9750 Final Project

Sierra Levy, Urvaksh Irani, Adela Dervisevic

2022-05-09

Bike Sharing Dataset - Linear Regression and Time Series Modeling

We have chosen to analyze the "London Bike Sharing" dataset which observes the number of bikes rented by day and by hour. The dataset also includes other variables, like humidity, temperature, weathercode, and seasonal aspects like holiday and or season. We are looking to see how we can use machine learning and linear regression analysis to predict bike count and make sure we built a reliable, and useful model.

Reading and cleaning the dataset

First, we'll read in the dataset, make sure it's cleaned (checking for null values, etc.) and ready to analyze. Since the dataframe is large, we're going to table a sample size of 500 rows so we can better read and interpret plots for the linear regression models.

```
library(tidyverse)
library(dplyr)
library(ALSM)
library(ggplot2)
library(lubridate)
library(scales)
library(forecast)
library(tseries)
library(kableExtra)
#setwd("/Users/slevy/Desktop/Homework")

#Reading in the data
df = read_csv('london_merged.csv')

#No NA Values
which(is.na(df), arr.ind=TRUE)
```

```
##      row col
```

```
#creating a smaller data sample to work with for the linear models
df2<-df[sample(1:nrow(df), 500),]
str(df2)
```

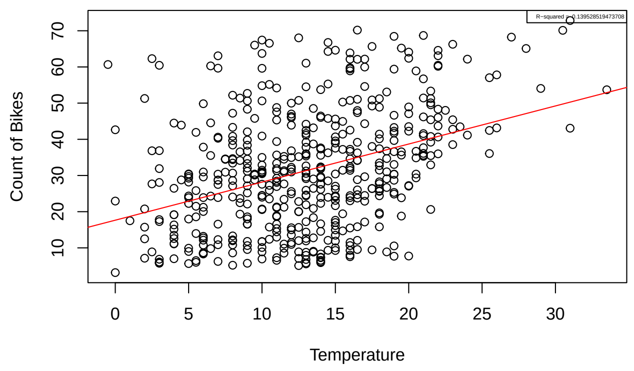
```
## tibble [500 x 10] (S3: tbl_df/tbl/data.frame)
##  $ timestamp   : POSIXct[1:500], format: "2015-10-01 18:00:00" "2015-05-13 14:00:00" ...
##  $ cnt         : num [1:500] 3536 1573 4238 81 923 ...
##  $ t1          : num [1:500] 16 19 28 5 19 11 9.5 8 6 16 ...
##  $ t2          : num [1:500] 16 19 29 3 19 11 9.5 5 2 16 ...
##  $ hum         : num [1:500] 59 33.5 51 81 80.5 74.5 91 87 68.5 68 ...
##  $ wind_speed  : num [1:500] 20 8 6.5 8 19.5 28.5 4 15 26 6 ...
##  $ weather_code: num [1:500] 1 1 1 1 2 3 1 4 1 1 ...
##  $ is_holiday  : num [1:500] 0 0 0 0 0 0 0 0 0 0 ...
##  $ is_weekend  : num [1:500] 0 0 0 0 1 1 0 1 1 0 ...
##  $ season      : num [1:500] 2 0 1 2 1 3 2 3 3 1 ...
```

Model 1 - Regression Analysis

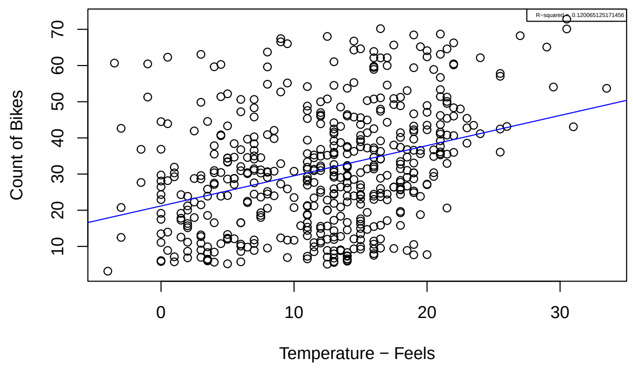
Pick our response and predictor variable

Since the dataset contains two variables that are similar to each other, Temperature and Temperature Feels, we're going to plot and analyze both variables to see which is a better choice as the predictor variable for the model. Additionally, we're going to do a Y transformation so we can better fit the model. After plotting both and fitting the regression lines, we're going to move forward with Temperature as it has a better R-Squared value.

```r
X = df2$t1
X1 = df2$t2

#Transforming the data to better fit the model.
new.y=sqrt(df2$cnt)

m1=lm(new.y~X, data=df2)
m2=lm(new.y~X1, data = df2)

plot(new.y~df2$t1, data=df2, xlab="Temperature", ylab = "Count of Bikes")
abline(m1, col="red")
legend("topright",
  legend=paste("R-squared = ", summary(m1)$r.squared),cex=0.3)

plot(new.y~df2$t2, data=df2, xlab="Temperature - Feels", ylab = "Count of Bikes")
abline(m2, col="blue")
legend("topright",
  legend=paste("R-squared = ", summary(m2)$r.squared),cex=0.3)
```



(a) (1)



(b) (1)

Figure 1: Count of Bike based on temperature

Understanding the relationship between Bike Count and Temperature

To better understand if there is a relationship between Temperature and Bike Count, and if temperature is a good variable to use for our model, we're going to perform a a t-test to help us draw a conclusion. Since our test indicates that there is a relationship between the two variables, this is a good predictor variable to use in our model.

Our hypothesis is as follows:

Ho: Beta1 = 0 - there is not a linear relationship Ha: Beta1 does not equal 0 , there is a linear relationship.

If our t* value is less than or equal to our t-critical value, we'll conclude Ho (that there is not a relationship between Temperature and Bike Count), otherwise, we'll conclude Ha (there is a relationship between the two variables).

Since our t* > t-critical, we'll reject the null hypothesis and conclude that there is a relationship between Bike Count and Temperature.

```
summary(m1)
```

```
##
## Call:
## lm(formula = new.y ~ X, data = df2)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -30.951 -10.864  -1.061   8.936  43.552
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  17.6453     1.6658  10.593   <2e-16 ***
## X             1.0526     0.1171   8.986   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 15.16 on 498 degrees of freedom
## Multiple R-squared:  0.1395, Adjusted R-squared:  0.1378
## F-statistic: 80.75 on 1 and 498 DF,  p-value: < 2.2e-16
```

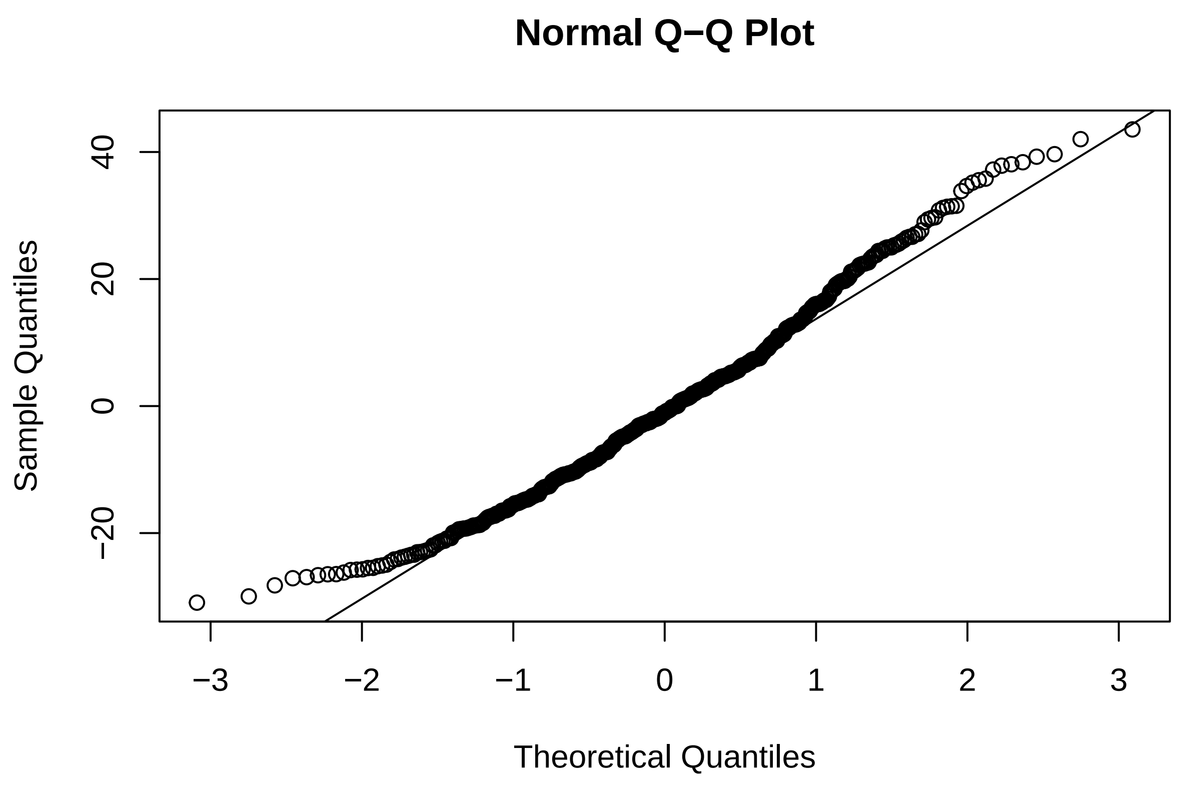
```
qt(.95,498)
```

```
## [1] 1.647919
```

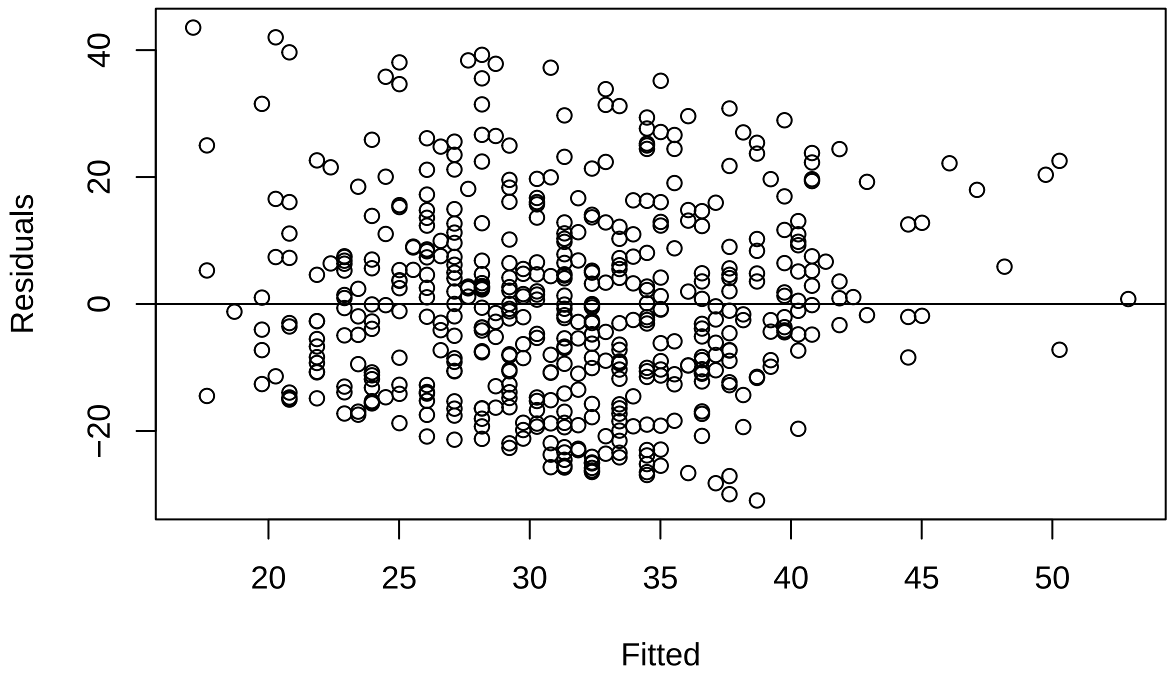
Checking for normality

Looking at the graphs below, we can clearly say that our data is normal. The plot of residuals against the predicted values shows randomly distributed residuals along the baseline 0 and balanced distribution between negative and positive residual values. The QQ plot shows the residuals closely distributed along the straight line indicating that there is no violation of normality.

```
qqnorm(resid(m1))
qqline(resid(m1))
```



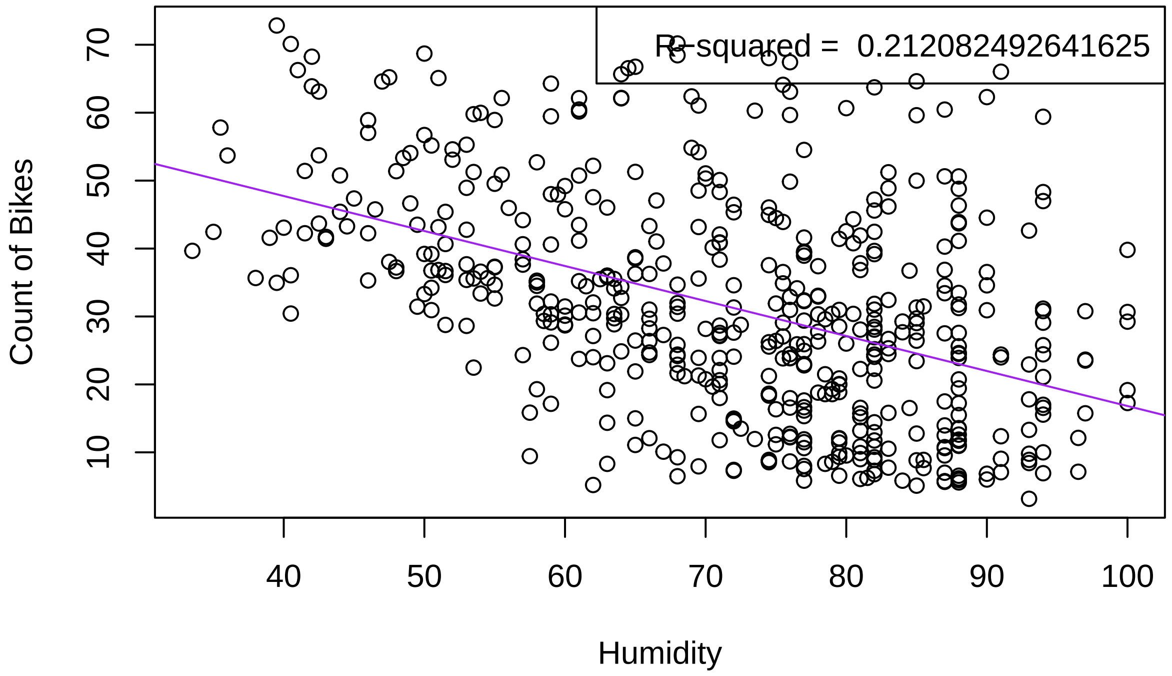
```
plot(fitted(m1), resid(m1),
  xlab = "Fitted", ylab = "Residuals")
abline(h = 0)
```



Analyze the relationship between Bike Count and Humidity and introducing new variables into the model.

There is a negative relationship between bike count and humidity. As humidity rises, bike count decreases. We'll use this variable and introduce into our new model. We'll use both temperature and humidity for our last linear regression model. Introducing both humidity and temperature into the regression model, we see R-squared value increase.

```r
new.y=sqrt(df2$cnt)

plot(new.y~df2$hum, data=df2, xlab="Humidity", ylab = "Count of Bikes")
m3=lm(new.y~df2$hum, data=df2)
abline(m3, col="purple")
legend("topright",
  legend=paste("R-squared = ", summary(m3)$r.squared))
```



```r
new.y=sqrt(df2$cnt)
m4=lm(new.y~df2$hum+df2$t1, data=df2)
summary(m4)$r.squared
```

```
## [1] 0.2423276
```

Automatically find the best fit model with Backward Elimination.

While we can manually look for the best option to help us better understand and predict bike count, we can also use automatic procedures which help us identify the best models and help improve overall adjusted R squared. After performing backward elimination procedure, our best fit model includes Temperature, Humidity, Is Holiday, Is Weekend, and Season. Since adjusted R squared only went up a bit, our analysis shows that we can predict bike count, however, our model only accounts for a small portion of variation by introducing those 4 variables into the model. While our R-squared value is relatively low, some companies might deem that variance OK and still use the model to analyze the relationship. However, more data or other variables might need to be pulled into to better model and predict bike count with linear regression analysis.

```
backward_model<-lm(new.y~df2$t1+df2$t2+df2$hum+df2$wind_speed+df2$weather_code+df2$is_holiday+df2$is_we
summary(backward_model)
```

```
##
## Call:
## lm(formula = new.y ~ df2$t1 + df2$t2 + df2$hum + df2$wind_speed +
##     df2$weather_code + df2$is_holiday + df2$is_weekend + df2$season)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -31.447  -9.574  -2.653   8.390  42.880
##
## Coefficients:
##                  Estimate Std. Error t value Pr(>|t|)
## (Intercept)      46.44811    6.00392   7.736 5.86e-14 ***
## df2$t1            2.32444    0.85479   2.719  0.00677 **
## df2$t2           -1.40743    0.71112  -1.979  0.04835 *
## df2$hum          -0.36331    0.05768  -6.299 6.66e-10 ***
## df2$wind_speed   -0.10533    0.09312  -1.131  0.25856
## df2$weather_code -0.39475    0.28899  -1.366  0.17257
## df2$is_holiday    0.13326    4.64815   0.029  0.97714
## df2$is_weekend   -3.24823    1.48885  -2.182  0.02961 *
## df2$season        0.93398    0.63906   1.461  0.14452
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 14.14 on 491 degrees of freedom
## Multiple R-squared:  0.2621, Adjusted R-squared:  0.2501
## F-statistic:  21.8 on 8 and 491 DF,  p-value: < 2.2e-16
```

```
m.backward<-step(backward_model,direction="backward") # backward elimination by AIC
```

```
## Start:  AIC=2657.74
## new.y ~ df2$t1 + df2$t2 + df2$hum + df2$wind_speed + df2$weather_code +
##     df2$is_holiday + df2$is_weekend + df2$season
##
##                    Df Sum of Sq    RSS    AIC
## - df2$is_holiday    1       0.2  98133 2655.7
## - df2$wind_speed    1     255.7  98389 2657.0
## - df2$weather_code  1     372.9  98506 2657.6
## <none>                           98133 2657.7
## - df2$season        1     426.9  98560 2657.9
## - df2$t2            1     782.9  98916 2659.7
## - df2$is_weekend    1     951.3  99084 2660.6
## - df2$t1            1    1477.9  99611 2663.2
## - df2$hum           1    7930.5 106064 2694.6
##
## Step:  AIC=2655.74
## new.y ~ df2$t1 + df2$t2 + df2$hum + df2$wind_speed + df2$weather_code +
##     df2$is_weekend + df2$season
##
##                    Df Sum of Sq    RSS    AIC
## - df2$wind_speed    1     255.7  98389 2655.0
## - df2$weather_code  1     373.0  98506 2655.6
## <none>                           98133 2655.7
## - df2$season        1     440.4  98574 2656.0
## - df2$t2            1     787.6  98921 2657.7
## - df2$is_weekend    1     959.0  99092 2658.6
## - df2$t1            1    1489.8  99623 2661.3
## - df2$hum           1    7939.0 106072 2692.6
##
## Step:  AIC=2655.04
## new.y ~ df2$t1 + df2$t2 + df2$hum + df2$weather_code + df2$is_weekend +
##     df2$season
##
##                    Df Sum of Sq    RSS    AIC
## - df2$season        1     364.5  98753 2654.9
## <none>                           98389 2655.0
## - df2$weather_code  1     523.7  98913 2655.7
## - df2$t2            1     626.9  99016 2656.2
## - df2$is_weekend    1     967.0  99356 2657.9
## - df2$t1            1    1285.0  99674 2659.5
## - df2$hum           1    7683.4 106072 2690.6
##
## Step:  AIC=2654.89
## new.y ~ df2$t1 + df2$t2 + df2$hum + df2$weather_code + df2$is_weekend
##
##                    Df Sum of Sq    RSS    AIC
## <none>                           98753 2654.9
## - df2$weather_code  1     587.9  99341 2655.8
## - df2$t2            1     716.8  99470 2656.5
## - df2$is_weekend    1     958.0  99711 2657.7
## - df2$t1            1    1356.7 100110 2659.7
## - df2$hum           1    7328.6 106082 2688.7
```

```
summary(m.backward)
```

```
##
## Call:
## lm(formula = new.y ~ df2$t1 + df2$t2 + df2$hum + df2$weather_code +
##     df2$is_weekend)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -31.111  -9.521  -2.719   9.313  43.619
##
## Coefficients:
##                  Estimate Std. Error t value Pr(>|t|)
## (Intercept)       45.5508     5.8095   7.841 2.78e-14 ***
## df2$t1             2.1515     0.8258   2.605  0.00946 **
## df2$t2            -1.2996     0.6863  -1.894  0.05886 .
## df2$hum           -0.3379     0.0558  -6.055 2.79e-09 ***
## df2$weather_code  -0.4849     0.2828  -1.715  0.08700 .
## df2$is_weekend    -3.2496     1.4844  -2.189  0.02906 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 14.14 on 494 degrees of freedom
## Multiple R-squared:  0.2575, Adjusted R-squared:  0.2499
## F-statistic: 34.26 on 5 and 494 DF,  p-value: < 2.2e-16
```

Model 2

Building time-series analysis

In order to build a time-series analysis, we are first going to clean and prepare the data for the time series analysis. We need to break out the date and time, sum the total number of count for the day, then we can plot and observe our data. We have grouped the dates and summed the count column, and for rest of the columns we have used Mean to convert our dataset from hourly data to daily data. We have also removed weather code column as it did not have any impact on the count.

```
df$hour  <- hour(ymd_hms(df$timestamp))
df$timestamp <- format(as.POSIXct(df$timestamp,format='%Y-%m-%d %H:%M:%S'),format='%Y-%m-%d')
df$date = df$timestamp
str(df)
```

```
## spec_tbl_df [17,414 x 12] (S3: spec_tbl_df/tbl_df/tbl/data.frame)
##  $ timestamp   : chr [1:17414] "2015-01-04" "2015-01-04" "2015-01-04" "2015-01-04" ...
##  $ cnt         : num [1:17414] 182 138 134 72 47 46 51 75 131 301 ...
##  $ t1          : num [1:17414] 3 3 2.5 2 2 2 1 1 1.5 2 ...
##  $ t2          : num [1:17414] 2 2.5 2.5 2 0 2 -1 -1 -1 -0.5 ...
##  $ hum         : num [1:17414] 93 93 96.5 100 93 93 100 100 96.5 100 ...
##  $ wind_speed  : num [1:17414] 6 5 0 0 6.5 4 7 7 8 9 ...
##  $ weather_code: num [1:17414] 3 1 1 1 1 1 4 4 4 3 ...
##  $ is_holiday  : num [1:17414] 0 0 0 0 0 0 0 0 0 0 ...
##  $ is_weekend  : num [1:17414] 1 1 1 1 1 1 1 1 1 1 ...
##  $ season      : num [1:17414] 3 3 3 3 3 3 3 3 3 3 ...
##  $ hour        : int [1:17414] 0 1 2 3 4 5 6 7 8 9 ...
##  $ date        : chr [1:17414] "2015-01-04" "2015-01-04" "2015-01-04" "2015-01-04" ...
##  - attr(*, "spec")=
##   .. cols(
##   ..   timestamp = col_datetime(format = ""),
##   ..   cnt = col_double(),
##   ..   t1 = col_double(),
##   ..   t2 = col_double(),
##   ..   hum = col_double(),
##   ..   wind_speed = col_double(),
##   ..   weather_code = col_double(),
##   ..   is_holiday = col_double(),
##   ..   is_weekend = col_double(),
##   ..   season = col_double()
##   .. )
##  - attr(*, "problems")=<externalptr>
```

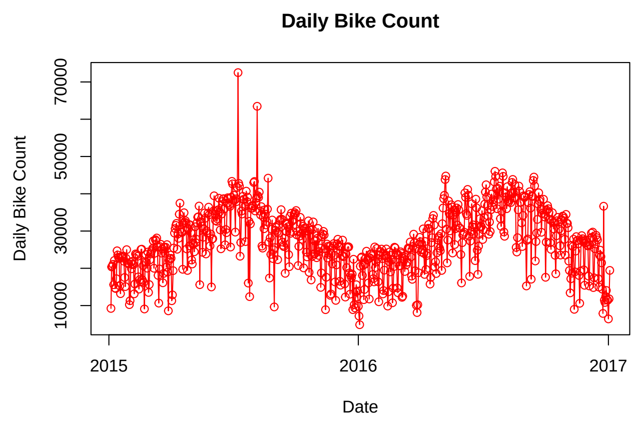
```
df = select(df,-c(weather_code,hour))
df1 = select(df,c(date,cnt))
df1 = aggregate(df1$cnt, list(df1$date), FUN=sum)
df1 = df1 %>%
  rename(
    date = Group.1,
    cnt = x)
df2 = select(df,c(date,t1,t2,hum,wind_speed,is_holiday,is_weekend,season))
df2 = df2 %>%
  group_by(date) %>%
  summarise_at(vars(t1,t2,hum,wind_speed,is_holiday,is_weekend,season),list(name = mean))
nd = inner_join(df1,df2,by='date')
nd$Date = nd$date
nd$Date= as.Date(nd$Date)
str(nd)
```

```
## 'data.frame':    730 obs. of  10 variables:
##  $ date           : chr  "2015-01-04" "2015-01-05" "2015-01-06" "2015-01-07" ...
##  $ cnt            : num  9234 20372 20613 21064 15601 ...
##  $ t1_name        : num  2.48 8.04 7.85 7.46 9.75 ...
##  $ t2_name        : num  0.646 6.708 5.333 4.5 7.792 ...
##  $ hum_name       : num  94.3 80.3 78.9 78.1 79.3 ...
##  $ wind_speed_name: num  7.5 8.85 16 19.76 20.48 ...
##  $ is_holiday_name: num  0 0 0 0 0 0 0 0 0 0 ...
##  $ is_weekend_name: num  1 0 0 0 0 0 1 1 0 0 ...
##  $ season_name    : num  3 3 3 3 3 3 3 3 3 3 ...
##  $ Date           : Date, format: "2015-01-04" "2015-01-05" ...
```

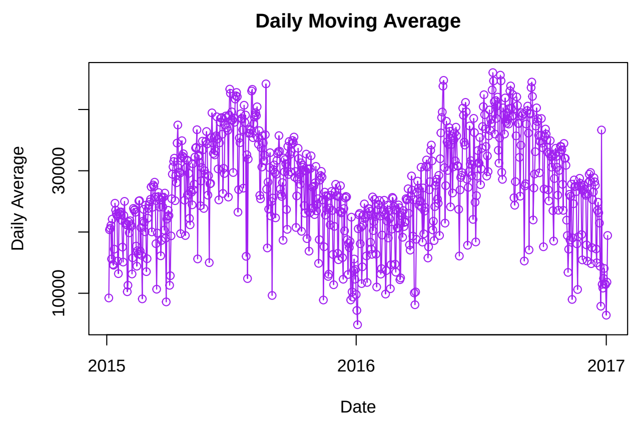
EDA

Looking at the plots below, we can see a lot of up and down movement with the daily bike count, and it is hard to tell if there are any patterns with the data. It looks like there could potentially be some seasonality to the data, which could be due to monthly, weekly, or daily patterns. Plotting the data also helps us identify outliers as you can see with some of the dates. After we clean the data, we can see the plot improves. Lastly, since we can see there is a lot of variance in the daily bike count, we will explore Daily, Weekly, and Monthly moving averages to see what is better to run a time series analysis on. Looking at the moving average plots, it appears it will be best to work with either Weekly or Monthly moving averages as they have a smoother line.

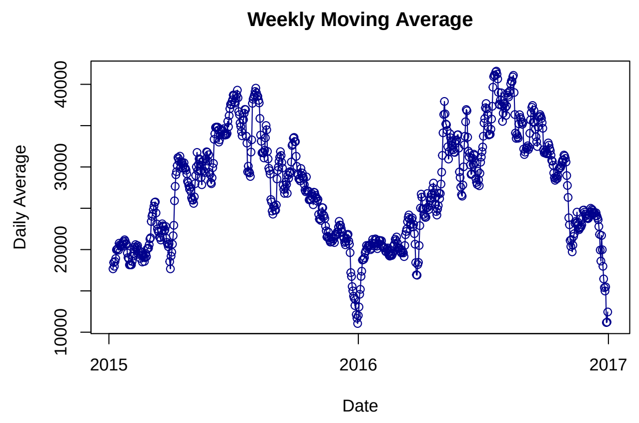
```
#Plot the data
plot(nd$cnt~nd$Date, data = nd, type = "o", col = "red", xlab = "Date", ylab = "Daily Bike Count", main

#cleaning the data
time_series_data = ts(nd[, c('cnt')])
nd$cleaned_count = tsclean(time_series_data)

#Creating moving averages
nd$daily_moving_average = ma(nd$cleaned_count, order=1)
nd$weekly_moving_average = ma(nd$cleaned_count, order=7)
nd$monthly_moving_average = ma(nd$cleaned_count, order=30)

#Plot Daily MA
plot(y = nd$daily_moving_average, x = nd$Date, data = nd, type = "o", col = "purple", xlab = "Date", yl
     main = "Daily Moving Average")
#Plot Weekly MA
plot(y = nd$weekly_moving_average, x= nd$Date, data = nd, type = "o", col = "dark blue", xlab = "Date",
     main = "Weekly Moving Average")
#Plot Monthly MA
plot(y = nd$monthly_moving_average, x =nd$Date, data = nd, type = "o", col = "orange", xlab = "Date", y
     main = "Monthly Moving Average")
```



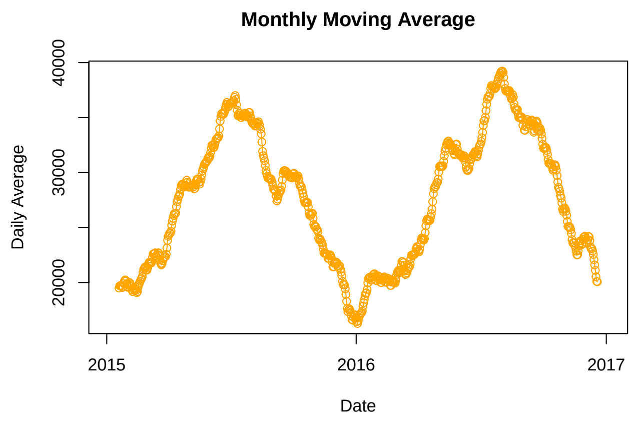
(a) (Daily)



(b) (Daily after cleaning)



(c) (Weekly Moving Averages)



(d) (Monthly Moving Averages)

Figure 2: Moving Averages

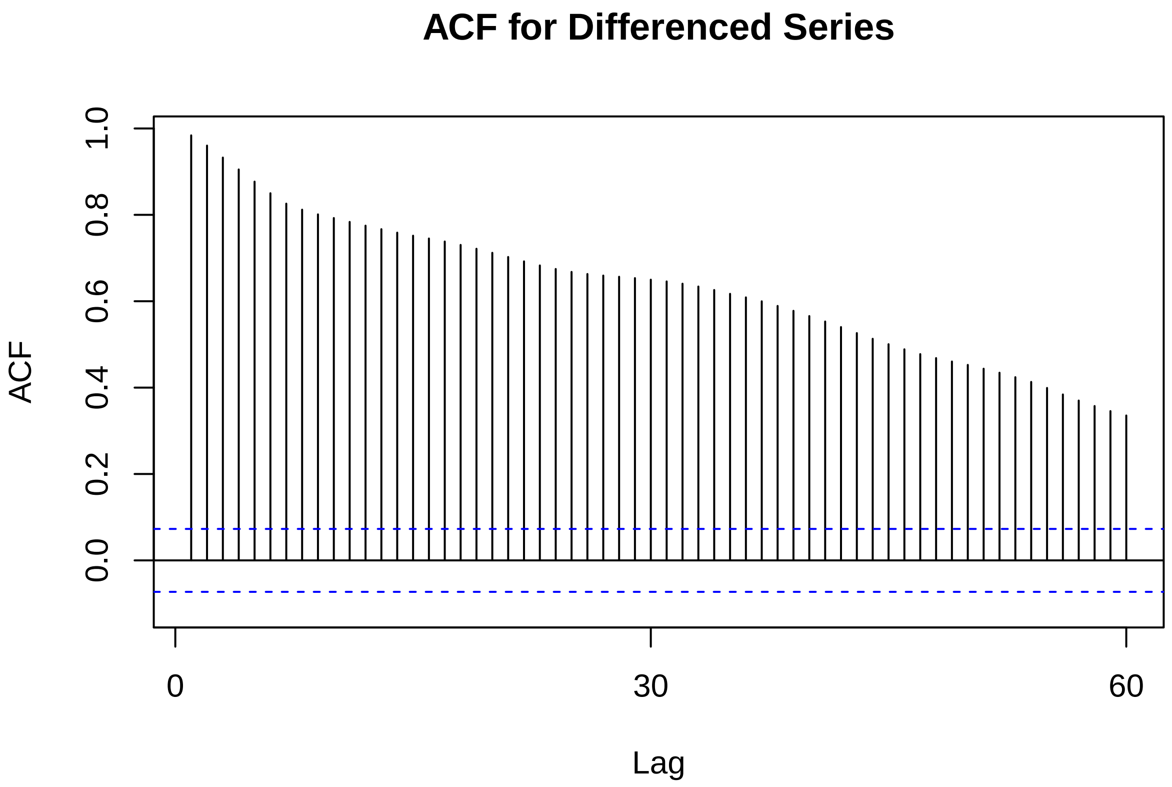
Testing the data

Using the ADF test, we have a high p value indicating our data is not stationary and that we need to differentiate the data. After differentiating the data, we see that our p-value is low and the data is now stationary. ACF lag indicates spikes at last a 1,2,3,6,7 and so on. We can see in the partial ACF significant spikes at lag 1 and 7. Because of this we are most likely going to use AR or MA components in the analysis. Lastly, we can see with the Ljung Box test that the p value is very small, indicating that it is not white noise.

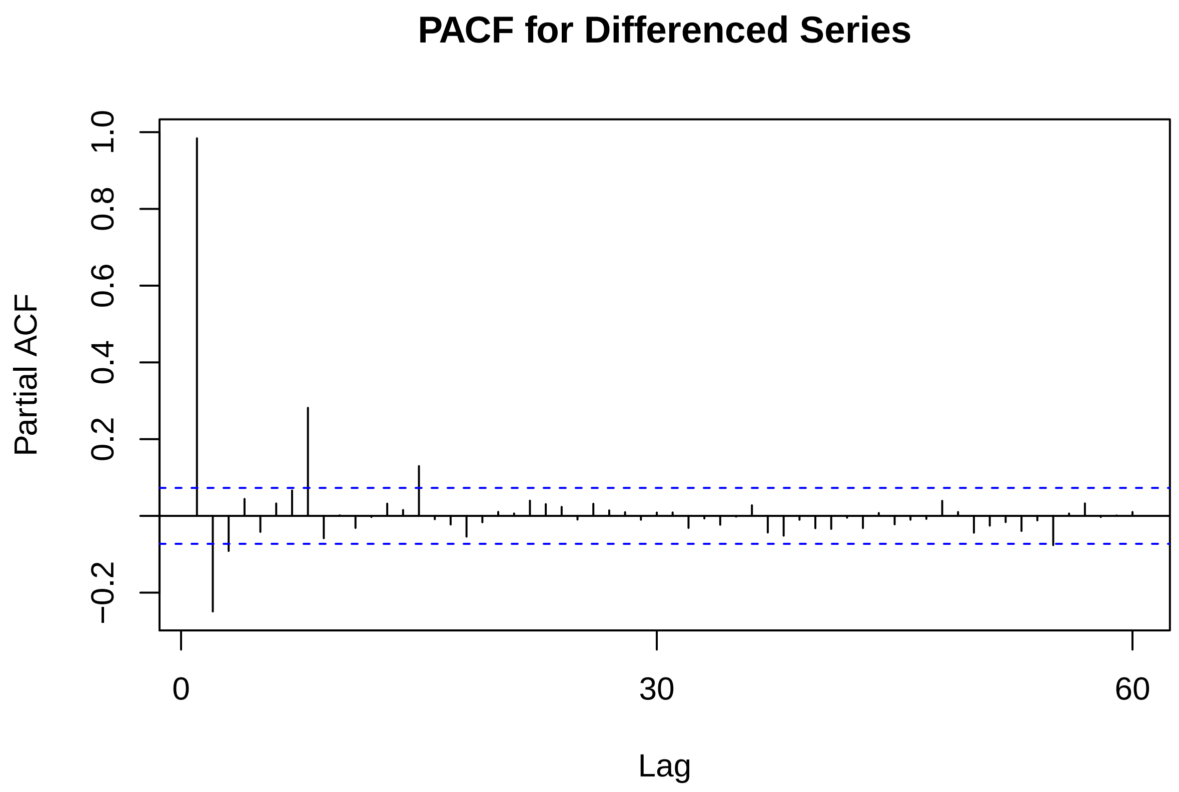
```
final_data = ts(na.omit(nd$weekly_moving_average), frequency=30)

#Testing to see if data is stationary
adf.test(final_data, alternative = "stationary")
```

```
##
##  Augmented Dickey-Fuller Test
##
## data:  final_data
## Dickey-Fuller = -1.604, Lag order = 8, p-value = 0.7459
## alternative hypothesis: stationary
```

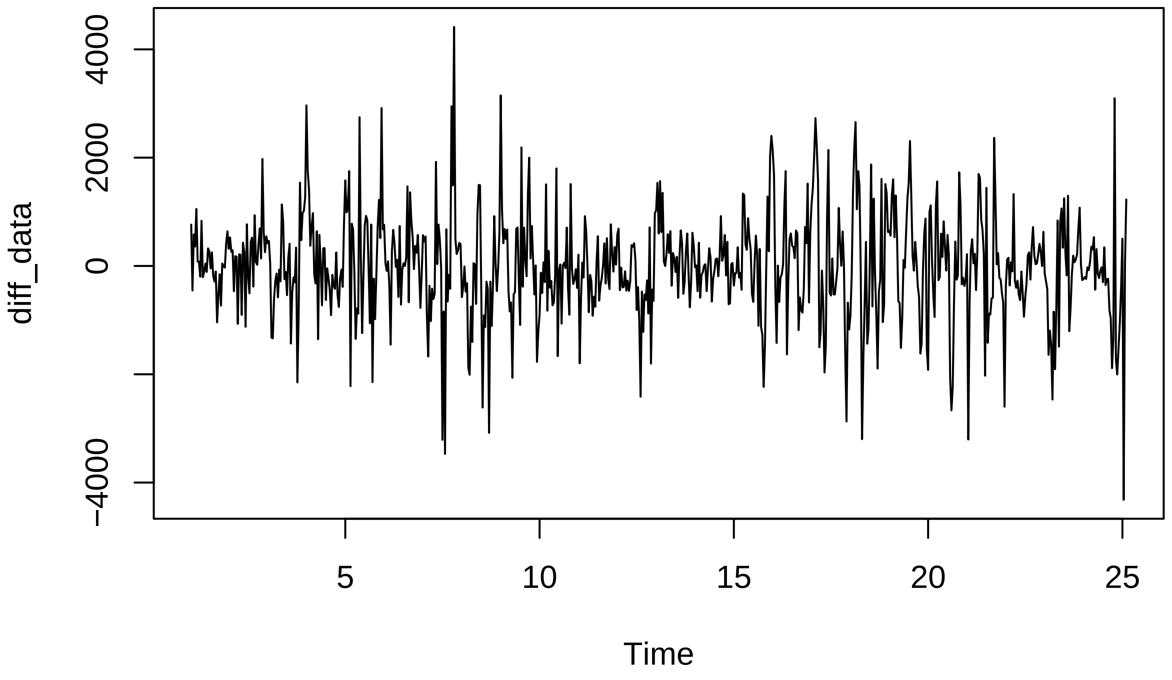
```
#Testing for white noise
Acf(final_data, main='ACF for Differenced Series')
```



```
Pacf(final_data, main='PACF for Differenced Series')
```



```
#Difference
diff_data = diff(final_data)
plot(diff_data)
```

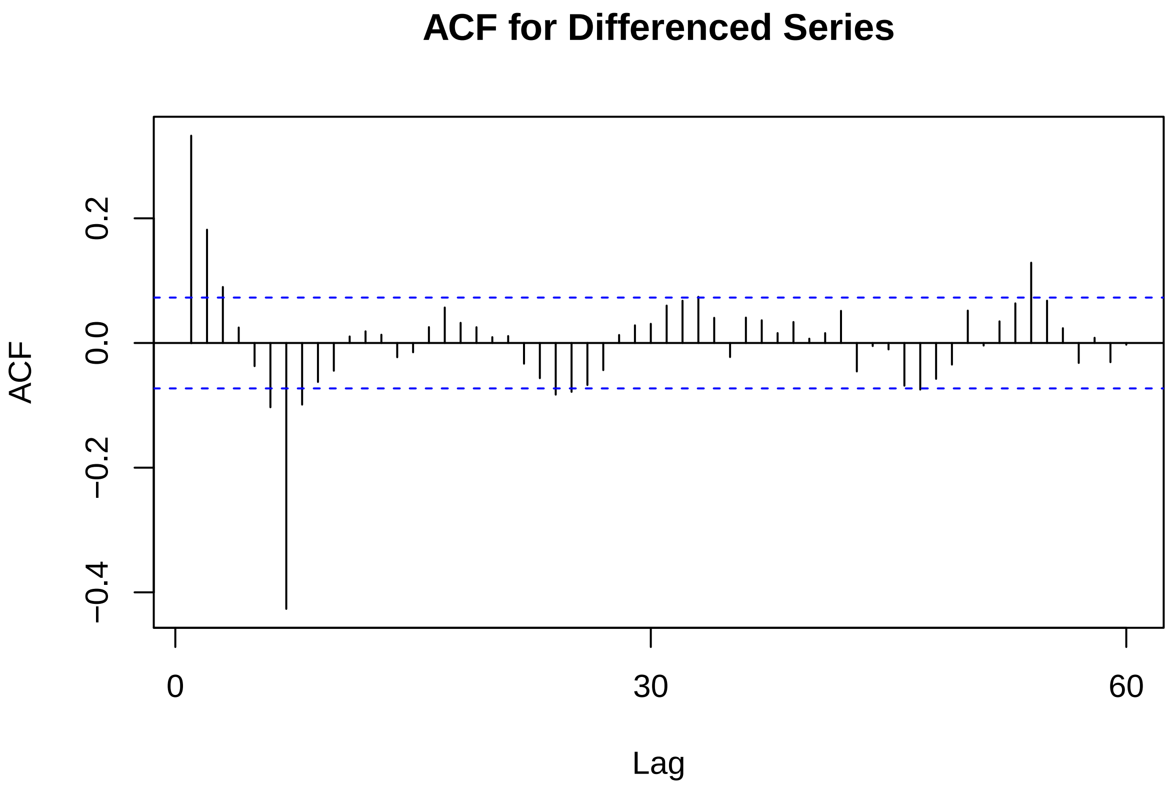


```
adf.test(diff_data, alternative = "stationary")
```

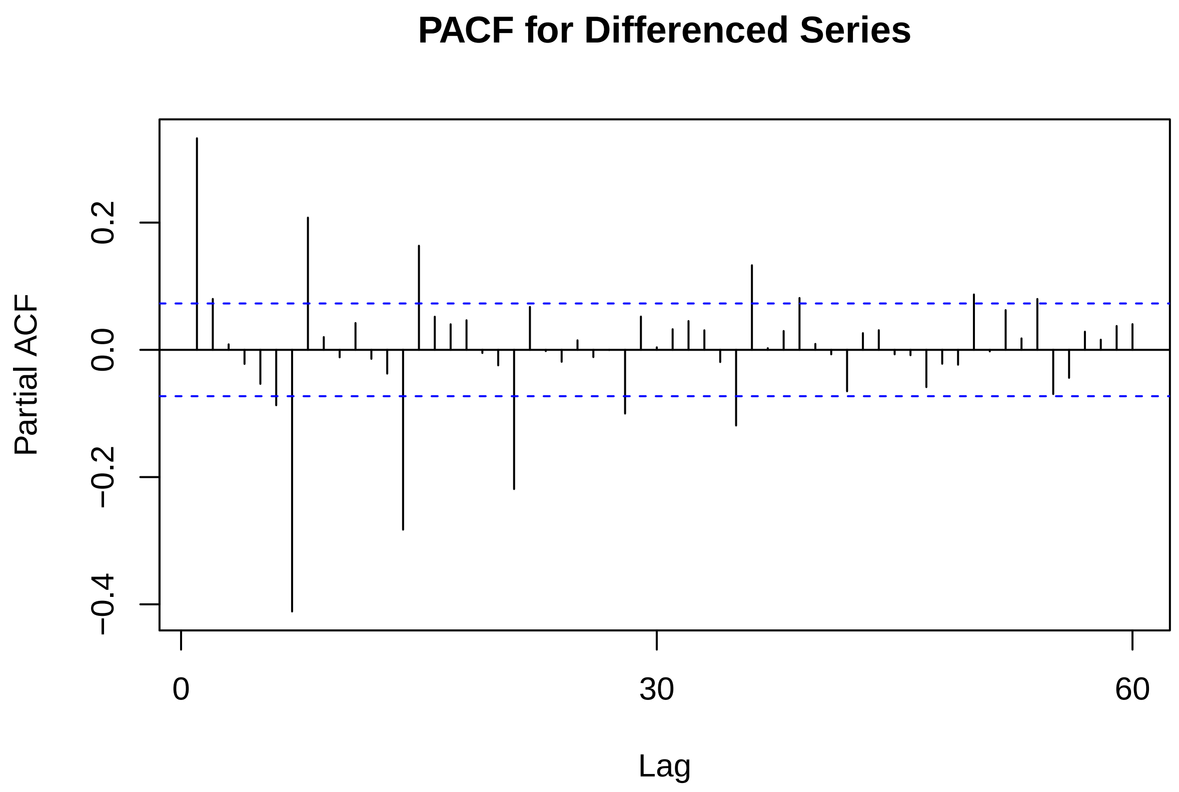
```
##
##  Augmented Dickey-Fuller Test
```

```
##
## data:  diff_data
## Dickey-Fuller = -10.053, Lag order = 8, p-value = 0.01
## alternative hypothesis: stationary
```

```
#Re-testing lag
Acf(diff_data, main='ACF for Differenced Series')
```



```
Pacf(diff_data, main='PACF for Differenced Series')
```



```
#Ljung Box Test
Box.test(diff(diff_data), lag = 10, type = "Ljung-Box")
```

```
##
##  Box-Ljung test
##
## data:  diff(diff_data)
## X-squared = 345.85, df = 10, p-value < 2.2e-16
```

Building our time series models

Since we are now confident that our data is stationary, we are going to use the forecast function to automatically identify the best model for us. After auto-calibrating the model, we end up with ARIMA(2,0,0) as the best option. Our first model coefficient is .3065.

```
auto.arima(diff_data, trace=TRUE)
```

```
##
##  Fitting models using approximations to speed things up...
##
##  ARIMA(2,0,2)(1,0,1)[30] with non-zero mean : 11951.76
##  ARIMA(0,0,0)            with non-zero mean : 12005.2
##  ARIMA(1,0,0)(1,0,0)[30] with non-zero mean : 11948.42
##  ARIMA(0,0,1)(0,0,1)[30] with non-zero mean : 11940.89
##  ARIMA(0,0,0)            with zero mean     : 12003.23
##  ARIMA(0,0,1)            with non-zero mean : 11939.02
##  ARIMA(0,0,1)(1,0,0)[30] with non-zero mean : 11964.17
##  ARIMA(0,0,1)(1,0,1)[30] with non-zero mean : 11964.41
##  ARIMA(1,0,1)            with non-zero mean : 11920.49
##  ARIMA(1,0,1)(1,0,0)[30] with non-zero mean : 11946.4
##  ARIMA(1,0,1)(0,0,1)[30] with non-zero mean : 11922.51
##  ARIMA(1,0,1)(1,0,1)[30] with non-zero mean : 11947.28
##  ARIMA(1,0,0)            with non-zero mean : 11922.58
##  ARIMA(2,0,1)            with non-zero mean : 11922.44
##  ARIMA(1,0,2)            with non-zero mean : 11922.12
##  ARIMA(0,0,2)            with non-zero mean : 11924.48
##  ARIMA(2,0,0)            with non-zero mean : 11920.41
##  ARIMA(2,0,0)(1,0,0)[30] with non-zero mean : 11947.21
##  ARIMA(2,0,0)(0,0,1)[30] with non-zero mean : 11922.43
##  ARIMA(2,0,0)(1,0,1)[30] with non-zero mean : 11947.69
##  ARIMA(3,0,0)            with non-zero mean : 11922.88
##  ARIMA(3,0,1)            with non-zero mean : Inf
##  ARIMA(2,0,0)            with zero mean     : 11918.4
##  ARIMA(2,0,0)(1,0,0)[30] with zero mean     : 11945.21
##  ARIMA(2,0,0)(0,0,1)[30] with zero mean     : 11920.42
##  ARIMA(2,0,0)(1,0,1)[30] with zero mean     : 11945.78
##  ARIMA(1,0,0)            with zero mean     : 11920.59
##  ARIMA(3,0,0)            with zero mean     : 11920.87
##  ARIMA(2,0,1)            with zero mean     : 11920.42
##  ARIMA(1,0,1)            with zero mean     : 11918.49
##  ARIMA(3,0,1)            with zero mean     : Inf
##
##  Now re-fitting the best model(s) without approximations...
##
##  ARIMA(2,0,0)            with zero mean     : 11917.74
##
##  Best model: ARIMA(2,0,0)            with zero mean
```

```
## Series: diff_data
## ARIMA(2,0,0) with zero mean
##
## Coefficients:
##          ar1     ar2
##       0.3065  0.0796
## s.e.  0.0371  0.0371
##
## sigma^2 = 839113:  log likelihood = -5955.85
## AIC=11917.71   AICc=11917.74   BIC=11931.46
```

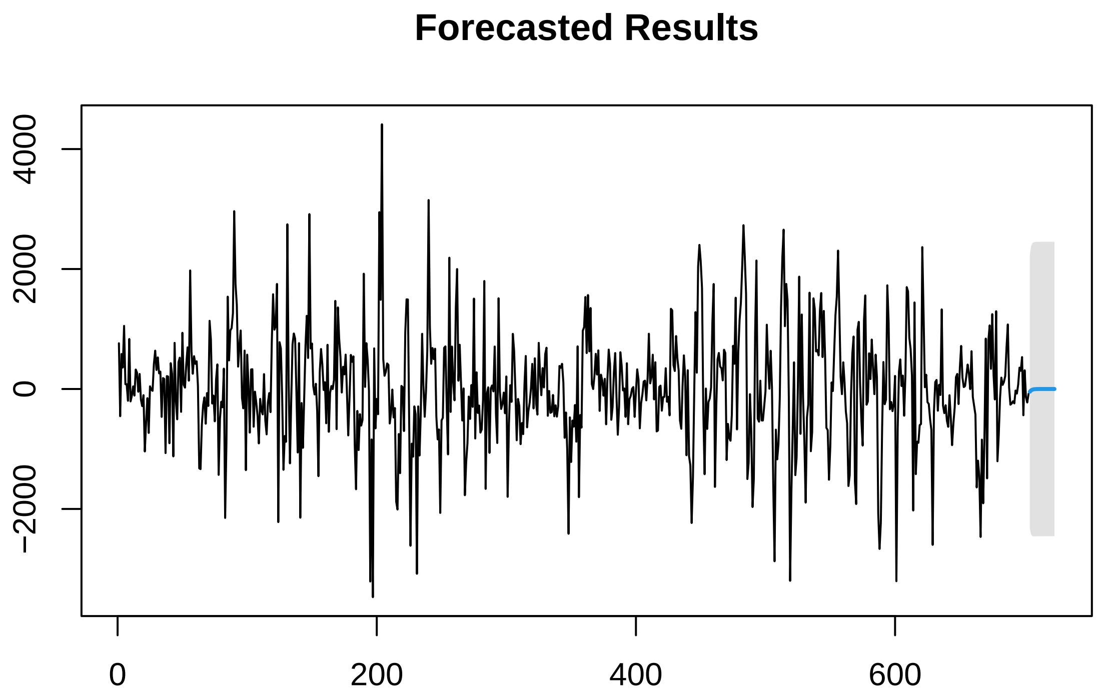
Forecasting

After creating the model and forecasting the results, we can now plot and analyze our time-series model. The plot below shows that the forecasted data is in a straight line, and doesn't replicate the ups and downs we see when we plotted the data earlier. This tells us that our model is not an accurate model and we should try another approach to ARIMA modeling.

```r
#Forecasting the data
number.of.days = length(diff_data)
days.out.of.sample = 20
in.sample =
  diff_data[1:(number.of.days-days.out.of.sample)]
model = auto.arima(in.sample)
future.returns =
  forecast(model,
    h=days.out.of.sample,
    level=c(99)) #confidence level 99%
str(future.returns)
```

```
## List of 10
##  $ method   : chr "ARIMA(2,0,0) with zero mean"
##  $ model    :List of 18
##   ..$ coef     : Named num [1:2] 0.3233 0.0882
##   .. ..- attr(*, "names")= chr [1:2] "ar1" "ar2"
##   ..$ sigma2   : num 787756
##   ..$ var.coef : num [1:2, 1:2] 0.00141 -0.000502 -0.000502 0.00141
##   .. ..- attr(*, "dimnames")=List of 2
##   .. .. ..$ : chr [1:2] "ar1" "ar2"
##   .. .. ..$ : chr [1:2] "ar1" "ar2"
##   ..$ mask     : logi [1:2] TRUE TRUE
##   ..$ loglik   : num -5769
##   ..$ aic      : num 11544
##   ..$ arma     : int [1:7] 2 0 0 0 1 0 0
##   ..$ residuals: Time-Series [1:703] from 1 to 703: 711 -721 663 209 885 ...
##   ..$ call     : language auto.arima(y = in.sample, x = list(x = c(762.999999999996, -453.2857142857
##   ..$ series   : chr "in.sample"
##   ..$ code     : int 0
##   ..$ n.cond   : int 0
##   ..$ nobs     : int 703
##   ..$ model    :List of 10
##   .. ..$ phi  : num [1:2] 0.3233 0.0882
##   .. ..$ theta: num 0
##   .. ..$ Delta: num(0)
##   .. ..$ Z    : num [1:2] 1 0
##   .. ..$ a    : num [1:2] -80.4 -19.9
##   .. ..$ P    : num [1:2, 1:2] 0 0 0 0
##   .. ..$ T    : num [1:2, 1:2] 0.3233 0.0882 1 0
##   .. ..$ V    : num [1:2, 1:2] 1 0 0 0
##   .. ..$ h    : num 0
##   .. ..$ Pn   : num [1:2, 1:2] 1 0 0 0
##   ..$ bic      : num 11557
##   ..$ aicc     : num 11544
##   ..$ x        : Time-Series [1:703] from 1 to 703: 763 -453 583 357 1052 ...
##   ..$ fitted   : Time-Series [1:703] from 1 to 703: 52.4 267.7 -79.2 148.6 167 ...
##   ..- attr(*, "class")= chr [1:3] "forecast_ARIMA" "ARIMA" "Arima"
##  $ level    : num 99
##  $ mean     : Time-Series [1:20] from 704 to 723: -45.86 -21.92 -11.13 -5.53 -2.77 ...
##  $ lower    : Time-Series [1:20, 1] from 704 to 723: -2332 -2425 -2454 -2457 -2457 ...
##   ..- attr(*, "dimnames")=List of 2
##   .. ..$ : NULL
##   .. ..$ : chr "99%"
##  $ upper    : Time-Series [1:20, 1] from 704 to 723: 2240 2381 2432 2446 2451 ...
##   ..- attr(*, "dimnames")=List of 2
##   .. ..$ : NULL
##   .. ..$ : chr "99%"
##  $ x        : Time-Series [1:703] from 1 to 703: 763 -453 583 357 1052 ...
##  $ series   : chr "in.sample"
##  $ fitted   : Time-Series [1:703] from 1 to 703: 52.4 267.7 -79.2 148.6 167 ...
##  $ residuals: Time-Series [1:703] from 1 to 703: 711 -721 663 209 885 ...
##  - attr(*, "class")= chr "forecast"
```

```
plot(forecast(future.returns), main = "Forecasted Results")
```



```
compounded.forecasts = 1+future.returns$mean
(compounded.forecasts = cumprod(compounded.forecasts))
```

```
##  [1]    -44.85575    938.44079  -9509.32779  43110.34450 -76359.91381
##  [6]  29335.76909   9034.89739   5910.11119   4888.40975   4466.02418
## [11]   4273.14645   4180.90502   4135.79566   4113.49215   4102.40442
## [16]   4096.87744   4094.11864   4092.74065   4092.05212   4091.70804
```

Re-testing our ARIMA model

Since we did not see the desired results with our first ARIMA model, we are going to re-test on the daily average to see if we can better fit the ANOVA time-series model. After re-running the data, you can see that we have a slightly better model as the prediction is not a straight line, however, the forecast clearly does not align with the trends and outputs we see in the data. This tells us that time-series modeling might not be the best fit for analyzing and forecasting how many people will rent/use bikes in the future, and we should consider other types of analysis to better forecast results.

```
arima_v2 = ts(na.omit(nd$daily_moving_average), frequency=30)
#Testing to see if data is stationary
adf.test(arima_v2, alternative = "stationary")
```

```
##
##  Augmented Dickey-Fuller Test
##
## data:  arima_v2
## Dickey-Fuller = -2.7707, Lag order = 8, p-value = 0.2521
## alternative hypothesis: stationary
```

```
#Testing for white noise
Acf(arima_v2, main='ACF for Differenced Series')
```

```r
Pacf(arima_v2, main='PACF for Differenced Series')
#Difference
diff_data_v2 = diff(arima_v2)
plot(diff_data_v2)
adf.test(diff_data_v2, alternative = "stationary")
```

```
##
##  Augmented Dickey-Fuller Test
##
## data:  diff_data_v2
## Dickey-Fuller = -12.561, Lag order = 8, p-value = 0.01
## alternative hypothesis: stationary
```

```r
#Re-testing lag
Acf(diff_data_v2, main='ACF for Differenced Series')
Pacf(diff_data_v2, main='PACF for Differenced Series')
#Ljung Box Test
Box.test(diff(diff_data_v2), lag = 10, type = "Ljung-Box")
```

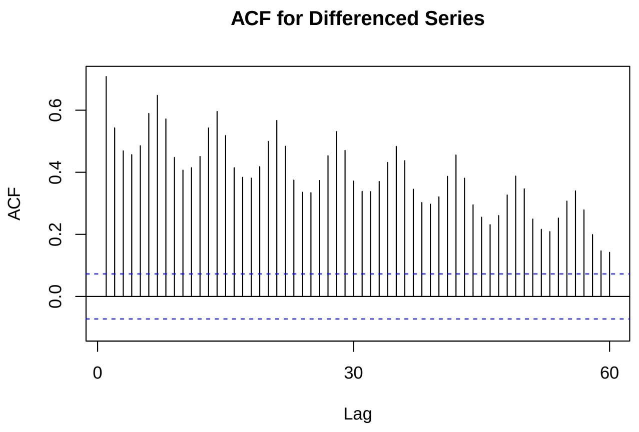
```
##
##  Box-Ljung test
##
## data:  diff(diff_data_v2)
## X-squared = 229.64, df = 10, p-value < 2.2e-16
```

```r
auto.arima(diff_data_v2, seasonal =TRUE)
```

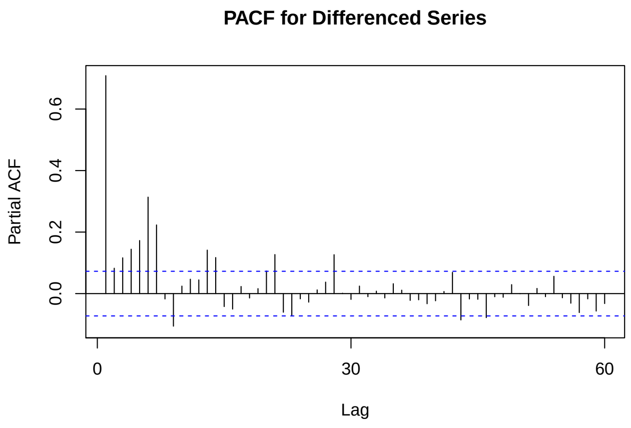
```
## Series: diff_data_v2
## ARIMA(5,0,2)(0,0,1)[30] with zero mean
##
## Coefficients:
##          ar1      ar2      ar3      ar4      ar5      ma1     ma2     sma1
##       0.3687  -0.5880  -0.2729  -0.2340  -0.2702  -0.8868  0.5745  -0.0481
## s.e.  0.1363   0.0671   0.0468   0.0405   0.0519   0.1341  0.1321   0.0382
##
## sigma^2 = 27293184:  log likelihood = -7272.26
## AIC=14562.53   AICc=14562.78   BIC=14603.85
```

```r
#Forecast data
number.of.days = length(diff_data_v2)
days.out.of.sample = 20
in.sample =
  diff_data_v2[1:(number.of.days-days.out.of.sample)]
model = auto.arima(in.sample)
future.returns =
  forecast(model,
    h=days.out.of.sample,
    level=c(99)) #confidence level 99%
```

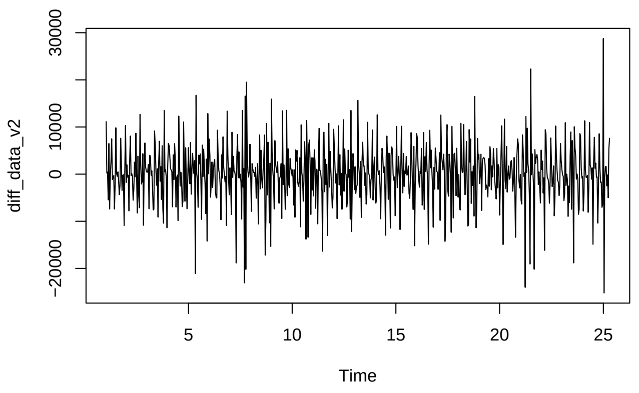
```r
plot(forecast(future.returns), main = "Forecasted Results")
```


ACF for Differenced Series
ACF
Lag
0.0
0.2
0.4
0.6
0
30
60

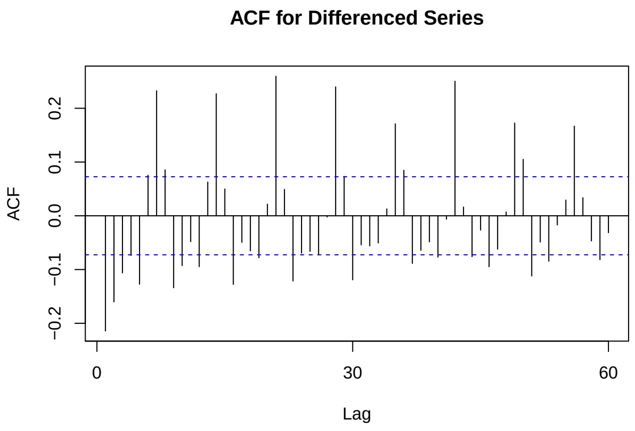
(a) (1)


PACF for Differenced Series
Partial ACF
Lag
0.0
0.2
0.4
0.6
0
30
60

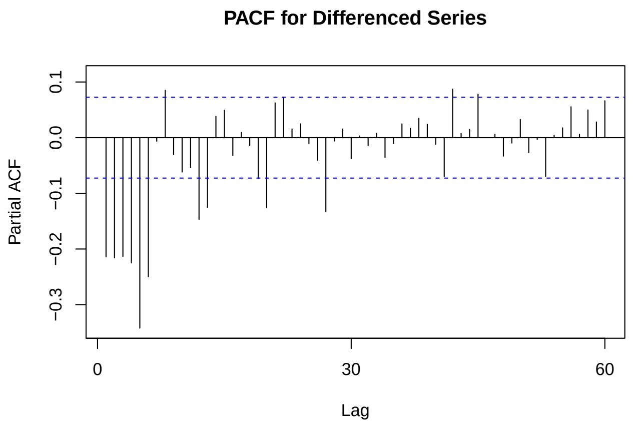
(b) (1)


diff_data_v2
Time
−20000
0
10000
30000
5
10
15
20
25

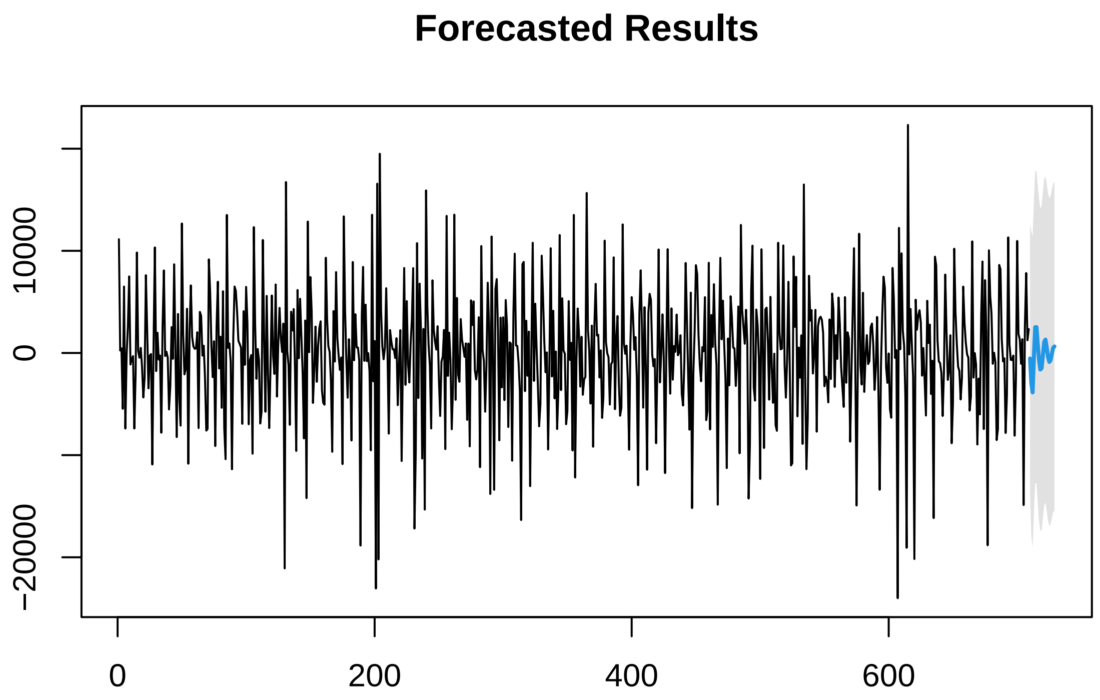
(c) (1)


ACF for Differenced Series
ACF
Lag
−0.2
−0.1
0.0
0.1
0.2
0
30
60

(d) (1)


PACF for Differenced Series
Partial ACF
Lag
−0.3
−0.2
−0.1
0.0
0.1
0
30
60

(e) (1)



```
compounded.forecasts = 1+future.returns$mean
(compounded.forecasts = cumprod(compounded.forecasts))
```

```
##  [1] -5.369796e+02  1.577321e+06 -6.086893e+09  2.259065e+12  5.706025e+15
##  [6]  1.444668e+19  1.251750e+22 -8.439632e+24  1.368907e+28 -2.035687e+31
## [11]  4.616927e+33  4.986893e+36  6.549249e+39  3.531594e+42 -1.440912e+45
## [16]  1.264921e+48 -8.772339e+50  5.924147e+52  3.074444e+55  1.970646e+58
```

Conclusion

Unfortunately, both the linear regression and ANOVA models were unsuccessful in producing the results we wanted. In our linear regression analysis, we were unable to identify variables to accurately predict bike count, resulting in a low R-squared value. Even after manually fitting the models and automatically finding the best solution, both our models had a low R-squared value and would not be useful in predicting bike count. Due to the volatility of the daily bike count, we also struggled to use time-series modeling to predict bike count. Since our time-series data has no real apparent patterns or seasonality to it, creating the time-series analysis was unsuccessful and ANOVA modeling is also not the best choice for predicting bike count. If predicting bike count is still desired, exploring other variables and other types of models might produce better results.