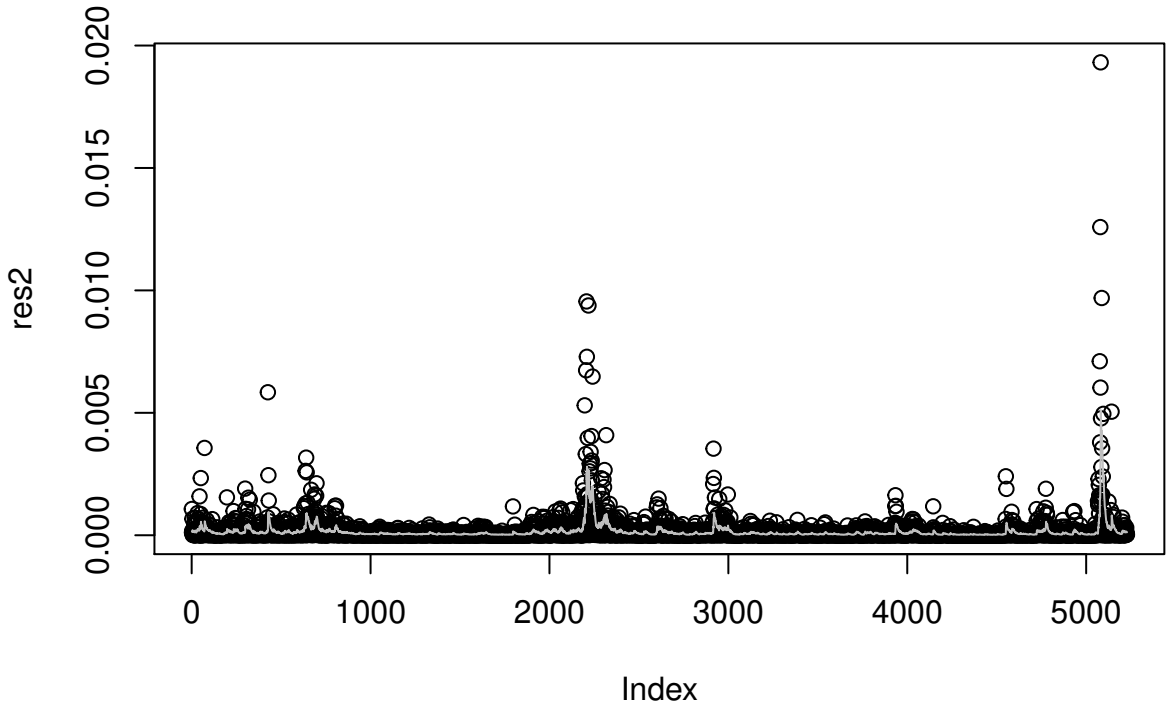

R Console
data.frame
4 x 3


Forecast Unconditional Sigma
(n.roll = 0)
Horizon: 40
Actual
Forecast
Sigma
400
600
800
1000
1200
Sep 15
Oct 01
Oct 15
Nov 01
Nov 15
GARCH model : sGARCH

```r
res <- garchfit2@fit$residuals #stroing estimated normal residuals
plot(res)
```




```r
var <- garchfit2@fit$var #storing estimated variance
res2 <- (res)^2 # calculating square of residuals

plot(res2) #plotting square of residuals
lines(var, col = "gray")
```



The above plots show the residual plots.The first graph is the residual plot with normal residuals after fitting the model. The second plot is plot of the squared residuals and the estimated conditional variance

Now, we forecast the log-return values using the model garchfit2 for next 30 days.

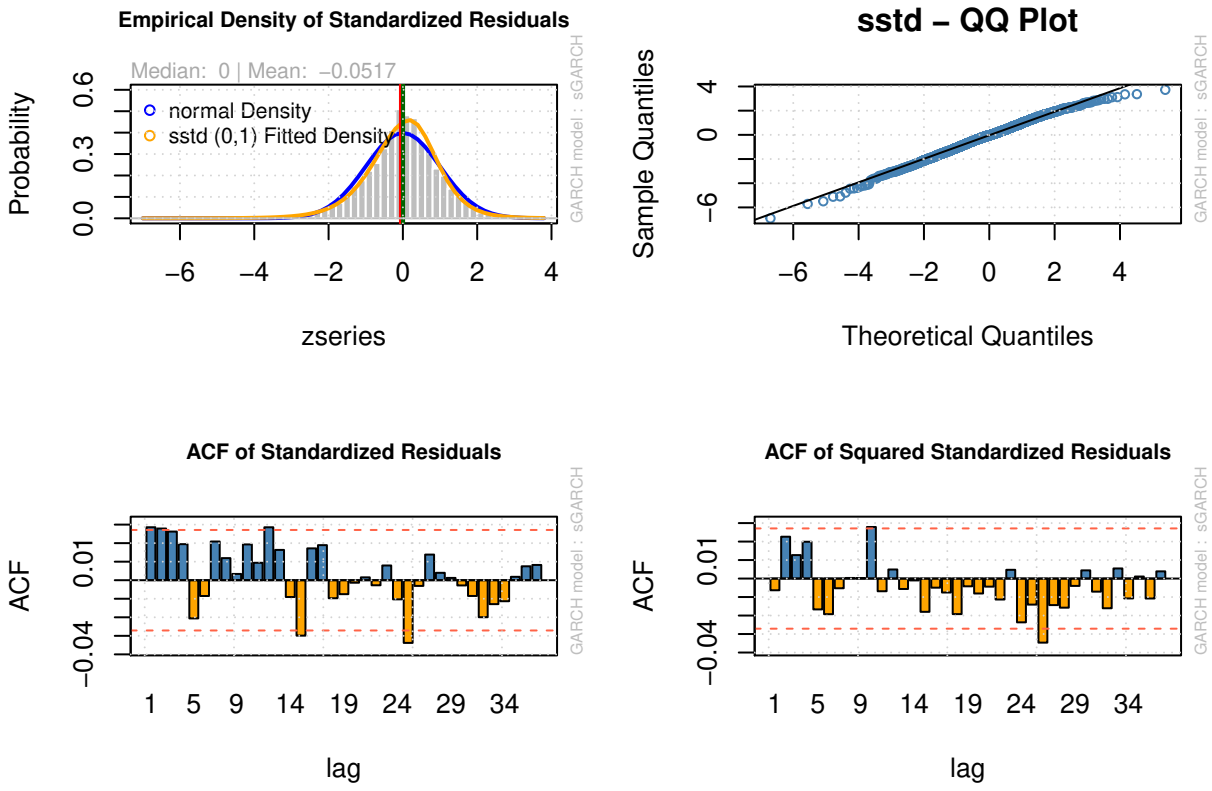
```
#GARCH Forecasting
garchforecast <- ugarchforecast(garchfit2, n.ahead = 30 )
garchforecast
```

```
##
## *----------------------------------*
## *       GARCH Model Forecast       *
## *----------------------------------*
## Model: sGARCH
## Horizon: 30
## Roll Steps: 0
## Out of Sample: 30
##
## 0-roll forecast [T0=2020-10-13]:
##         Series   Sigma
## T+1  0.0003117 0.01055
## T+2  0.0001480 0.01057
## T+3  0.0001628 0.01059
## T+4  0.0001898 0.01062
## T+5  0.0002159 0.01064
## T+6  0.0002403 0.01066
```

```
## T+7  0.0002629 0.01068
## T+8  0.0002838 0.01070
## T+9  0.0003033 0.01072
## T+10 0.0003214 0.01074
## T+11 0.0003382 0.01076
## T+12 0.0003538 0.01078
## T+13 0.0003683 0.01080
## T+14 0.0003817 0.01082
## T+15 0.0003942 0.01084
## T+16 0.0004058 0.01086
## T+17 0.0004165 0.01088
## T+18 0.0004265 0.01090
## T+19 0.0004358 0.01092
## T+20 0.0004444 0.01094
## T+21 0.0004524 0.01096
## T+22 0.0004598 0.01097
## T+23 0.0004667 0.01099
## T+24 0.0004731 0.01101
## T+25 0.0004791 0.01103
## T+26 0.0004846 0.01105
## T+27 0.0004897 0.01106
## T+28 0.0004944 0.01108
## T+29 0.0004989 0.01110
## T+30 0.0005030 0.01112
```

Below are some diagnostic plots for our model working on log-returns data.

```
#plots
par(mfrow=c(2,2))
plot(garchfit2, which=8)
plot(garchfit2, which=9)
plot(garchfit2, which=10)
plot(garchfit2, which=11)
```

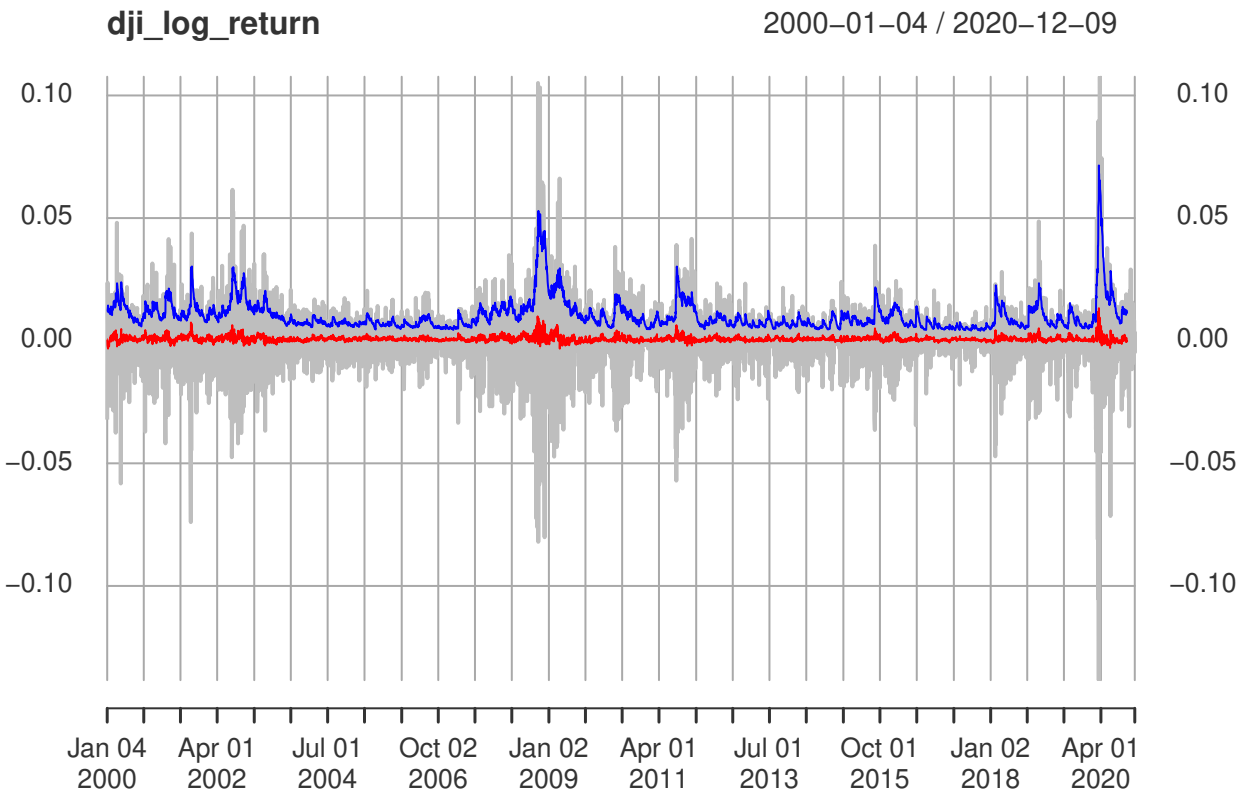


Now, we show the original DJIA adjusted close price time series with the mean model fit (red line).

```
par(mfrow=c(1,1))
cond_volatility <- sigma(garchfit)
mean_model_fit <- fitted(garchfit)
p <- plot(dji_adjusted_close, col = "gray")
p <- addSeries(mean_model_fit, col = 'red', on = 1)
#p <- addSeries(cond_volatility, col = 'red', on = 1)
p
```

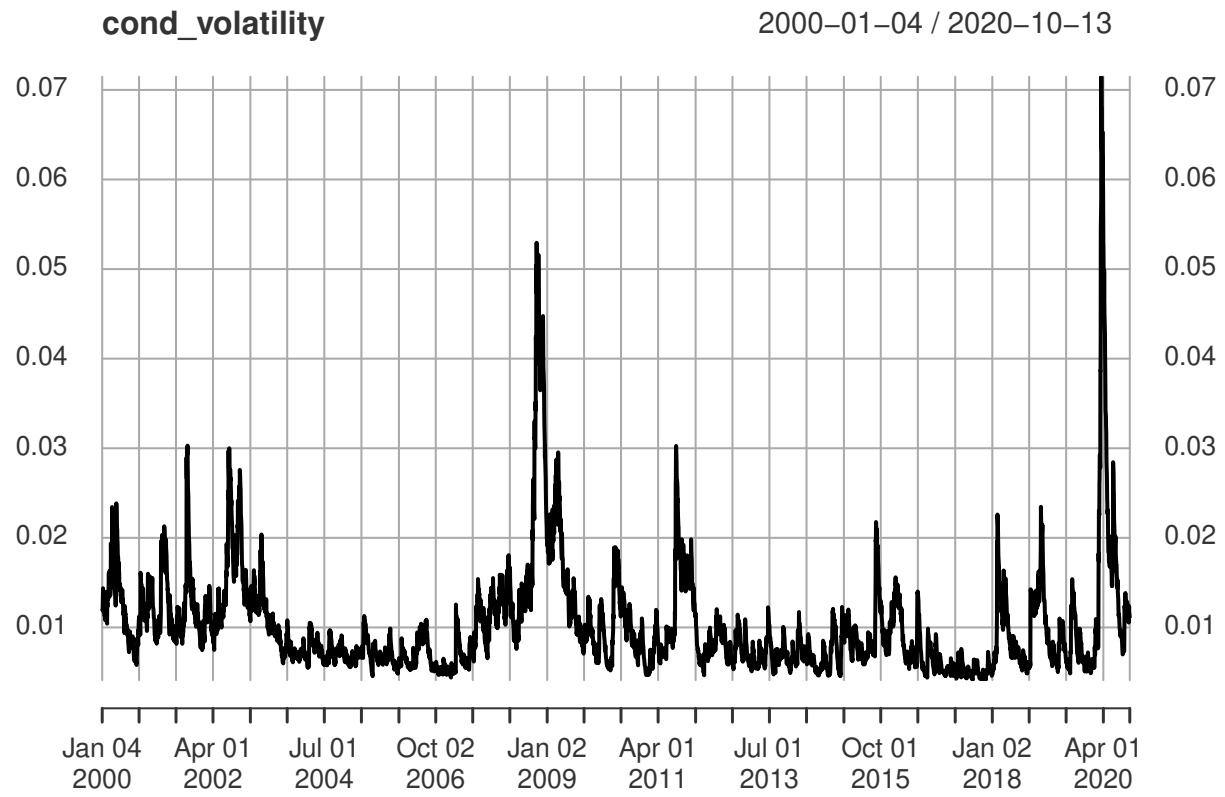
Here is the plot showing the original DJIA log-returns time series with the mean model fit (red line) and the conditional volatility (blue line).

```
par(mfrow=c(1,1))
cond_volatility <- sigma(garchfit2)
mean_model_fit <- fitted(garchfit2)
p <- plot(dji_log_return, col = "grey")
p <- addSeries(mean_model_fit, col = 'red', on = 1)
p <- addSeries(cond_volatility, col = 'blue', on = 1)
p
```



Below is the plot of conditional volatility as a result of our GARCH model on log-returns.

```
#Conditional Volatility Analysis

plot(cond_volatility)
```



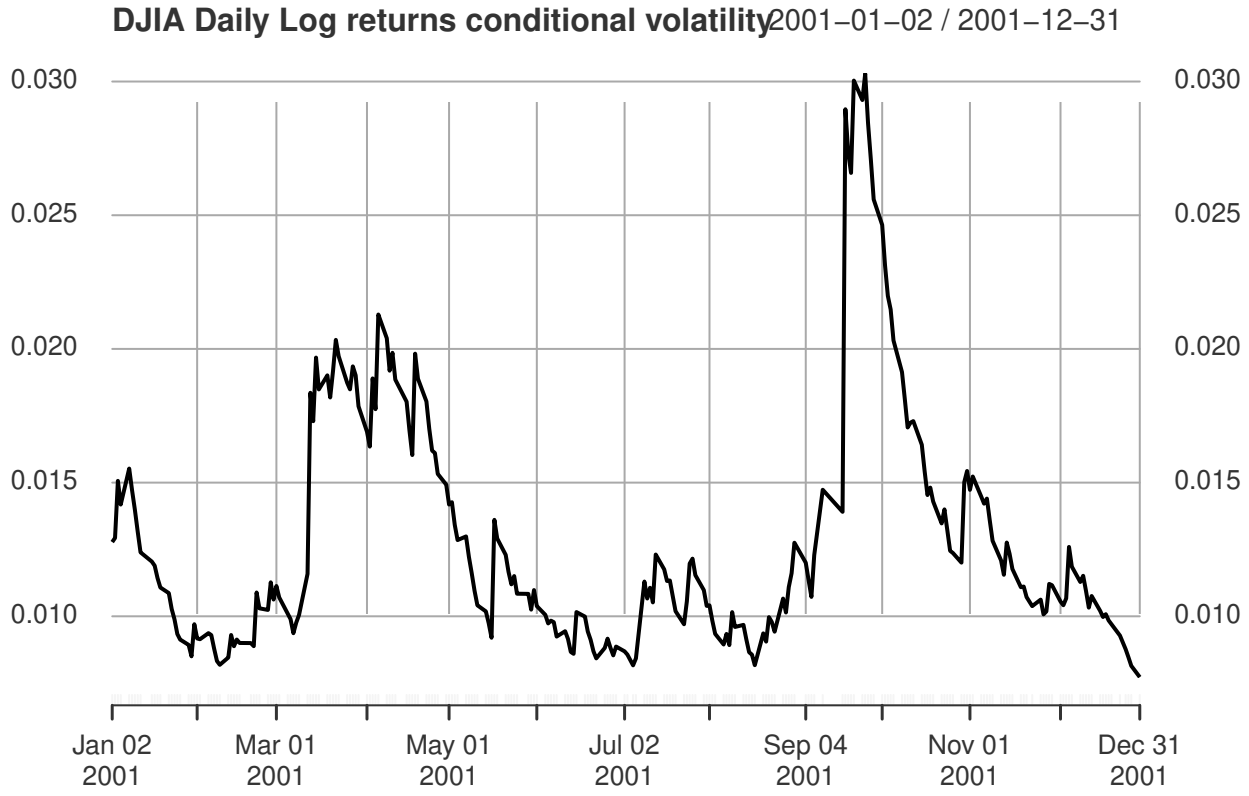
Line plots of conditional volatility by year are shown.

```
par(mfrow=c(1,1))
pl <- lapply(2000:2020, function(x) { plot(cond_volatility[as.character(x)], main = "DJIA Daily Log retu
pl
```

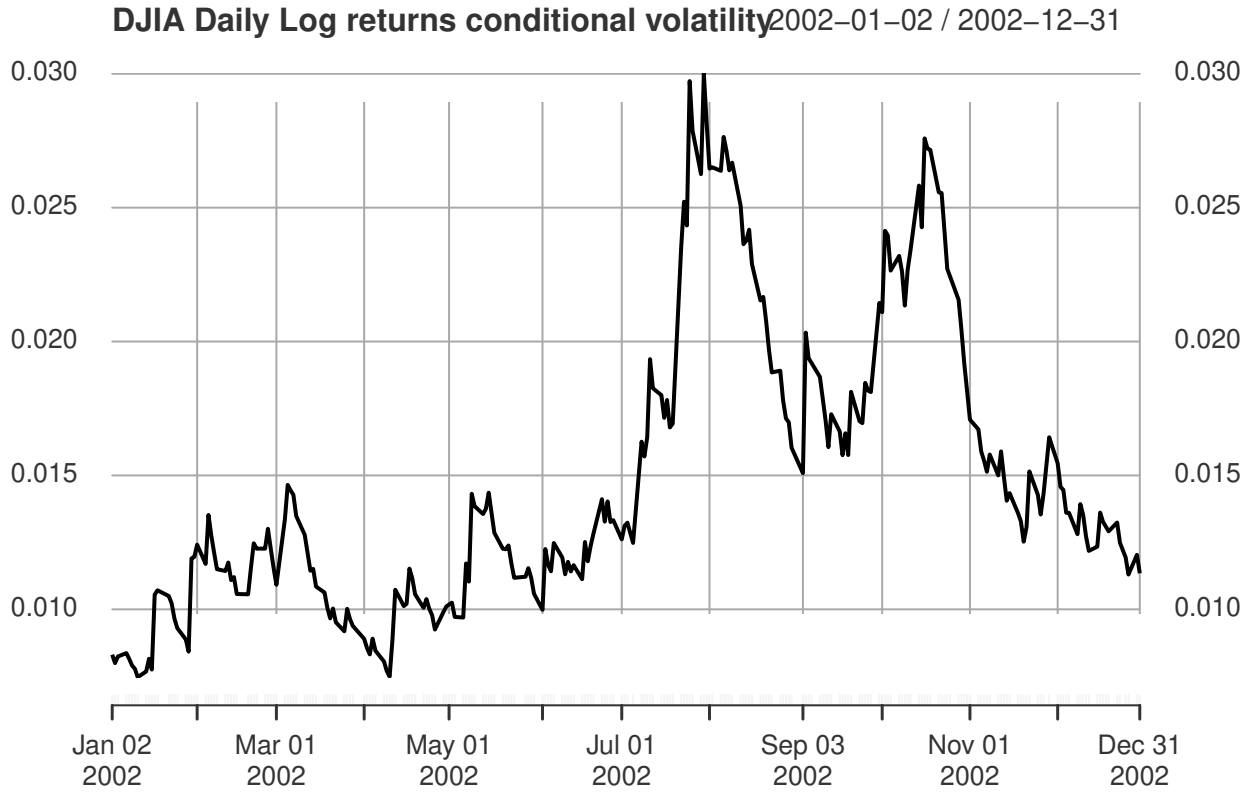
```
## [[1]]
```


DJIA Daily Log returns conditional volatility 2000−01−04 / 2000−12−29
0.020
0.015
0.010
Jan 04 2000
Mar 01 2000
May 01 2000
Jul 03 2000
Sep 01 2000
Nov 01 2000
Dec 29 2000

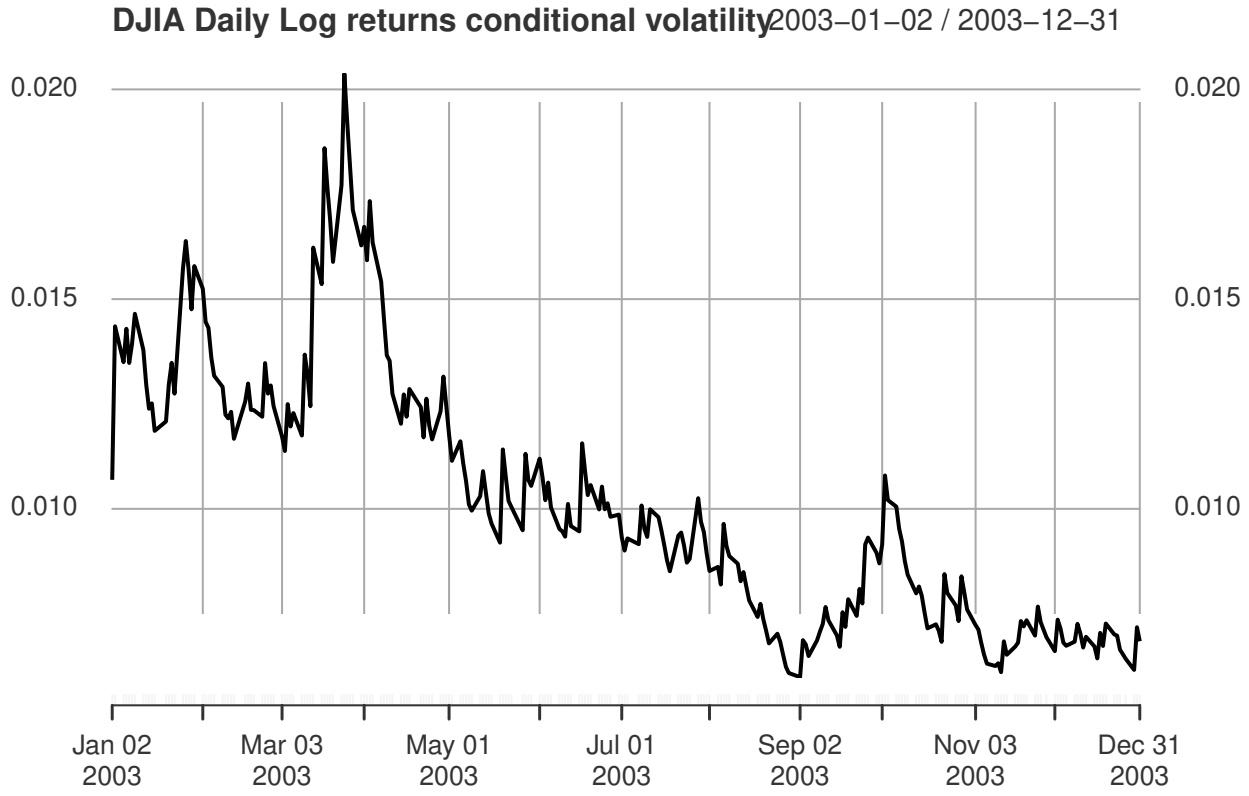
```
##
## [[2]]
```


DJIA Daily Log returns conditional volatility 2001−01−02 / 2001−12−31
0.030
0.025
0.020
0.015
0.010
Jan 02 2001
Mar 01 2001
May 01 2001
Jul 02 2001
Sep 04 2001
Nov 01 2001
Dec 31 2001

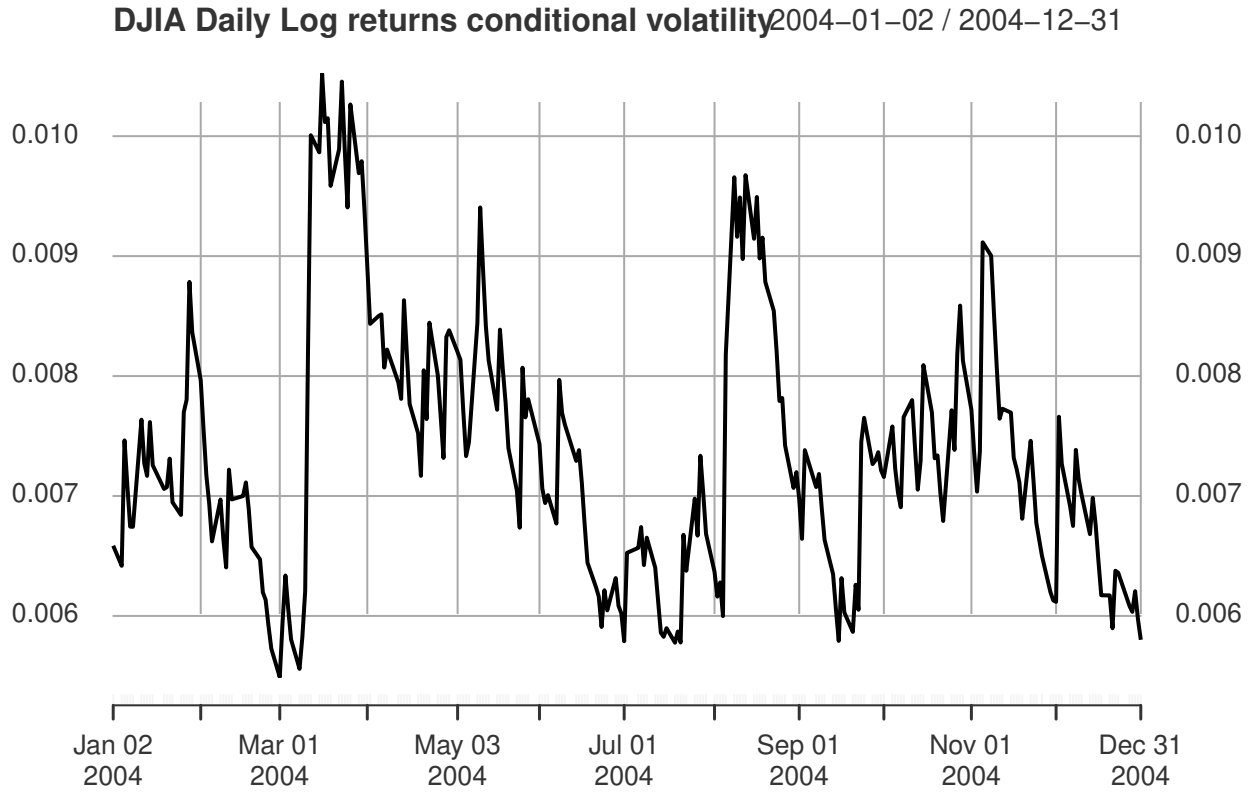
```
##
## [[3]]
```


DJIA Daily Log returns conditional volatility 2002−01−02 / 2002−12−31
0.030
0.025
0.020
0.015
0.010
Jan 02 2002
Mar 01 2002
May 01 2002
Jul 01 2002
Sep 03 2002
Nov 01 2002
Dec 31 2002

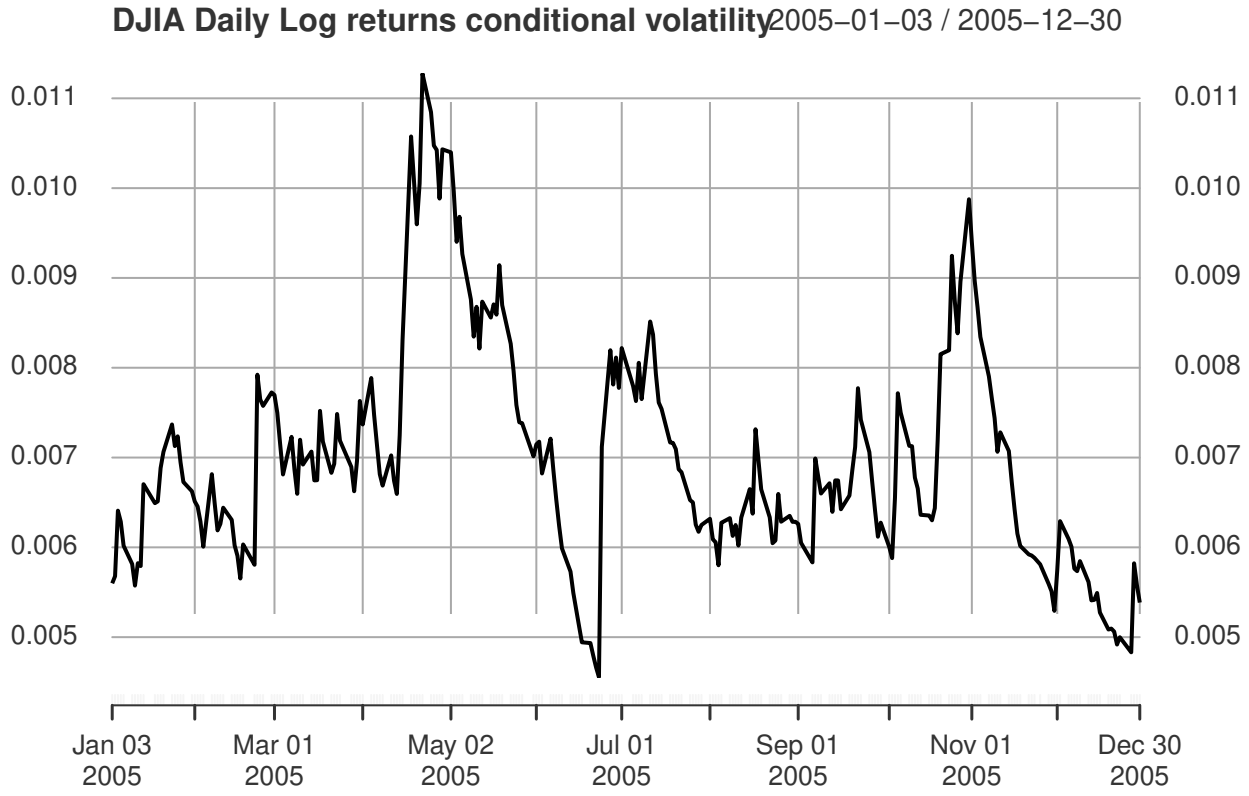
```
##
## [[4]]
```


DJIA Daily Log returns conditional volatility 2003−01−02 / 2003−12−31
0.020
0.015
0.010
Jan 02 2003
Mar 03 2003
May 01 2003
Jul 01 2003
Sep 02 2003
Nov 03 2003
Dec 31 2003

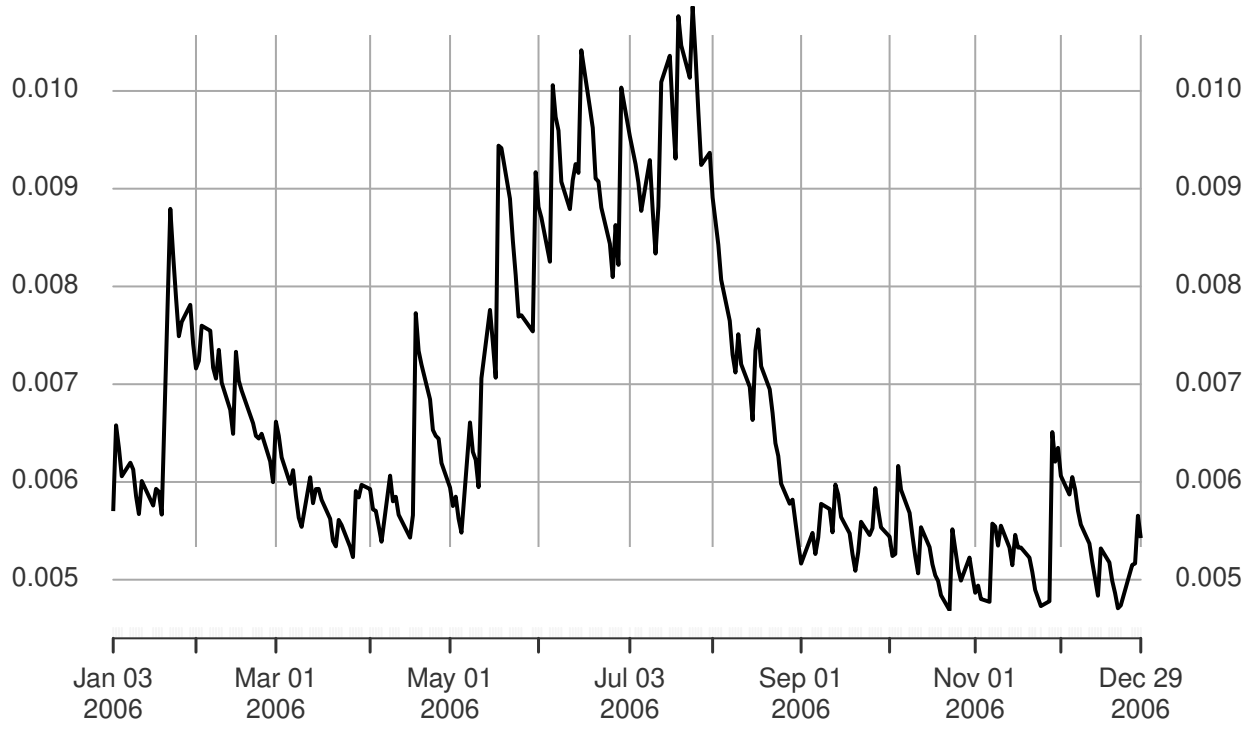
```
##
## [[5]]
```


DJIA Daily Log returns conditional volatility 2004−01−02 / 2004−12−31
0.010
0.009
0.008
0.007
0.006
Jan 02 2004
Mar 01 2004
May 03 2004
Jul 01 2004
Sep 01 2004
Nov 01 2004
Dec 31 2004

```
##
## [[6]]
```


DJIA Daily Log returns conditional volatility
2005−01−03 / 2005−12−30
0.011
0.010
0.009
0.008
0.007
0.006
0.005
Jan 03
2005
Mar 01
2005
May 02
2005
Jul 01
2005
Sep 01
2005
Nov 01
2005
Dec 30
2005

```
##
## [[7]]
```


DJIA Daily Log returns conditional volatility 2006−01−03 / 2006−12−29
0.010
0.009
0.008
0.007
0.006
0.005
Jan 03
2006
Mar 01
2006
May 01
2006
Jul 03
2006
Sep 01
2006
Nov 01
2006
Dec 29
2006

```
##
## [[8]]
```


DJIA Daily Log returns conditional volatility 2007−01−03 / 2007−12−31
0.014
0.012
0.010
0.008
0.006
Jan 03
2007
Mar 01
2007
May 01
2007
Jul 02
2007
Sep 04
2007
Nov 01
2007
Dec 31
2007

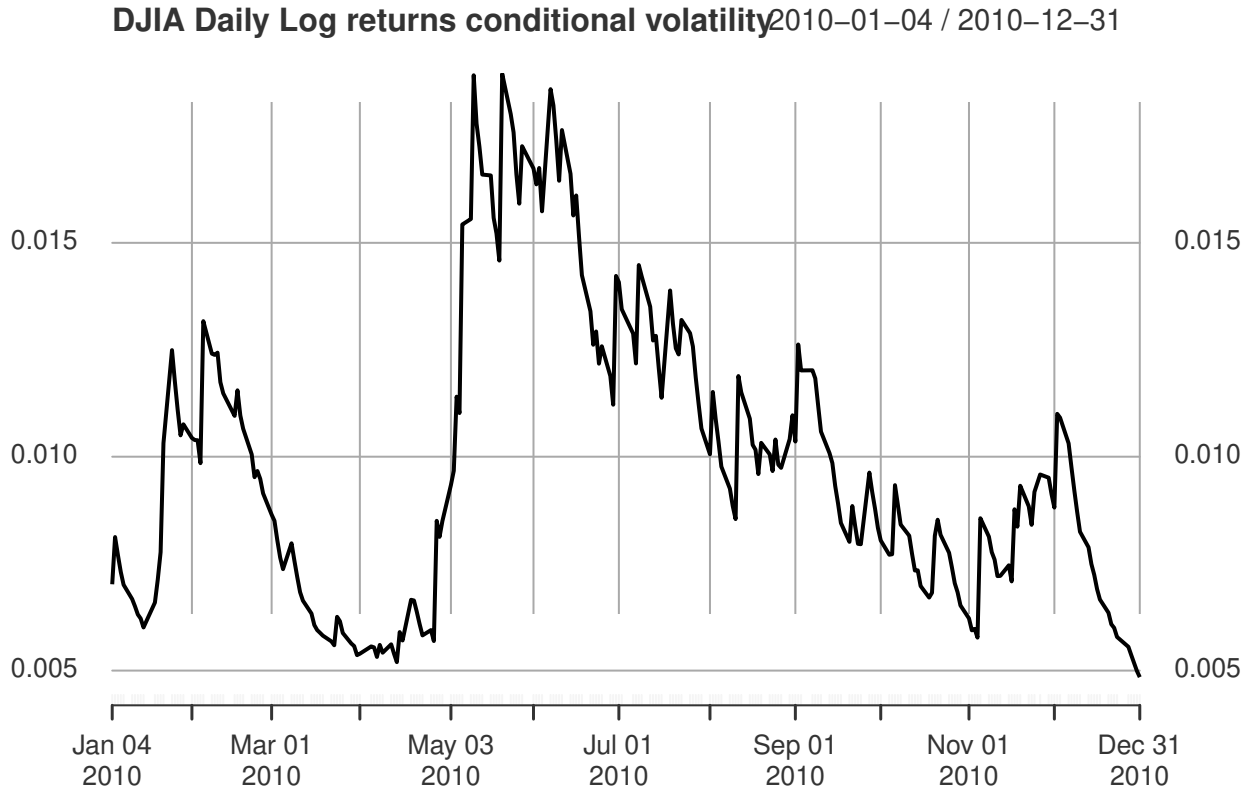
```
##
## [[9]]
```


DJIA Daily Log returns conditional volatility 2008−01−02 / 2008−12−31
0.05
0.04
0.03
0.02
0.01
Jan 02 2008
Mar 03 2008
May 01 2008
Jul 01 2008
Sep 02 2008
Nov 03 2008
Dec 31 2008

```
##
## [[10]]
```


DJIA Daily Log returns conditional volatility 2009−01−02 / 2009−12−31
0.025
0.020
0.015
0.010
Jan 02 2009
Mar 02 2009
May 01 2009
Jul 01 2009
Sep 01 2009
Nov 02 2009
Dec 31 2009

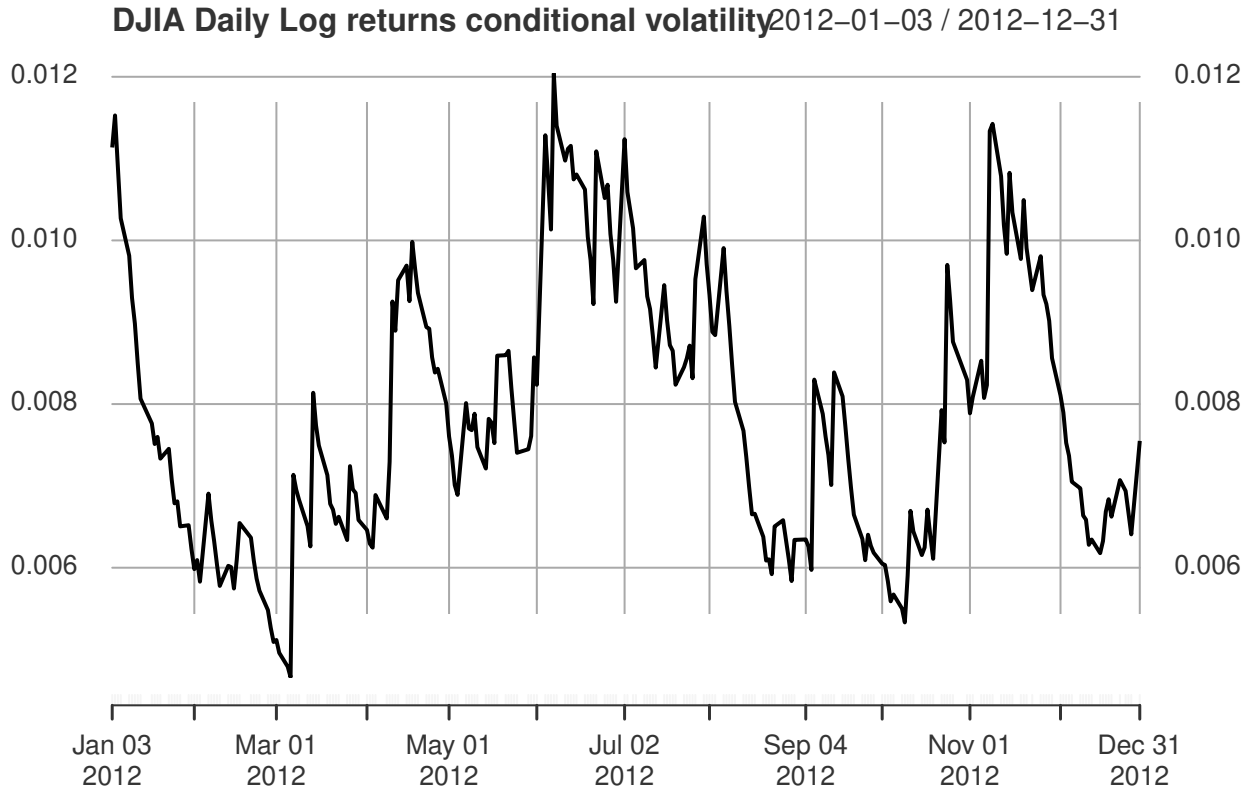
```
##
## [[11]]
```


DJIA Daily Log returns conditional volatility 2010−01−04 / 2010−12−31
0.015
0.010
0.005
Jan 04 2010
Mar 01 2010
May 03 2010
Jul 01 2010
Sep 01 2010
Nov 01 2010
Dec 31 2010

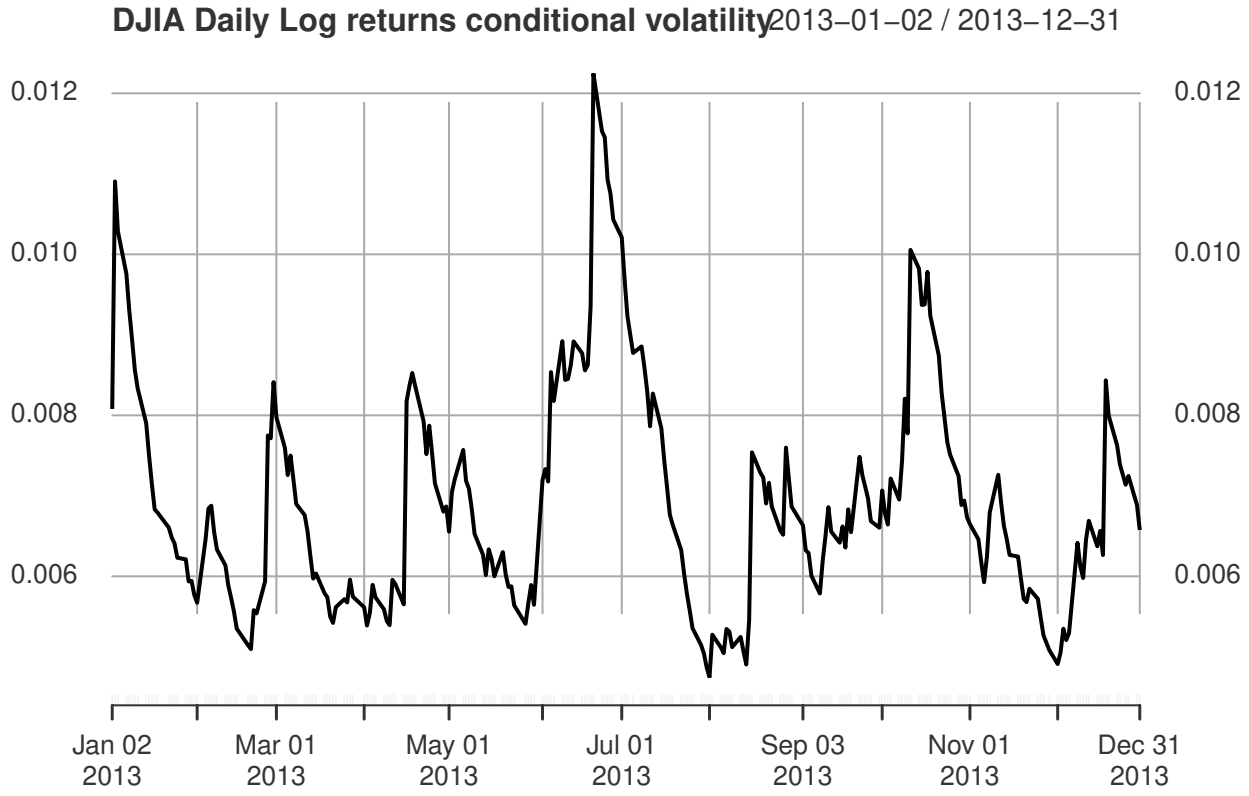
```
##
## [[12]]
```


DJIA Daily Log returns conditional volatility 2011−01−03 / 2011−12−30
0.030
0.025
0.020
0.015
0.010
0.005
Jan 03 2011
Mar 01 2011
May 02 2011
Jul 01 2011
Sep 01 2011
Nov 01 2011
Dec 30 2011

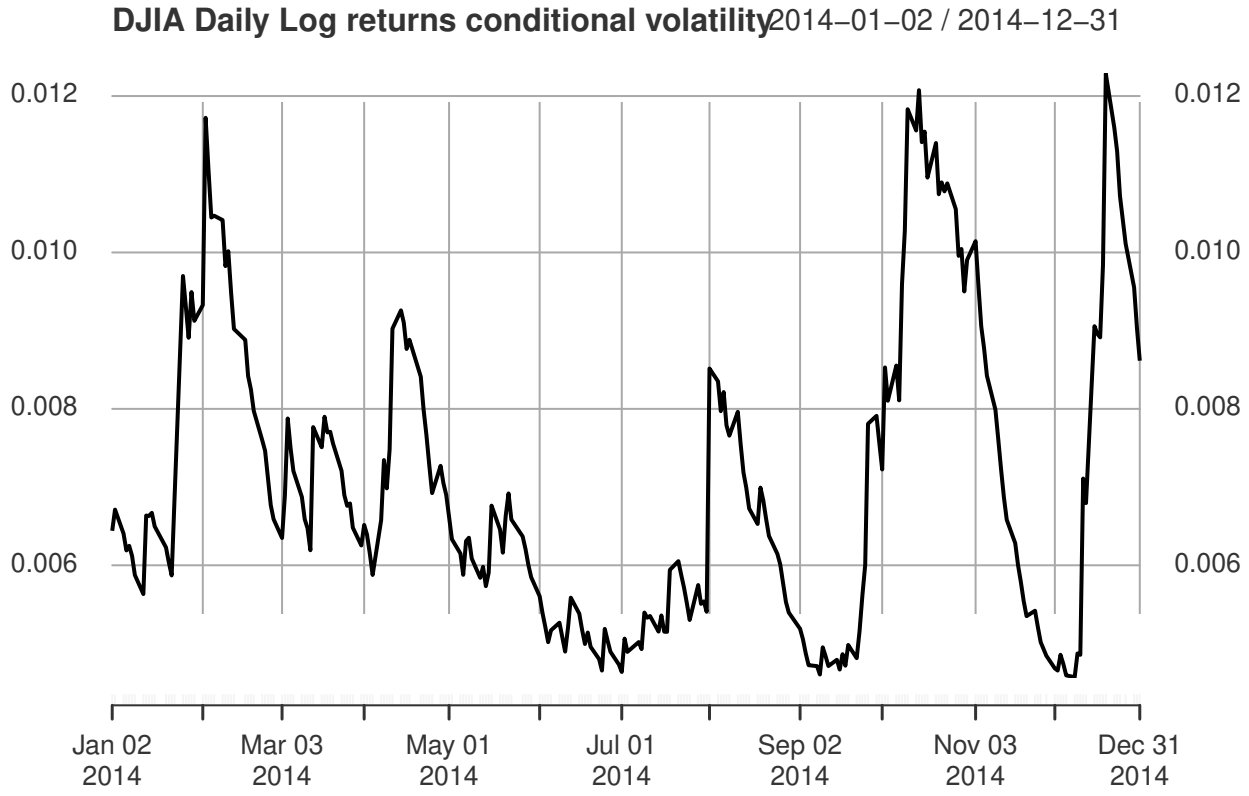
```
##
## [[13]]
```


DJIA Daily Log returns conditional volatility
2012−01−03 / 2012−12−31
0.012
0.010
0.008
0.006
Jan 03
2012
Mar 01
2012
May 01
2012
Jul 02
2012
Sep 04
2012
Nov 01
2012
Dec 31
2012

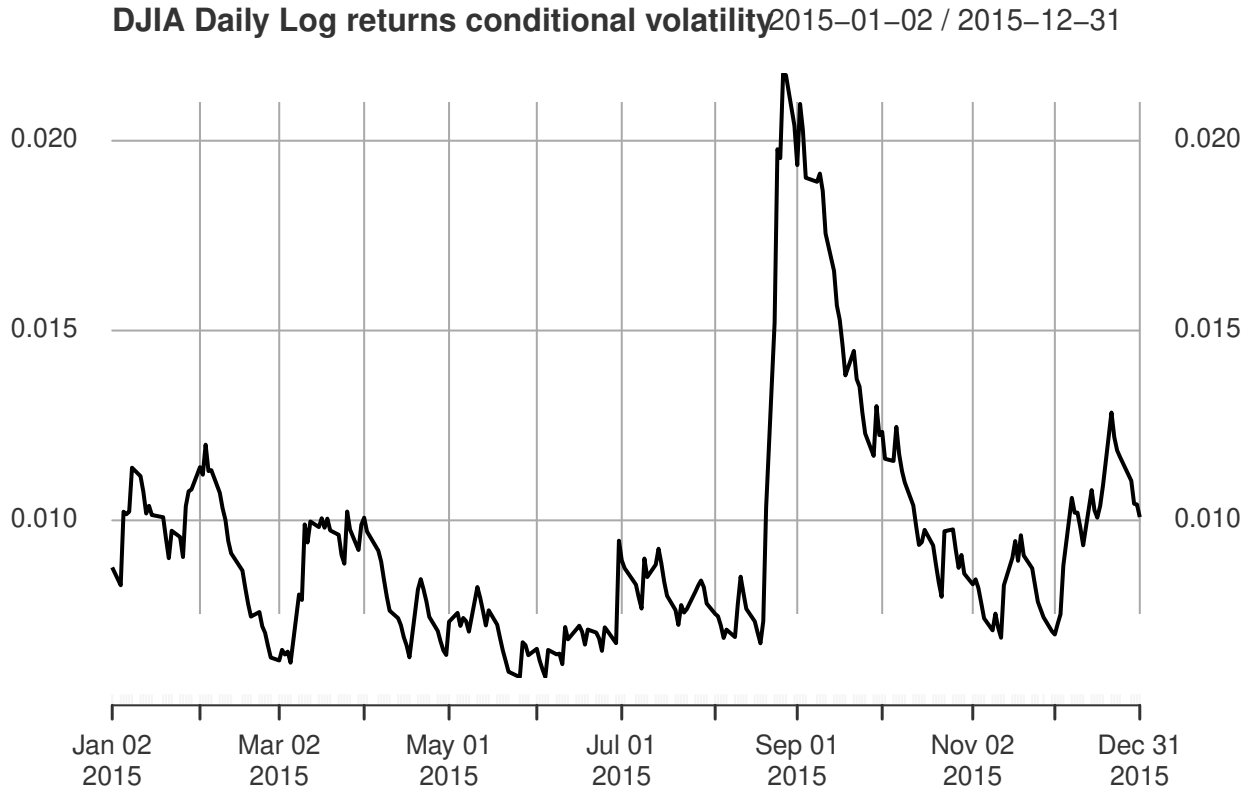
```
##
## [[14]]
```


DJIA Daily Log returns conditional volatility 2013−01−02 / 2013−12−31
0.012
0.010
0.008
0.006
Jan 02 2013
Mar 01 2013
May 01 2013
Jul 01 2013
Sep 03 2013
Nov 01 2013
Dec 31 2013

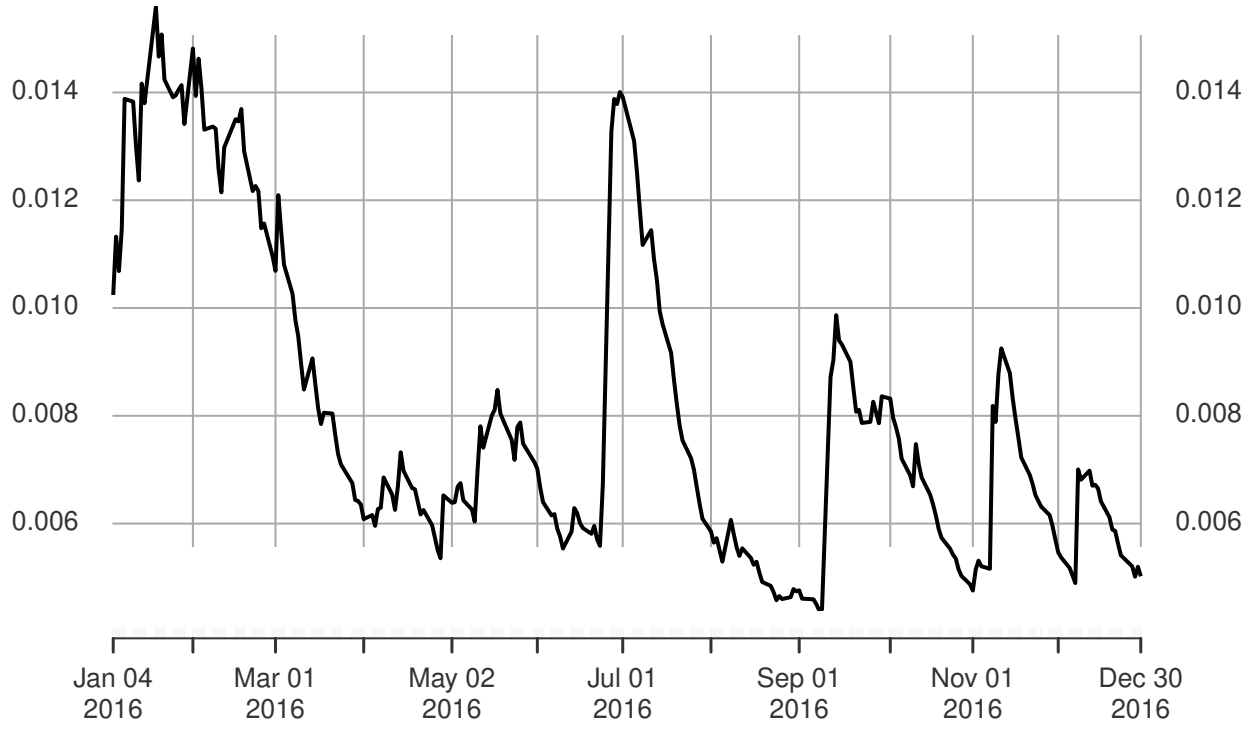
```
##
## [[15]]
```


DJIA Daily Log returns conditional volatility 2014−01−02 / 2014−12−31
0.012
0.010
0.008
0.006
Jan 02 2014
Mar 03 2014
May 01 2014
Jul 01 2014
Sep 02 2014
Nov 03 2014
Dec 31 2014

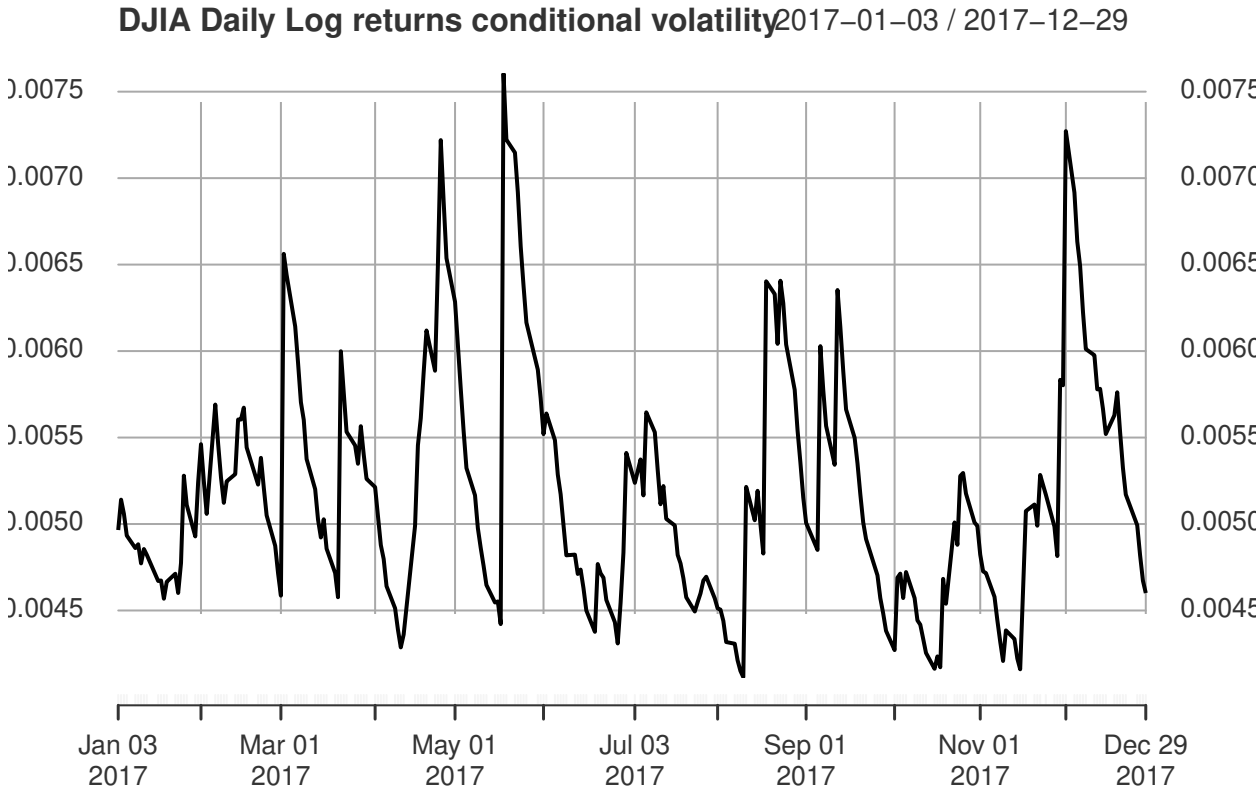
```
##
## [[16]]
```


DJIA Daily Log returns conditional volatility 2015−01−02 / 2015−12−31
0.020
0.015
0.010
Jan 02 2015
Mar 02 2015
May 01 2015
Jul 01 2015
Sep 01 2015
Nov 02 2015
Dec 31 2015

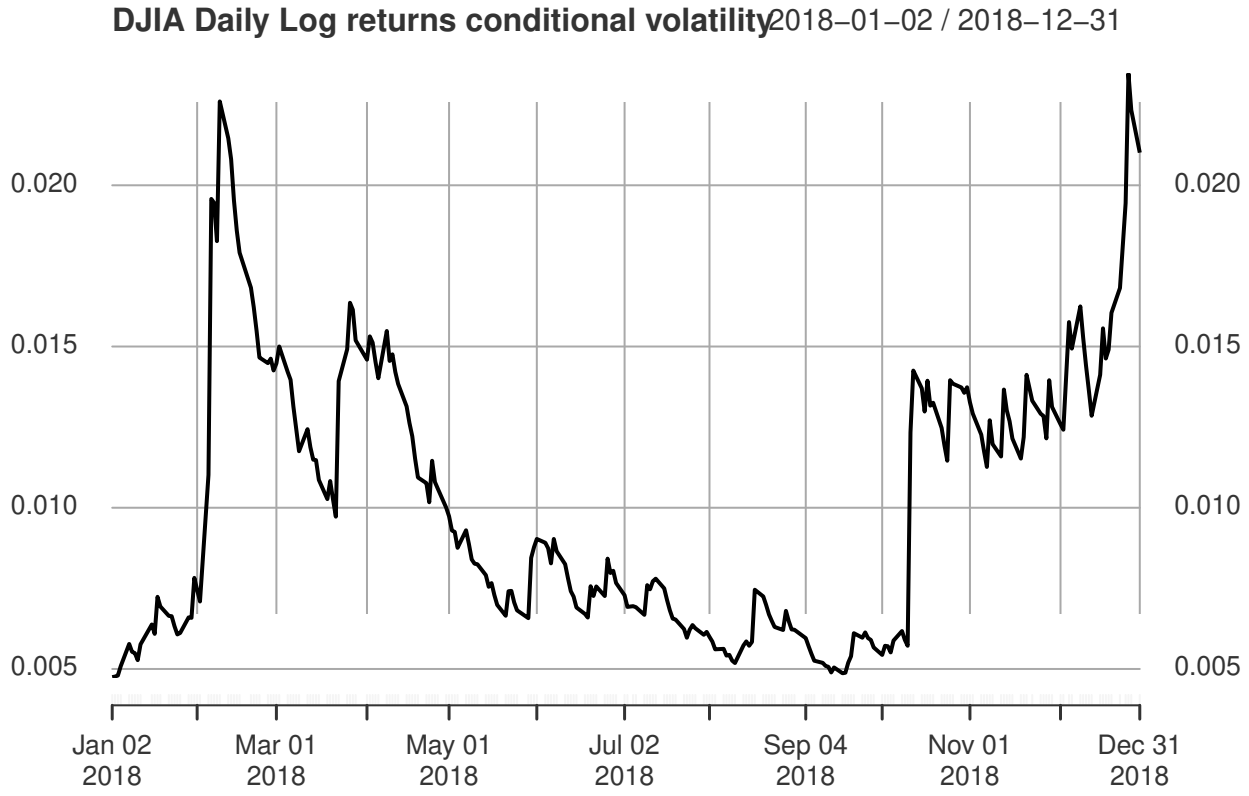
```
##
## [[17]]
```


DJIA Daily Log returns conditional volatility 2016−01−04 / 2016−12−30
0.014
0.012
0.010
0.008
0.006
Jan 04 2016
Mar 01 2016
May 02 2016
Jul 01 2016
Sep 01 2016
Nov 01 2016
Dec 30 2016

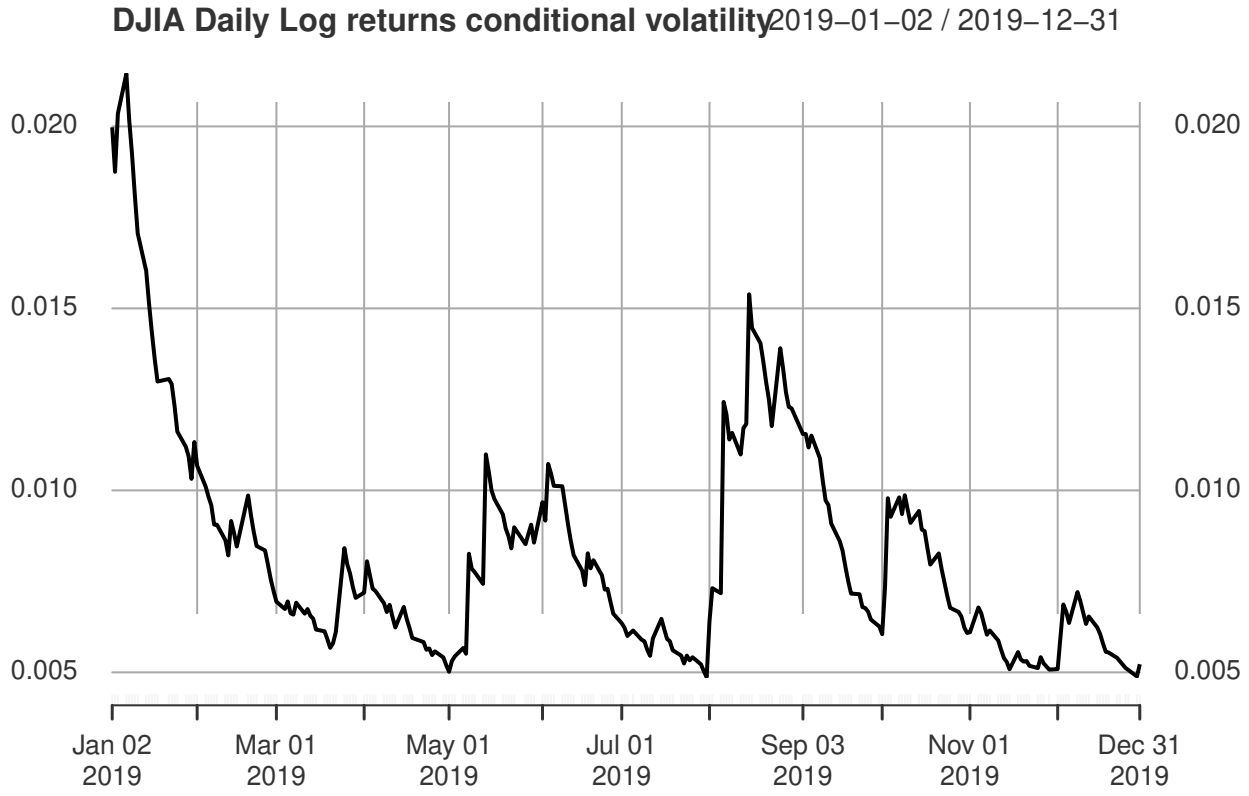
```
##
## [[18]]
```


DJIA Daily Log returns conditional volatility 2017−01−03 / 2017−12−29
0.0075
0.0070
0.0065
0.0060
0.0055
0.0050
0.0045
Jan 03 2017
Mar 01 2017
May 01 2017
Jul 03 2017
Sep 01 2017
Nov 01 2017
Dec 29 2017

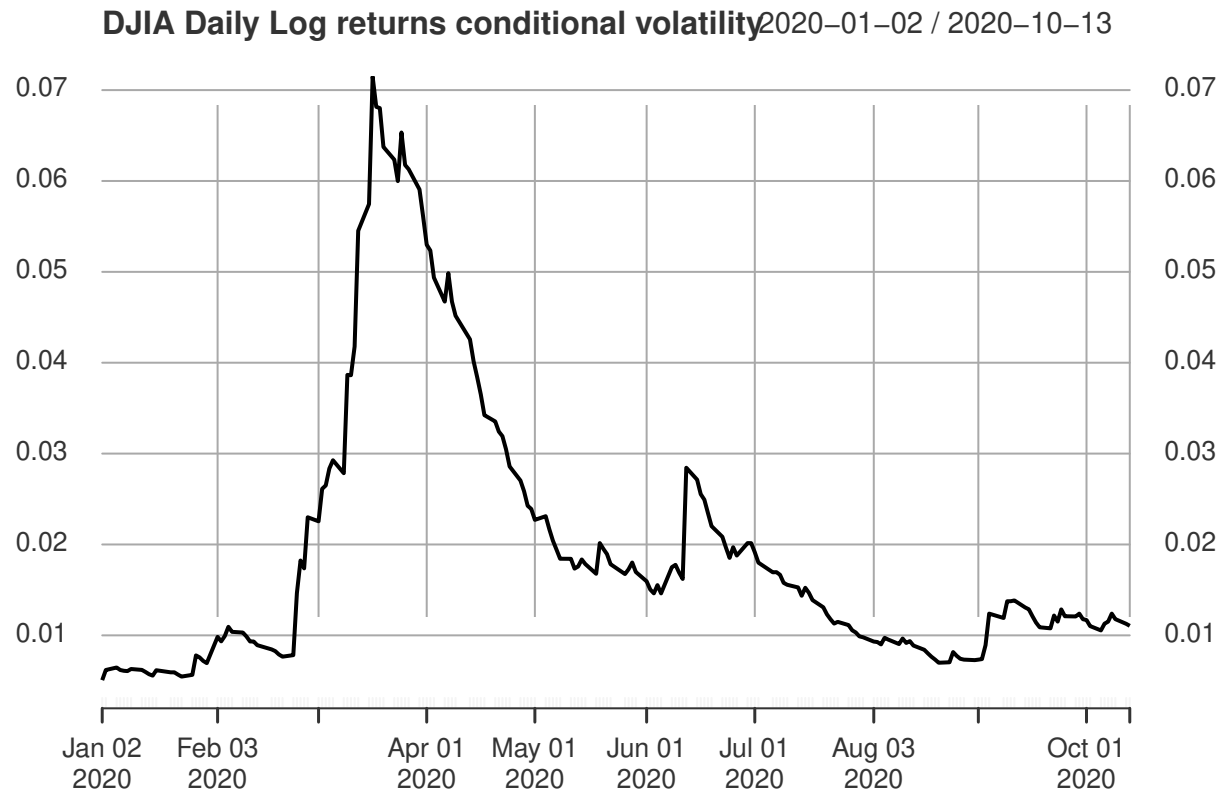
```
##
## [[19]]
```


DJIA Daily Log returns conditional volatility 2018−01−02 / 2018−12−31
0.020
0.015
0.010
0.005
Jan 02 2018
Mar 01 2018
May 01 2018
Jul 02 2018
Sep 04 2018
Nov 01 2018
Dec 31 2018

```
##
## [[20]]
```

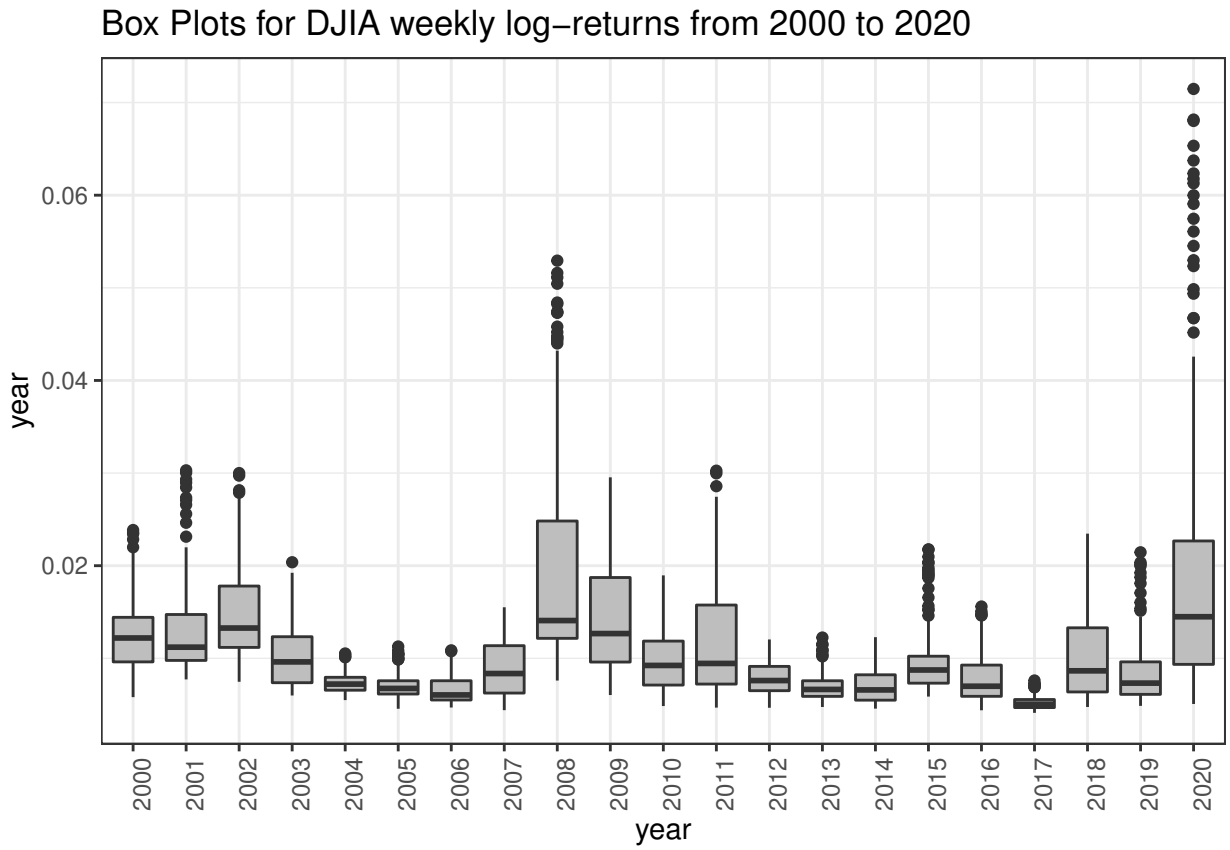

DJIA Daily Log returns conditional volatility 2019−01−02 / 2019−12−31
0.020
0.015
0.010
0.005
Jan 02 2019
Mar 01 2019
May 01 2019
Jul 01 2019
Sep 03 2019
Nov 01 2019
Dec 31 2019

```
##
## [[21]]
```



Box Plots of conditional volatility by year are shown.

```
par(mfrow=c(1,1))

cond_volatility_df <- data.frame(year = factor(year(index(cond_volatility))), value = coredata(cond_vola
colnames(cond_volatility_df) <- c( "year", "value")

boxplot <- ggplot(data = cond_volatility_df, aes(x = year, y = value)) + theme_bw() + theme(legend.posi

boxplot
```


Box Plots for DJIA weekly log−returns from 2000 to 2020
year
0.06
0.04
0.02
2000
2001
2002
2003
2004
2005
2006
2007
2008
2009
2010
2011
2012
2013
2014
2015
2016
2017
2018
2019
2020
year

Feed Forward Neural network

R Markdown

Feed Forward Neural network

Researching deeper and deeper in machine learning fields, we have reached some new neural network models in the forecast package called nnetar. A single hidden layer neural network is the most simple neural networks form. In this kind of single hidden layer form, there is only one layer of input nodes that send weighted inputs to the next layer of receiving nodes. The nnetar function inside the forecast package fits a single hidden layer neural network model to a timeSeries. The approach of this function model is to use lagged values of the time series as input data, reaching to a non-linear autoregressive model.

Load the data

```
#Getting data
suppressMessages(getSymbols("^DJI", from = "2005-01-01", to = "2020-12-05"))
```

```
## [1] "^DJI"
```

```
head(DJI)
```

```
##            DJI.Open DJI.High  DJI.Low DJI.Close DJI.Volume DJI.Adjusted
## 2005-01-03 10783.75 10867.39 10710.07  10729.43  270620000     10729.43
## 2005-01-04 10727.81 10769.56 10605.15  10630.78  293280000     10630.78
## 2005-01-05 10629.53 10684.43 10597.75  10597.83  263550000     10597.83
## 2005-01-06 10593.19 10667.58 10589.33  10622.88  232850000     10622.88
## 2005-01-07 10624.80 10653.25 10571.74  10603.96  283770000     10603.96
## 2005-01-10 10603.44 10663.74 10582.38  10621.03  279500000     10621.03
```

```
tail(DJI)
```

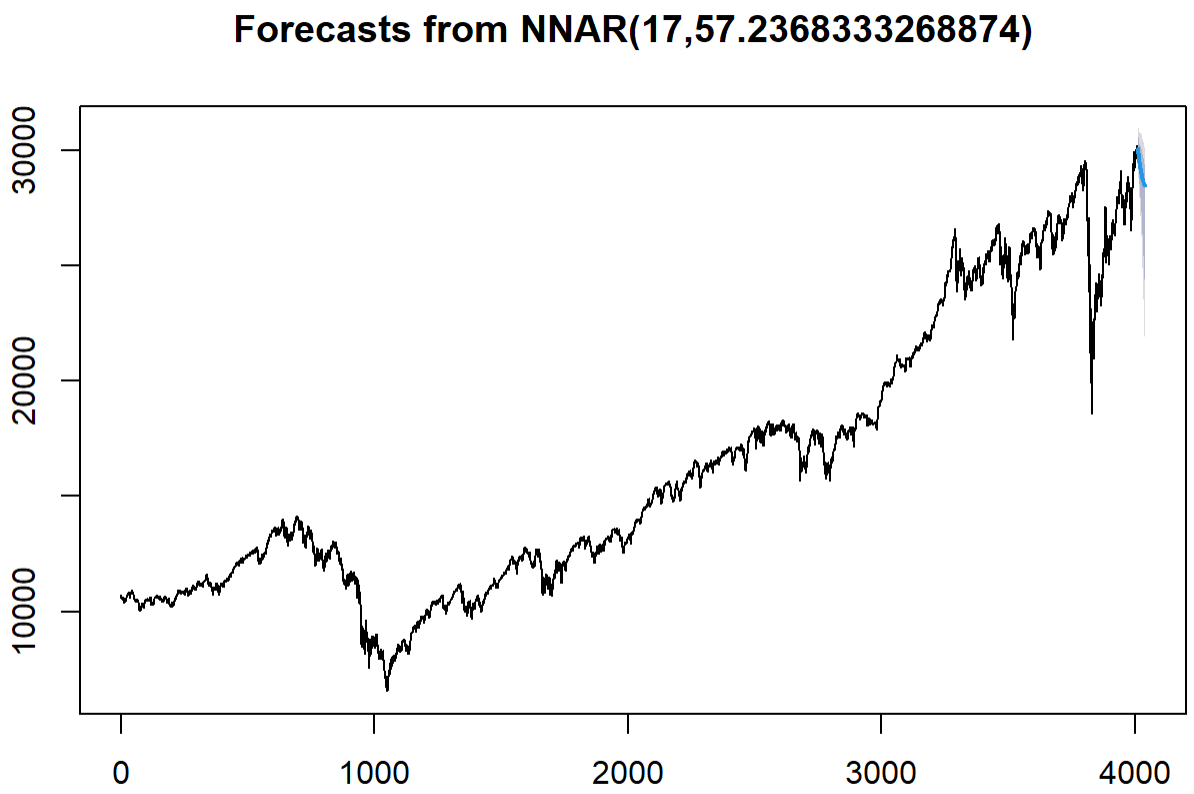
```
##            DJI.Open DJI.High  DJI.Low DJI.Close DJI.Volume DJI.Adjusted
## 2020-11-27 29911.33 30015.13 29819.98  29910.37  177040000     29910.37
## 2020-11-30 29854.51 29854.51 29463.64  29638.64  551350000     29638.64
## 2020-12-01 29797.50 30083.31 29797.50  29823.92  429510000     29823.92
## 2020-12-02 29695.09 29902.51 29599.29  29883.79  385280000     29883.79
## 2020-12-03 29920.83 30110.88 29877.27  29969.52  405680000     29969.52
## 2020-12-04 29989.56 30218.26 29989.56  30218.26  356590000     30218.26
```

We can apply Feed Foward Neural network models to the index or any stock price of our choosing by n(h)=n(s)/(αn(i)+αn(0)), where n(i) stands for number of input neurons; n(0) stands for number of output neurons; n(s) stands for number of train samples and α=1.5^(−10).

```
dji_close <- DJI[,"DJI.Adjusted"]#closed value for dji
#With the hidden layers approach explained we proceed to calculate them:
alpha <- 1.5^(-10)
nh <- length(dji_close)/(alpha*(length(dji_close)+30))
Adjunclose <- DJI[,"DJI.Close"]
AdjunL1 <- Lag(Adjunclose,1)
#print(length(dji_close))
#Now with the hidden layers calculated we proceed to apply the nnetar function with the parameters selecte
d.
lambda <- BoxCox.lambda(dji_close)
dnn_pred <- nnetar(dji_close, size= nh, lambda = lambda,MaxNWts=1200)#Fitting nnetar
neural.price.model<- nnetar(AdjunL1, lambda = lambda)
```

```
## Warning in nnetar(AdjunL1, lambda = lambda): Missing values in x, omitting rows
```

```
#We use a Box Cox lambda to ensure the residuals will be roughly homoscedastic. We forecast the next 30 day
s with the neural net fitted.
dnn_forecast <- forecast(dnn_pred, h= 30, PI = TRUE)
plot(dnn_forecast)#Fitting nnetar
```



#Artificial Neural Network

Then we continue working on Feed Foward Neural network. this time we found a multilayer perception which contains a three-layer with one hidden layer model trained with back-propagation algorithm. According to some research it can be the most effective to forecast financial time series.

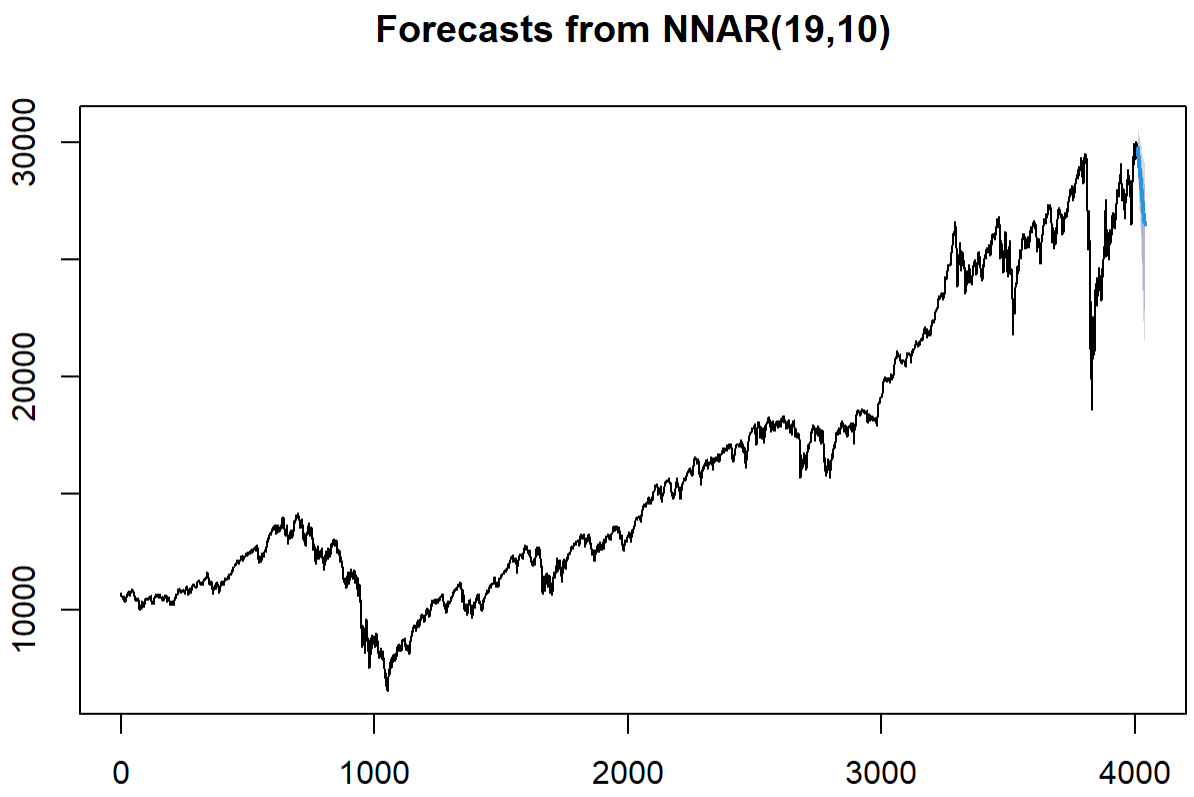
```
CloseL1 <- Lag(dji_close,1)# Preparing input data for ANN
#print(CloseL1[4000:4010])
#CloseL1 <- na.omit(CloseL1)
length <- length(dji_close)
CloseL1.train <- CloseL1[1:(length-30)]# Delimit training range
CloseL1.test <- CloseL1[(length-29):length]# Delimit testing range
```

The training parameters were set as follows: decay rate = 0.00001, number of units in the hidden layer = 10, and epoch size = 10000.

```
#N Net Function for Close Price
set.seed(1)
neural.price.mode1 <- nnet(dji_close ~ Adjunclose, data = CloseL1.train,
                           size = 100, decay = 0.00001, linout = 1, skip=TRUE, MaxNWts=2600, trace=FALSE)
#Predict Close Price on test data
ann_forecast <- forecast(neural.price.model, h= 30, PI = TRUE)
```

Finally, we plot out the line chart to compare them.

```
plot(ann_forecast)
```




```
plot(CloseL1)
```


```
#calculate RMS error
#Rmse <- sqrt(mean((CloseL1.test-CloseL1$Lag.1)^2))
#Rmse
```

knn.R

ashah

2020-12-16

```
# KNN model can be used for both classification and regression problems. The
most popular application is to use it for classification problems.
# Here we use the tsfknn package with h = 30 and get a forecast 30 days in
the future of DJI.
# The accuracy is shown and plots show the forecast.
#The first thing we do is read the data and create the time series. Our main
function is knn_forecasting which returns a knn forecast object.
#This object is then passed to rolling origin to gives us the accuracy. KNN
algorithm uses 'feature similarity' to predict the values of any new data
points.
# This means that the new point is assigned a value based on how closely it
resembles the points in the training set. The k parameter determines the
number of k closest features vectors which are called k nearest neighbors.
# Targets are the time-series data that come right after the nearest
neighbors and their number is the value of the h parameter.
# The lag parameter is used to build an instance which is in turn used to
find  the closest vectors using the Eucledian formula.
if(!require(tsfknn)) install.packages("tsfknn")
```

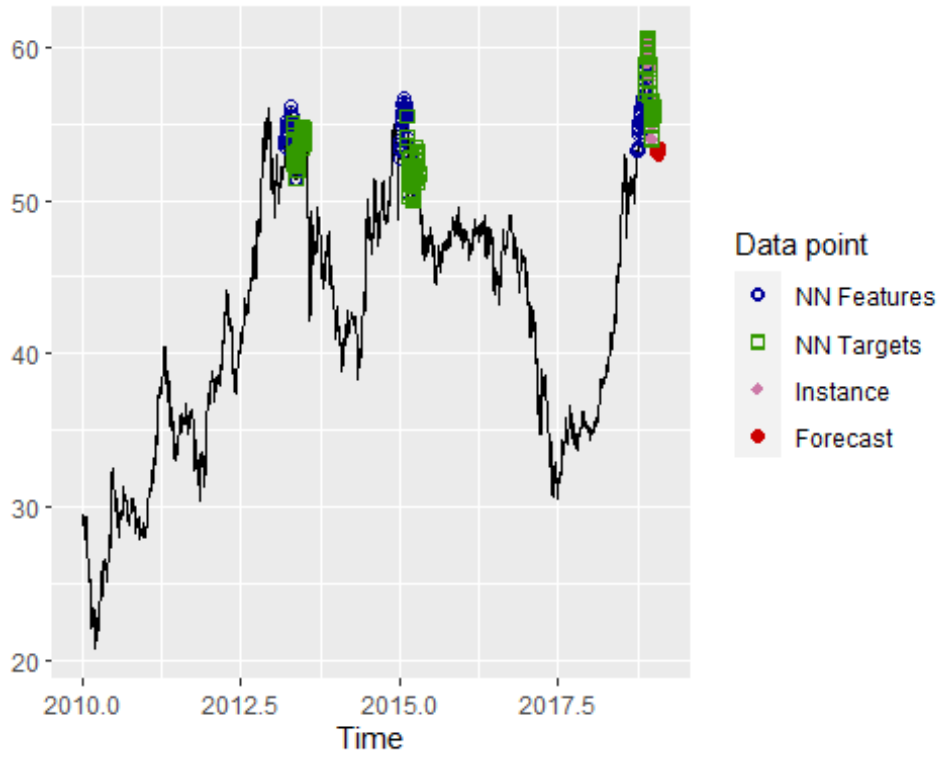
```
## Loading required package: tsfknn
```

```
library(tsfknn)
library(ggplot2)
```

```
# Reading the data
dji_d<-read.csv("C:\\Users\\ashah\\Documents\\dji_d.csv")

# creating time series of opening price for DJ1
dji_d.ts <-ts(dji_d$Open, start=2010, end= 2019, frequency= 250)

# KNN forecast -
predknn <- knn_forecasting(dji_d.ts, lags = 1 : 30, h = 30, k = 40, msas =
"MIMO")
ro <- rolling_origin(predknn)

# showing the accuracy
print(ro$global_accu)
```

```
##     RMSE      MAE     MAPE
## 4.416086 3.679488 6.408087
```

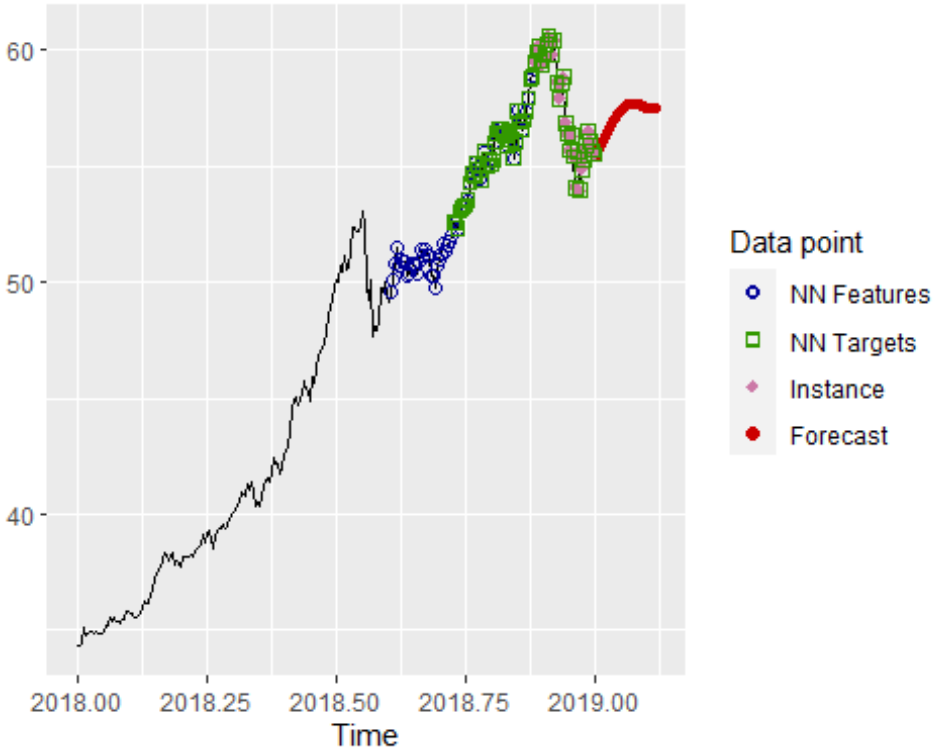
```r
#plotting the  graph
autoplot(predknn, highlight = "neighbors", faceting = FALSE)
```




```r
#Create a second time series with a small time gap for a better visualization
timeS <- window(dji_d.ts, start = 2018, end = 2019)
pred <- knn_forecasting(timeS, lags = 1 : 30, h = 30, k = 40, msas = "MIMO")
ro2 <- rolling_origin(pred)

# showing accuracy for 1 year time period and forecast of 30 days
print(ro2$global_accu)
```

```
##     RMSE      MAE     MAPE
## 3.531710 2.610383 4.503961
```

```r
# Plot second graph
autoplot(pred, highlight = "neighbors", faceting = FALSE)
```




```
# conclusion - The second graph starts in 2018 and  goes till the end of the
year. From our second  graph we see that it forecasts a clear upwards trend
which starts smoothing out at the end.
# This is what was observed in the stock market at the beginning of 2019 in
the first 30 days more or less. In the first graph the forecast is similar
over a larger time scale. Based on this observation and the accuracy values,
# we can conclude Knn forecasting gives a decent forecast.
```

Conclusion:

The Dow Jones Index is very difficult to predict because so many different outside events factor into the index. Because of this unpredictability of factors no one model is better than the other. It may appear in our report that one model provided better predictions than other but with different market conditions this may not be the case.

For example, there was a huge dip in the market in early 2020 due to covid lockdowns. No such model would be able to predict something like this without being given that information.

The Prophet model can give a direction to the market trend but cannot exactly predict the prize of a particular share as, the share price can be impacted by many events such as news, economic scenario, insider updates etc. We use two different stocks to predict the momentum and in both we found that prophet is successfully able to predict the momentum but fails to predict the share price. Also, the more volatile the stock is, more deviating results or get.

The Arima model is a very powerful tool to make stationary predicitons. Unfortuantly the Dow Jones Index is non-stationary, so it makes the Arima prediction not totally accurate. The Arima model provides multiple ways to analyize its predictions and this shows the upper, lower and mean predictions of our index. With so many outside factors affecting the index our prediction is most likely not accurate.

In the Garch Model we saw that the conditional volatility changes within and between years. This shows that the volatility is an indicator of the amplitude of variations and is a basic measure of risk when applied to log returns of an asset. The conditional volatility at time t is the volatility of a random variable given the knowledge of historical data up to time t-1. To compute the conditional volatility, we have to build a GARCH model. Volatility has a wide range of applications in finance, options pricing.

In the Feed Forward Neural Network we can see the predicted and actual trend have vast difference. Maybe because we have enough train set, or we didn't consider the other fact of market trend like rumors, economic scenario, insider updates etc. Hence that we get deviating results.