Regression Analysis of Bike Sharing Demand

Chance Robinson, Jayson Barker and Neha Dixit

Master of Science in Data Science, Southern Methodist University, USA

1 Introduction

[Intro]

2 Data Description

The dataset we chose for this project was a publicly shared, hourly bike sharing dataset made available through Kaggle in csv format.

This data set is divided into two distinct sets – a train and test set. The train set consists of 10,886 rows (titled "train.csv") and the test set consists of 6,493 rows (titled "test.csv"). Within the training set, the first 19 days of each month are captured whereas in the test data set, the 20th day to the end of each month is present. The entirity of the data spans from 1/1/2011 through 12/31/2012 - encompassing two full years of bike sharing data. Interstingly, the time component of this analysis is captured in hours of each day meaning we have a calendar date represented 24 times (for each hour of that day) in the data, along with it's associated attribute values.

In train data set, there are a total of 12 attributes which capture multiple variables related to bike rentals. Some of these attributes are categorical, and others are continuous. All attributes are summarized in the table below:

Column Name	Type	Description
1. datetime	Date	YYYY-MM-DD HH24 (example: 2011-01-01 04:00:00)
2. season	Integer	(1-4)
3. holiday	Integer	(0 or 1)
4. workingday	Integer	(0 or 1)
5. weather	Integer	(1-4)
6. temp	Float	temparture in Celcius
7. atemp	Float	"feels like" temperature in Celsius
8. humidity	Integer	relative humidity
9. windspeed	Float	wind speed
10. casual	Integer	count of casual users
11. registered	Integer	count of registered users
12. count	Integer	count of total users (*response variable*)

Note that the test data set lacks the casual, registered and count variables.

3 Exploratory Data Analysis

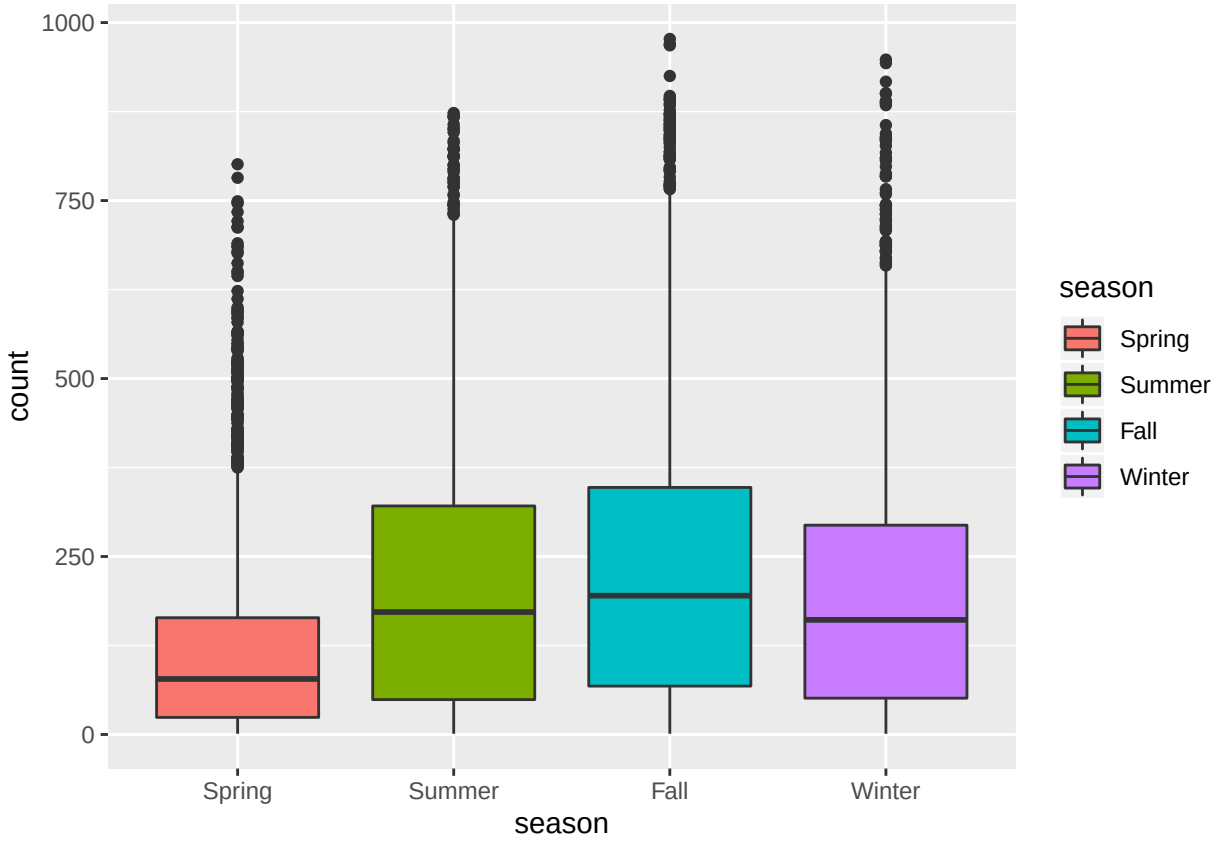
3.1 Plotting against Response Variable

3.1.1 Categorical Variables

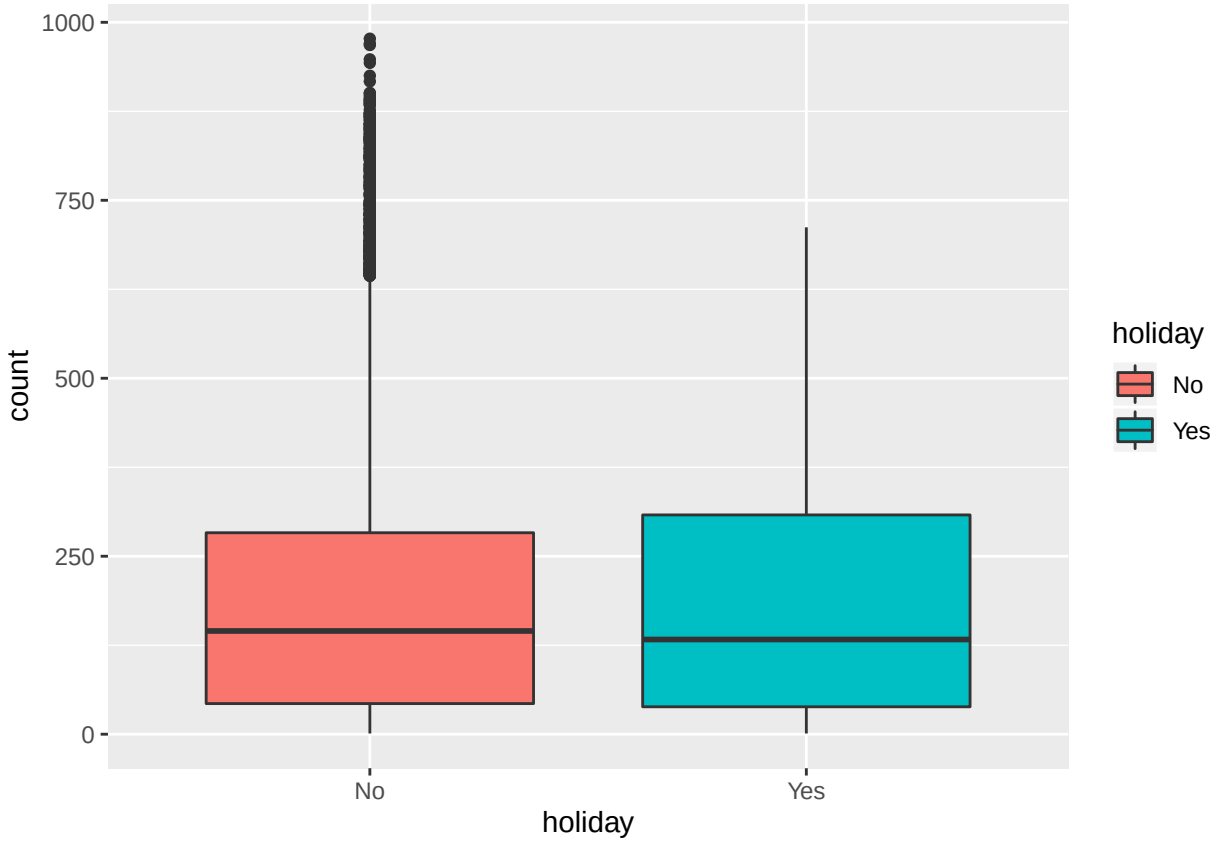
Several numeric variables were found to be better suited to categorization and were converted to factors.

3.1.1.1 Season

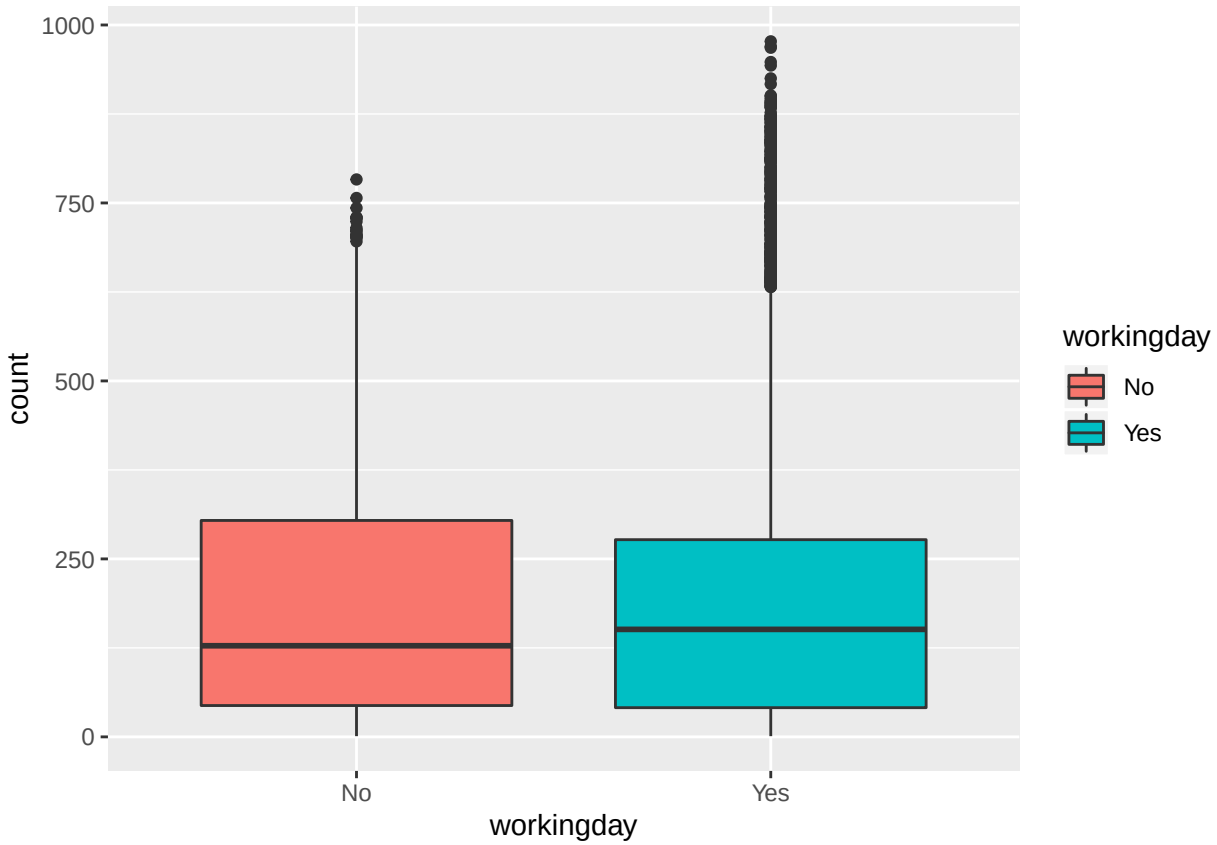
Season	Label	Description
1	Spring	Dec 21 ~ Mar 20
2	Summer	Mar 21 ~ Jun 20
3	Fall	Jun 21 ~ Sep 20
4	Winter	Sep 21 ~ Dec 20



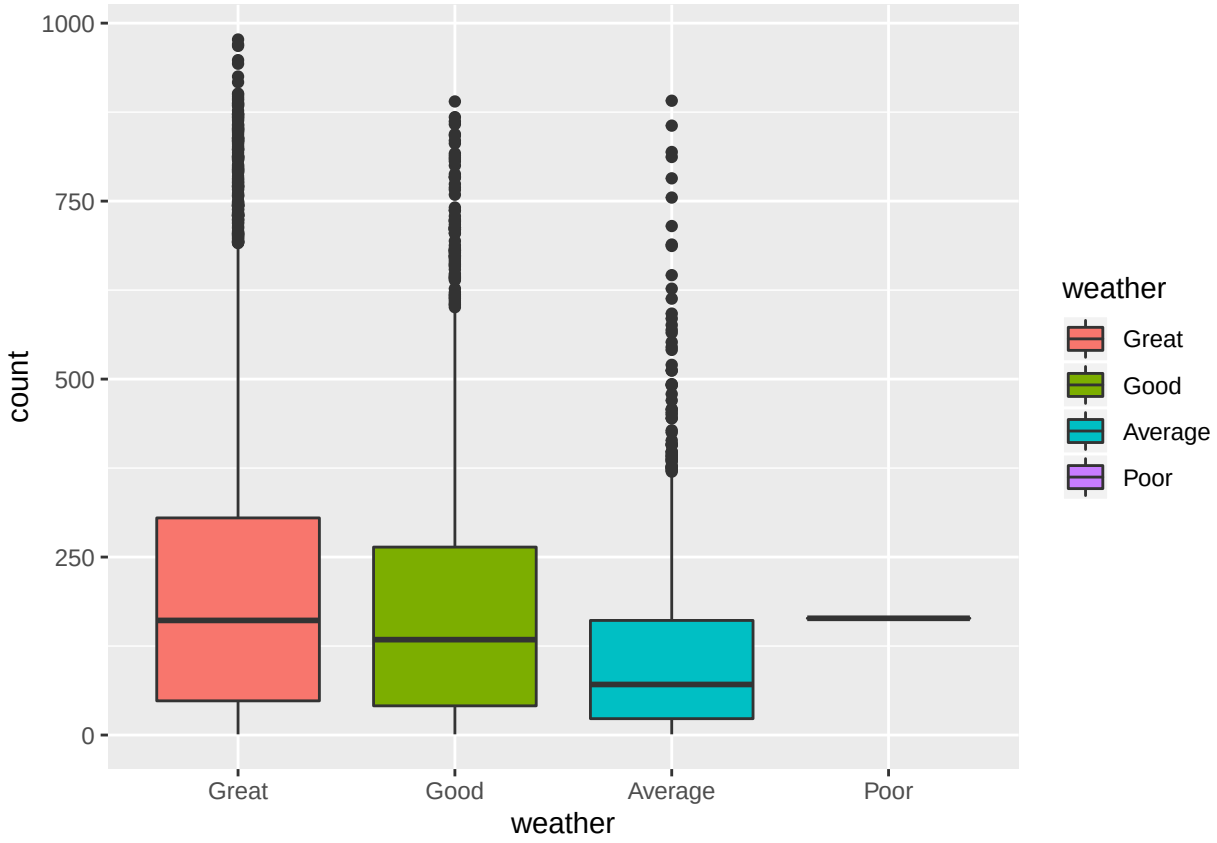
3.1.1.2 Holiday



3.1.1.3 Working Day


1000
750
500
250
0
count
No
Yes
workingday
workingday
No
Yes

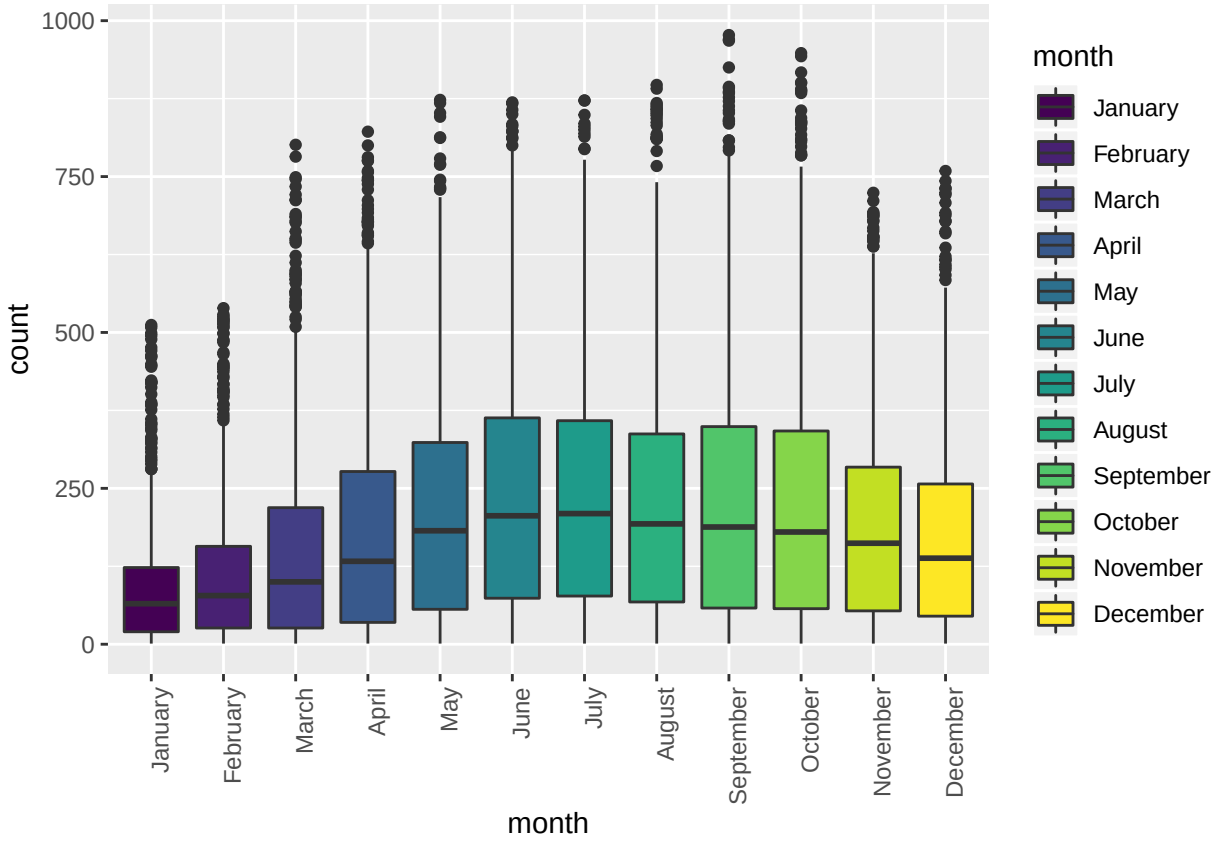
3.1.1.4 Weather



3.1.1.5 Count by Month

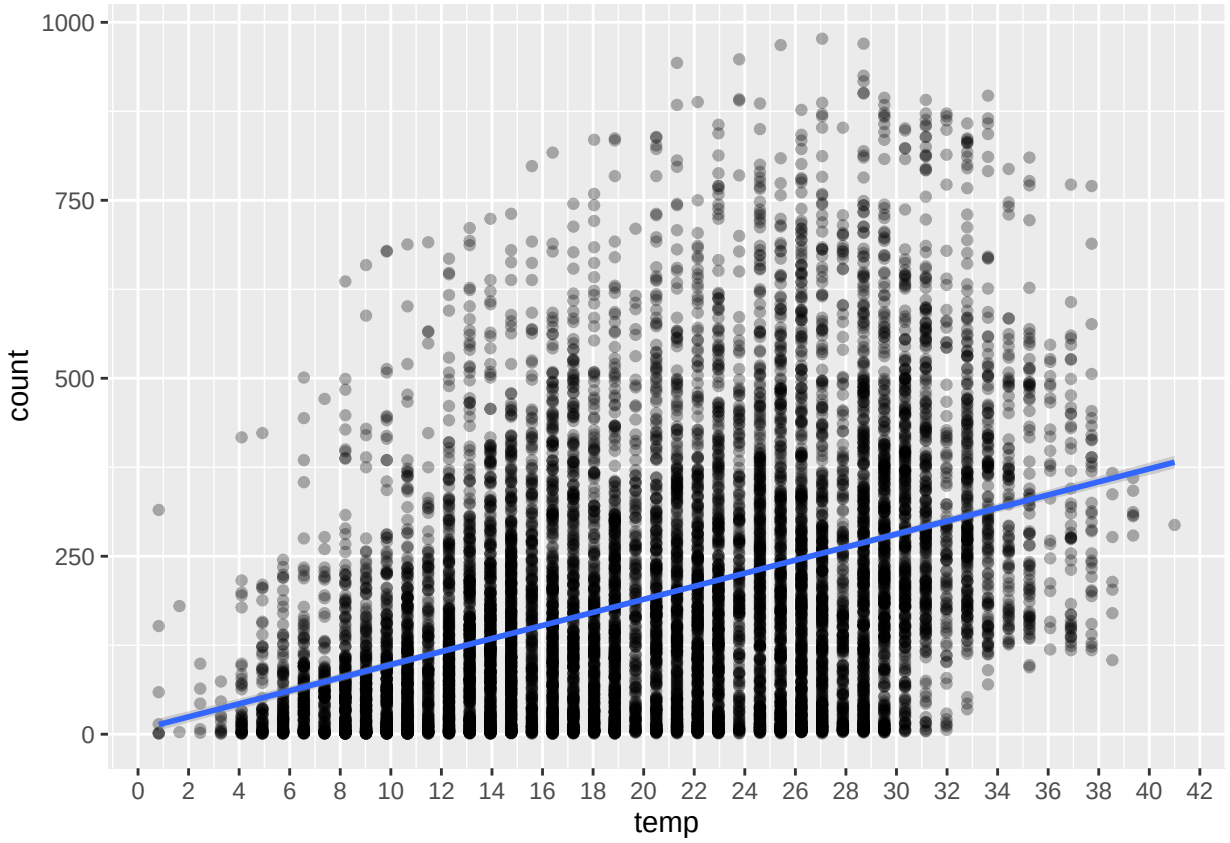
The datetime column was broken out into multiple factors as well so that we could visualize the components of each date and aggregate by different dimensions of the timestamp. We felt this was also necessary due to the nature of how the train/ test data sets had been pre-split. (i.e. . . with the first 19 days of the month holding the only true counts to validate our models against.)

- Year
- Month
- Day
- Hour

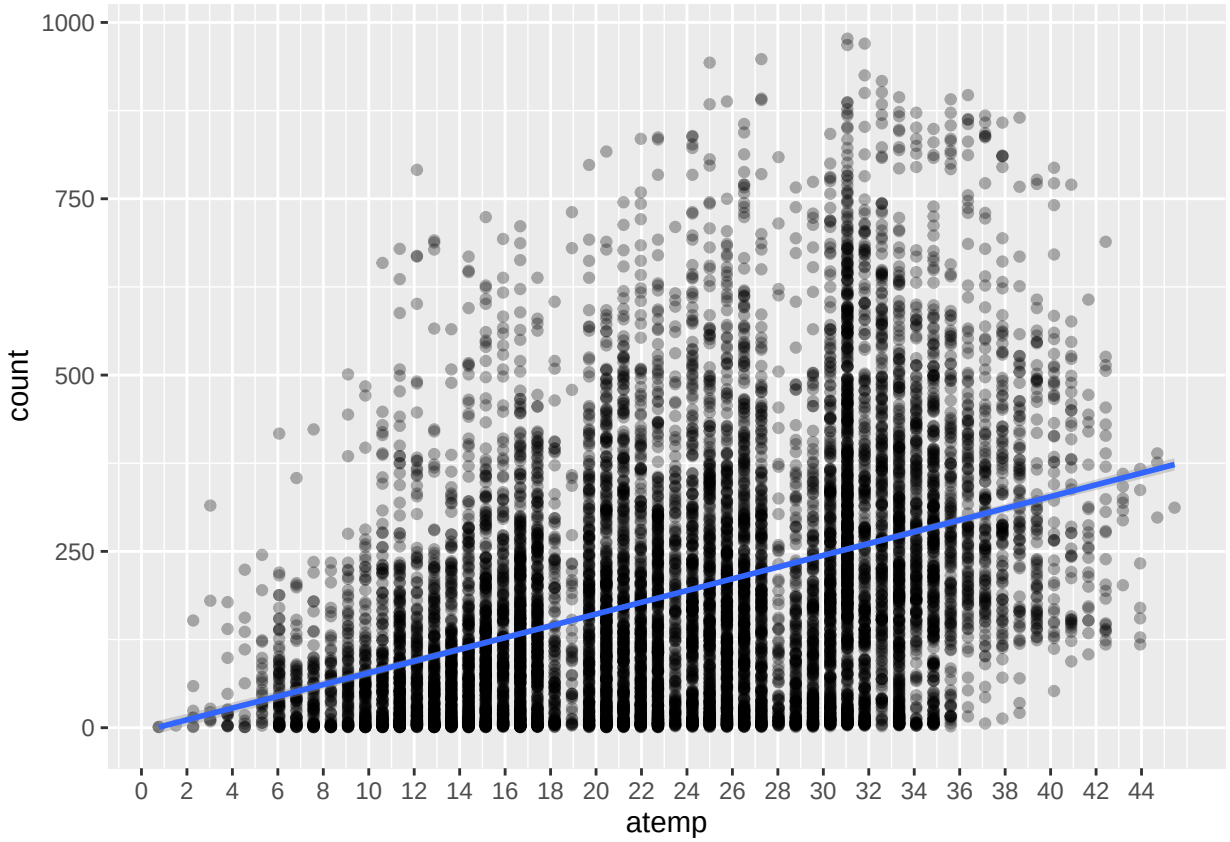

1000
750
500
250
0
count
January
February
March
April
May
June
July
August
September
October
November
December
month
month
January
February
March
April
May
June
July
August
September
October
November
December

3.1.2 Continuous Variables

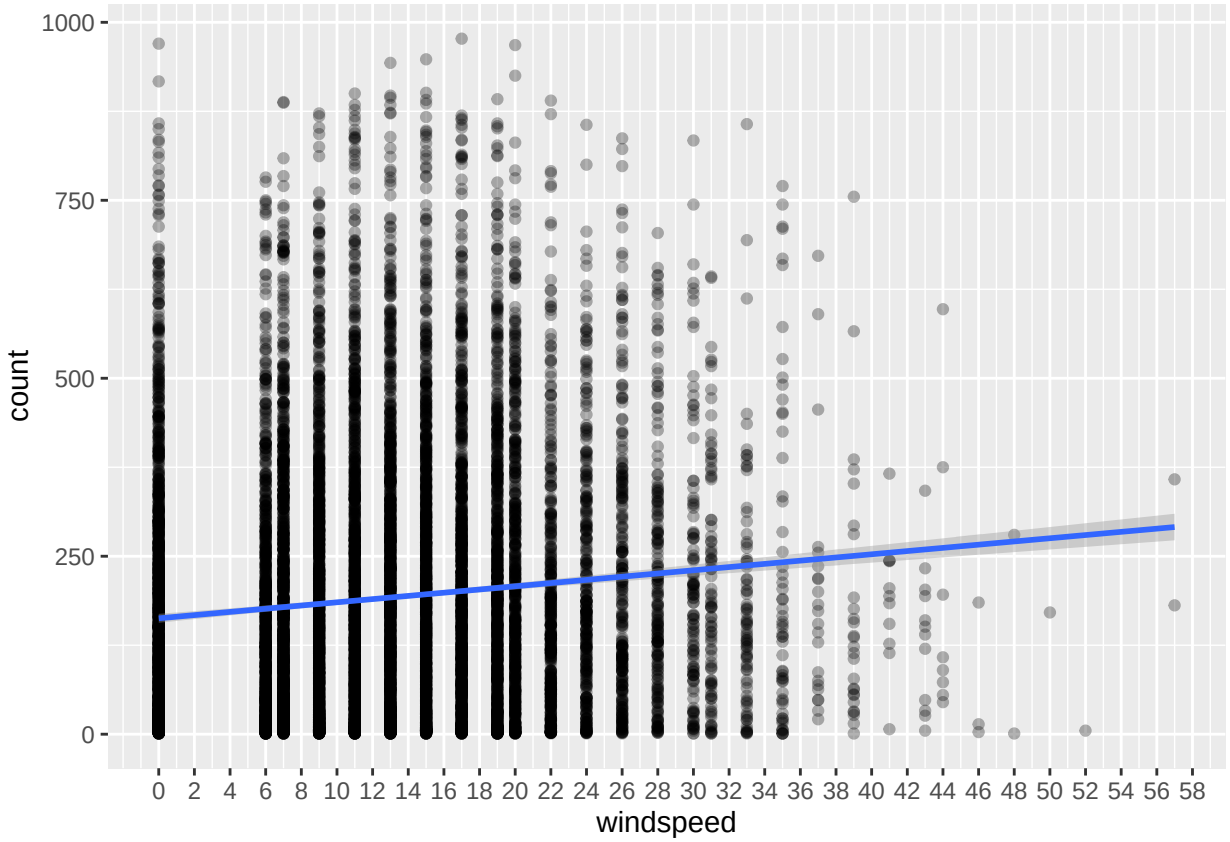
3.1.2.1 Count by Temperature



3.1.2.2 Count by "Feels like" Temperature



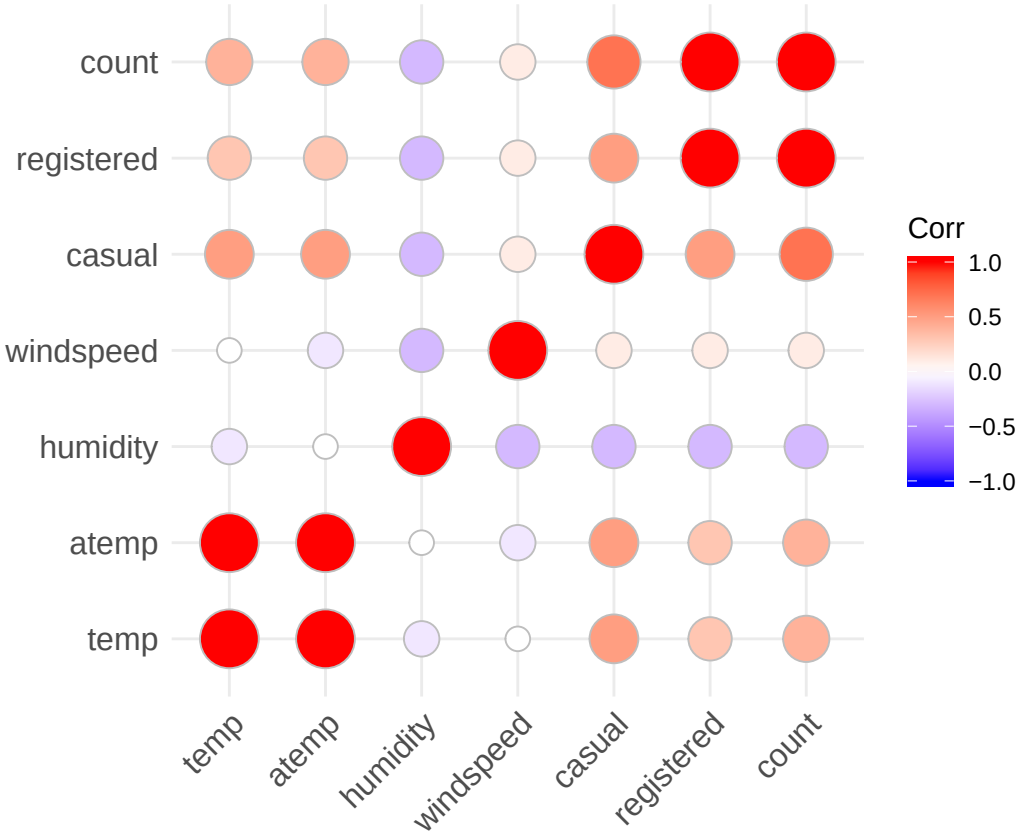
3.1.2.3 Count by Wind Speed



3.2 Correlation Matrix

There are several variables with a relatively high level of covariance from the training set. The following columns should therefore be removed so as not to be picked up by any automated modelling techniques.

- atemp
- causual
- registered



4 Objective I Analysis

4.1 Question of Interest

The team be utilizing the multiple linear regression techniques we've learned up to this point in the program to predict the bike rental deman on a given date and time. The model will be evaluated on the Root Mean Squared Logarithmic Error, or RMSLE. As this data represents hourly data collected, there is an obvious time component associated with this particular competition. We wanted to gauge how effective mutiple linear regression would be when the assumption of indepdence is clearly violated.

4.2 Model Selection

4.2.1 Lasso

```
split.perc = .80

train.indices = sample(1:dim(train.mod.1)[1],round(split.perc * dim(train.mod.1)[1]))

train.mod.1.train = train.mod.1[train.indices,]
train.mod.1.test  = train.mod.1[-train.indices,]



train.mod.1.train$datetime <- NULL
train.mod.1.test$datetime <- NULL



train.mod.1.train
```

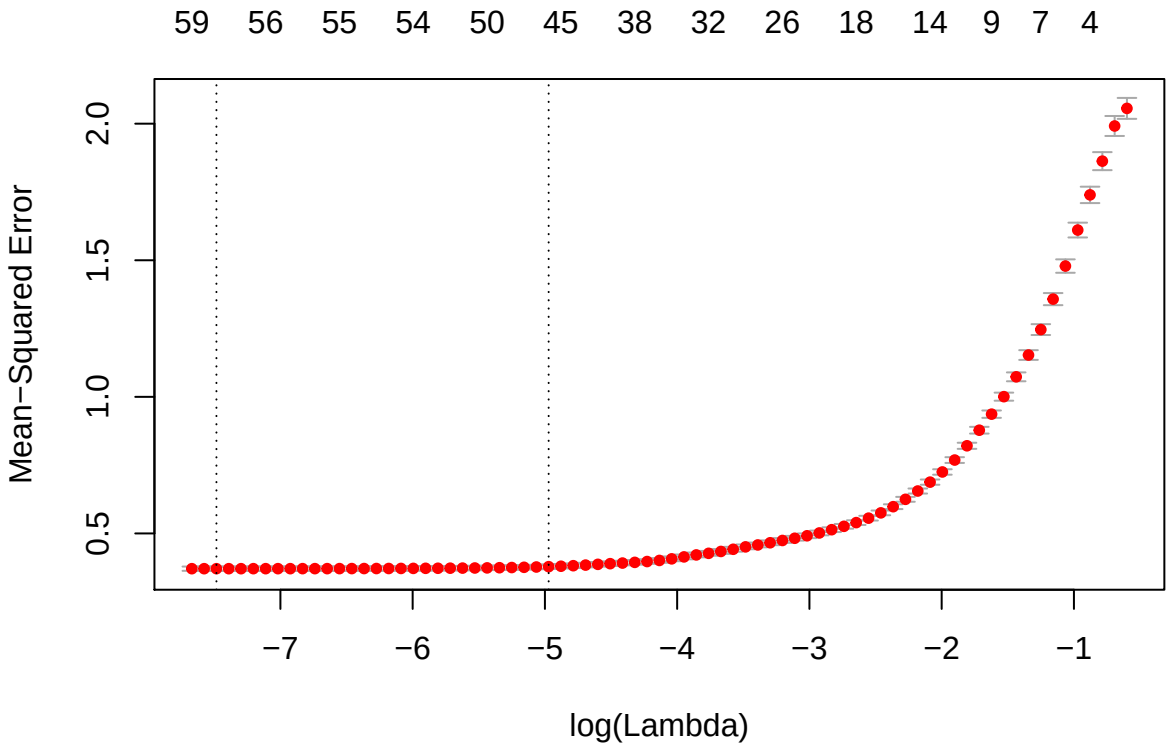
```
## # A tibble: 8,624 x 12
##    season holiday workingday weather  temp humidity windspeed count year
##    <fct>  <fct>   <fct>      <fct>   <dbl>    <dbl>     <dbl> <dbl> <fct>
##  1 Fall   No      Yes        Average 28.7        89     13.0   1.61 2012
##  2 Spring No      Yes        Great   21.3        27     31.0   4.77 2011
##  3 Fall   No      No         Great   28.7        54     11.0   6.17 2012
##  4 Winter No      Yes        Great    9.84       75      0     2.30 2011
##  5 Spring No      Yes        Good     8.2        34     19.0   4.16 2011
##  6 Summer No      No         Good    28.7        48      6.00  5.48 2011
##  7 Winter No      Yes        Great   13.9        46     15.0   6.30 2012
##  8 Spring No      Yes        Great    9.02       51     20.0   4.04 2011
##  9 Fall   No      Yes        Great   20.5        82      0     5.27 2012
## 10 Fall   No      Yes        Great   30.3        70     19.0   5.19 2012
## # ... with 8,614 more rows, and 3 more variables: month <ord>, day <fct>,
## #   hour <fct>
```

```
x <- model.matrix(count~.,train.mod.1.train)[,-7]
y <- train.mod.1.train$count
```

```
grid=10^seq(10,-2, length =100)
lasso.model <- glmnet(x,y,alpha=1, lambda=grid)

xtest<-model.matrix(count~.,train.mod.1.test)[,-7]
ytest <- train.mod.1.test$count


cv.out=cv.glmnet(x,y,alpha=1) #alpha=1 performs LASSO
plot(cv.out)
```




```
best.lambda <-cv.out$lambda.min  #Optimal penalty parameter.  You can make this call visually.
lasso.pred=predict(lasso.model ,s=best.lambda ,newx=xtest)

testMSE_LASSO <- mean((ytest-lasso.pred)^2)
testMSE_LASSO
```

```
## [1] 0.3535176
```

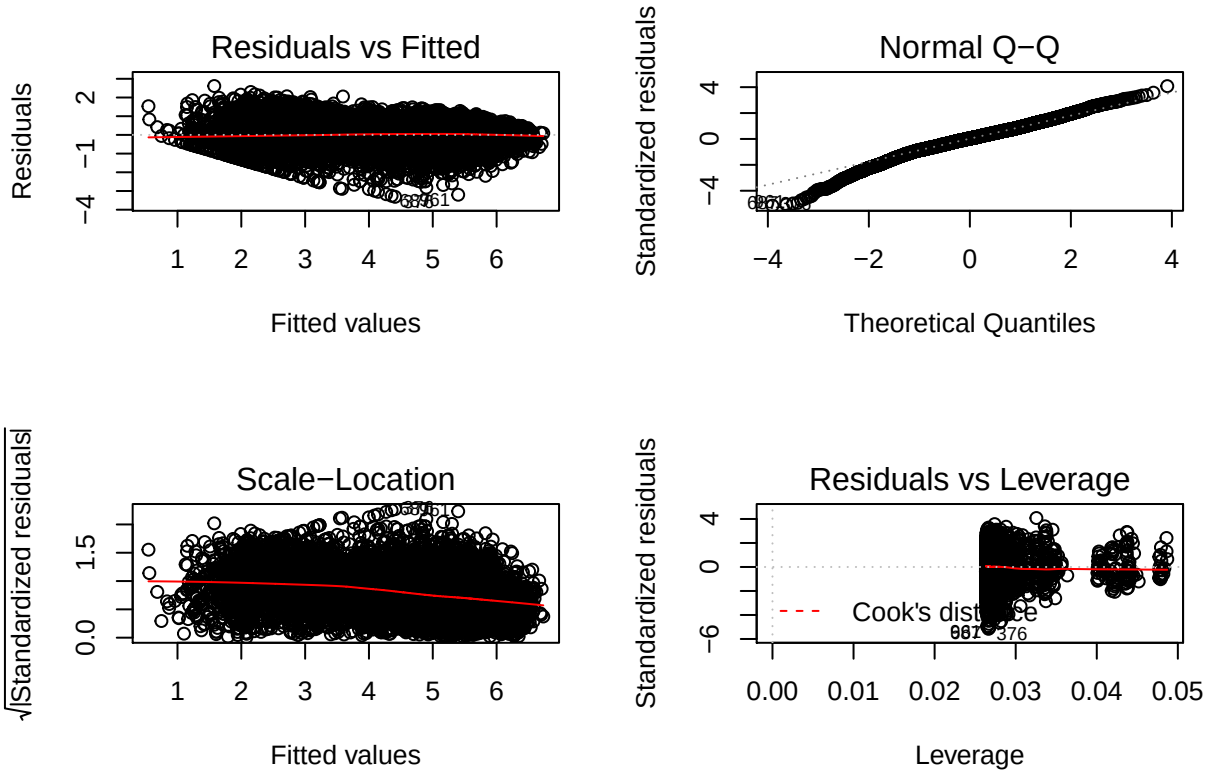
4.2.2 Custom Variable Selection

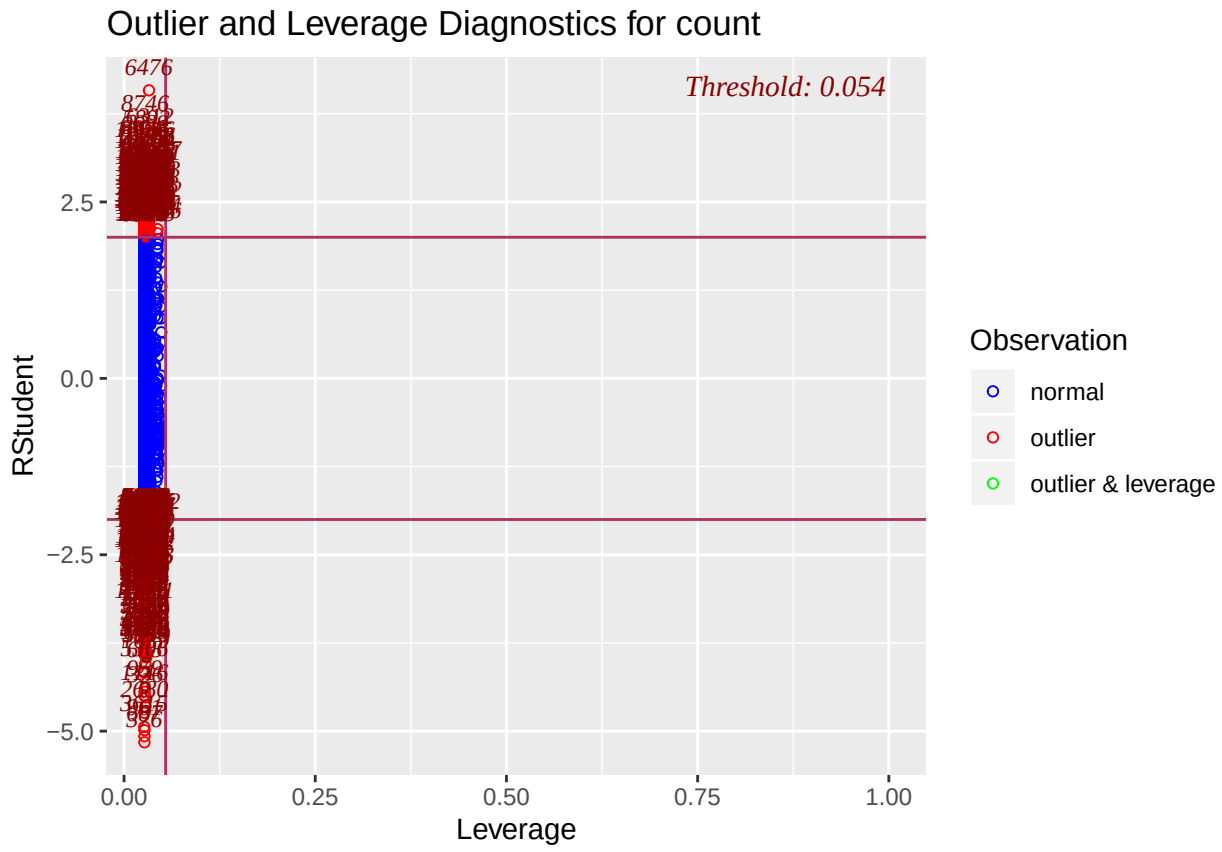
We developed the best fitting model with a custom selection of variables after having accounted for those with high correlation and adding interactive terms such as month/ hour based on the seasonal nature that

the box plots exhibited.

```
## Warning: not plotting observations with leverage one:
##   5555

## Warning: not plotting observations with leverage one:
##   5555
```





4.2.2.1 Custom RMSLE

```
## [1] 0.1534782
```

4.2.2.2 Kaggle Score

The Root Mean Squared Logarithmic Error Loss (RMSLE) for the kaggle submission was 0.67517 for our custom model. We scored better than around 24% of all public submission for the competition with this technique.

4.2.3 Stepwise

4.2.3.1 Stepwise RMSLE

The Root Mean Squared Logarithmic Error Loss calculation against the train data set.

```
## [1] 0.1469294
```

4.2.3.2 Kaggle Score

The Root Mean Squared Logarithmic Error Loss (RMSLE) for the kaggle submission was 0.64765 for our stepwise model. We scored better than around 24% of all public submission for the competition with this technique, which was almost identical to that of the custom model with an interaction.

4.3 Model Assumptions Assessment

- The response variable is linear
- The data is normally distributed
- Independence

4.4 Comparing Competing Models

4.5 Parameters

Parameters

4.6 Model Interpretation

Model Interpretation

4.7 Conclusion

Conclusion

5 Objective II Analysis

5.1 Question of Interest

As the independence assumption from Objective I appears to have been violated, the team wanted to apply the time series analysis techniques we've learned up to this point in the MSDS program in an attempt to address. The primary goal was to compare and contrast the performance of auto arima models versus those that we modeled on our own.

Additionally, we wanted to compare the Kaggle submission to that from Objective I to see if our predictions were better or worse than from the prior objective.

5.2 Data Preperation

As the data had been pre-split monthly, we needed to make our predictions from the 20th of each month and on for each year of recorded bike rentals. (2011 and 2012) The approach has been summarized with the steps below.

1. Log the response variable
2. Loop through years
3. Loop through months
4. Fit AR model
5. Forcast x number of observations based on the number of rows from test dataframe and impute the count from the time series forecast

5.3 Comparing Competing Models

5.3.1 Auto Arima

```
##
##  Ljung-Box test
##
## data:  Residuals from ARIMA(4,0,4) with non-zero mean
## Q* = 10.049, df = 3, p-value = 0.01816
##
## Model df: 9.   Total lags used: 12
```

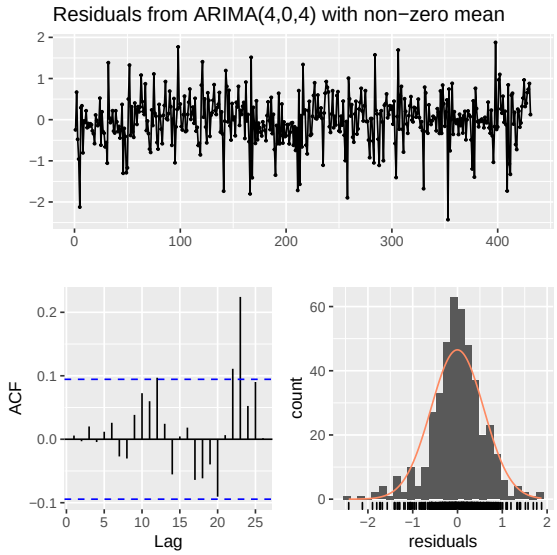


Figure 1: Auto Arima Residual Plots

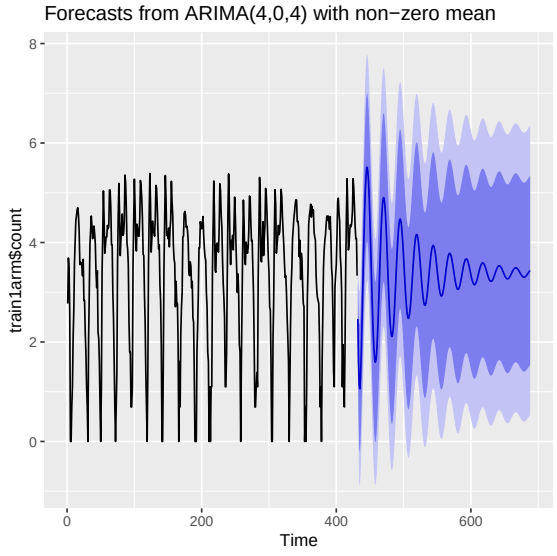


Figure 2: Auto Arima Forecasts

5.3.2 Auto Regression

```
##
##  Ljung-Box test
##
## data:  Residuals from ARIMA(25,0,0) with non-zero mean
## Q* = 11.131, df = 3, p-value = 0.01104
##
## Model df: 26.   Total lags used: 29
```

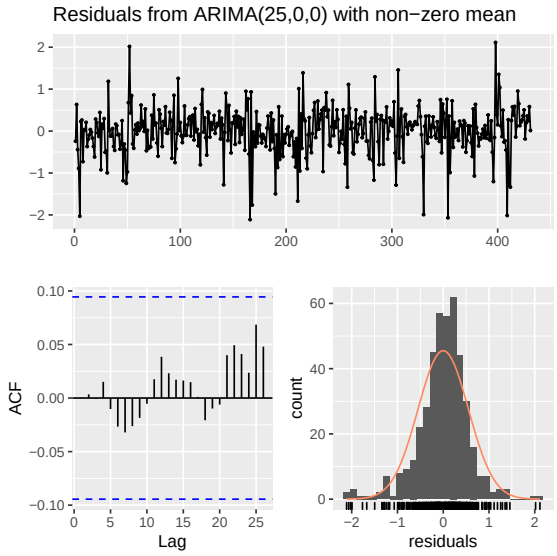


Figure 3: Arima (25) Residual Plots

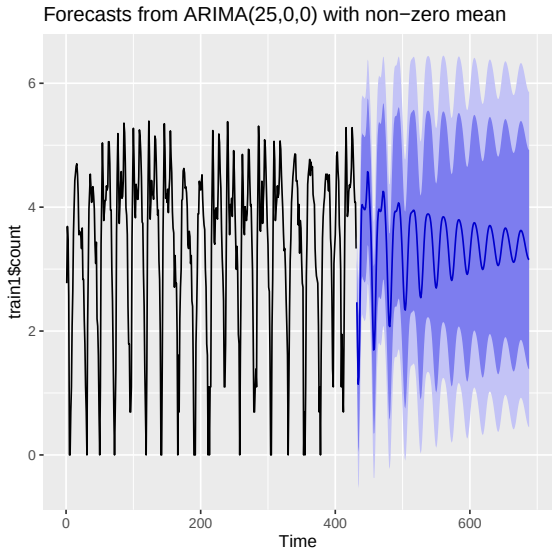


Figure 4: Arima (25) Forecasts

5.3.2.1 Kaggle Score

The Root Mean Squared Logarithmic Error Loss (RMSLE) for the kaggle submission was 1.01847 for our AR25 model. We scored better than only 14.48% of all public submission for the competition which was considerably worse than our score from both of the multiple linear regression models above.

5.4 Model Assumption Assessment

Both the ARIMA and AR 25 models showed a constant mean and variance for the timespan of the observations. The auto-correlations however were noticably better for the AR 25 graph. The error terms were also slightly improved which is the reason we elected to not go with the automated selection approach.

- Constant Mean
- Constant Variance
- Constant auto-correlations

5.5 Conclusion

Although the data set was found to be serially correlated, ultimately applying time series techniques was not sufficient to improve our RMSLE above and beyond the multiple linear regression techniques from objective one. Again, this likely had more to due with the fact that we were attempting to predict future observations 10 plus days in advance on a monthly basis. And with the seasonal nature of our hourly observations, the accuracy of the preduction became less accurate over time and showed a regression to the mean after only a few days worth of predictions. The immediate observations however, within 24 to 48 hours, visually at least showed much better predictive trends.

6 Appendix

6.1 Code

6.1.1 R Code For Objective I

6.1.2 R Code For Objective II

```
### Library Imports

library(tidyverse)
# Date manipulation
library(lubridate)
# RMLSE
library(MLmetrics)
# Time Series Analysis
library(tseries)
library(forecast)


### Load the csv data
train <- read_csv('../../../data/train.csv')
test <- read_csv('../../../data/test.csv')


### Categorical Factors
train$season <- factor(train$season, labels = c("Spring", "Summer", "Fall", "Winter"))
test$season <- factor(test$season, labels = c("Spring", "Summer", "Fall", "Winter"))

table(train$season)

train$holiday <- factor(train$holiday, labels = c("No", "Yes"))
test$holiday <- factor(test$holiday, labels = c("No", "Yes"))

table(train$holiday)

train$workingday <- factor(train$workingday, labels = c("No", "Yes"))
test$workingday <- factor(test$workingday, labels = c("No", "Yes"))

table(train$workingday)

train$weather <- factor(train$weather, labels = c("Great", "Good", "Average", "Poor"))
test$weather <- factor(test$weather, labels = c("Great", "Good", "Average", "Poor"))

table(train$weather)


### Split Date-Time
train <- train %>%
```

```
  mutate(year = as.factor(format(datetime, format = "%Y")),
         month = as.numeric(format(datetime, format = "%m")),
         day = as.factor(format(datetime, format = "%d")),
         hour = as.factor(format(datetime, format = "%H")))

test <- test %>%
  mutate(year = as.factor(format(datetime, format = "%Y")),
         month = as.numeric(format(datetime, format = "%m")),
         day = as.factor(format(datetime, format = "%d")),
         hour = as.factor(format(datetime, format = "%H")))


### Convert months to ordinal factor
train$month <-month(train$datetime, label = TRUE, abbr = FALSE)
test$month <-month(test$datetime, label = TRUE, abbr = FALSE)


## 2011

### January

#### Auto Arima

train1arm <- train %>%
  filter(year == '2011' & month == 'January') %>%
  select(datetime, count)

test1arm <- test %>%
  filter(year == '2011' & month == 'January') %>%
  mutate(count = NA) %>%
  select(datetime, count)


### Log the response variable
train1arm$count = log(train1arm$count)


autoarm <- auto.arima(train1arm$count, D=1)


number = nrow(test1arm)

acf(autoarm$residuals)
```

```r
pacf(autoarm$residuals)

checkresiduals(autoarm)

fcst <- forecast(autoarm, h=number)

autoplot(fcst)

# point estimate (mean)
test1arm$count <- fcst$mean

RMSLE(y_pred = fcst$fitted, y_true = train1arm$count)

summary(autoarm)


#### AR 25

train1 <- train %>%
  filter(year == '2011' & month == 'January') %>%
  select(datetime, count)

test1 <- test %>%
  filter(year == '2011' & month == 'January') %>%
  mutate(count = NA) %>%
  select(datetime, count)


### Log the response variable
train1$count = log(train1$count)


AR25 <- arima(train1$count,order=c(25,0,0))

number = nrow(test1)

acf(AR25$residuals)
pacf(AR25$residuals)

checkresiduals(AR25)

fcst <- forecast(AR25, h=number)
```

```r
autoplot(fcst)

# point estimate (mean)
test1$count <- fcst$mean


RMSLE(y_pred = fcst$fitted, y_true = train1$count)
summary(AR25)


#
#
#

## 2012

### December

train24 <- train %>%
  filter(year == '2012' & month == 'December') %>%
  select(datetime, count)

test24 <- test %>%
  filter(year == '2012' & month == 'December') %>%
  mutate(count = NA) %>%
  select(datetime, count)


### Log the response variable
train24$count = log(train24$count)


AR25 <- arima(train24$count,order=c(25,0,0))

number = nrow(test24)

acf(AR25$residuals)
pacf(AR25$residuals)

checkresiduals(AR25)

fcst <- forecast(AR25, h=number)
```

```r
autoplot(fcst)

# point estimate (mean)
test24$count <- fcst$mean


RMSLE(y_pred = fcst$fitted, y_true = train24$count)

summary(AR25)


### Combine all of the individual data frames

combined <- data.frame(datetime=character(),
                 count=double(),
                 stringsAsFactors=FALSE)



combined <- bind_rows(test1, test2, test3, test4, test5,
                      test6, test7, test8, test9, test10,
                      test11, test12,test13, test14, test15,
                      test16, test17, test18, test19, test20,
                      test21, test22, test23, test24)


combined <- combined %>%
  mutate(count = round(exp(count)))


# combined
# write.csv(combined, file = "./ts_kaggle_submission.csv", row.names = F)
```