# Baseball Forecasts

Joe Jenkins and Anthony Stachowski

11/11/2020

```r
library(dplyr)
```

```
## Warning: package 'dplyr' was built under R version 3.6.2

##
## Attaching package: 'dplyr'

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

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

```r
library(fitdistrplus)
```

```
## Loading required package: MASS

##
## Attaching package: 'MASS'

## The following object is masked from 'package:dplyr':
##
##     select

## Loading required package: survival

## Warning: package 'survival' was built under R version 3.6.2

## Loading required package: npsurv

## Loading required package: lsei
```

```r
library(ggplot2)
```

```
## Warning: package 'ggplot2' was built under R version 3.6.2
```

```r
library(fBasics)
```

```
## Loading required package: timeDate

## Loading required package: timeSeries
```

```r
library(fpp)
```

```
## Loading required package: forecast

## Registered S3 method overwritten by 'xts':
##   method     from
##   as.zoo.xts zoo

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

## Registered S3 methods overwritten by 'forecast':
##   method             from
##   fitted.fracdiff    fracdiff
##   residuals.fracdiff fracdiff

## Loading required package: fma

##
## Attaching package: 'fma'

## The following objects are masked from 'package:MASS':
##
##     cement, housing, petrol

## Loading required package: expsmooth

## Loading required package: lmtest

## Loading required package: zoo

##
## Attaching package: 'zoo'

## The following object is masked from 'package:timeSeries':
##
##     time<-

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

## Loading required package: tseries
```

```r
library(forecast)

# Importing data that was scraped from baseball-reference.com:
data <- read.csv("Data/BOS_NYY_2009_2019.csv", stringsAsFactors = FALSE)

# Creating an index that is a combination of the season (Year) and the game number (Rk):
data$Game_Year <- paste(data$Rk, data$Year, sep = "_")
data$yearGame <- as.integer(paste(as.character(data$Year),
                                  ifelse(data$Rk < 10,
                                         paste("00", as.character(data$Rk), sep = ""),
                                         ifelse(data$Rk >= 10 & data$Rk < 100,
                                                paste("0", as.character(data$Rk), sep = ""),
                                                as.character(data$Rk))), sep = ""))
```

### Average

Computing the average percentage of strikes thrown by game by assessing both NY and Bos:

```r
avgData <- data %>%
  group_by(yearGame, Year) %>%
  summarise(mean_PerSK = mean(PerSK))
```

```
## `summarise()` regrouping output by 'yearGame' (override with `.groups` argument)
```

Creating a time series from our data Also, separating data into testing set (2019 season) and training set (2009-2018 seasons)

Testing set:

```r
avgData2019 <- avgData %>%
  dplyr::filter(Year == 2019)
```

Testing set time series:

```r
avgIndex2019 = ts(avgData2019$mean_PerSK,start=c(2019,1), frequency = 162)
```

Training set:

```r
avgData2009_2018 <- avgData %>%
  dplyr::filter(Year < 2019)
```

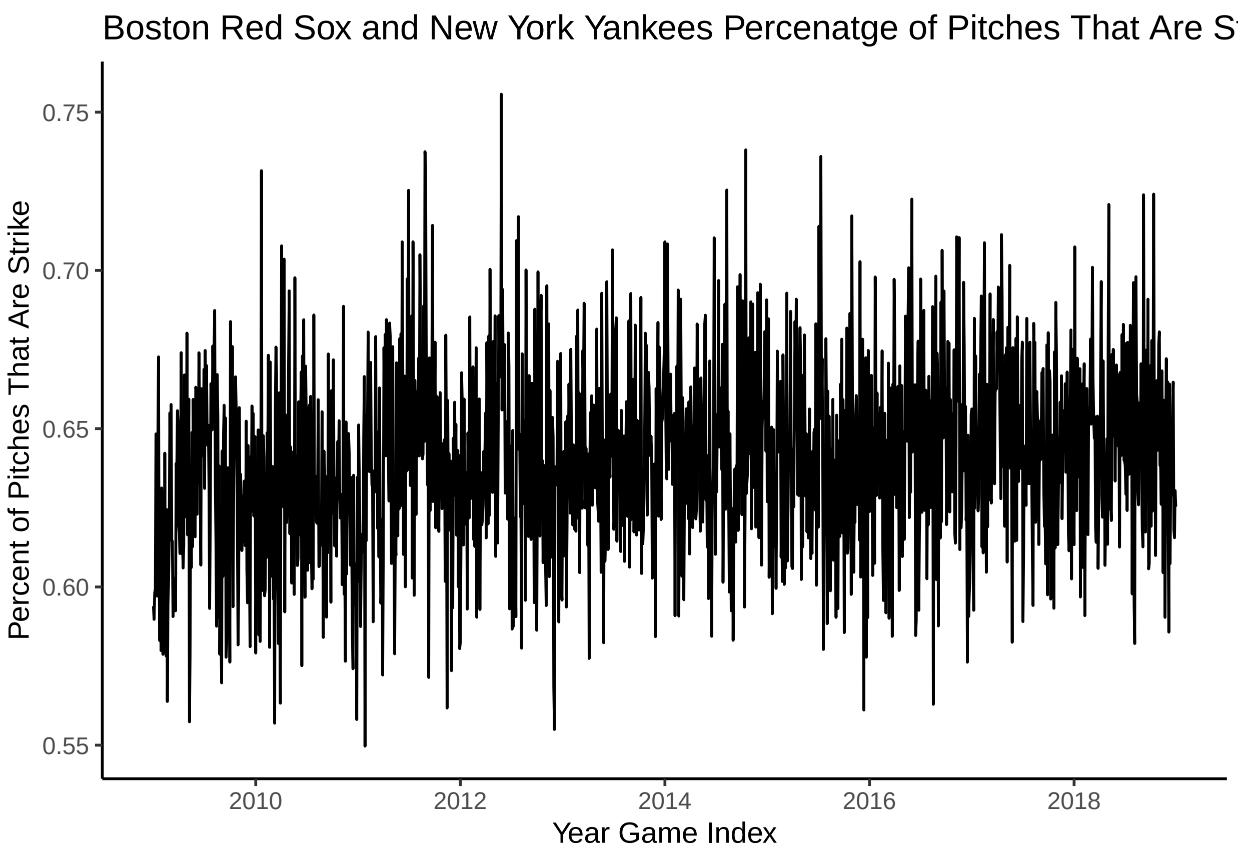
Training set time series:

```r
avgIndex2009_2018 = ts(avgData2009_2018$mean_PerSK,start=c(2009,1), frequency = 162)
```

### Time Plot

Time plot of training set:

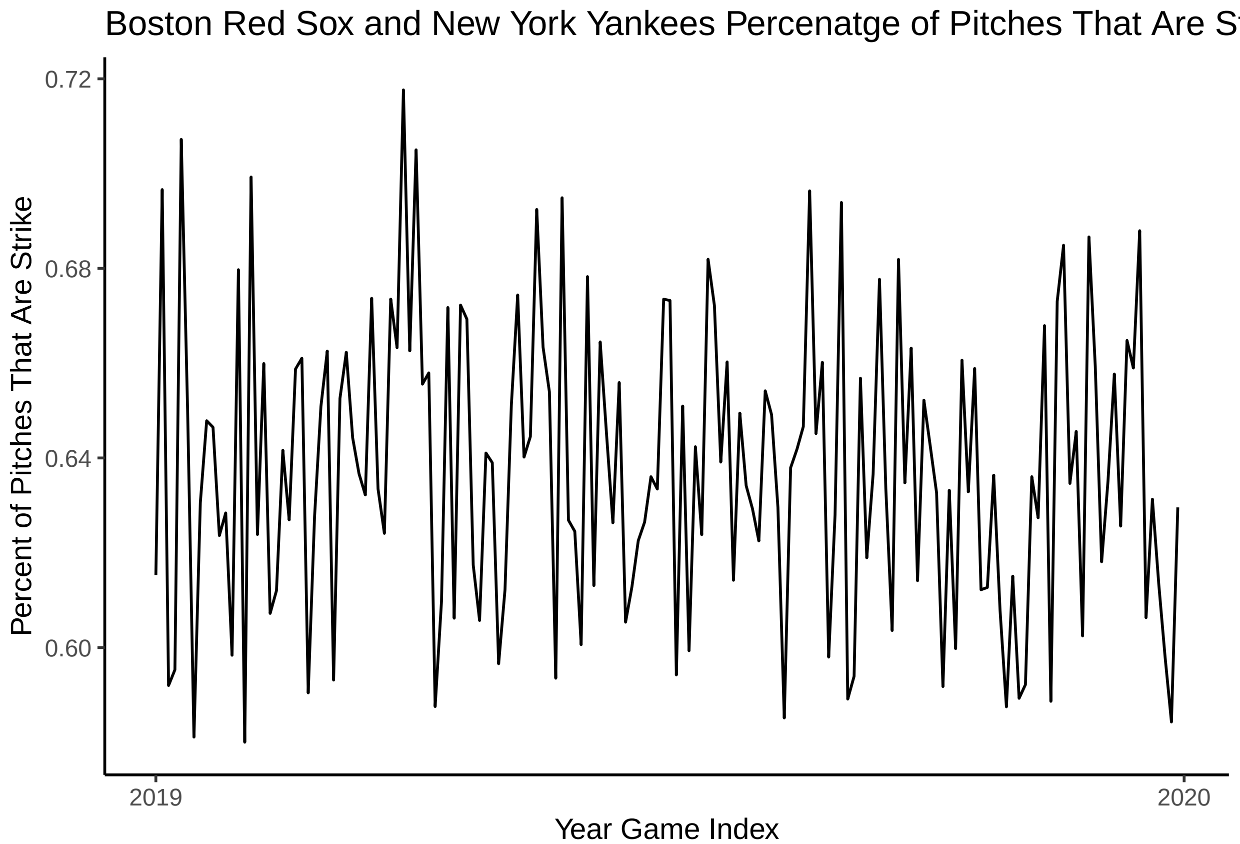
```r
autoplot(avgIndex2009_2018) + xlab("Year Game Index") +
  ylab("Percent of Pitches That Are Strike") + ggtitle("Boston Red Sox and New York Yankees Percenatge o
  theme_classic()
```




There appears to be quite a bit of random fluctuation within our time series. There may be a slight trend upwards over our training set, but seems very minimal. No obvious seasonality or cycles based on visual assessment. Our average percentages vary between 55% and 75%.

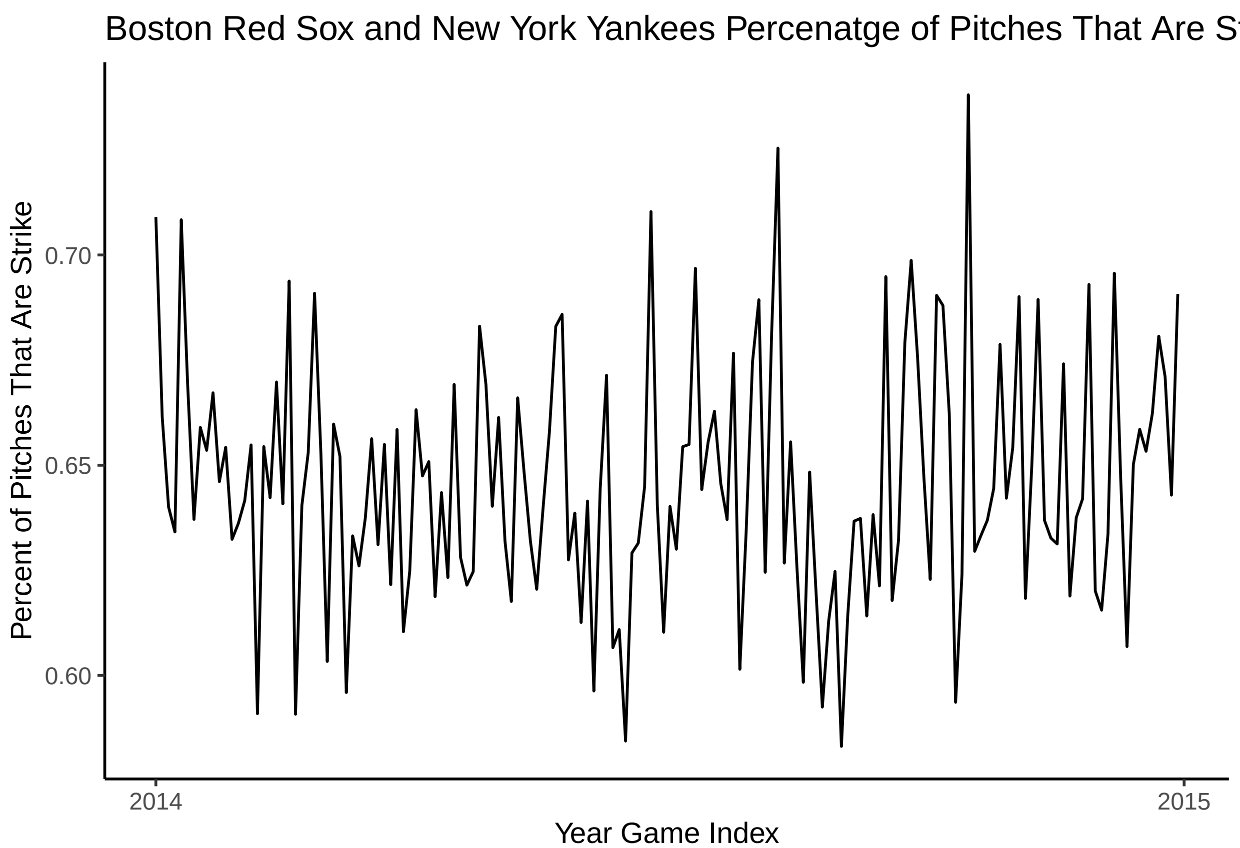
Time plot of testing set:

```r
autoplot(avgIndex2019) + xlab("Year Game Index") +
  ylab("Percent of Pitches That Are Strike") + ggtitle("Boston Red Sox and New York Yankees Percenatge o
  theme_classic()
```




Time plot of training set, season 2014 for closer view of individual year:

```
avgData2014 <- avgData %>%
  dplyr::filter(Year == 2014)
avgIndex2014 = ts(avgData2014$mean_PerSK,start=c(2014,1), frequency = 162)
autoplot(avgIndex2014) + xlab("Year Game Index") +
  ylab("Percent of Pitches That Are Strike") + ggtitle("Boston Red Sox and New York Yankees Percenatge o
  theme_classic()
```




#### Check Distribution Aspects

```r
# Examining Basic Statistics of our training set and testing set:
basicStats(avgIndex2019)
```

```
##             avgIndex2019
## nobs          162.000000
## NAs             0.000000
## Minimum         0.580023
## Maximum         0.717657
## 1. Quartile     0.613594
## 3. Quartile     0.659761
## Mean            0.637638
## Median          0.636047
## Sum           103.297359
## SE Mean         0.002412
## LCL Mean        0.632874
## UCL Mean        0.642402
## Variance        0.000943
## Stdev           0.030705
## Skewness        0.177281
## Kurtosis       -0.625211
```

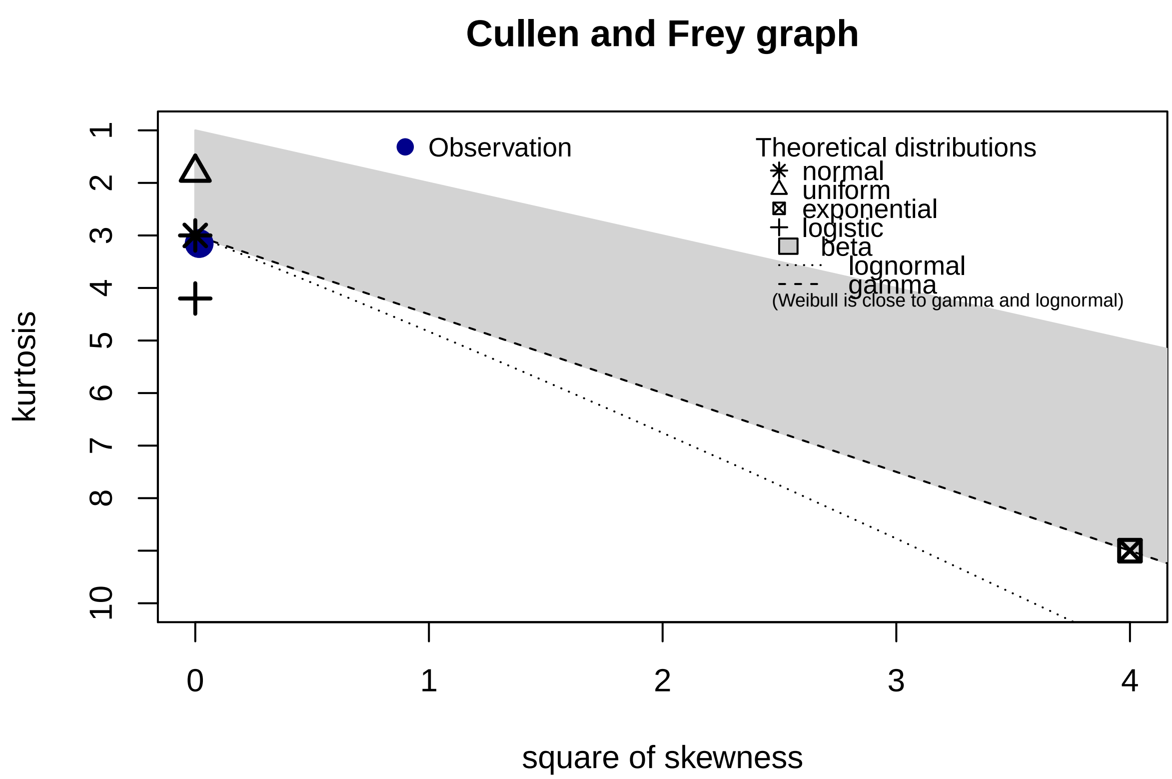
```r
basicStats(avgIndex2009_2018)
```

```
##             avgIndex2009_2018
## nobs              1620.000000
## NAs                  0.000000
## Minimum              0.549695
## Maximum              0.755686
## 1. Quartile          0.620429
## 3. Quartile          0.659326
## Mean                 0.640046
## Median               0.639623
## Sum               1036.873742
## SE Mean              0.000739
## LCL Mean             0.638596
## UCL Mean             0.641495
## Variance             0.000885
## Stdev                0.029743
## Skewness             0.128970
## Kurtosis             0.146038
```

Examining the nature of our distribution and checking for normality:

Training Set:

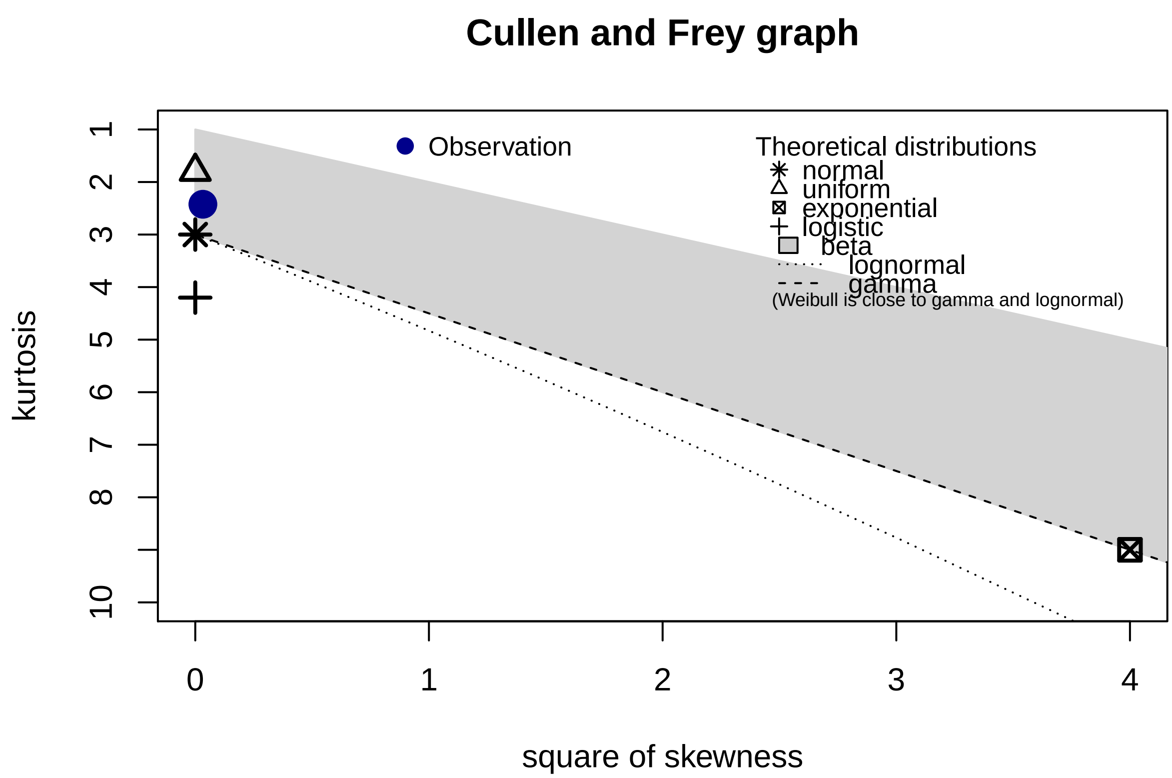
```r
descdist(as.numeric(avgIndex2009_2018), discrete = FALSE)
```



```
## summary statistics
## ------
## min:  0.5496951   max:  0.755686
## median:  0.6396226
## mean:  0.6400455
## estimated sd:  0.02974314
## estimated skewness:  0.1292089
## estimated kurtosis:  3.154103
```

Testing Set:

```r
descdist(as.numeric(avgIndex2019), discrete = FALSE)
```




```
## summary statistics
## ------
## min:  0.5800226   max:  0.7176573
## median:  0.636047
## mean:  0.637638
## estimated sd:  0.03070498
## estimated skewness:  0.1806119
## estimated kurtosis:  2.423552
```

Computing normal test using Jarque-Bera test, here we are examining if p-value is greater than alpha. We will compare the p-value against alpha values of 0.01 and 0.05. Training set:

```r
fBasics::normalTest(avgIndex2009_2018, method = 'jb')
```

```
##
## Title:
##  Jarque - Bera Normalality Test
##
## Test Results:
##   STATISTIC:
##     X-squared: 6.0165
##   P VALUE:
##     Asymptotic p Value: 0.04938
##
## Description:
##  Mon Nov 16 22:04:00 2020 by user:
```

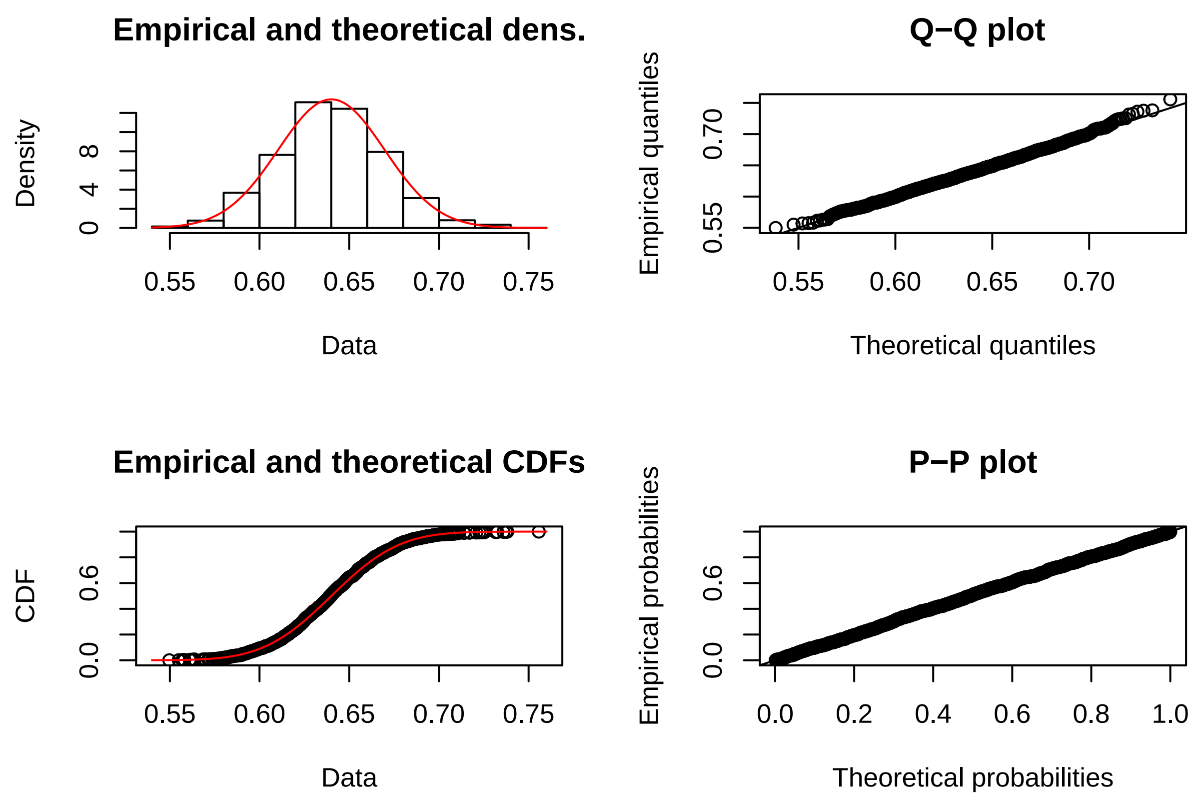
Testing set:

```r
fBasics::normalTest(avgIndex2019, method = "jb")
```

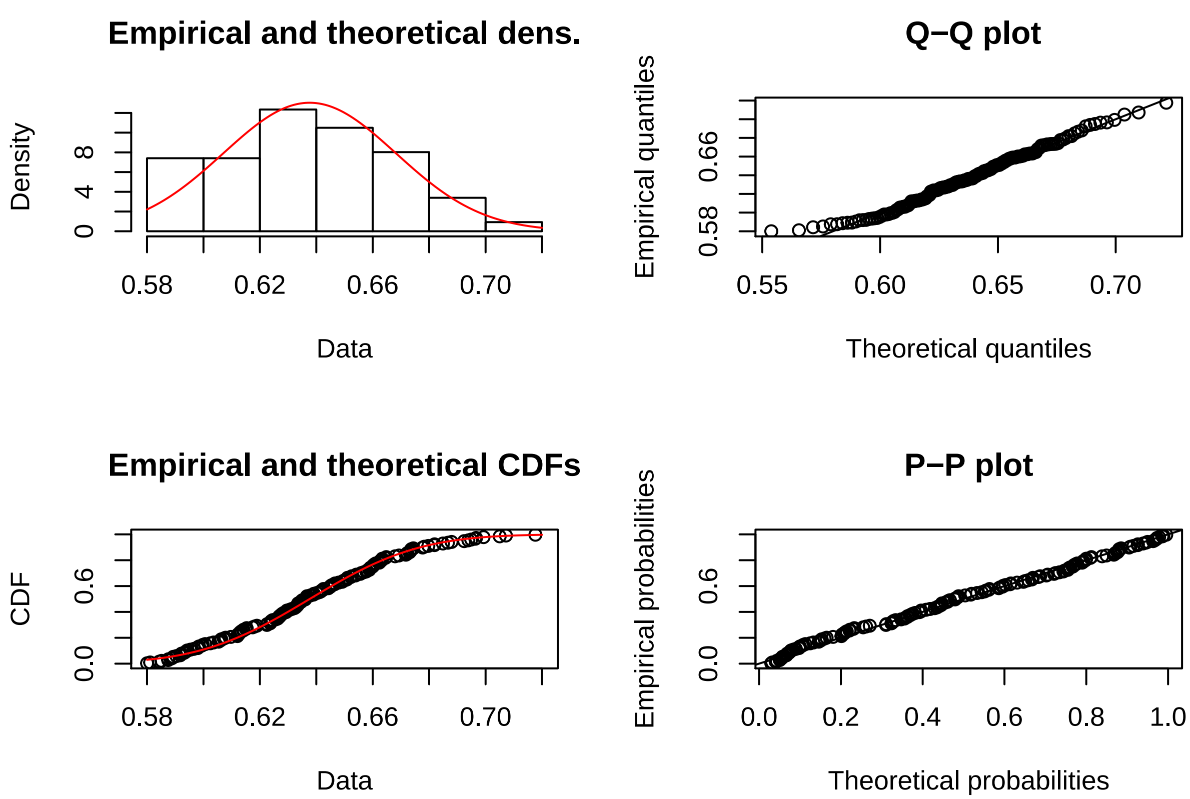
Conclusion: our training set passes the J-B Test at an alpha of 0.01, but our testing set does not.

Displaying normal plots for our data:

```r
plot(fitdist(as.numeric(avgIndex2009_2018), "norm"))
```



```r
plot(fitdist(as.numeric(avgIndex2019), "norm"))
```




Conclusion: our training set appears to align with a normal distribution when examining normality plots. Again the testing set does not, but this is not as problematic as we will be building the models using only the training set.

Overall we conclude that our training set is normally distributed.

Checking mean of our testing set to see whether it is different from zero:

```r
t.test(avgIndex2009_2018)
```

```
##
##  One Sample t-test
##
## data:  avgIndex2009_2018
## t = 866.13, df = 1619, p-value < 2.2e-16
## alternative hypothesis: true mean is not equal to 0
## 95 percent confidence interval:
##  0.6385961 0.6414950
## sample estimates:
## mean of x
## 0.6400455
```

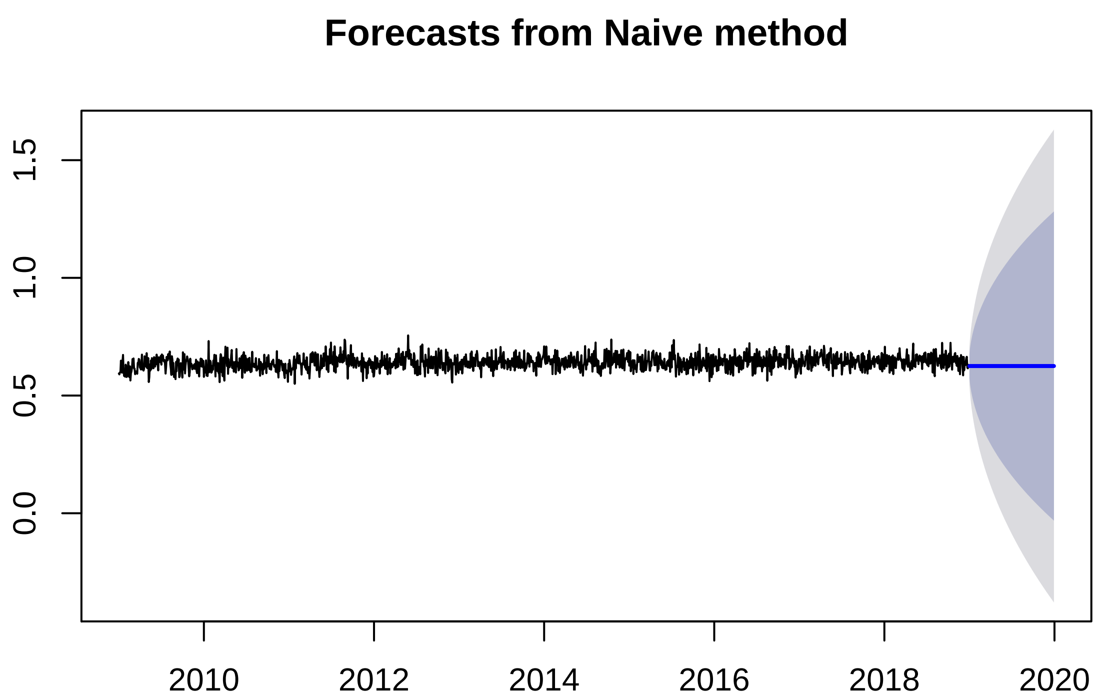
Resulting p-value is very small and therefore we can conclude that our mean value is different from zero.

**Forecasting Models:**

**Naive**

Uses most recent point as data for forecast values:

```
averageModelNaive <- naive(avgIndex2009_2018, h = 162)
plot(averageModelNaive)
```




Assessing the accuracy of this model versus the actual data:

```
naive_acc <- accuracy(averageModelNaive, avgIndex2019)
naive_acc
```

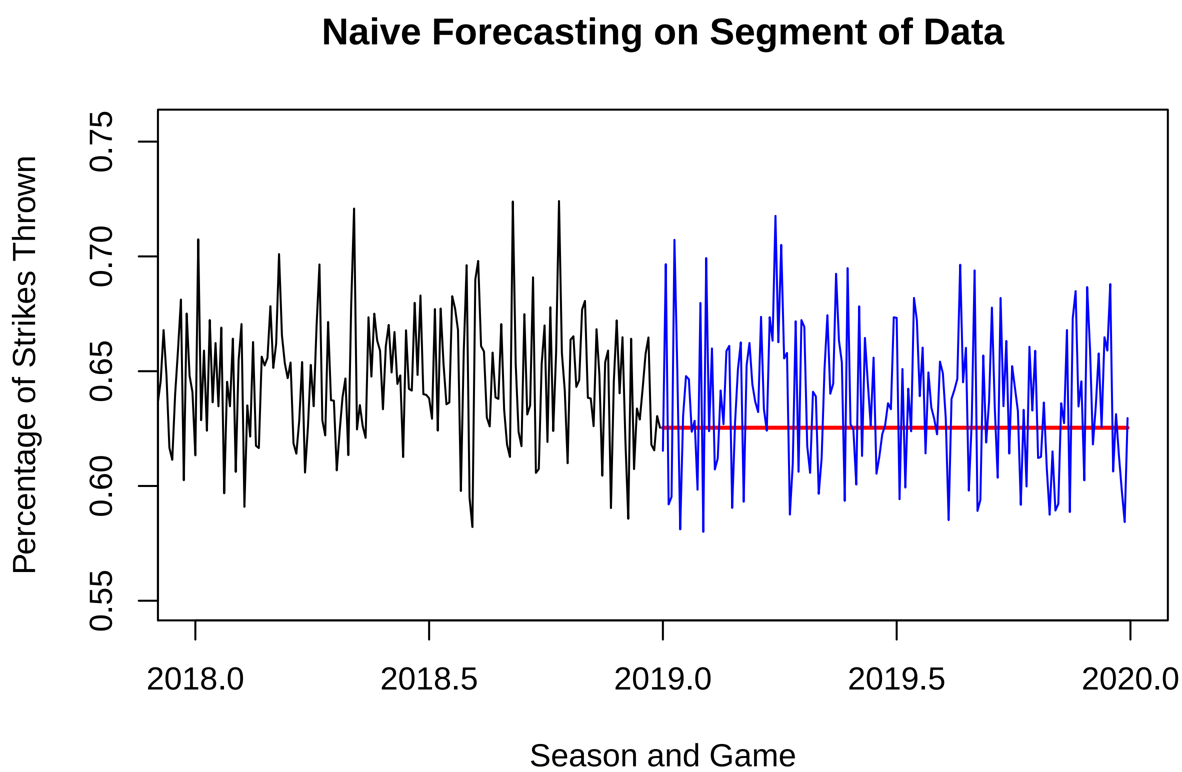
```
##                        ME       RMSE        MAE        MPE     MAPE      MASE
## Training set 1.943324e-05 0.04026719 0.03218030 -0.1943073 5.029071 1.0043165
## Test set     1.227252e-02 0.03297864 0.02677076  1.6993852 4.130311 0.8354899
##                     ACF1 Theil's U
## Training set -0.50808919        NA
## Test set     -0.07056665 0.7397499
```

Creating list of error values for assessing all models:

```
naive_errors_test <- naive_acc[2,]
```

Plotting portion of data for easier interpretation:

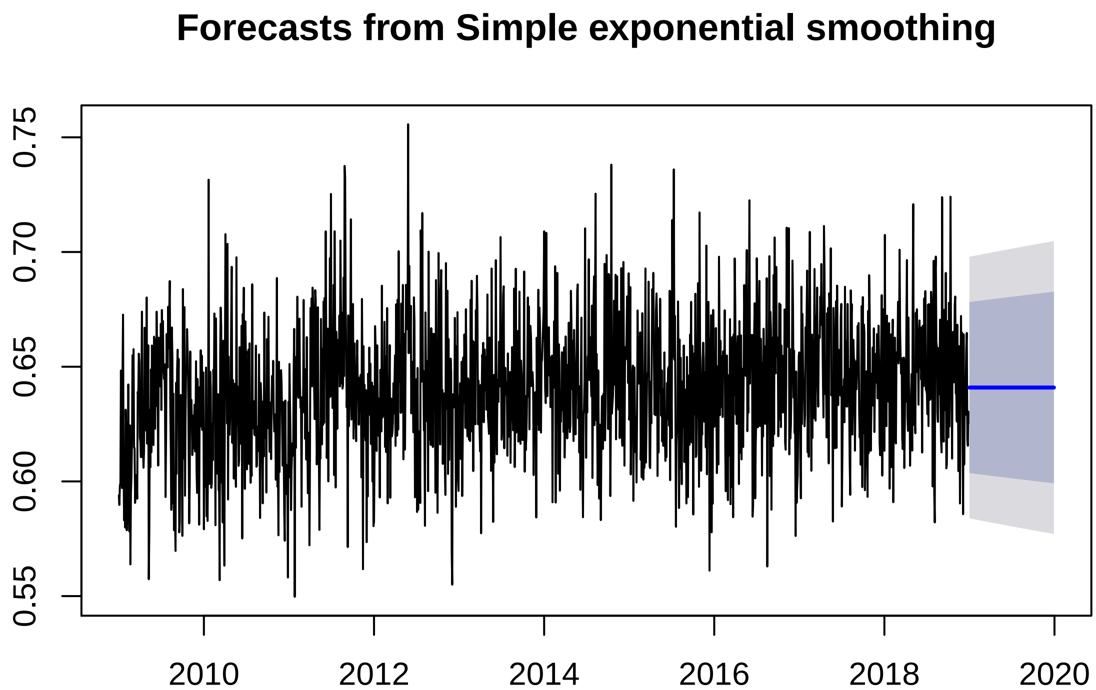
```
plot(avgIndex2009_2018,
     xlim = c(2018.0, 2020.0),
     xlab = "Season and Game",
     ylab = "Percentage of Strikes Thrown",
     main = "Naive Forecasting on Segment of Data")
lines(naive(avgIndex2009_2018, h=162)$mean, col="red", lwd=2)
lines(avgIndex2019, col="blue")
```




**SES**

Using weighted averages of past values to create a flat forecast:

```
averageModelSES <- ses(avgIndex2009_2018, h = 162)
plot(averageModelSES)
```




Assessing the accuracy of this model versus the actual data:

```
ses_acc <- accuracy(averageModelSES, avgIndex2019)
ses_acc
```

```
##                         ME       RMSE        MAE        MPE     MAPE      MASE
## Training set  0.0004086347 0.02906774 0.02303544 -0.1367957 3.603191 0.7189143
## Test set     -0.0032830463 0.03078562 0.02535308 -0.7457803 4.001794 0.7912455
##                      ACF1 Theil's U
## Training set  0.001780073        NA
## Test set     -0.070566653 0.6873529
```

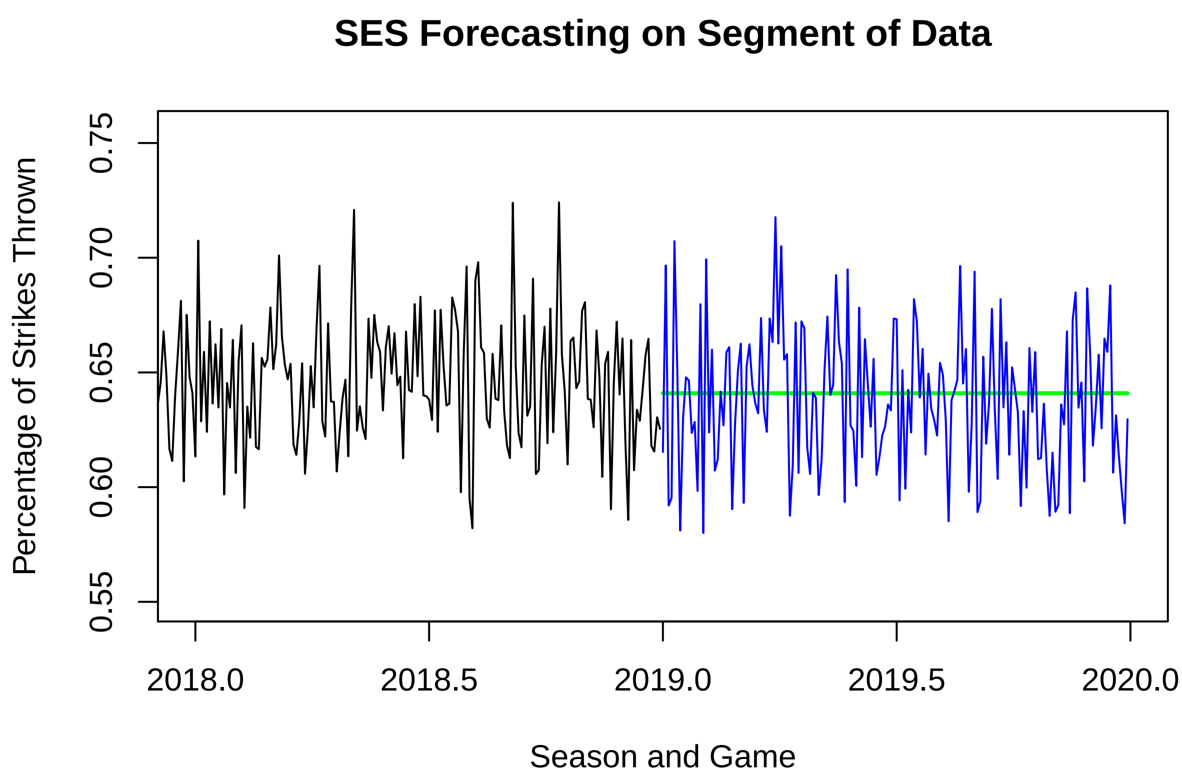
Creating list of error values for assessing all models:

```
ses_errors_test <- ses_acc[2,]
```

## Plotting a portion of the data for easier interpretation:

```
plot(avgIndex2009_2018,
     xlim = c(2018.0, 2020.0),
     xlab = "Season and Game",
     ylab = "Percentage of Strikes Thrown",
```

```r
     main = "SES Forecasting on Segment of Data")
lines(ses(avgIndex2009_2018, h=162)$mean, col="green", lwd=2)
lines(avgIndex2019, col="blue")
```




**ETS**

Using ETS model:

```r
averageETS <- ets(avgIndex2009_2018)
```

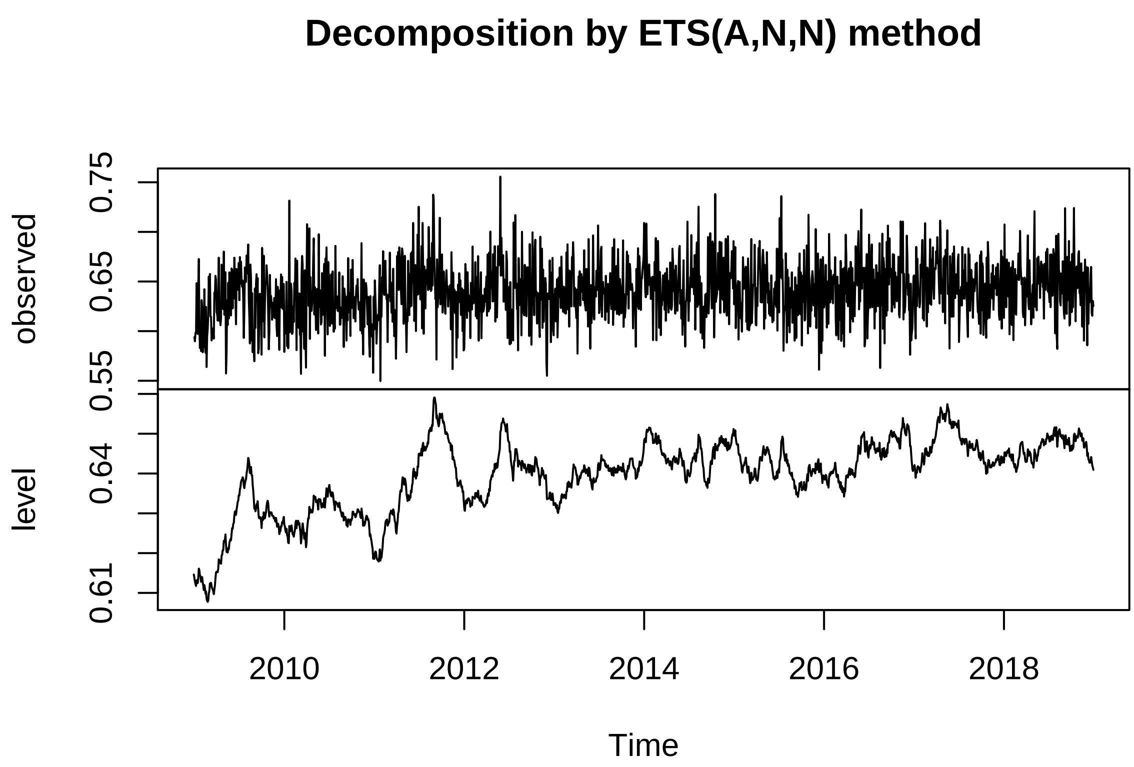
```
## Warning in ets(avgIndex2009_2018): I can't handle data with frequency greater
## than 24. Seasonality will be ignored. Try stlf() if you need seasonal forecasts.
```

```r
summary(averageETS)
```

```
## ETS(A,N,N)
##
## Call:
##  ets(y = avgIndex2009_2018)
##
##   Smoothing parameters:
##     alpha = 0.0397
##
```

```
##   Initial states:
##     l = 0.6146
##
##   sigma:  0.0291
##
##      AIC     AICc      BIC
## 514.5648 514.5797 530.7353
##
## Training set error measures:
##                        ME       RMSE        MAE        MPE     MAPE      MASE
## Training set 0.0004090071 0.02906774 0.02303542 -0.1367379 3.603185 0.7189136
##                     ACF1
## Training set 0.001798721
```

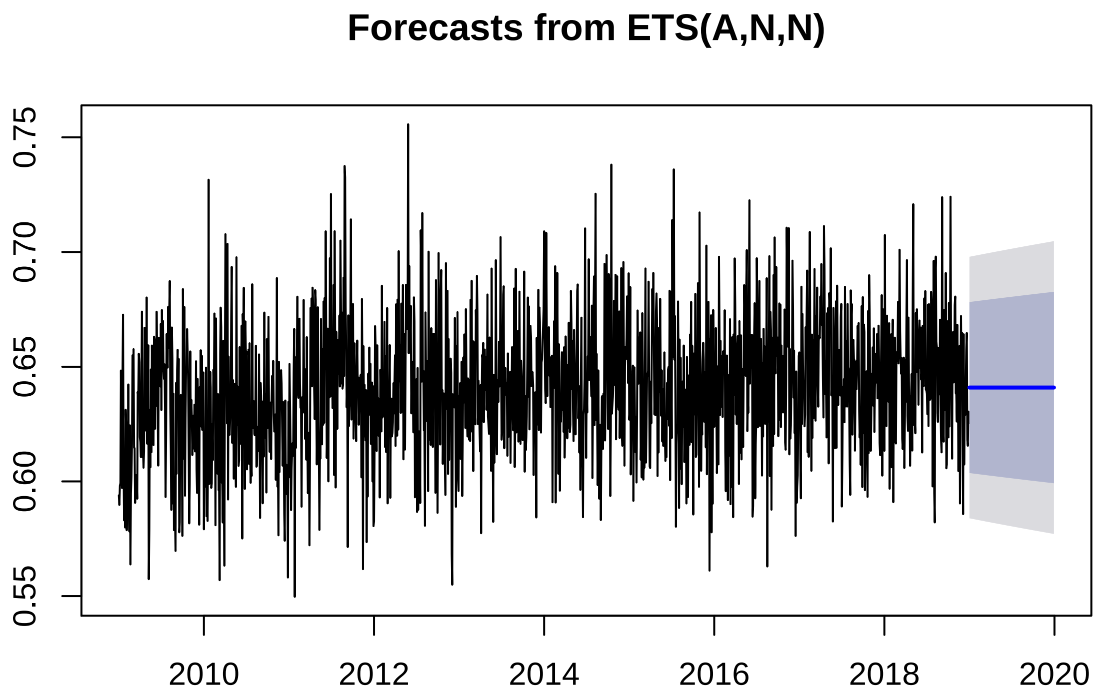
```
plot(averageETS)
```




Output from model is ETS(A,N,N), this has additive error, no trend and no seasonality. The ETS plot seems to indicate that there may be a slight trend, but this was not strong enough that the model felt the need to account for the trend.

Assessing the accuracy of the ETS Model:

```
ETSForecast <- forecast(averageETS, h = 162)
plot(ETSForecast)
```




```
ETS_acc <- accuracy(ETSForecast, avgData2019$mean_PerSK)
ETS_acc
```

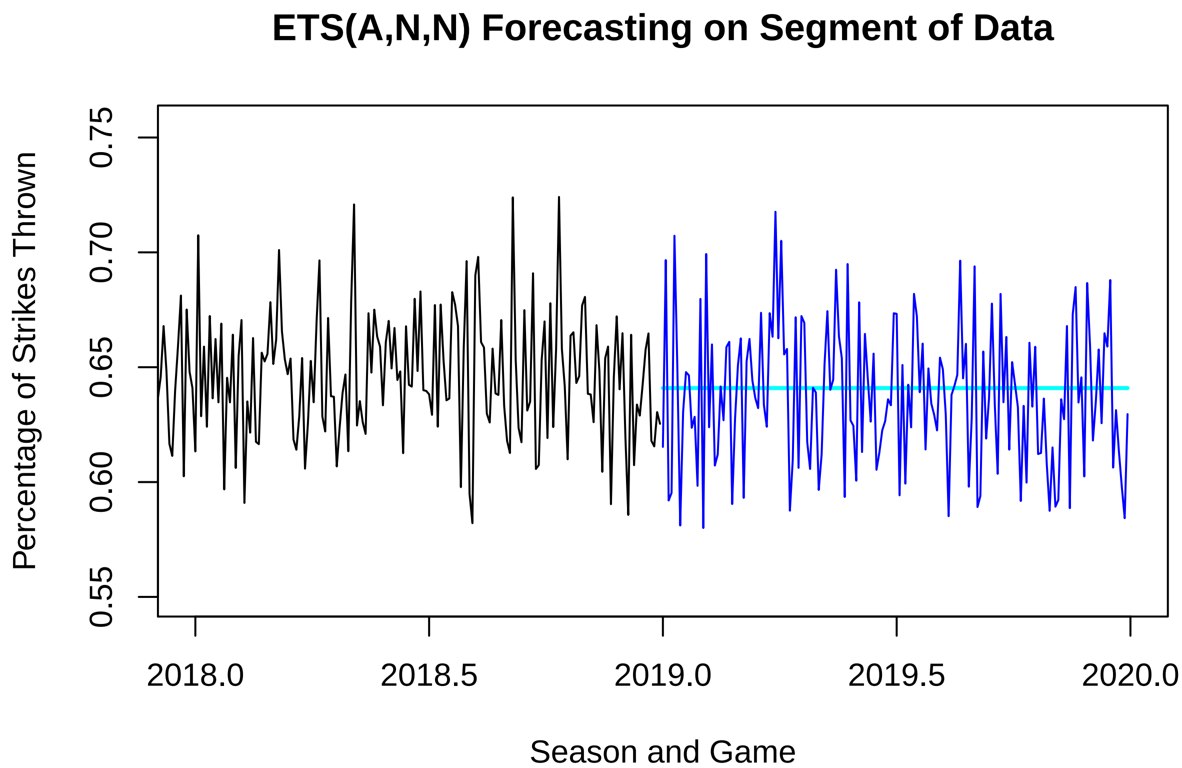
```
##                         ME       RMSE        MAE        MPE     MAPE      MASE
## Training set  0.0004090071 0.02906774 0.02303542 -0.1367379 3.603185 0.7158238
## Test set     -0.0032857070 0.03078590 0.02535334 -0.7461986 4.001852 0.7878529
##                     ACF1
## Training set 0.001798721
## Test set              NA
```

Creating list of error values for assessing all models:

```
ets_errors_test <- ETS_acc[2,]
```

Plotting a portion of the data for easier interpretation:

```
plot(avgIndex2009_2018,
     xlim = c(2018.0, 2020.0),
     xlab = "Season and Game",
     ylab = "Percentage of Strikes Thrown",
     main = "ETS(A,N,N) Forecasting on Segment of Data")
lines(ETSForecast$mean, col="cyan", lwd=2)
lines(avgIndex2019, col="blue")
```




**Pre-Processing for ARIMA:**

### Box-Cox Transformation

We will assess the Box Cox lambda to see if our data needs to be transformed:

```
BoxCox.lambda(avgIndex2009_2018)
```

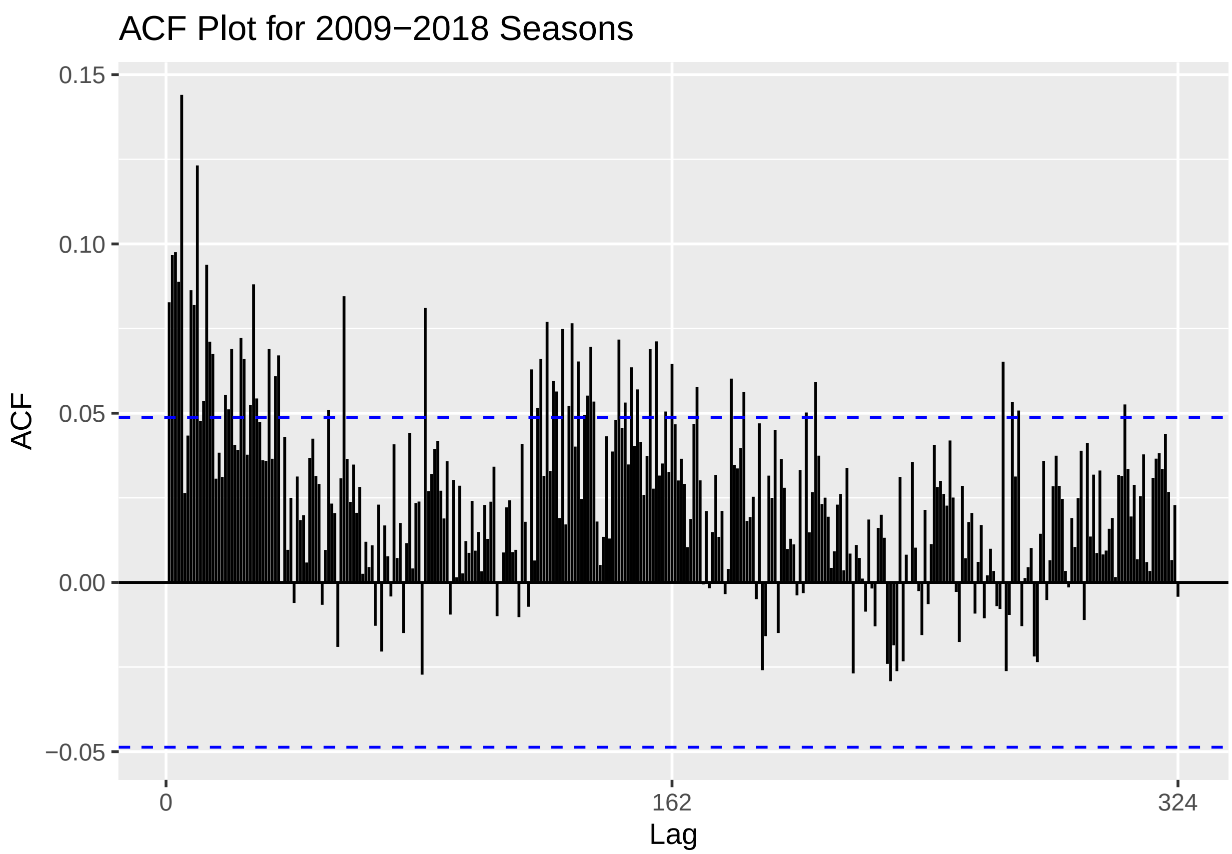
```
## [1] 1.999924
```

This is indicating a lambda greater than 1 and therefore we will not transform our data.

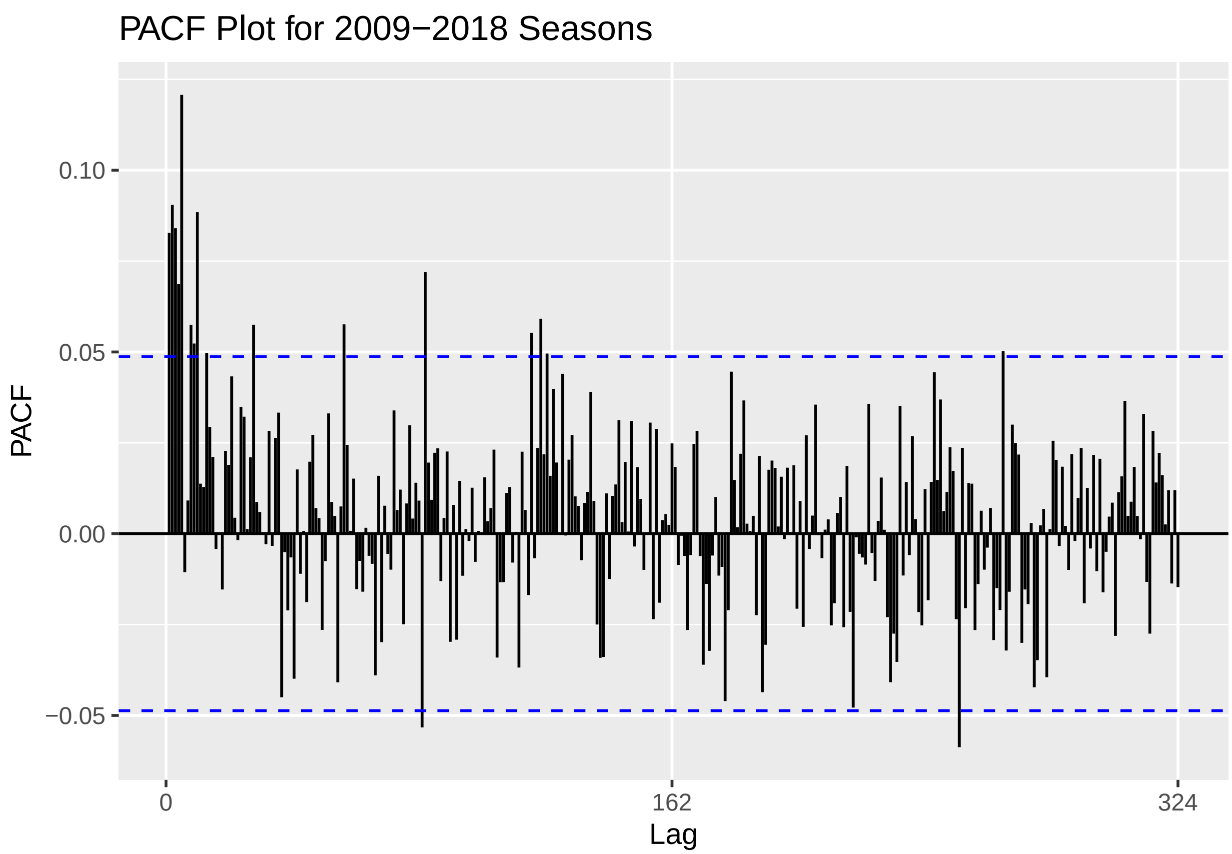
### ACF

Examine the data visually to see if there are concerns around our data not being stationary:

```
ggAcf(avgIndex2009_2018) + ggtitle("ACF Plot for 2009-2018 Seasons")
```




```
ggPacf(avgIndex2009_2018)+ ggtitle("PACF Plot for 2009-2018 Seasons")
```




We will conduct a Dickey-Fuller test to confirm the visual assessment:

```
adf.test(avgIndex2009_2018)
```

```
## Warning in adf.test(avgIndex2009_2018): p-value smaller than printed p-value
```

```
##
##  Augmented Dickey-Fuller Test
##
## data:  avgIndex2009_2018
## Dickey-Fuller = -9.4069, Lag order = 11, p-value = 0.01
## alternative hypothesis: stationary
```

The Dickey-Fuller test outputs a p-value smaller 0.01, which would indicate our data is stationary. However, we will test using ndiffs and nsdiffs to see if there might be an indication of adjustments that should be made for differencing:

```
ndiffs(avgIndex2009_2018)
```

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

Suggests that 1 difference should be done to make data more stationary.

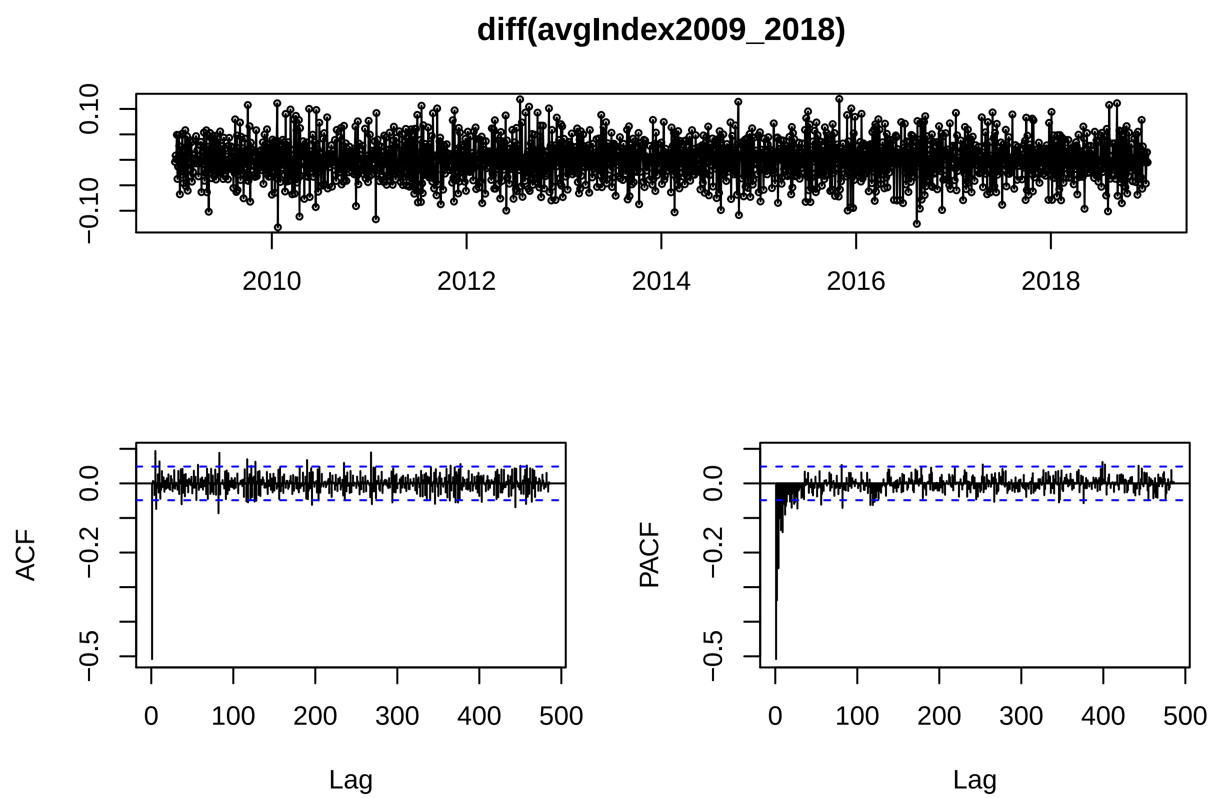
```r
nsdiffs(avgIndex2009_2018)
```

```
## [1] 0
```

Suggests that no seasonal differencing is needed.

Examine visual details once one differencing is done:

```r
tsdisplay(diff(avgIndex2009_2018))
```




Data now looks much more stationary, although there ADF Test did not indicate a strong need to difference our data.

**Ljung-Box Test:**

```r
log(length(avgIndex2009_2018)) # Based on our data, we should really only check up to around 7.4
```

```
## [1] 7.390181
```

```r
# Lag 1
Box.test(avgIndex2009_2018,lag=1,type='Ljung')
```

```
##
##  Box-Ljung test
##
## data:  avgIndex2009_2018
## X-squared = 11.114, df = 1, p-value = 0.0008567
```

```
# Lag 2
Box.test(avgIndex2009_2018,lag=2,type='Ljung')
```

```
##
##  Box-Ljung test
##
## data:  avgIndex2009_2018
## X-squared = 26.293, df = 2, p-value = 1.952e-06
```

```
# Lag 3
Box.test(avgIndex2009_2018,lag=3,type='Ljung')
```

```
##
##  Box-Ljung test
##
## data:  avgIndex2009_2018
## X-squared = 41.757, df = 3, p-value = 4.519e-09
```

```
# Lag 5
Box.test(avgIndex2009_2018,lag=5,type='Ljung')
```

```
##
##  Box-Ljung test
##
## data:  avgIndex2009_2018
## X-squared = 88.34, df = 5, p-value < 2.2e-16
```

```
# Lag 7
Box.test(avgIndex2009_2018,lag=7,type='Ljung')
```

```
##
##  Box-Ljung test
##
## data:  avgIndex2009_2018
## X-squared = 92.54, df = 7, p-value < 2.2e-16
```

```
# Lag 9
Box.test(avgIndex2009_2018,lag=9,type='Ljung')
```

```
##
##  Box-Ljung test
##
## data:  avgIndex2009_2018
## X-squared = 115.63, df = 9, p-value < 2.2e-16
```

Based on these results it looks like we have serial correlation occurring in our data as the p-values are all very small

### ARIMA

We will now examine an ARIMA forecasting model for our data set using auto.arima. Based on the above pre-processing information and testing it looks like an ARIMA model will include a differencing of 1. The Ljung-Test has also shown that we may have serial correlation occurring and this may also impact the ARIMA model that is chosen. Examining the ACF and PACF plots it is likely that there is an autoregression component that may be output by the model. As indicated in our class notes, the ARIMA model includes three components: p = order of the autoregression component d = degree of differencing involved q = order of moving average component

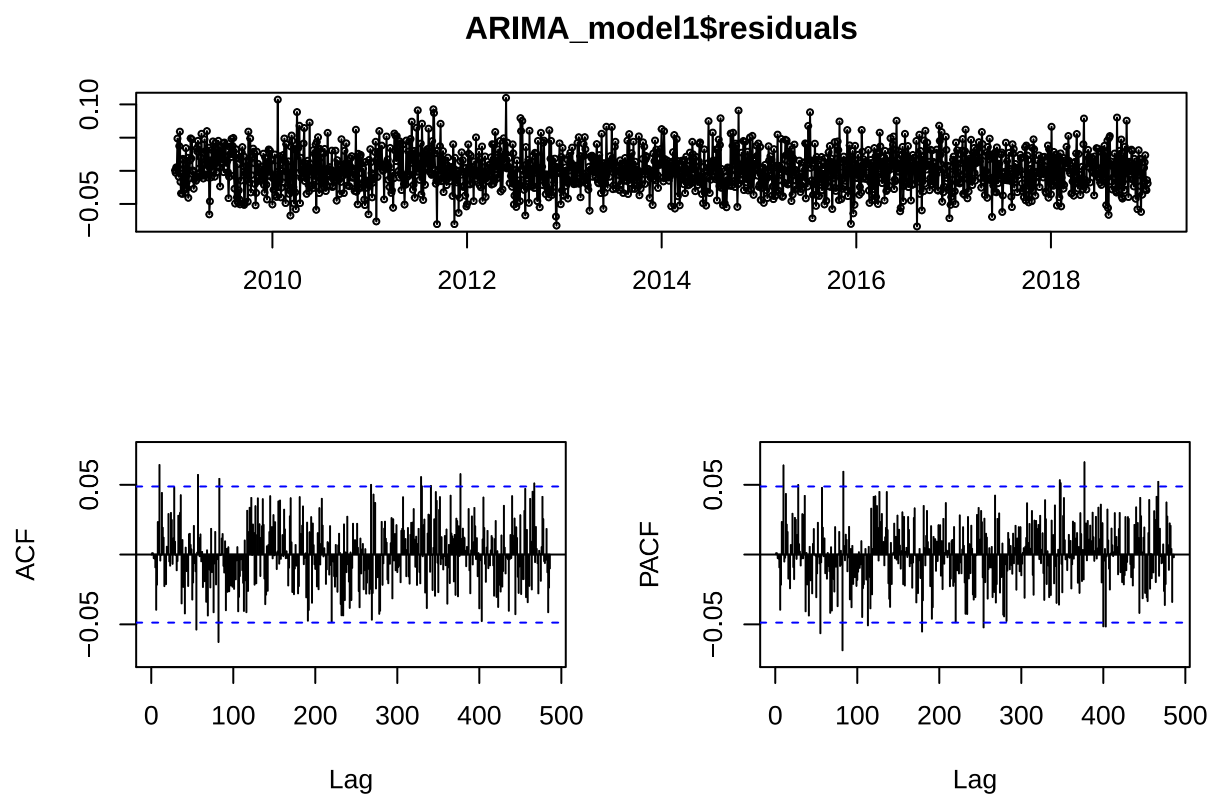
```
ARIMA_model1 = auto.arima(avgIndex2009_2018)
```

Output is a ARIMA(5,1,1) model

```
summary(ARIMA_model1)
```

```
## Series: avgIndex2009_2018
## ARIMA(5,1,1)
##
## Coefficients:
##          ar1     ar2     ar3     ar4     ar5      ma1
##       0.0273  0.0430  0.0448  0.0366  0.0960  -0.9899
## s.e.  0.0252  0.0252  0.0252  0.0252  0.0252   0.0048
##
## sigma^2 estimated as 0.0008393:  log likelihood=3437.7
## AIC=-6861.39   AICc=-6861.33   BIC=-6823.67
##
## Training set error measures:
##                        ME       RMSE        MAE         MPE     MAPE      MASE
## Training set 0.001313684 0.02890756 0.02286085 0.006897879 3.570202 0.7134653
##                      ACF1
## Training set 0.0009492317
```

```
tsdisplay(ARIMA_model1$residuals) # Residuals look pretty close to white noise
```

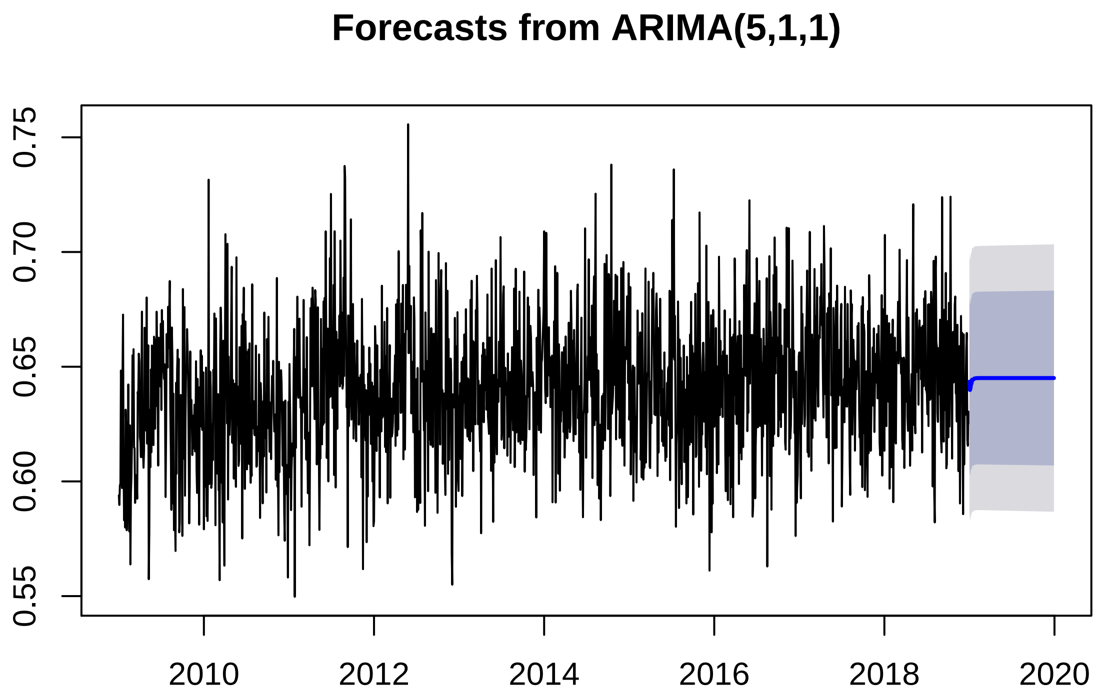



```
Box.test(ARIMA_model1$residuals, lag = 7, type = 'Ljung') # Residuals are white noise
```

```
##
##  Box-Ljung test
##
## data:  ARIMA_model1$residuals
## X-squared = 3.4681, df = 7, p-value = 0.8386
```

Assessing the accuracy of the model:

```
ARIMAForecast <- forecast(ARIMA_model1, h = 162)
plot(ARIMAForecast)
```




```
ARIMA_acc <- accuracy(ARIMAForecast, avgData2019$mean_PerSK)
ARIMA_acc
```

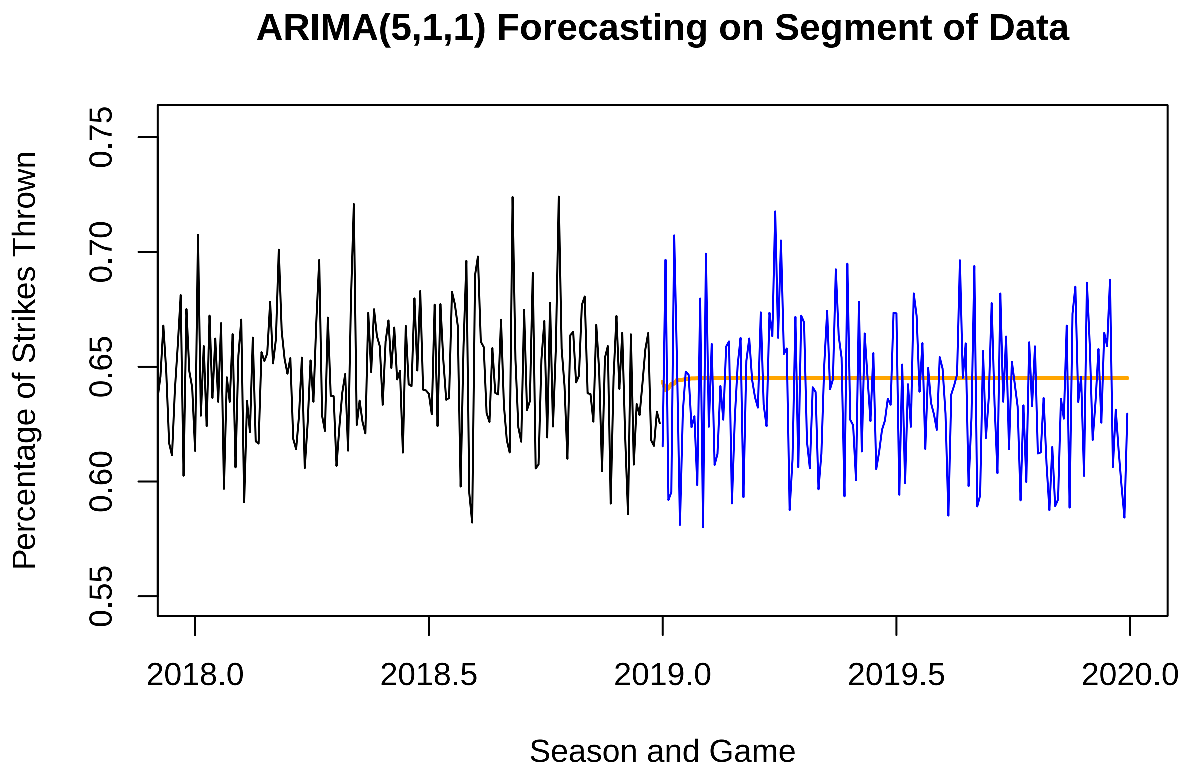
```
##                       ME       RMSE        MAE          MPE     MAPE      MASE
## Training set  0.001313684 0.02890756 0.02286085  0.006897879 3.570202 0.7103988
## Test set     -0.007301109 0.03149151 0.02596643 -1.377437395 4.121921 0.8069047
##                      ACF1
## Training set 0.0009492317
## Test set               NA
```

Creating list of error values for assessing all models:

```
arima_errors_test <- ARIMA_acc[2,]
```

Plotting a portion of the data for easier interpretation:

```
plot(avgIndex2009_2018,
     xlim = c(2018.0, 2020.0),
     xlab = "Season and Game",
     ylab = "Percentage of Strikes Thrown",
     main = "ARIMA(5,1,1) Forecasting on Segment of Data")
lines(ARIMAForecast$mean, col="orange", lwd=2)
lines(avgIndex2019, col="blue")
```




#### Examining Errors and AICc where appropriate:

```
print("Naive Forecast Errors")
```

```
## [1] "Naive Forecast Errors"
```

```
naive_errors_test
```

```
##          ME        RMSE         MAE         MPE        MAPE        MASE
##  0.01227252  0.03297864  0.02677076  1.69938516  4.13031120  0.83548990
##        ACF1   Theil's U
## -0.07056665  0.73974992
```

```
print("SES Forecast Errors")
```

```
## [1] "SES Forecast Errors"
```

```
ses_errors_test
```

```
##           ME         RMSE          MAE          MPE         MAPE         MASE
## -0.003283046  0.030785620  0.025353078 -0.745780338  4.001794285  0.791245468
##         ACF1    Theil's U
## -0.070566653  0.687352891
```

```r
print("ETS Forecast Errors")
```

```
## [1] "ETS Forecast Errors"
```

```r
ets_errors_test
```

```
##           ME         RMSE          MAE          MPE         MAPE         MASE
## -0.003285707  0.030785904  0.025353341 -0.746198572  4.001851899  0.787852931
##         ACF1
##           NA
```

```r
print("ARIMA Forecast Errors")
```

```
## [1] "ARIMA Forecast Errors"
```

```r
arima_errors_test
```

```
##           ME         RMSE          MAE          MPE         MAPE         MASE
## -0.007301109  0.031491509  0.025966434 -1.377437395  4.121920712  0.806904746
##         ACF1
##           NA
```

Based on analyzing the errors from the four methods used, SES performs the best for our data.

```r
print("ETS AICc")
```

```
## [1] "ETS AICc"
```

```r
averageETS$aicc
```

```
## [1] 514.5797
```

```r
print("ARIMA AICc")
```

```
## [1] "ARIMA AICc"
```

```r
ARIMA_model1$aicc
```

```
## [1] -6861.325
```

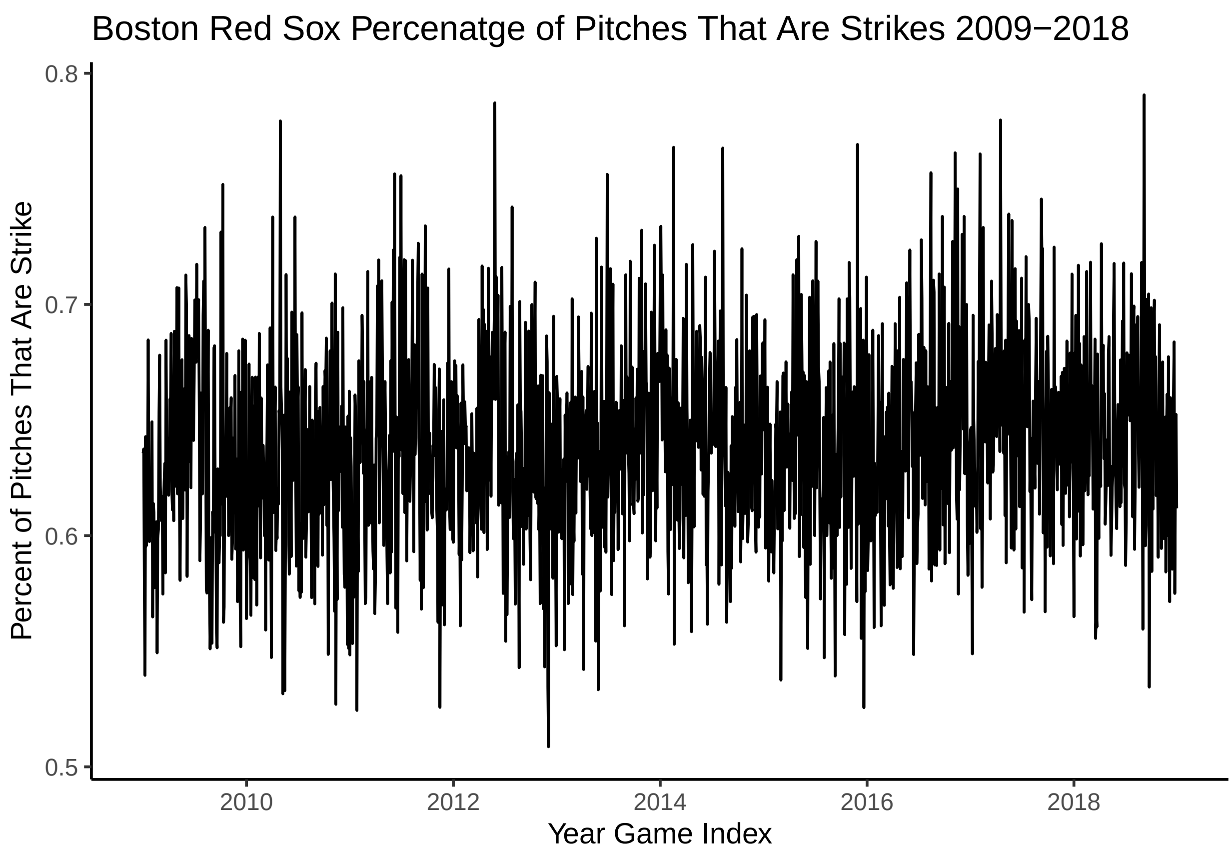
### Boston

```r
Bos2019 <- data %>%
  dplyr::filter(Team == "BOS" & Year == 2019)
BosIndex2019 = ts(Bos2019$PerSK,start=c(2019,1), frequency = 162)
Bos2009_2018 <- data %>%
  dplyr::filter(Team == "BOS" & Year < 2019)
BosIndex2009_2018 = ts(Bos2009_2018$PerSK,start=c(2009,1), frequency = 162)
```

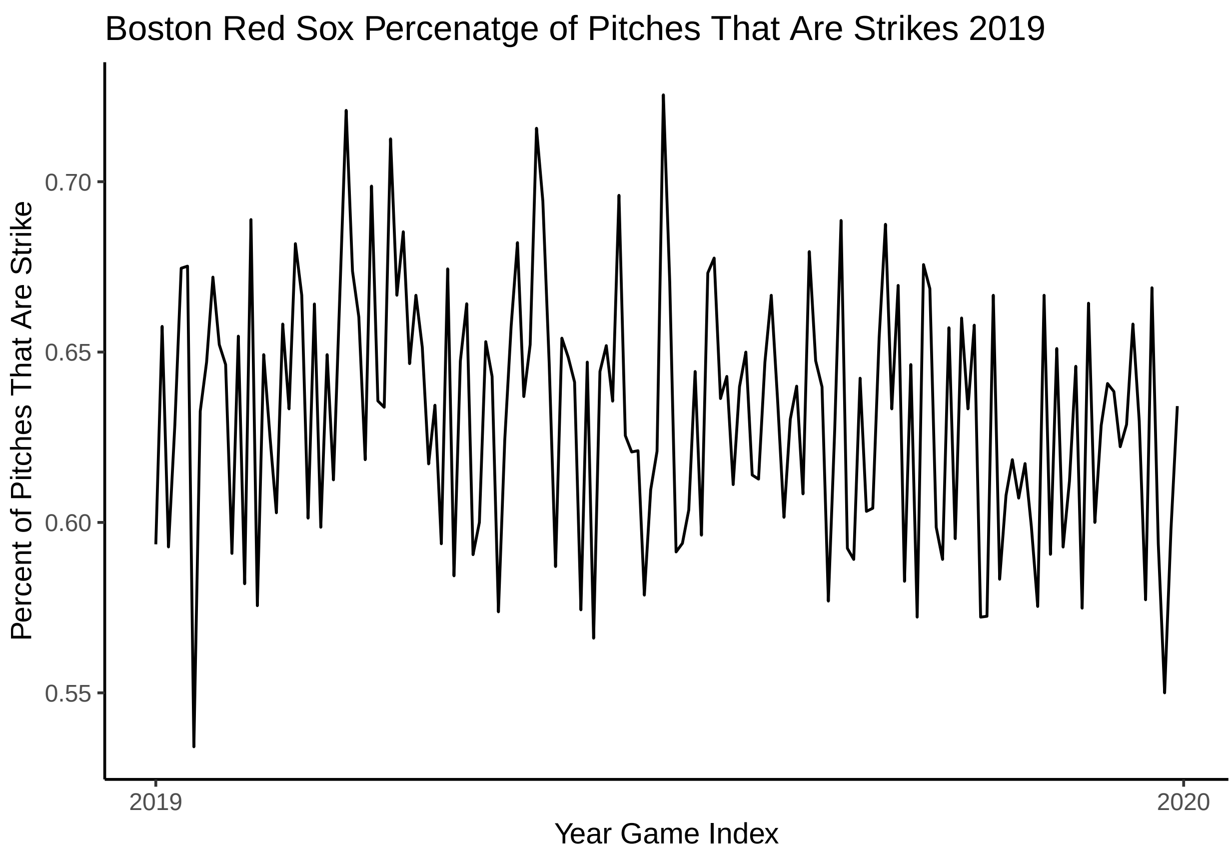
Above we created the training and test sets as well as making the data a time series.

**Time Plot**

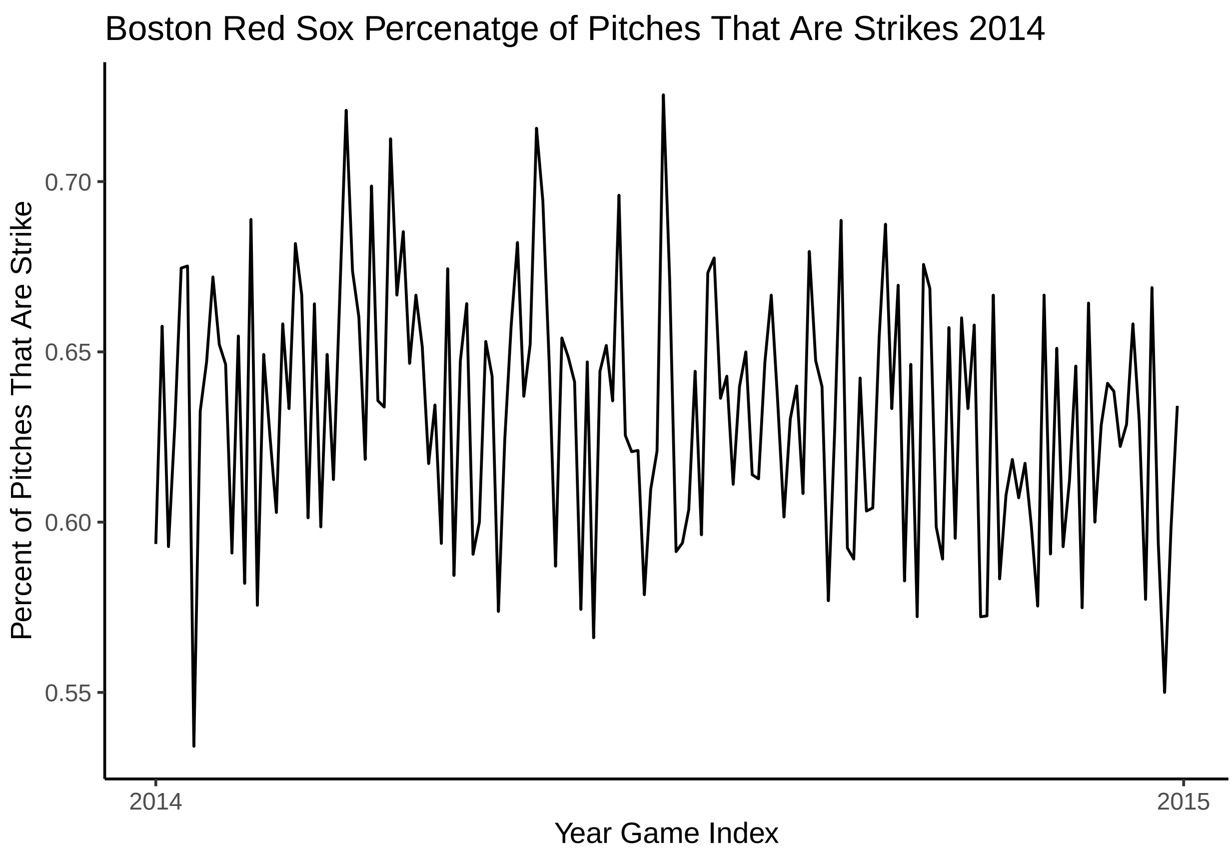
```
autoplot(BosIndex2009_2018) + xlab("Year Game Index") +
  ylab("Percent of Pitches That Are Strike") + ggtitle("Boston Red Sox Percenatge of Pitches That Are St
  theme_classic()
```




```
autoplot(BosIndex2019) + xlab("Year Game Index") +
  ylab("Percent of Pitches That Are Strike") + ggtitle("Boston Red Sox Percenatge of Pitches That Are St
  theme_classic()
```




```
Bos2014 <- data %>%
  dplyr::filter(Team == "BOS" & Year == 2014)
BosIndex2014 = ts(Bos2019$PerSK,start=c(2014,1), frequency = 162)
autoplot(BosIndex2014) + xlab("Year Game Index") +
  ylab("Percent of Pitches That Are Strike") + ggtitle("Boston Red Sox Percenatge of Pitches That Are St
  theme_classic()
```

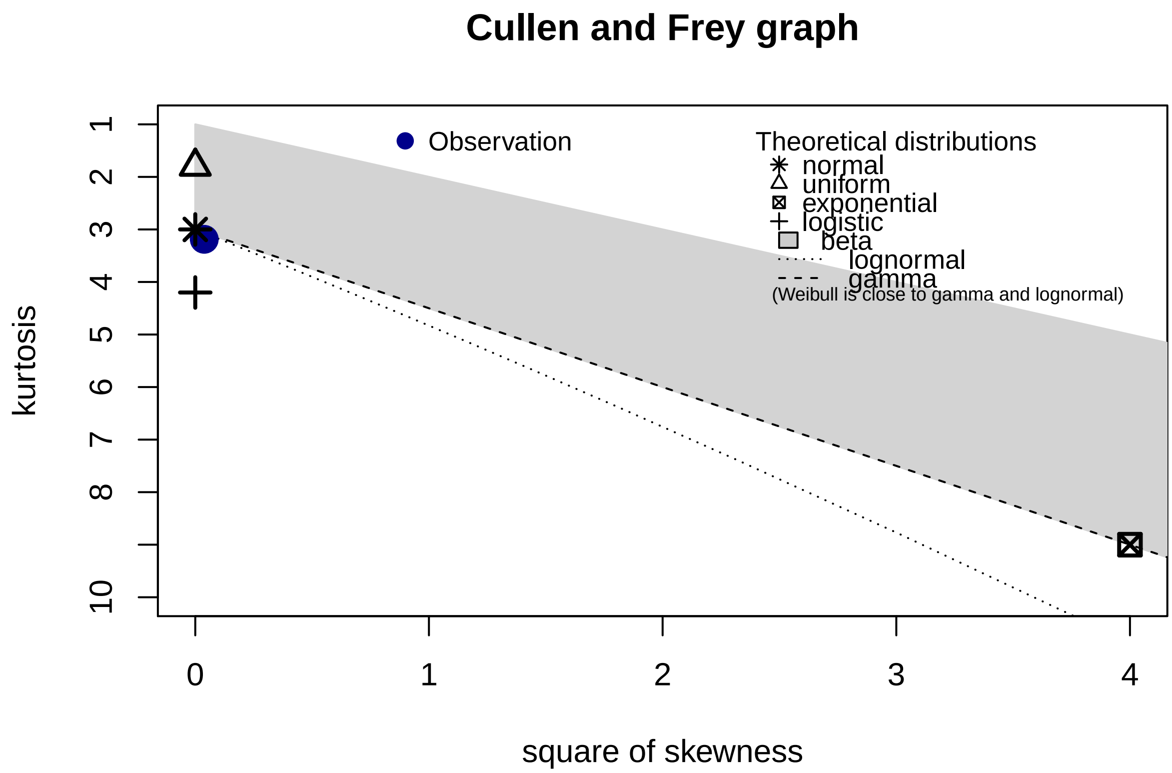



#### Check Distribution

Next, we examined the distribution to check the normality of the data.

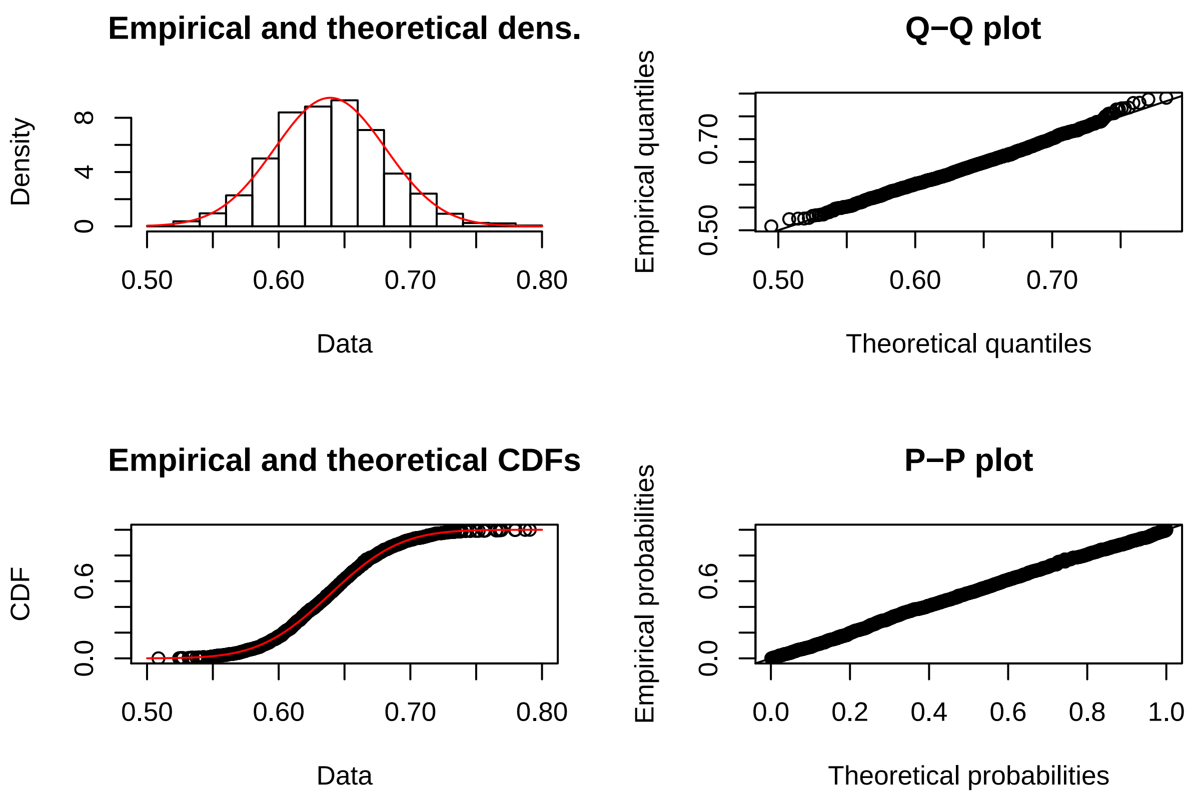
Training Set:

```
descdist(as.numeric(BosIndex2009_2018), discrete = FALSE)
```



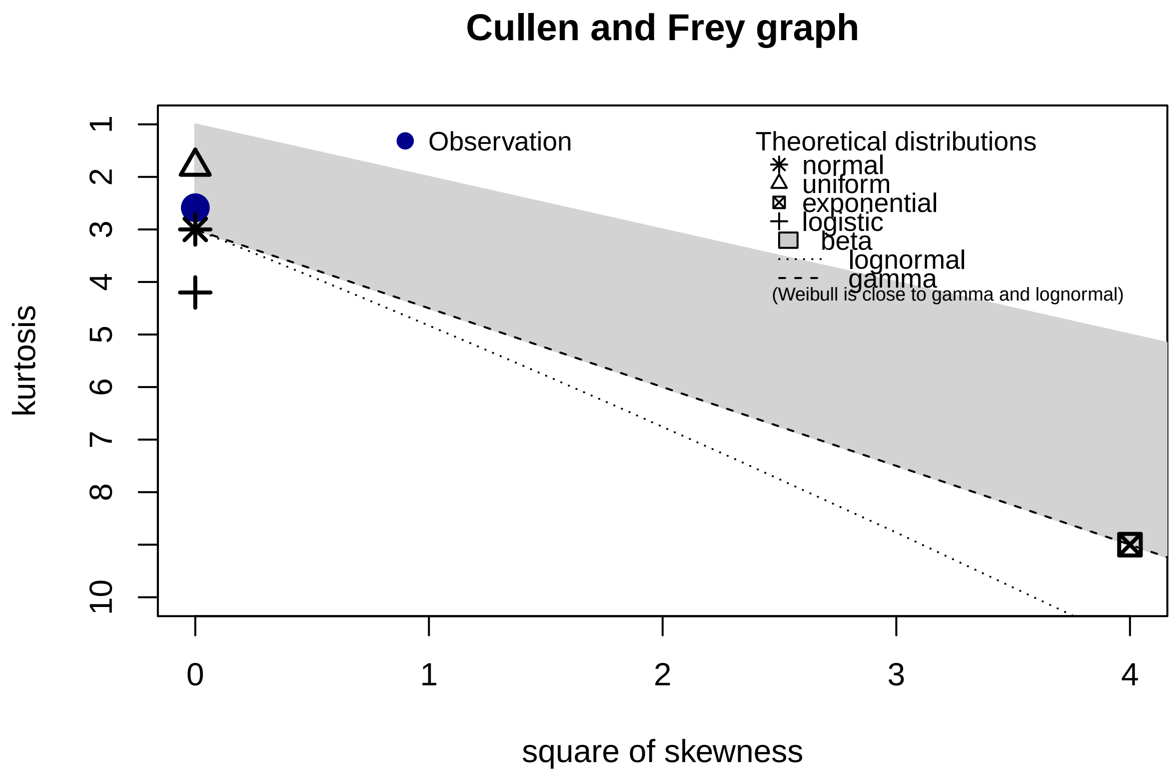

```
## summary statistics
## ------
## min:  0.5086705   max:  0.7906977
## median:  0.6381579
## mean:  0.6390978
## estimated sd:  0.04213107
## estimated skewness:  0.196141
## estimated kurtosis:  3.188347
```

```
plot(fitdist(as.numeric(BosIndex2009_2018), "norm"))
```



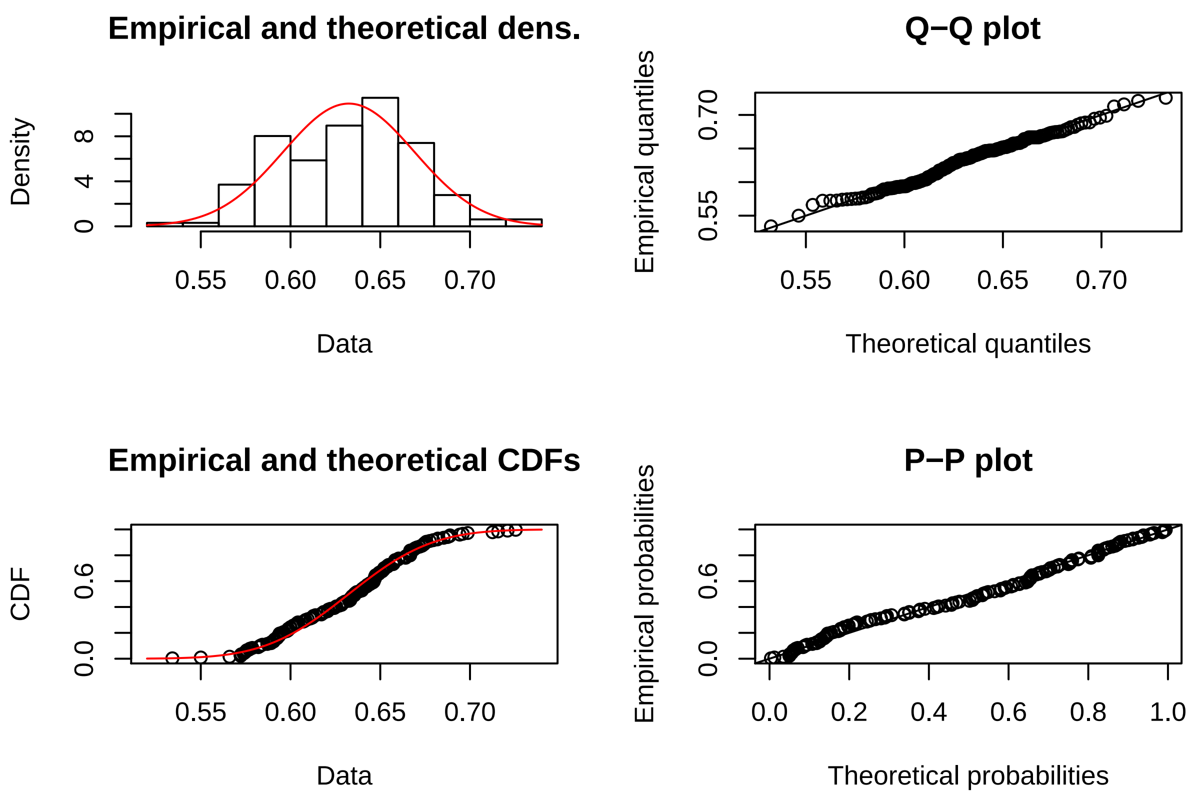

Test Set:

```
descdist(as.numeric(BosIndex2019), discrete = FALSE)
```




```
## summary statistics
## ------
## min:  0.5341615   max:  0.7254902
## median:  0.6357485
## mean:  0.6324597
## estimated sd:  0.03669672
## estimated skewness:  -0.01474578
## estimated kurtosis:  2.587301
```

```
plot(fitdist(as.numeric(BosIndex2019), "norm"))
```




Based off these graphs, it appears that both data sets are normally distributed.

Training Set:

```
fBasics::normalTest(BosIndex2009_2018, method = 'jb')
```

```
##
## Title:
##  Jarque - Bera Normalality Test
##
## Test Results:
##   STATISTIC:
##     X-squared: 12.6549
##   P VALUE:
##     Asymptotic p Value: 0.001787
##
## Description:
##  Mon Nov 16 22:04:34 2020 by user:
```

Testing Set:

```
fBasics::normalTest(BosIndex2019, method = "jb")
```

```
##
## Title:
```

```
##  Jarque - Bera Normalality Test
##
## Test Results:
##   STATISTIC:
##     X-squared: 1.2941
##   P VALUE:
##     Asymptotic p Value: 0.5236
##
## Description:
##  Mon Nov 16 22:04:34 2020 by user:
```

The training set passes the J-B Test, however, the testing set does not.

### Forecasting Models:

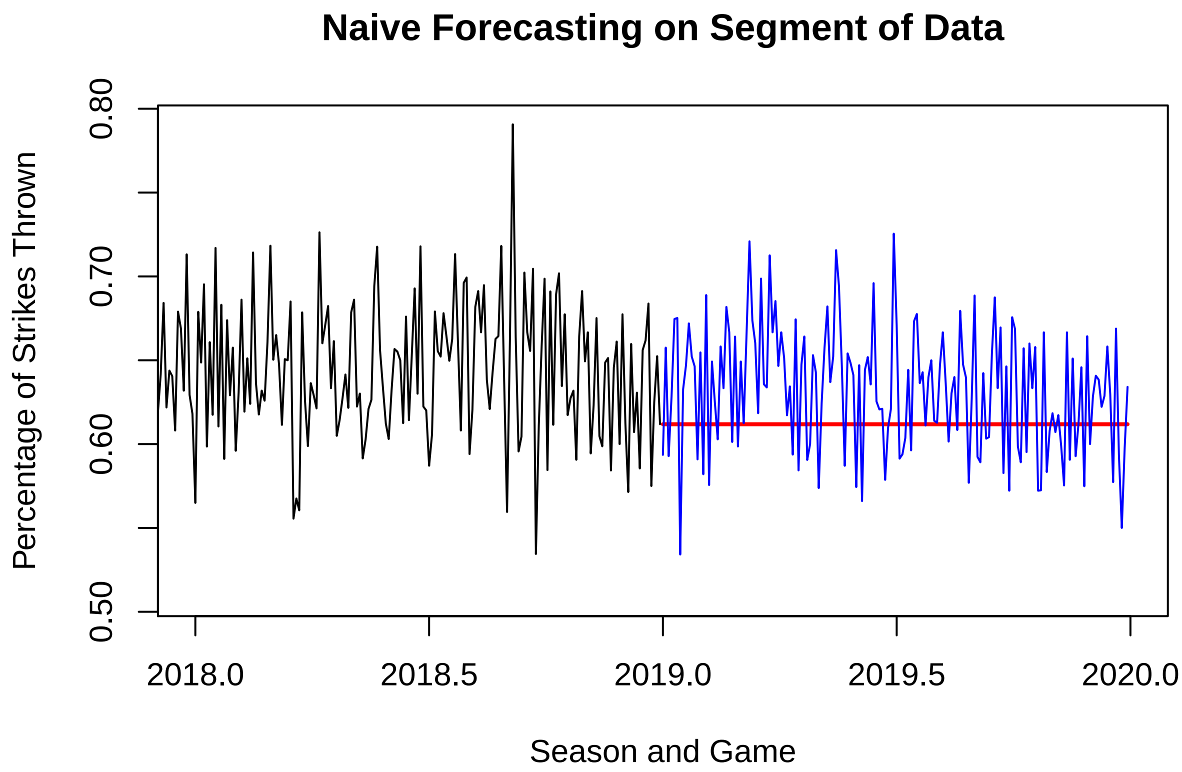
#### Naive

Uses the most recent data point as a foreast.

```r
BosNaive <- naive(BosIndex2009_2018, h = 162)
accuracy(BosNaive, BosIndex2019)
```

```
##                         ME       RMSE        MAE        MPE     MAPE      MASE
## Training set -1.471081e-05 0.05815381 0.04676407 -0.4168228 7.344388 1.0059049
## Test set      2.061758e-02 0.04199311 0.03505766  2.9331625 5.428761 0.7540977
##                     ACF1 Theil's U
## Training set -0.50874537        NA
## Test set     -0.03244388 0.7900361
```

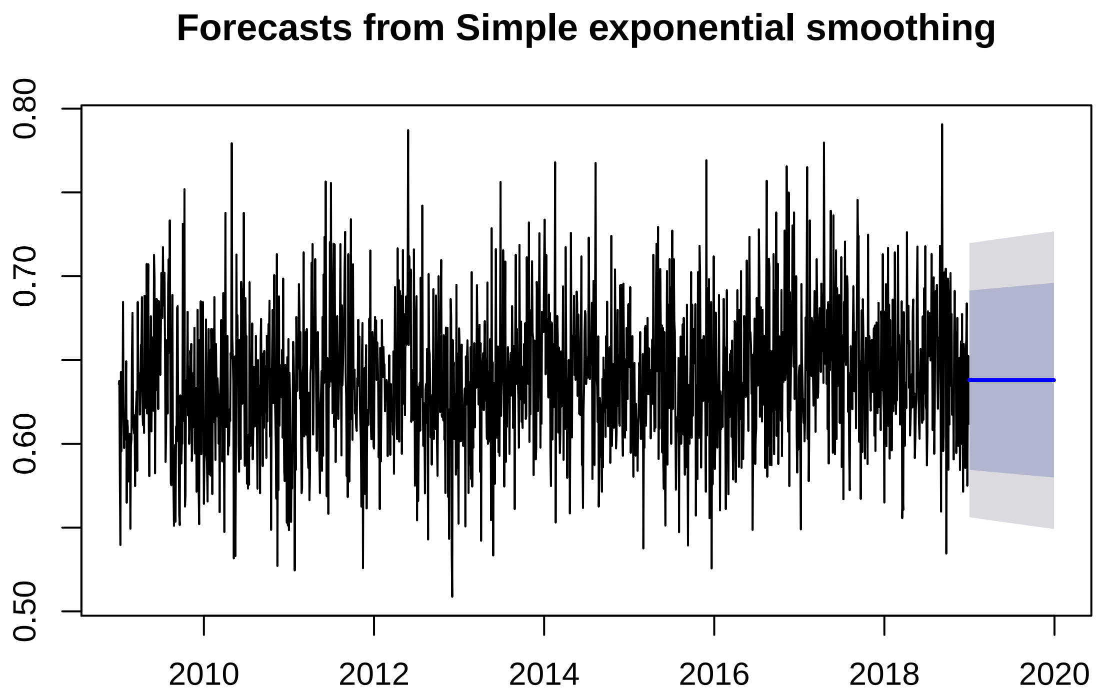
```r
plot(BosIndex2009_2018,
     xlim = c(2018.0, 2020.0),
     xlab = "Season and Game",
     ylab = "Percentage of Strikes Thrown",
     main = "Naive Forecasting on Segment of Data")
lines(naive(BosIndex2009_2018, h=162)$mean, col="red", lwd=2)
lines(BosIndex2019, col="blue")
```




**SES**

Using weighted averages of past values to crete a flat forecast:

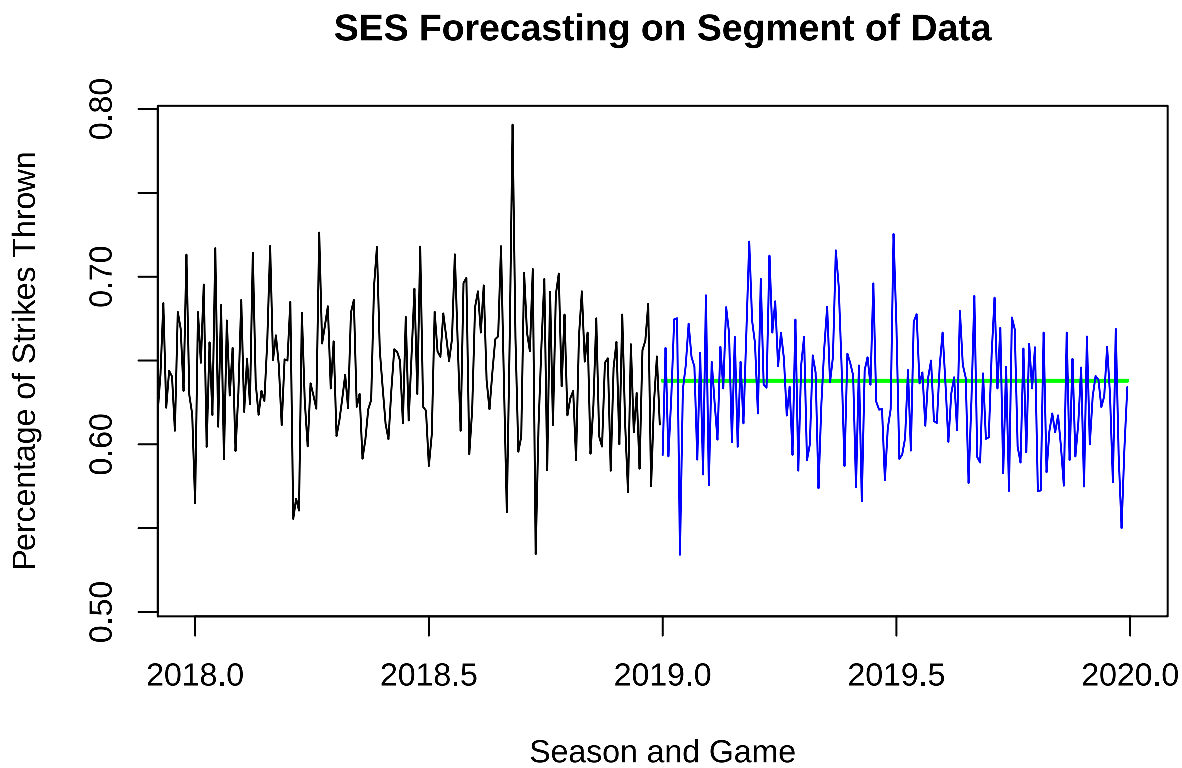
```
BosModelSES <- ses(BosIndex2009_2018, h = 162)
plot(BosModelSES)
```




```r
accuracy(BosModelSES, BosIndex2019)
```

```
##                         ME       RMSE        MAE        MPE     MAPE      MASE
## Training set  0.000304222 0.04170095 0.03312183 -0.370681 5.206483 0.7124575
## Test set     -0.005502127 0.03699473 0.03010262 -1.210648 4.839276 0.6475136
##                      ACF1 Theil's U
## Training set -0.004960156        NA
## Test set     -0.032443876 0.6896162
```

```r
plot(BosIndex2009_2018,
     xlim = c(2018.0, 2020.0),
     xlab = "Season and Game",
     ylab = "Percentage of Strikes Thrown",
     main = "SES Forecasting on Segment of Data")
lines(ses(BosIndex2009_2018, h=162)$mean, col="green", lwd=2)
lines(BosIndex2019, col="blue")
```



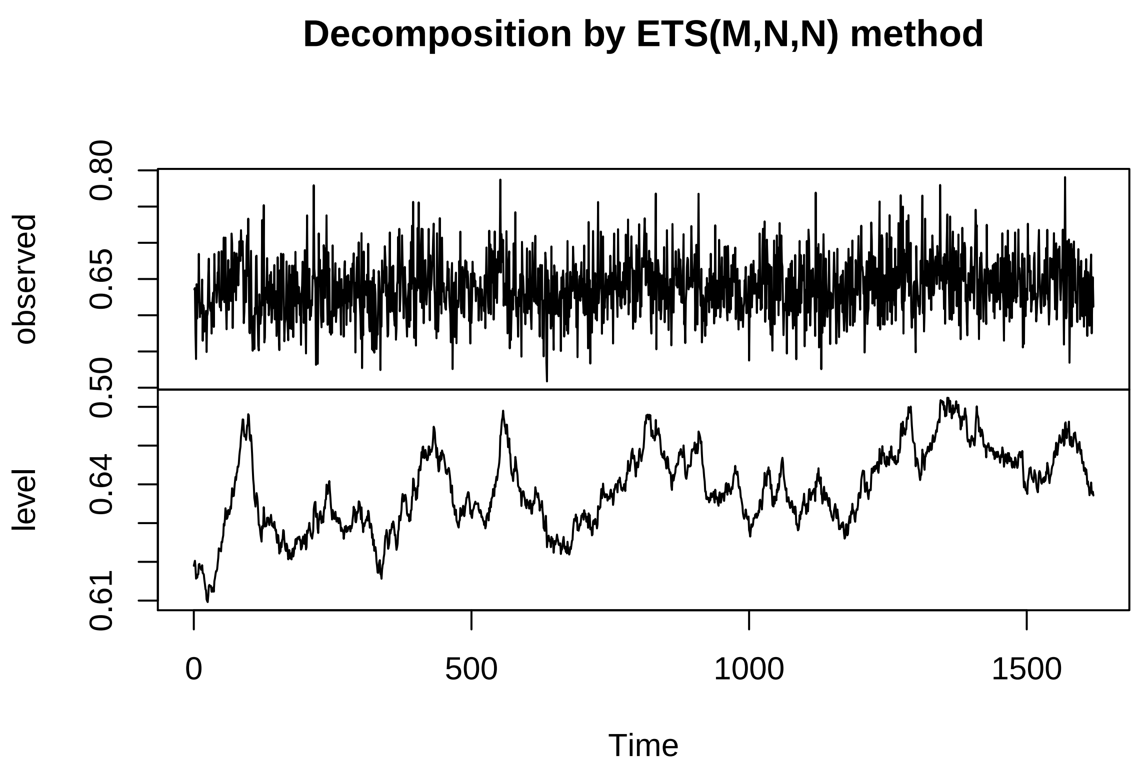

**ETS**

```
BosETS <- ets(Bos2009_2018$PerSK)
summary(BosETS)
```

```
## ETS(M,N,N)
##
## Call:
##  ets(y = Bos2009_2018$PerSK)
##
##   Smoothing parameters:
##     alpha = 0.0373
##
##   Initial states:
##     l = 0.619
##
##   sigma:  0.0653
##
##      AIC     AICc      BIC
## 1680.477 1680.492 1696.647
##
## Training set error measures:
##                         ME       RMSE        MAE       MPE     MAPE      MASE
## Training set 0.0003005453 0.04170309 0.03311298 -0.370391 5.205362 0.7080859
##                       ACF1
```

```
## Training set -0.008778531
```

```r
plot(BosETS)
```



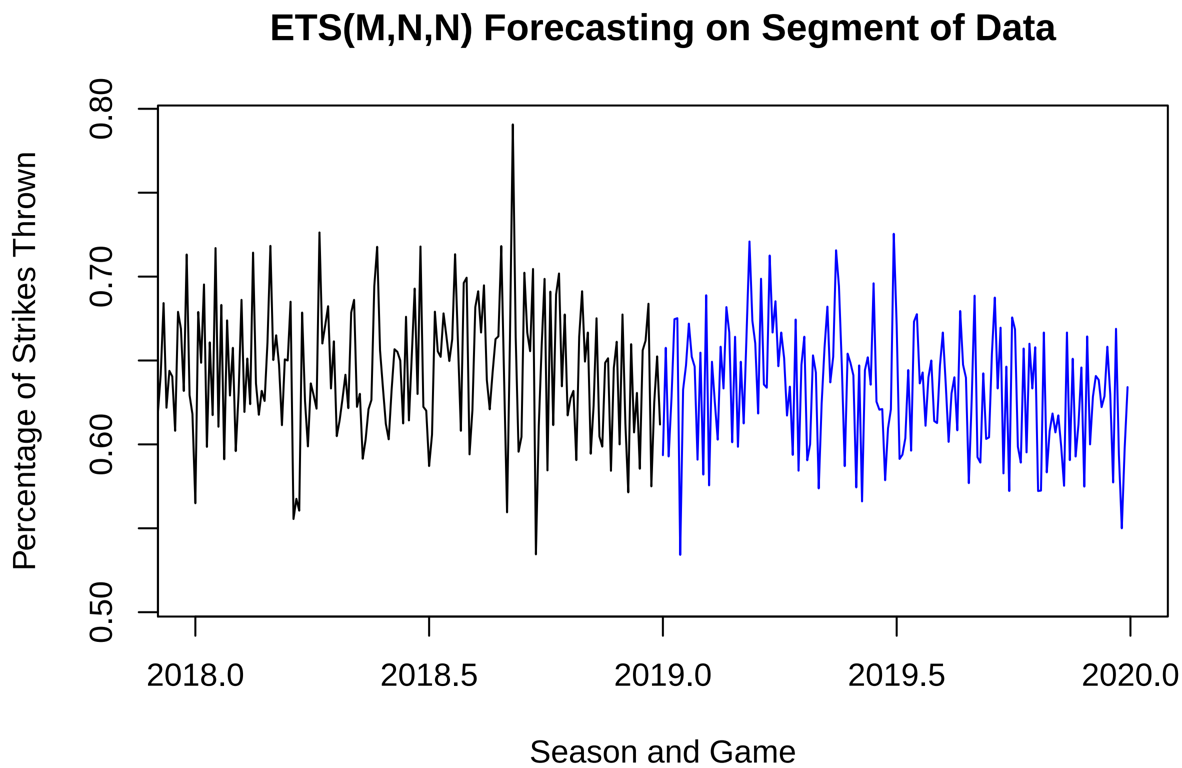

The output from the model is an ETS(M,N,N), which has multiplicative errors, no trend, and no seasonality.

Looking at the accuracy of the ETS Model

```r
BosETSForcast <- forecast(BosETS, h = 162)
accuracy(BosETSForcast, Bos2019$PerSK)
```

```
##                         ME       RMSE        MAE       MPE     MAPE      MASE
## Training set  0.0003005453 0.04170309 0.03311298 -0.370391 5.205362 0.7080859
## Test set     -0.0046614390 0.03687907 0.03007148 -1.077275 4.828019 0.6430467
##                      ACF1
## Training set -0.008778531
## Test set               NA
```

```r
plot(BosIndex2009_2018,
     xlim = c(2018.0, 2020.0),
     xlab = "Season and Game",
     ylab = "Percentage of Strikes Thrown",
     main = "ETS(M,N,N) Forecasting on Segment of Data")
lines(BosETSForcast$mean, col="cyan", lwd=2)
lines(BosIndex2019, col="blue")
```



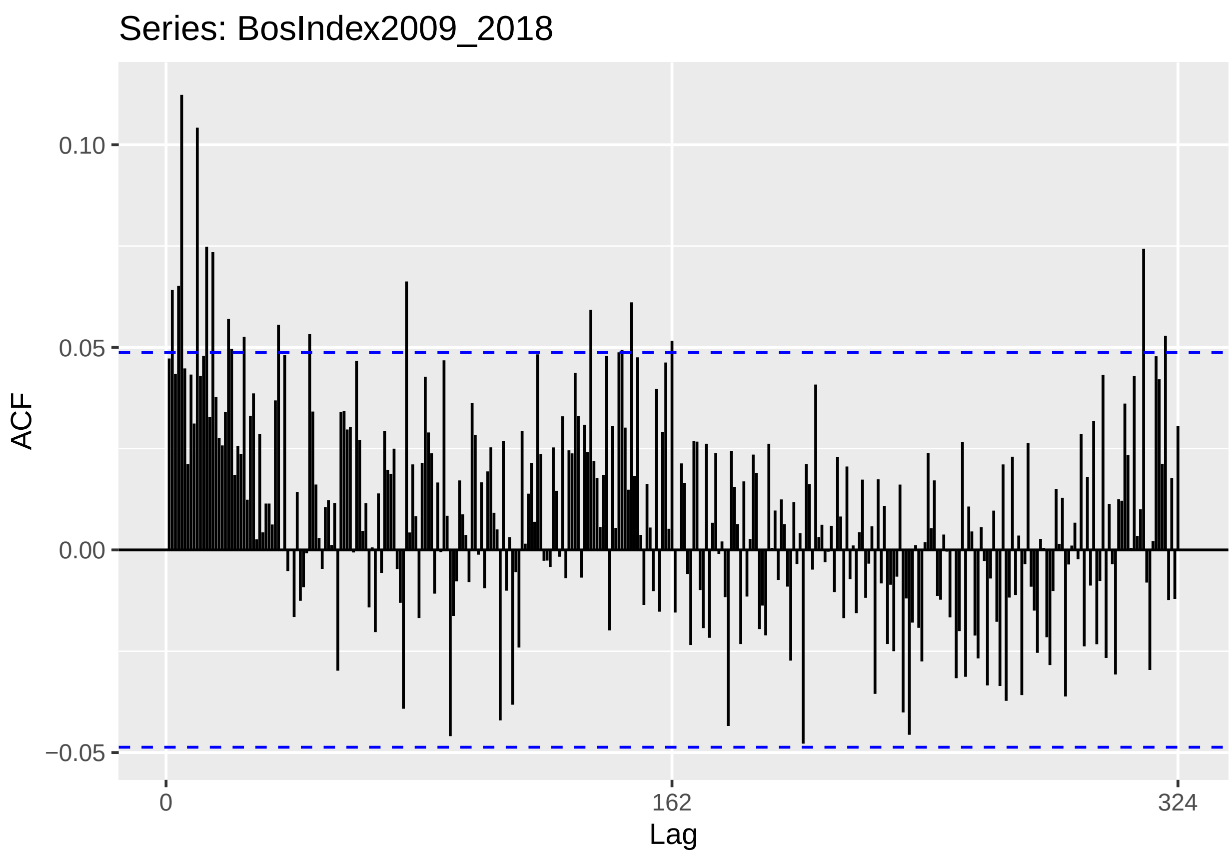

**ACF**

```
adf.test(BosIndex2009_2018)
```

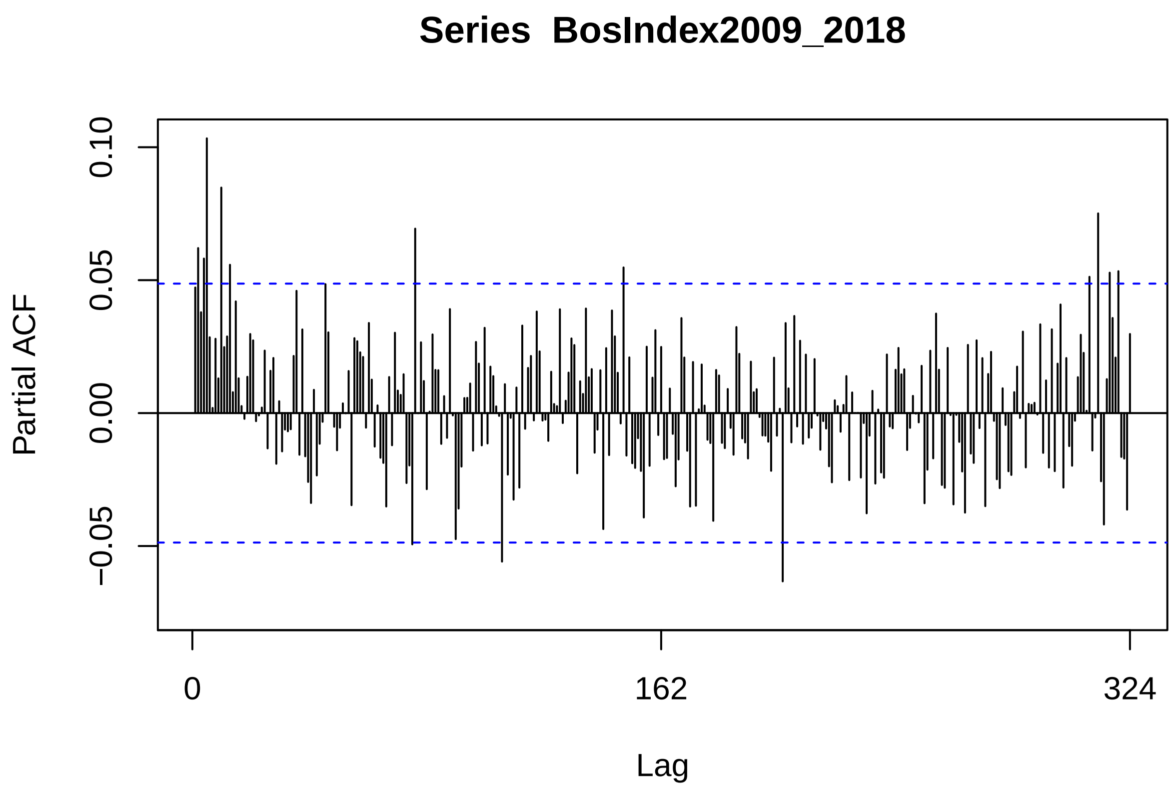
```
## Warning in adf.test(BosIndex2009_2018): p-value smaller than printed p-value
```

```
##
##  Augmented Dickey-Fuller Test
##
## data:  BosIndex2009_2018
## Dickey-Fuller = -9.3182, Lag order = 11, p-value = 0.01
## alternative hypothesis: stationary
```

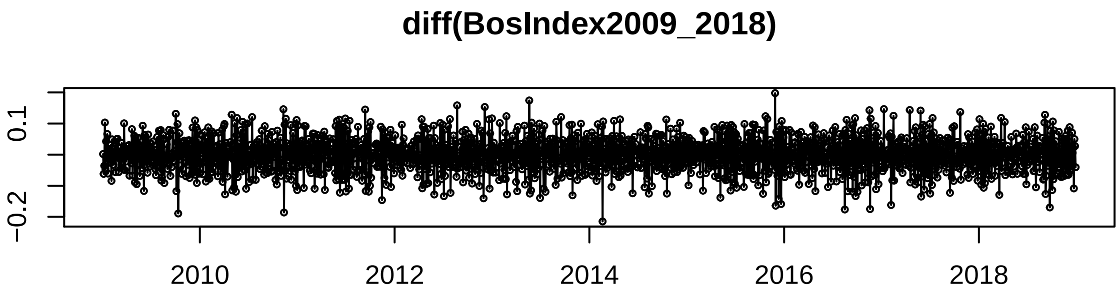
```
ggAcf(BosIndex2009_2018)
```

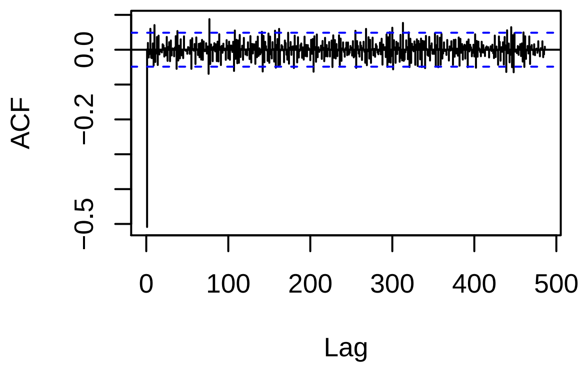


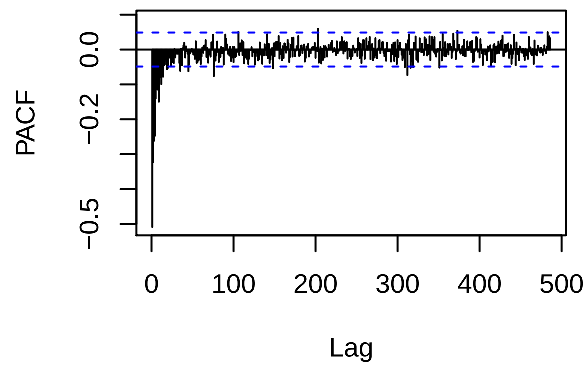

```
Pacf(BosIndex2009_2018)
```




```
tsdisplay(diff(BosIndex2009_2018))
```




```
ndiffs(BosIndex2009_2018)
```

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

```
nsdiffs(BosIndex2009_2018)
```

```
## [1] 0
```

**Arima**

Output is a ARIMA(4, 1, 1) with drift model

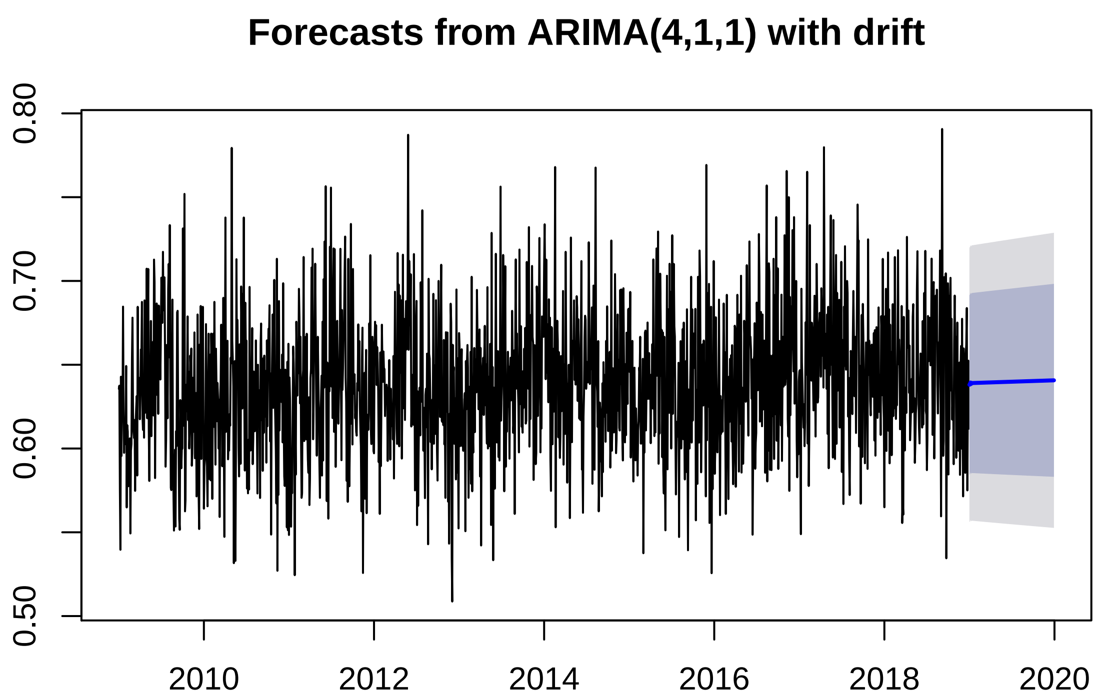
```
BosArima <- auto.arima(BosIndex2009_2018)
summary(BosArima)
```

```
## Series: BosIndex2009_2018
## ARIMA(4,1,1) with drift
##
## Coefficients:
##           ar1     ar2      ar3     ar4      ma1  drift
##       -0.0008  0.0169  -0.0036  0.0199  -0.9701      0
## s.e.   0.0288  0.0286   0.0283  0.0281   0.0146      0
##
## sigma^2 estimated as 0.001745:  log likelihood=2845.33
```

```
## AIC=-5676.66   AICc=-5676.59   BIC=-5638.93
##
## Training set error measures:
##                        ME       RMSE        MAE        MPE     MAPE      MASE
## Training set 0.0001282787 0.04168704 0.03307933 -0.3976575 5.200827 0.7115432
##                      ACF1
## Training set -0.001662638
```

Looking at the accuracy

```
BosArimaForecast <- forecast(BosArima, h = 162)
plot(BosArimaForecast)
```

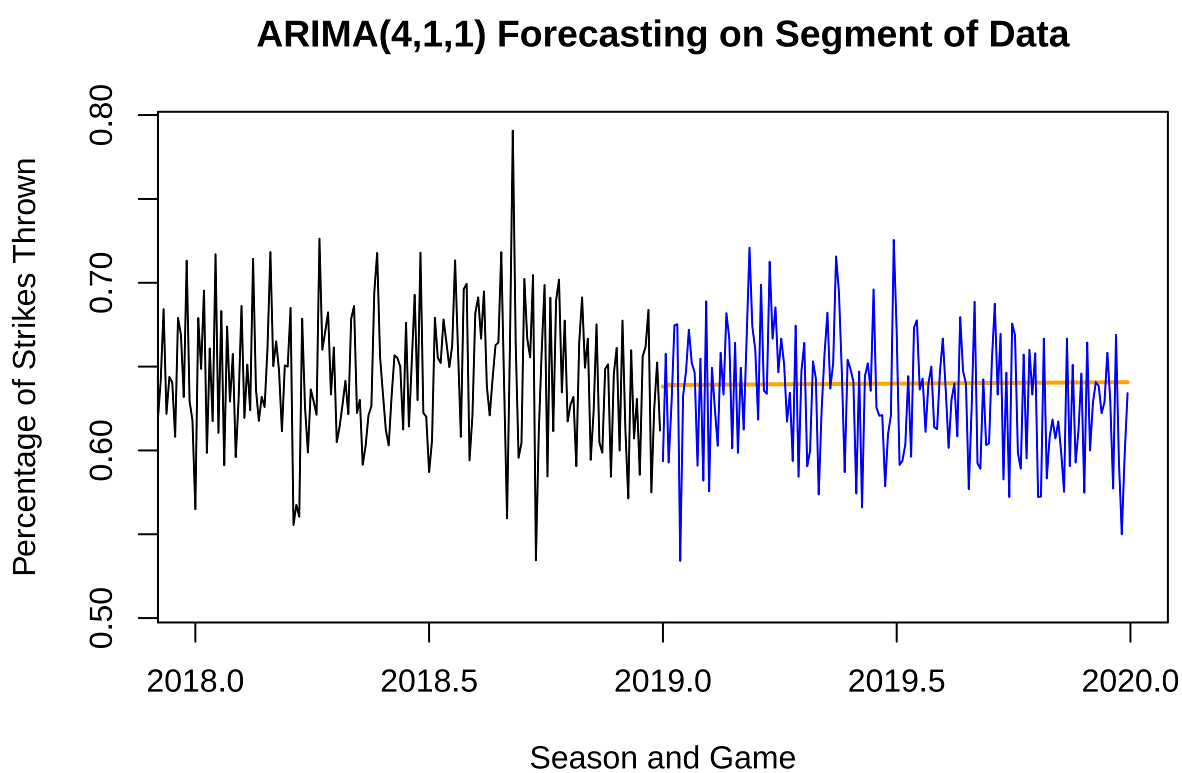



```
accuracy(BosArimaForecast, BosIndex2019)
```

```
##                         ME       RMSE        MAE        MPE     MAPE      MASE
## Training set  0.0001282787 0.04168704 0.03307933 -0.3976575 5.200827 0.7115432
## Test set     -0.0073902068 0.03742291 0.03028949 -1.5110785 4.882947 0.6515333
##                      ACF1 Theil's U
## Training set -0.001662638        NA
## Test set     -0.026704182 0.6977084
```

Plotting a portion of the data for easier interpretation:

```r
plot(BosIndex2009_2018,
     xlim = c(2018.0, 2020.0),
     xlab = "Season and Game",
     ylab = "Percentage of Strikes Thrown",
     main = "ARIMA(4,1,1) Forecasting on Segment of Data")
lines(BosArimaForecast$mean, col="orange", lwd=2)
lines(BosIndex2019, col="blue")
```



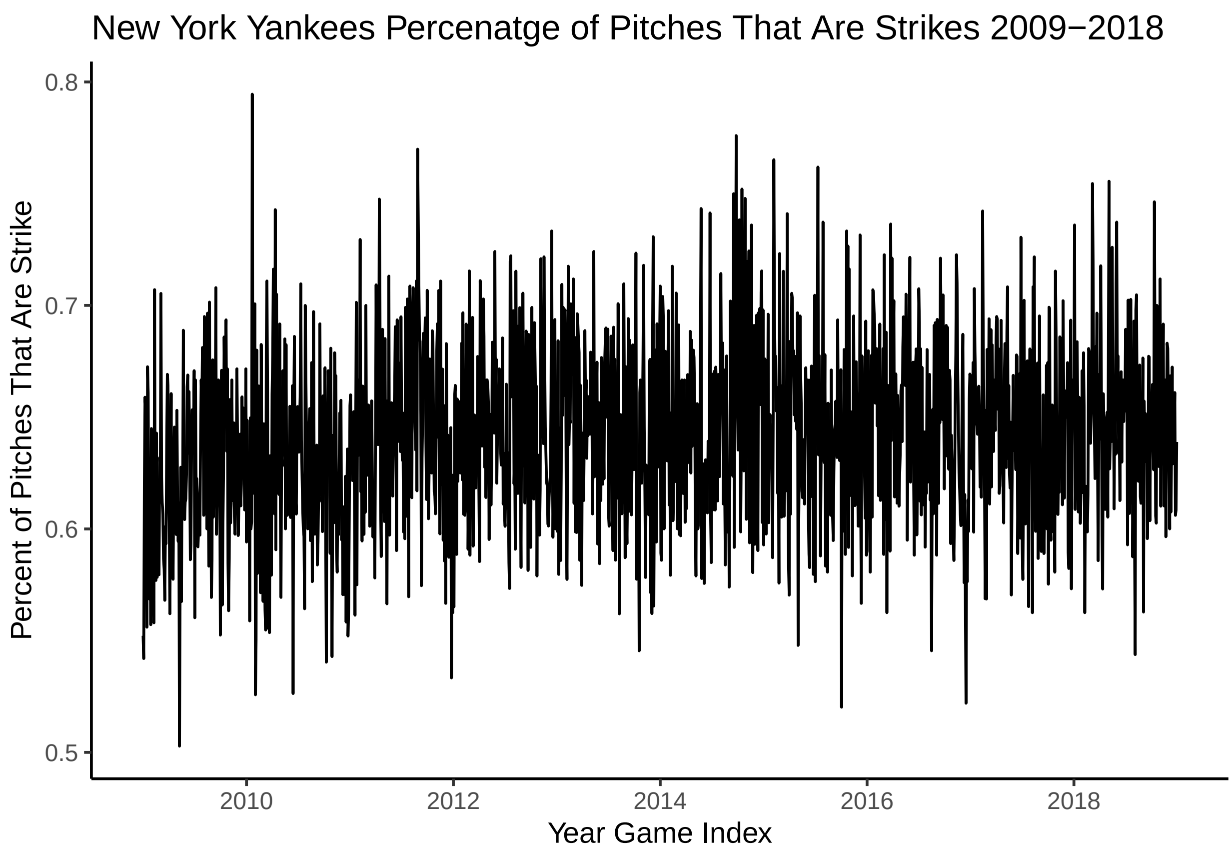

**New York**

```r
Nyy2019 <- data %>%
  dplyr::filter(Team == "NYY" & Year == 2019)
NyyIndex2019 = ts(Nyy2019$PerSK,start=c(2019,1), frequency = 162)
Nyy2009_2018 <- data %>%
  dplyr::filter(Team == "NYY" & Year < 2019)
NyyIndex2009_2018 = ts(Nyy2009_2018$PerSK,start=c(2009,1), frequency = 162)
```

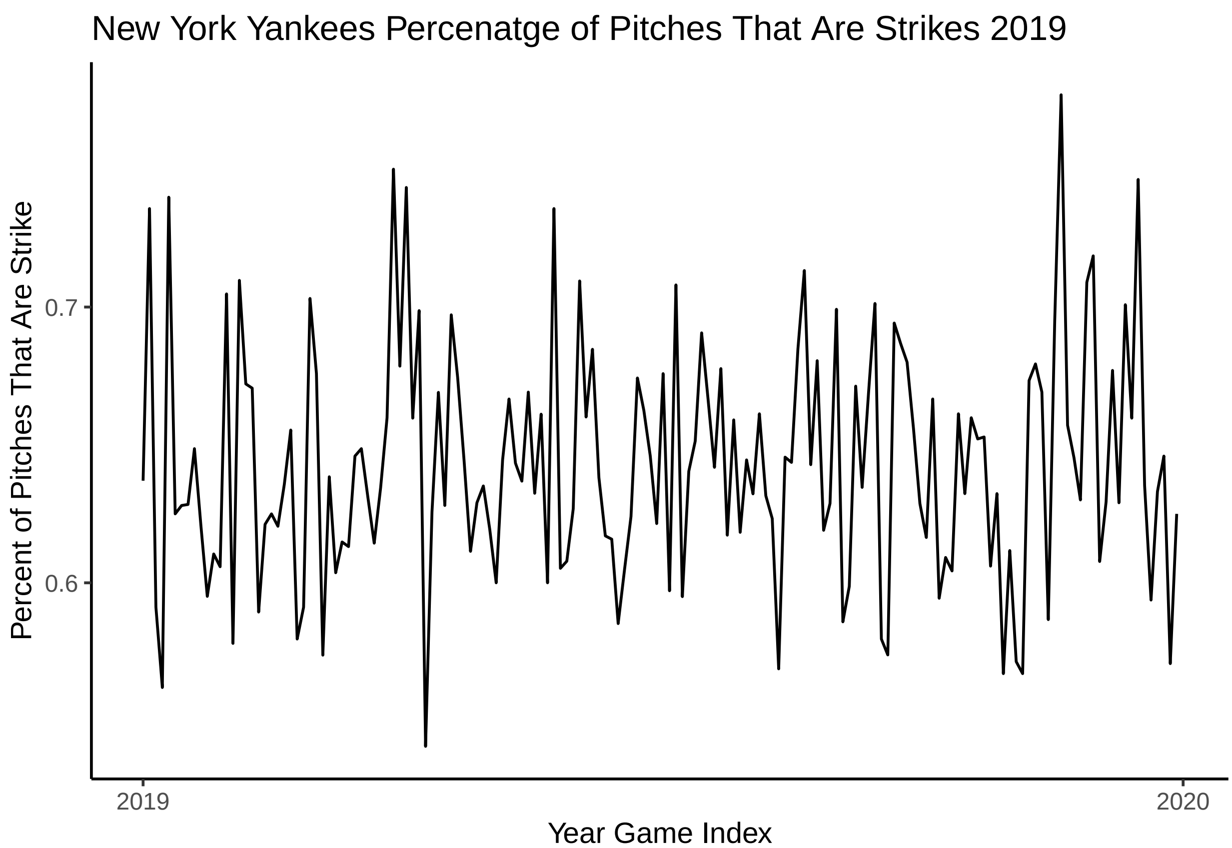
Above we created the training and test sets as well as making the data a time series.

**Time Plot**

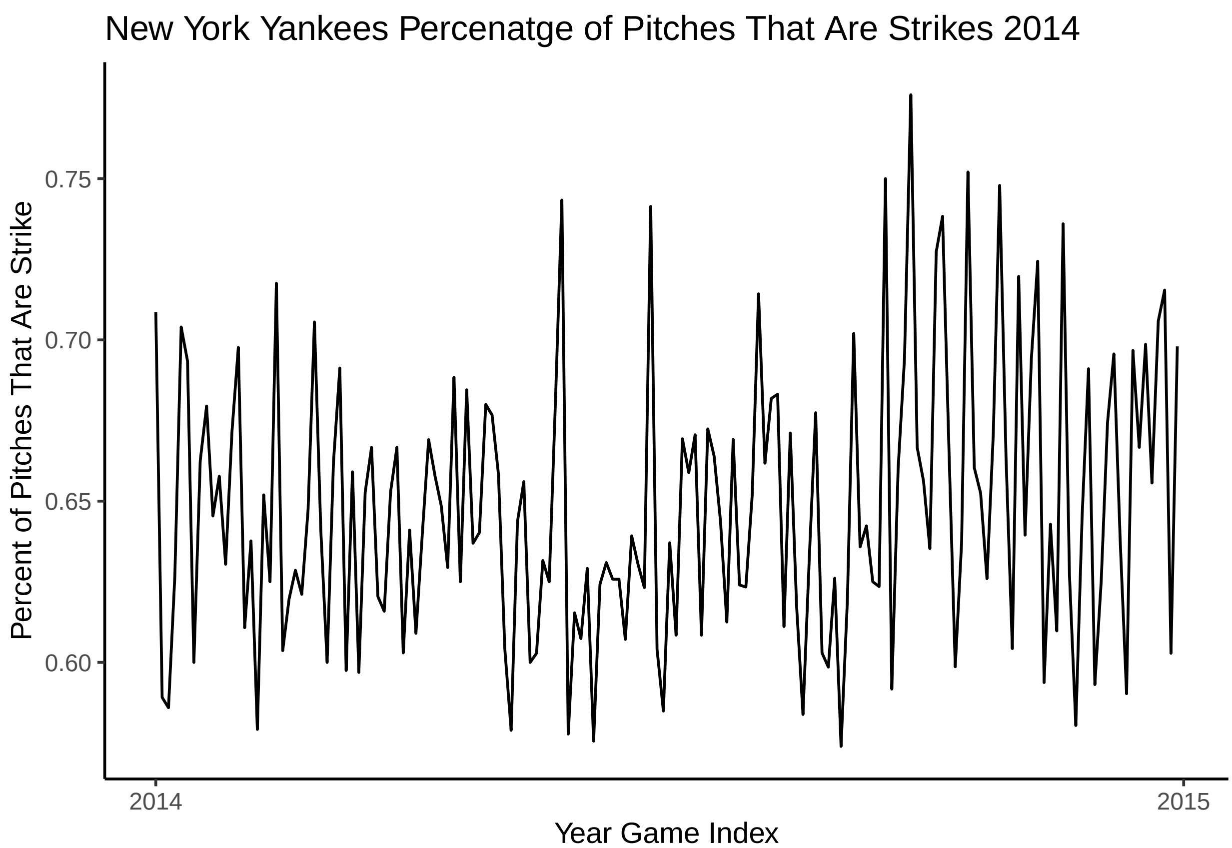
```r
autoplot(NyyIndex2009_2018) + xlab("Year Game Index") +
  ylab("Percent of Pitches That Are Strike") + ggtitle("New York Yankees Percenatge of Pitches That Are
  theme_classic()
```




```r
autoplot(NyyIndex2019) + xlab("Year Game Index") +
  ylab("Percent of Pitches That Are Strike") + ggtitle("New York Yankees Percenatge of Pitches That Are
  theme_classic()
```

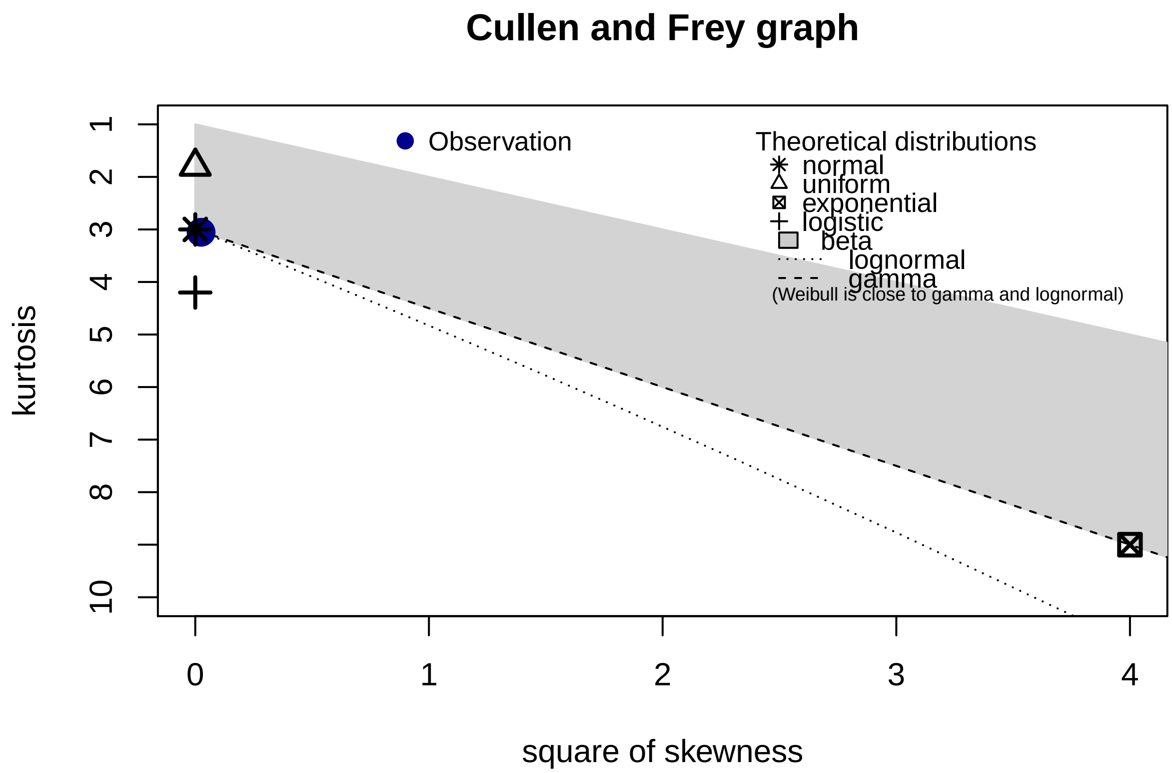



```
Nyy2014 <- data %>%
  dplyr::filter(Team == "NYY" & Year == 2014)
NyyIndex2014 = ts(Nyy2014$PerSK,start=c(2014,1), frequency = 162)
autoplot(NyyIndex2014) + xlab("Year Game Index") +
  ylab("Percent of Pitches That Are Strike") + ggtitle("New York Yankees Percenatge of Pitches That Are
  theme_classic()
```



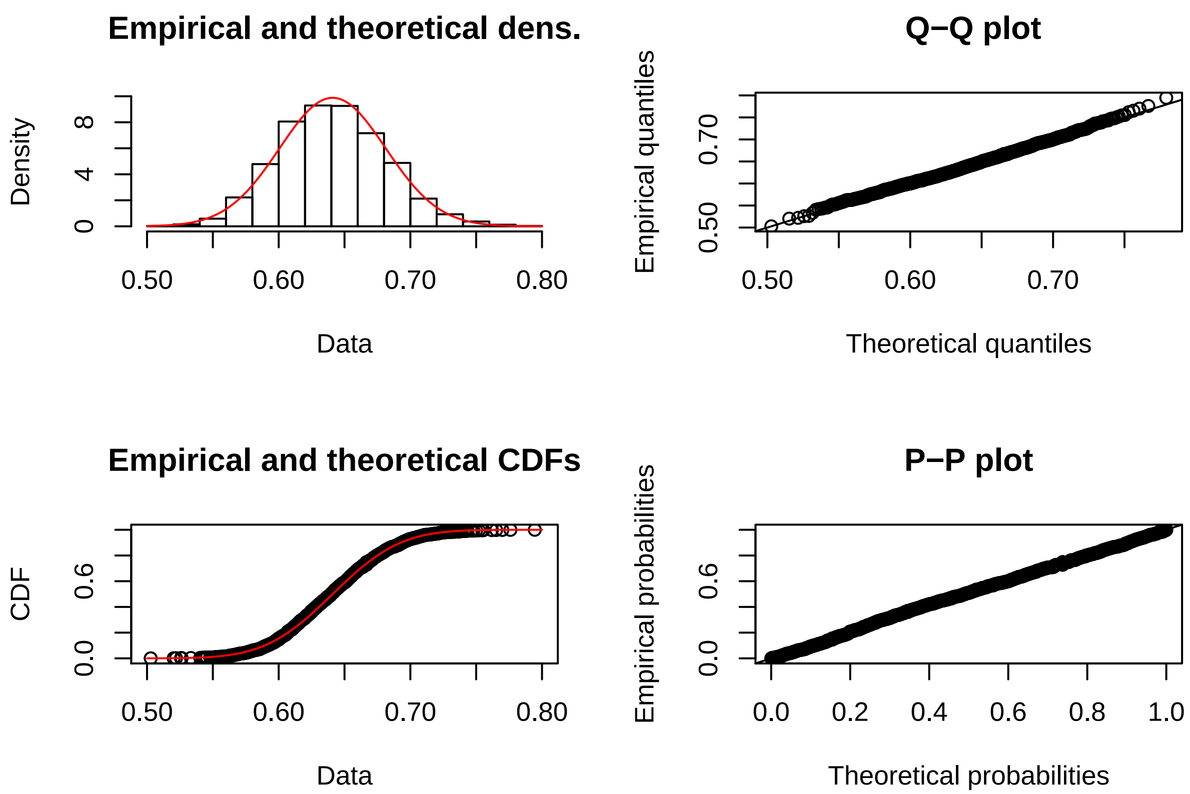

**Check Distribution**

```
descdist(as.numeric(NyyIndex2009_2018), discrete = FALSE)
```

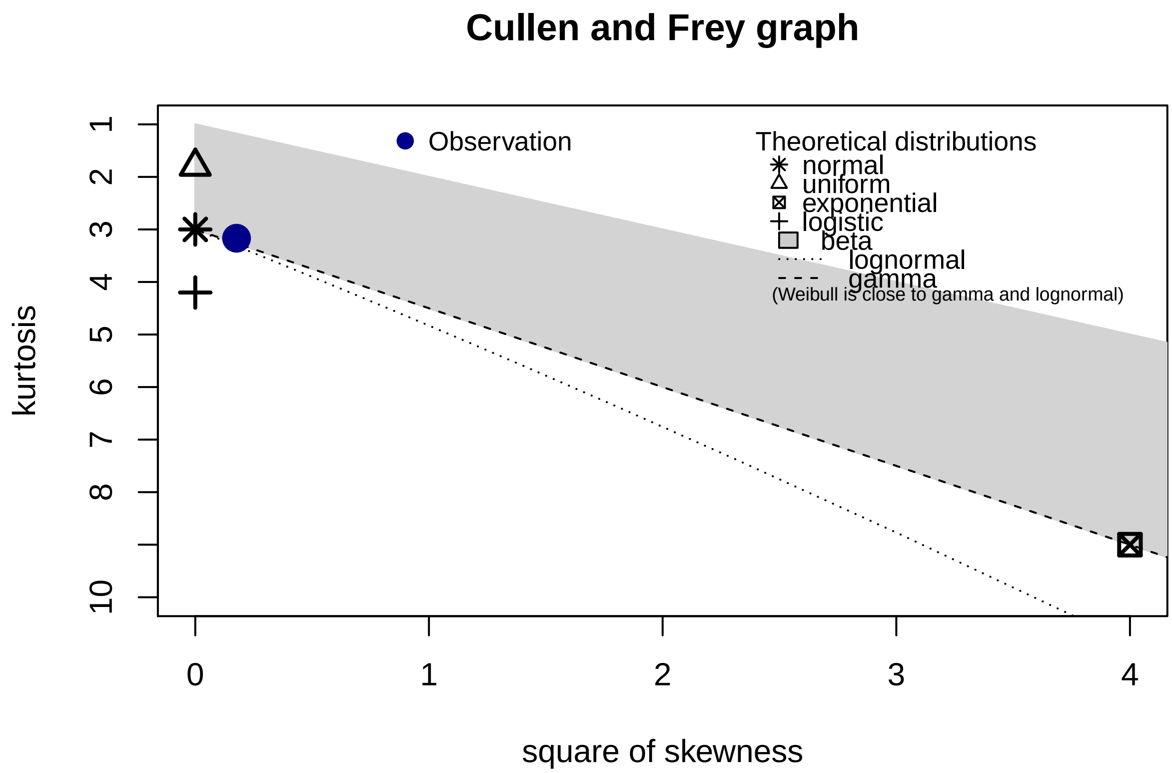



```
## summary statistics
## ------
## min:  0.5027933   max:  0.7945205
## median:  0.64
## mean:  0.6409933
## estimated sd:  0.04039069
## estimated skewness:  0.1581971
## estimated kurtosis:  3.054017
```

```
plot(fitdist(as.numeric(NyyIndex2009_2018), "norm"))
```

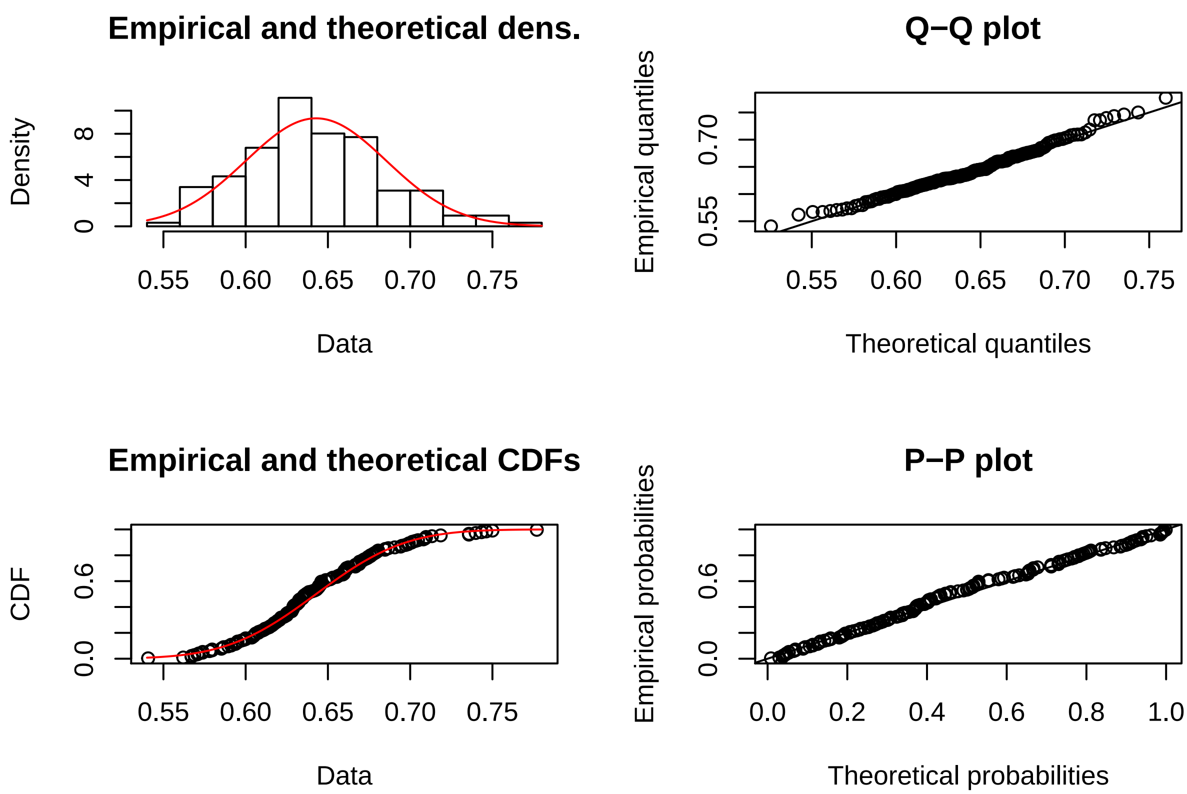



```
descdist(as.numeric(NyyIndex2019), discrete = FALSE)
```




```
## summary statistics
## ------
## min:  0.5406977   max:  0.7769784
## median:  0.6369543
## mean:  0.6428164
## estimated sd:  0.0428578
## estimated skewness:  0.4203004
## estimated kurtosis:  3.167513
```

```
plot(fitdist(as.numeric(NyyIndex2019), "norm"))
```




Based on these graphs it appears that the data is pretty close to a normal distribution.

Training Set:

```
fBasics::normalTest(NyyIndex2009_2018, method = 'jb')
```

```
##
## Title:
##  Jarque - Bera Normalality Test
##
## Test Results:
##   STATISTIC:
##     X-squared: 6.9144
##   P VALUE:
##     Asymptotic p Value: 0.03152
##
## Description:
##  Mon Nov 16 22:05:48 2020 by user:
```

Test Set:

```
fBasics::normalTest(NyyIndex2019, method = "jb")
```

```
##
## Title:
```

```
##  Jarque - Bera Normalality Test
##
## Test Results:
##   STATISTIC:
##     X-squared: 4.7879
##   P VALUE:
##     Asymptotic p Value: 0.09127
##
## Description:
##  Mon Nov 16 22:05:48 2020 by user:
```

The traing set passes the J-B Test but the test set does not.

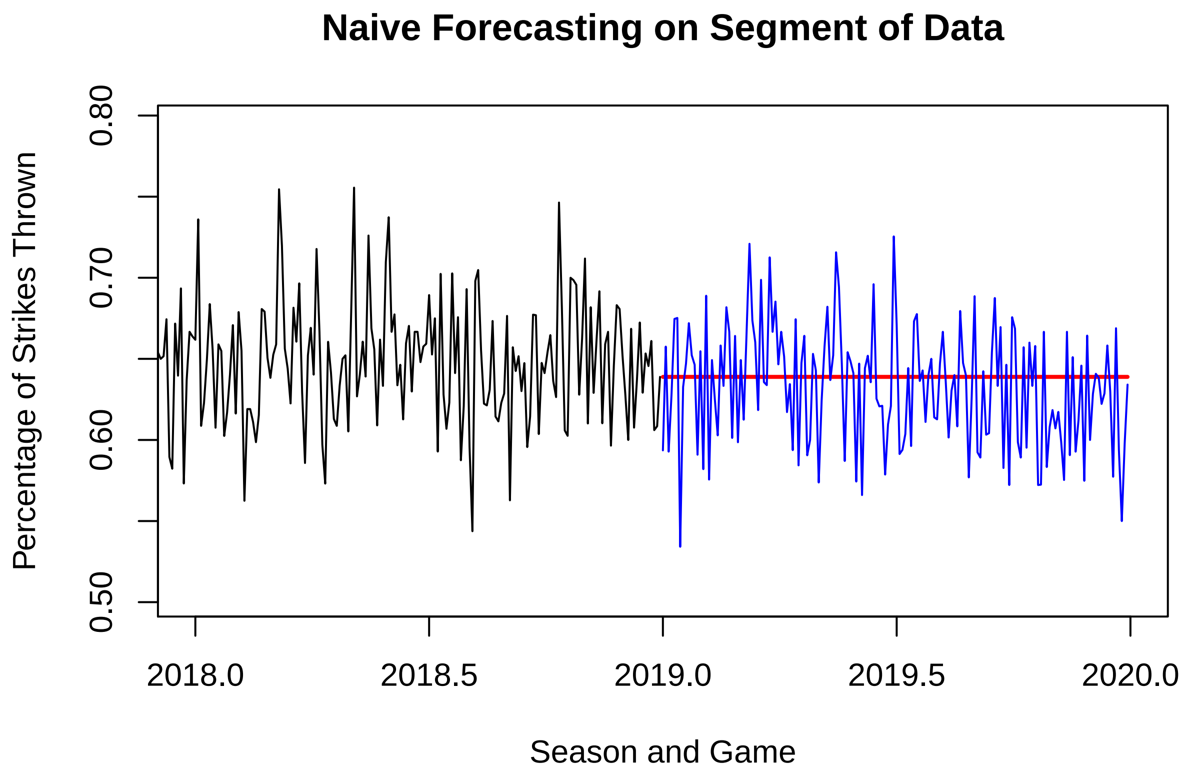
### Naive

Uses the most recent data point as a foreast.

```
NyyNaive <- naive(NyyIndex2009_2018, h = 162)
accuracy(NyyNaive, NyyIndex2019)
```

```
##                        ME       RMSE        MAE        MPE     MAPE      MASE
## Training set 0.0000535773 0.05444616 0.04308236 -0.3514503 6.736134 0.9589081
## Test set     0.0039274640 0.04290545 0.03363661  0.1783240 5.193493 0.7486687
##                     ACF1 Theil's U
## Training set -0.46607350        NA
## Test set      0.03665979 0.7297323
```

```
plot(NyyIndex2009_2018,
     xlim = c(2018.0, 2020.0),
     xlab = "Season and Game",
     ylab = "Percentage of Strikes Thrown",
     main = "Naive Forecasting on Segment of Data")
lines(naive(NyyIndex2009_2018, h=162)$mean, col="red", lwd=2)
lines(BosIndex2019, col="blue")
```



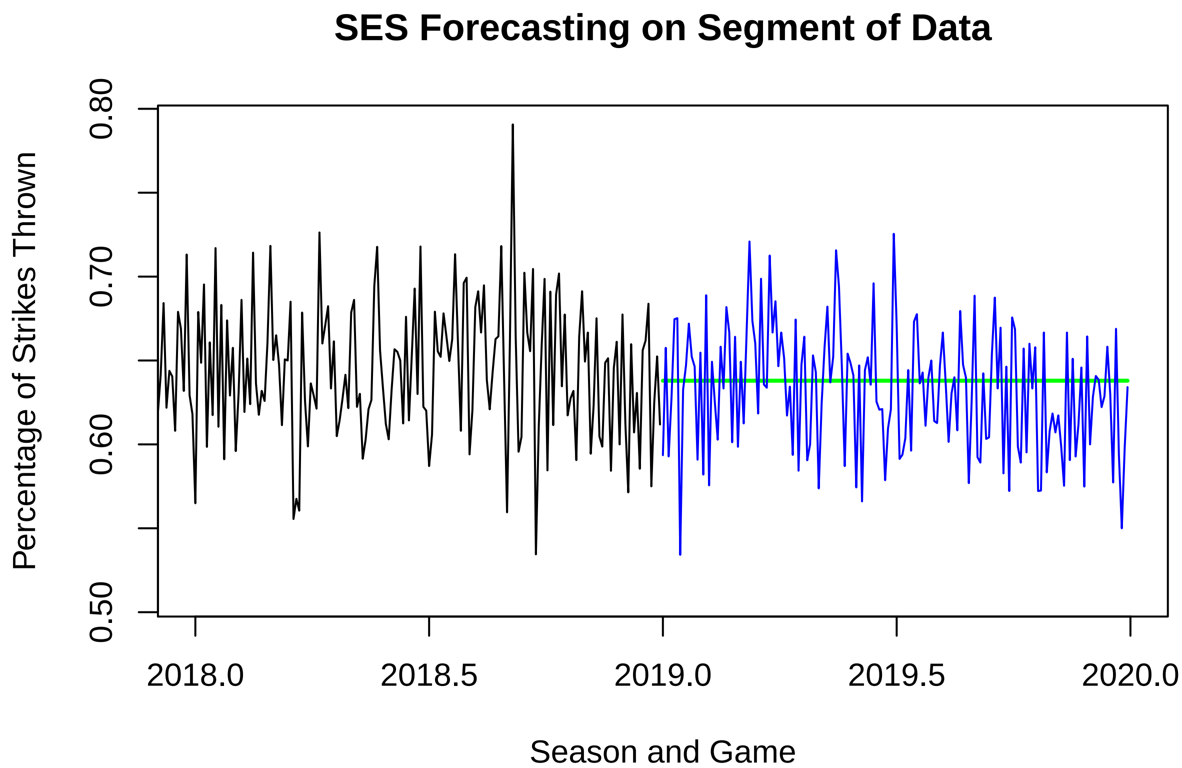

**SES**

Uses the weighted averages of past values to create a falt forecast:

```
NyyModelSES <- ses(NyyIndex2009_2018, h = 162)
accuracy(NyyModelSES, NyyIndex2019)
```

```
##                          ME       RMSE        MAE        MPE     MAPE      MASE
## Training set  0.0009848176 0.03974690 0.03188058 -0.2241641 4.984191 0.7095838
## Test set     -0.0037690740 0.04289124 0.03430517 -1.0242033 5.360313 0.7635494
##                    ACF1 Theil's U
## Training set 0.04235663        NA
## Test set     0.03665979 0.7300607
```

```
plot(BosIndex2009_2018,
     xlim = c(2018.0, 2020.0),
     xlab = "Season and Game",
     ylab = "Percentage of Strikes Thrown",
     main = "SES Forecasting on Segment of Data")
lines(ses(BosIndex2009_2018, h=162)$mean, col="green", lwd=2)
lines(BosIndex2019, col="blue")
```



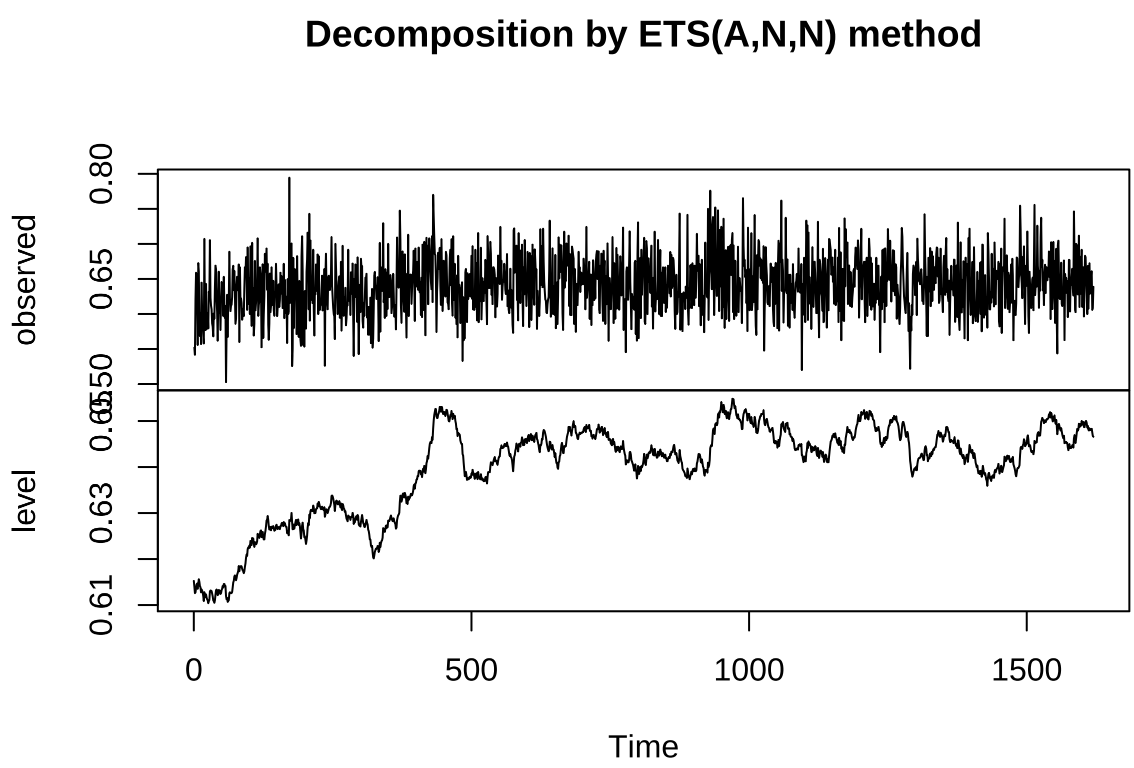

**ETS**

```
NyyETS <- ets(Nyy2009_2018$PerSK)
summary(NyyETS)
```

```
## ETS(A,N,N)
##
## Call:
##  ets(y = Nyy2009_2018$PerSK)
##
##   Smoothing parameters:
##     alpha = 0.0196
##
##   Initial states:
##     l = 0.6152
##
##   sigma:  0.0398
##
##       AIC     AICc      BIC
## 1528.370 1528.385 1544.541
##
## Training set error measures:
##                          ME       RMSE         MAE        MPE     MAPE      MASE
## Training set 0.0009848176 0.0397469 0.03188058 -0.2241641 4.984191 0.7399914
##                      ACF1
```

```
## Training set 0.04235663
```

```r
plot(NyyETS)
```

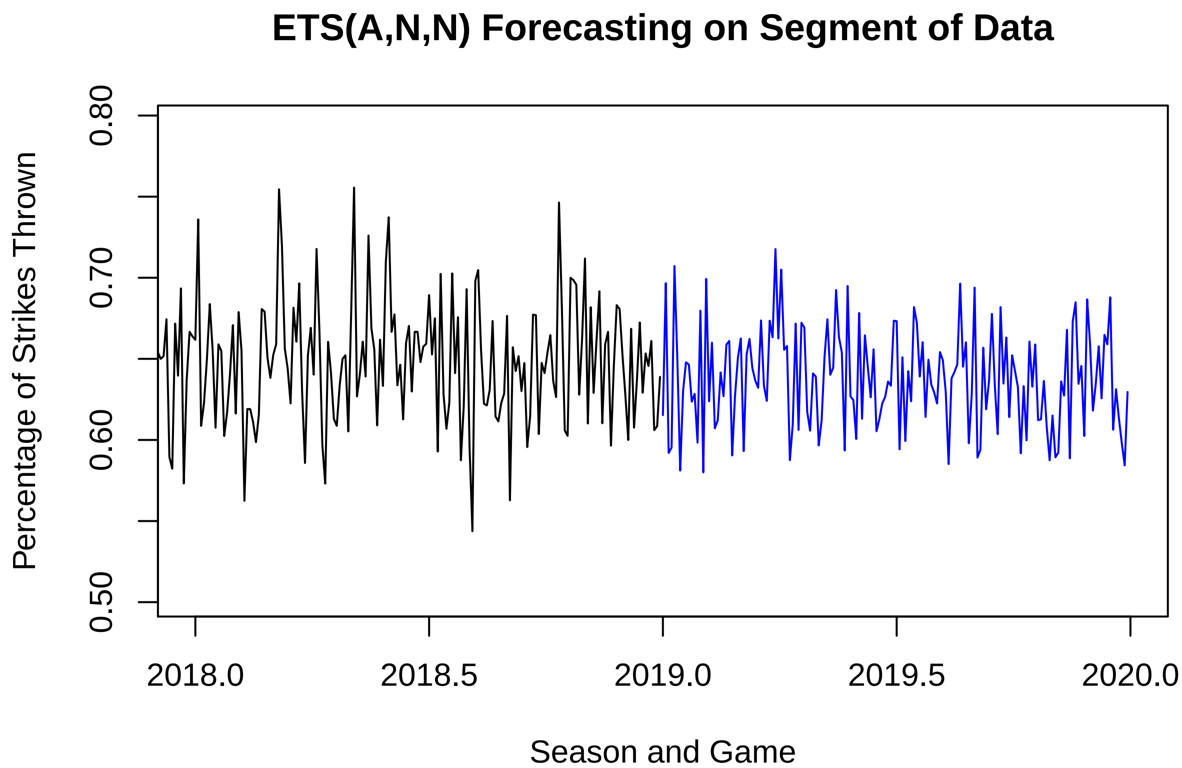



The output of the model is an ETS(A,N,N) model which is additive errors, no trend, and no seasonality.

```r
NyyETSForecast <- forecast(NyyETS, h = 162)
accuracy(NyyETSForecast, Nyy2019$PerSK)
```

```
##                         ME       RMSE        MAE        MPE     MAPE      MASE
## Training set  0.0009848176 0.03974690 0.03188058 -0.2241641 4.984191 0.7399914
## Test set     -0.0037690740 0.04289124 0.03430517 -1.0242033 5.360313 0.7962696
##                    ACF1
## Training set 0.04235663
## Test set             NA
```

```r
plot(NyyIndex2009_2018,
     xlim = c(2018.0, 2020.0),
     xlab = "Season and Game",
     ylab = "Percentage of Strikes Thrown",
     main = "ETS(A,N,N) Forecasting on Segment of Data")
lines(NyyETSForecast$mean, col="cyan", lwd=2)
lines(avgIndex2019, col="blue")
```



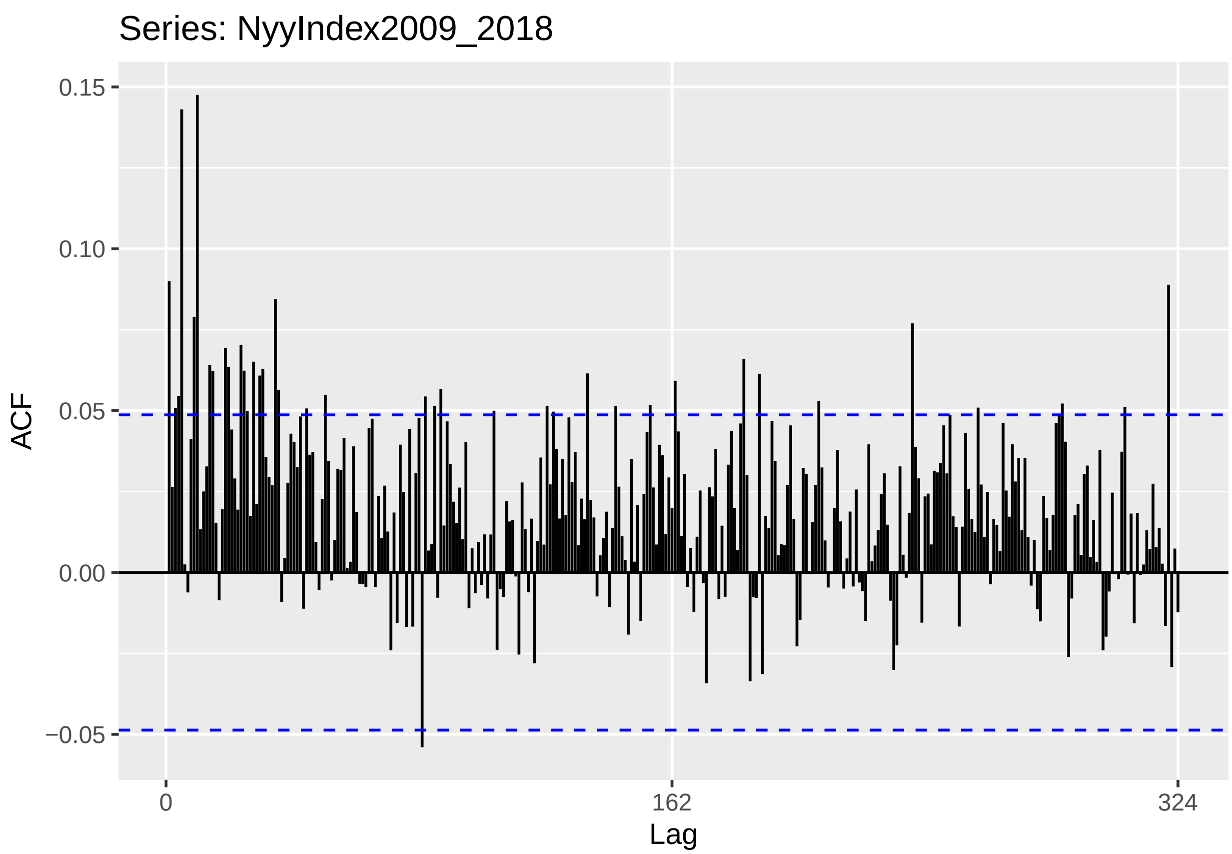

### ACF

```
adf.test(NyyIndex2009_2018)
```

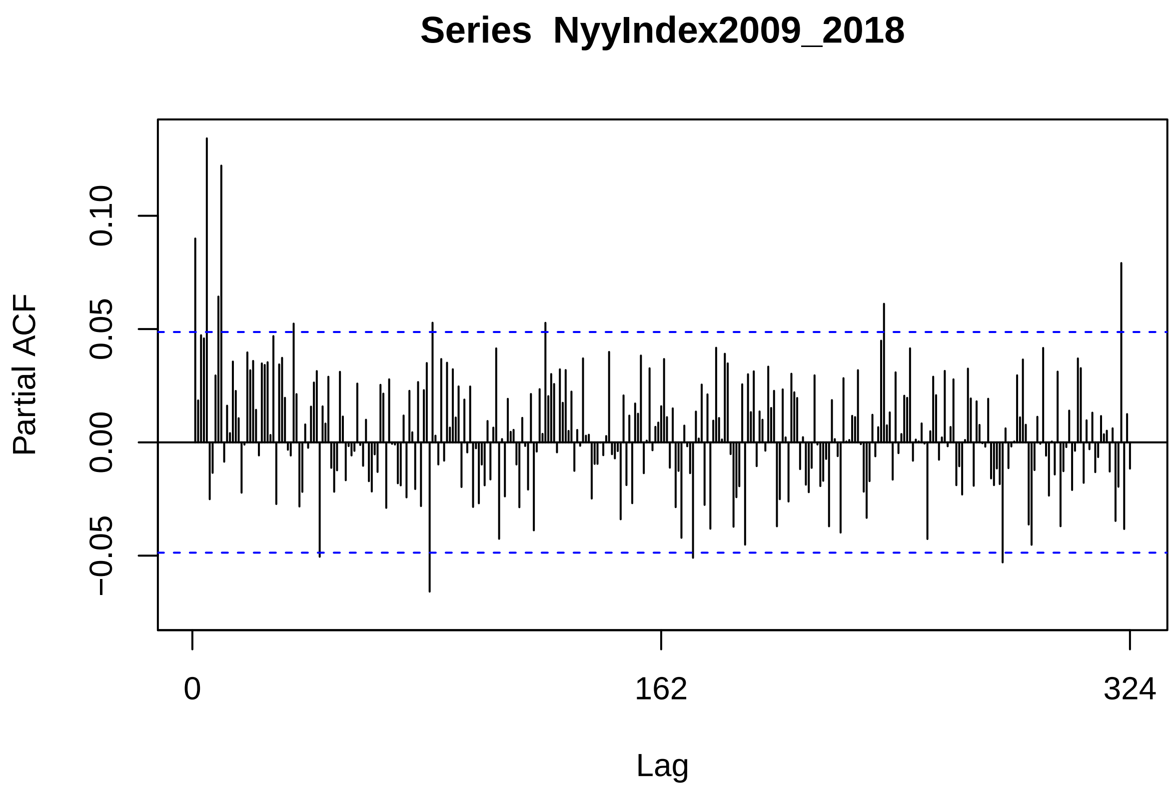
```
## Warning in adf.test(NyyIndex2009_2018): p-value smaller than printed p-value
```

```
##
##  Augmented Dickey-Fuller Test
##
## data:  NyyIndex2009_2018
## Dickey-Fuller = -9.6271, Lag order = 11, p-value = 0.01
## alternative hypothesis: stationary
```

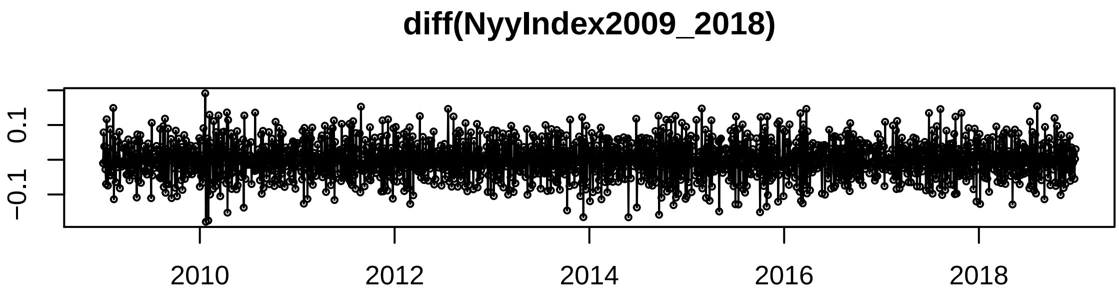
```
ggAcf(NyyIndex2009_2018)
```

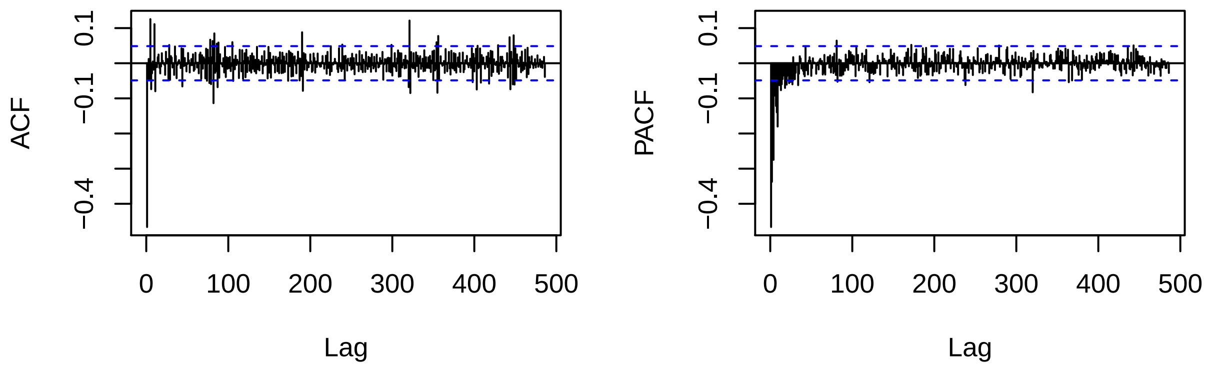



```
Pacf(NyyIndex2009_2018)
```




```
tsdisplay(diff(NyyIndex2009_2018))
```




```r
ndiffs(NyyIndex2009_2018)
```

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

```r
nsdiffs(NyyIndex2009_2018)
```

```
## [1] 0
```

### Arima

The output is a ARIMA(0,1,2) model

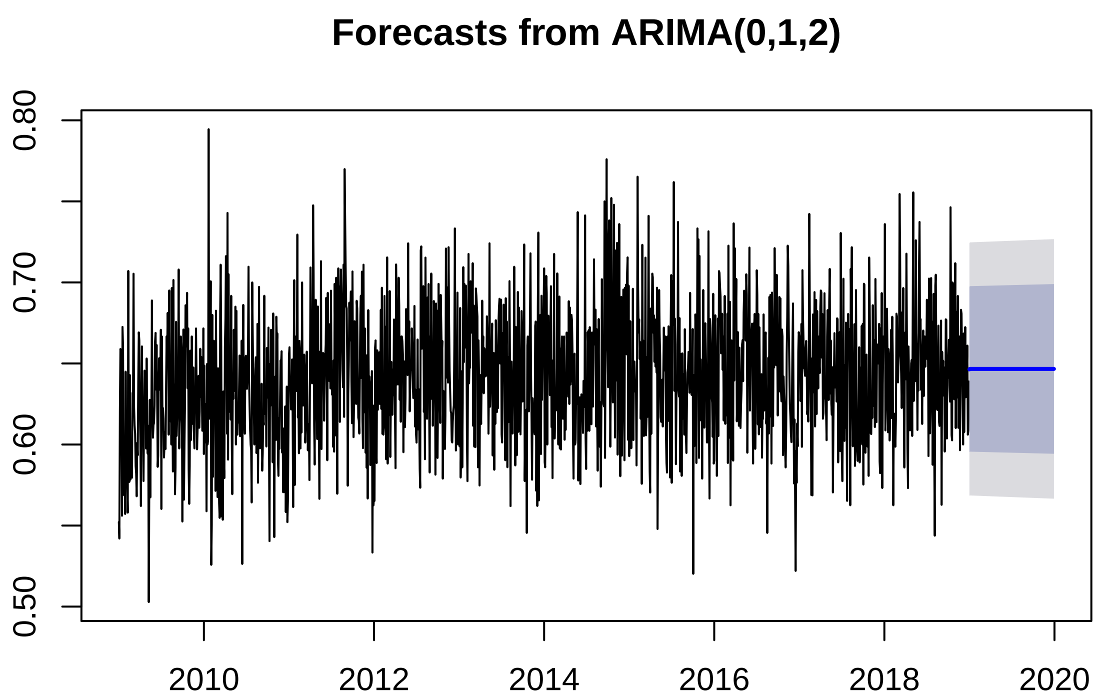
```r
NyyArima <- auto.arima(NyyIndex2009_2018)
summary(NyyArima)
```

```
## Series: NyyIndex2009_2018
## ARIMA(0,1,2)
##
## Coefficients:
##           ma1      ma2
##       -0.9347  -0.0472
## s.e.   0.0254   0.0259
##
## sigma^2 estimated as 0.001579:  log likelihood=2923.92
```

```
## AIC=-5841.84   AICc=-5841.83   BIC=-5825.68
##
## Training set error measures:
##                       ME       RMSE       MAE        MPE     MAPE      MASE
## Training set 0.001372986 0.03970542 0.0317614 -0.1593027 4.961353 0.7069313
##                      ACF1
## Training set -0.001633054
```

Looking at the accuracy:

```r
NyyArimaForecast <- forecast(NyyArima, h = 162)
plot(NyyArimaForecast)
```




```r
accuracy(NyyArimaForecast, NyyIndex2019)
```

```
##                        ME       RMSE        MAE        MPE     MAPE      MASE
## Training set  0.001372986 0.03970542 0.03176140 -0.1593027 4.961353 0.7069313
## Test set     -0.003817333 0.04289528 0.03431328 -1.0317421 5.361977 0.7637297
##                      ACF1 Theil's U
## Training set -0.001633054        NA
## Test set      0.036750745 0.7301394
```

Plotting a portion of the data for easier interpretation:

```r
plot(NyyIndex2009_2018,
     xlim = c(2018.0, 2020.0),
     xlab = "Season and Game",
     ylab = "Percentage of Strikes Thrown",
     main = "ARIMA(0,1,2) Forecasting on Segment of Data")
lines(NyyArimaForecast$mean, col="orange", lwd=2)
lines(NyyIndex2019, col="blue")
```