ST 558 Homework 7

Eric Warren

October 24, 2023

Contents

1 Reading in Data **1**

2 Split the Data **1**

3 Basic EDA **2**

3.1 Exploring the Data 3

3.2 Changing Variables 11

4 Fitting MLR Models **11**

5 Logistic Regression **16**

1 Reading in Data

The first thing we need to do is read in the data. This data is about Seoul bike sharing rentals. The option with `locale` will help us read in the data correctly.

```
library(tidyverse)
bikes <- read_csv("SeoulBikeData.csv", locale = locale(encoding = "latin1"))
```

2 Split the Data

Now we are going to split the data into 75% going to the training set and the other 25% going to the test set. We'll do our EDA and model fitting on the training set and see how the 'best' model does using the test set.

```
# To reproduce
set.seed(999)

# Indices to split on
train <- sample(1:nrow(bikes), size = nrow(bikes) * 0.75)
test <- dplyr::setdiff(1:nrow(bikes), train)
```

```
# Subset into train and test
(bikeTrain <- bikes[train, ])  # Show training data tibble
```

```
## # A tibble: 6,570 x 14
##    Date       `Rented Bike Count`  Hour `Temperature(°C)` `Humidity(%)`
##    <chr>                    <dbl> <dbl>             <dbl>         <dbl>
##  1 15/07/2018                1847    19              31.2            62
##  2 16/08/2018                 668    12              34.5            44
##  3 07/09/2018                 624     6              20.2            70
##  4 11/03/2018                 212     8               2.8            82
##  5 22/10/2018                1513    21              13.4            56
##  6 17/09/2018                1445    16              27.6            37
##  7 10/07/2018                  39     1              20.4            97
##  8 23/10/2018                 193     5              10              73
##  9 08/03/2018                 165    10               3.6            88
## 10 19/04/2018                 996    14              20.5            17
## # i 6,560 more rows
## # i 9 more variables: `Wind speed (m/s)` <dbl>, `Visibility (10m)` <dbl>,
## #   `Dew point temperature(°C)` <dbl>, `Solar Radiation (MJ/m2)` <dbl>,
## #   `Rainfall(mm)` <dbl>, `Snowfall (cm)` <dbl>, Seasons <chr>, Holiday <chr>,
## #   `Functioning Day` <chr>
```

```
(bikeTest <- bikes[test, ])  # Show testing data tibble
```

```
## # A tibble: 2,190 x 14
##    Date       `Rented Bike Count`  Hour `Temperature(°C)` `Humidity(%)`
##    <chr>                    <dbl> <dbl>             <dbl>         <dbl>
##  1 01/12/2017                 181     6              -6.6            35
##  2 01/12/2017                 490     9              -6.5            27
##  3 01/12/2017                 426    20              -0.3            79
##  4 01/12/2017                 405    21              -0.8            81
##  5 01/12/2017                 398    22              -0.9            83
##  6 01/12/2017                 323    23              -1.3            84
##  7 02/12/2017                 328     0              -1.8            87
##  8 02/12/2017                 262     2              -2.9            86
##  9 02/12/2017                  89     4              -3.8            79
## 10 02/12/2017                 328     9              -2.9            68
## # i 2,180 more rows
## # i 9 more variables: `Wind speed (m/s)` <dbl>, `Visibility (10m)` <dbl>,
## #   `Dew point temperature(°C)` <dbl>, `Solar Radiation (MJ/m2)` <dbl>,
## #   `Rainfall(mm)` <dbl>, `Snowfall (cm)` <dbl>, Seasons <chr>, Holiday <chr>,
## #   `Functioning Day` <chr>
```

3 Basic EDA

Here we are going to do the following things in this section:

- Go through a quick EDA on the data (again, feel free to ignore the Date column)
- You may want to rename the variables (not required)
- Your focus should be on the response variable we'll use: Rented Bike Count (renamed `rented_bike_count`)

- We'll also create a binary version of that variable for use with logistic regression. Create a new variable that is 1 if the number of bikes rented is greater than or equal to 650 and 0 otherwise.

Please note the EDA will take place on the training data.

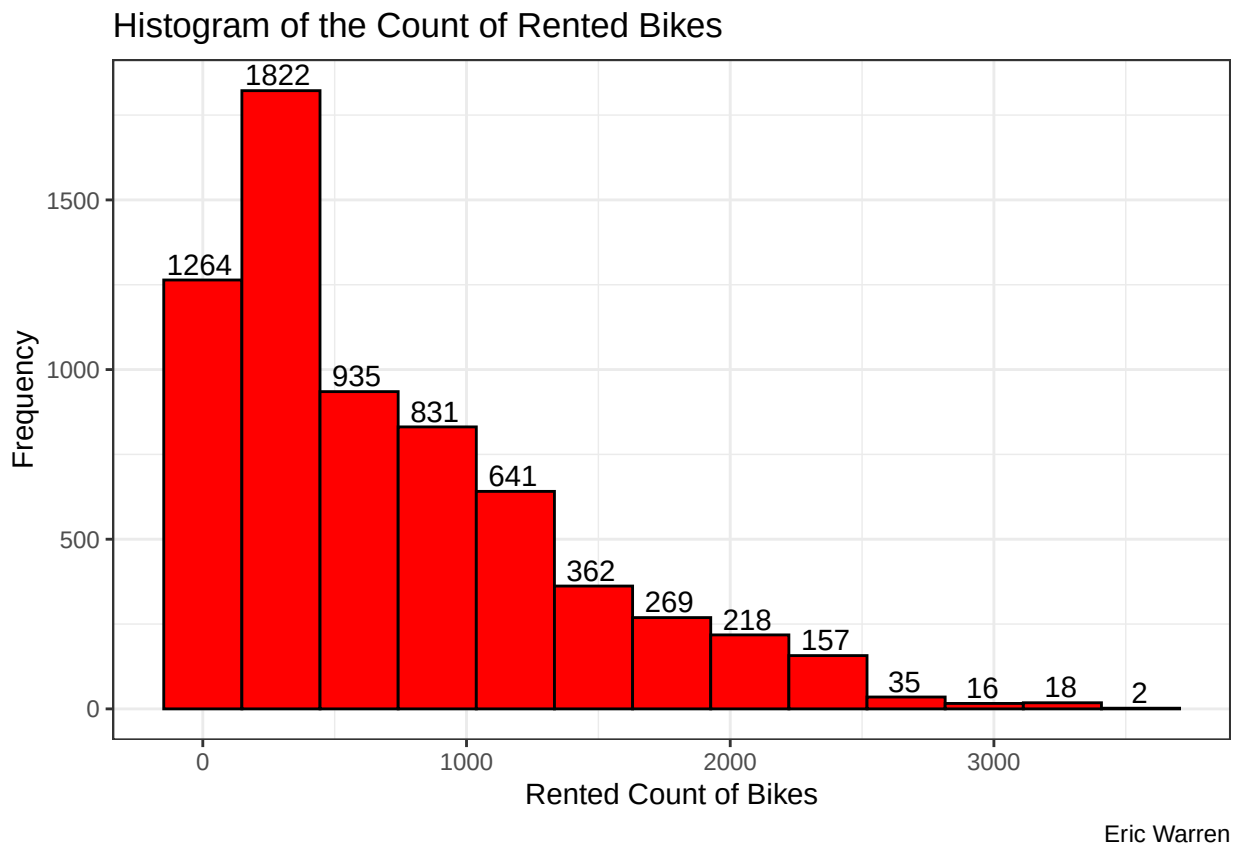
3.1 Exploring the Data

The first thing I am going to do is to clean the names of the training and testing data so they match up in the future. We will use the `janitor::clean_names()` function to do just this.

```
bikeTrain <- janitor::clean_names(bikeTrain)
bikeTest <- janitor::clean_names(bikeTest)
```

Now let us to begin some exploratory data analysis on the data. The first thing I want to do is to get a histogram of what the rented bike count is for each observation. I am going to make a histogram that shows this to see what the breakdown is and what is the most frequent bike counts.

```
# Rule to find the optimal number of bins
bins <- ceiling(min(sqrt(length(bikeTrain$rented_bike_count)),
    log2(length(bikeTrain$rented_bike_count))))
bikeTrain %>%
    ggplot(aes(x = rented_bike_count)) + geom_histogram(bins = bins,
    fill = "red", col = "black") + stat_bin(bins = bins, geom = "text",
    aes(label = ..count..), vjust = -0.25, hjust = 0.55) + labs(x = "Rented Count of Bikes",
    y = "Frequency", title = "Histogram of the Count of Rented Bikes",
    caption = "Eric Warren") + theme_bw()
```

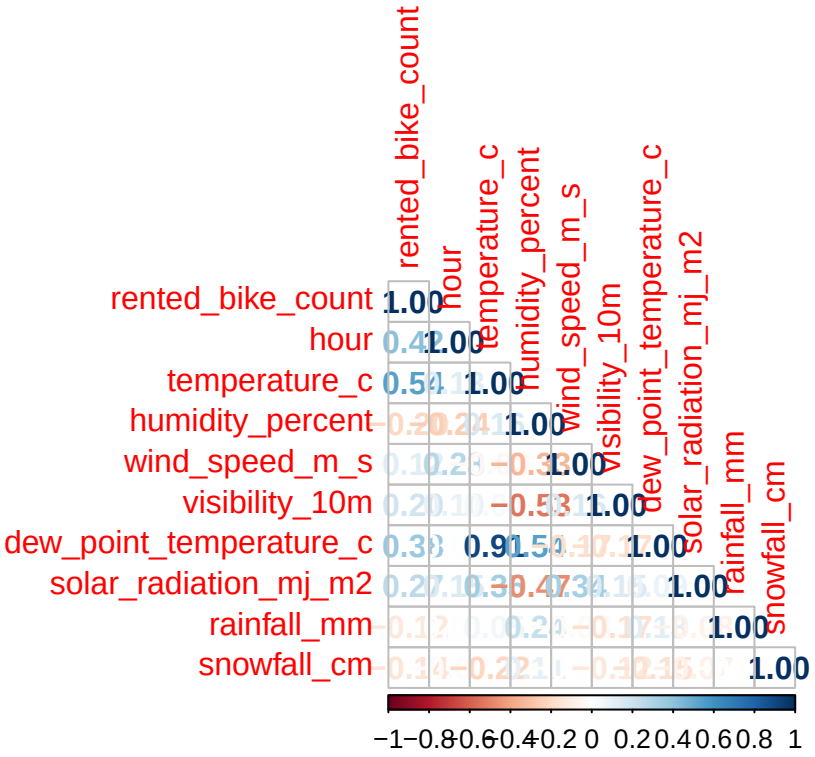


As we can see from this histogram, the majority for the bike counts are definitely less than 1000 and we can see how we have some outliers in the 3000s. This distribution seems to be very skewed right. This is something to keep in mind when we want to model later that our response does not look to be normal.

We can also take a look at all the different correlations between the numerical variables. Let's see if we spot any high correlations.

```
library(corrplot)

correlations <- cor(bikeTrain[sapply(bikeTrain, is.numeric)])
corrplot(correlations, method = "number", type = "lower")
```

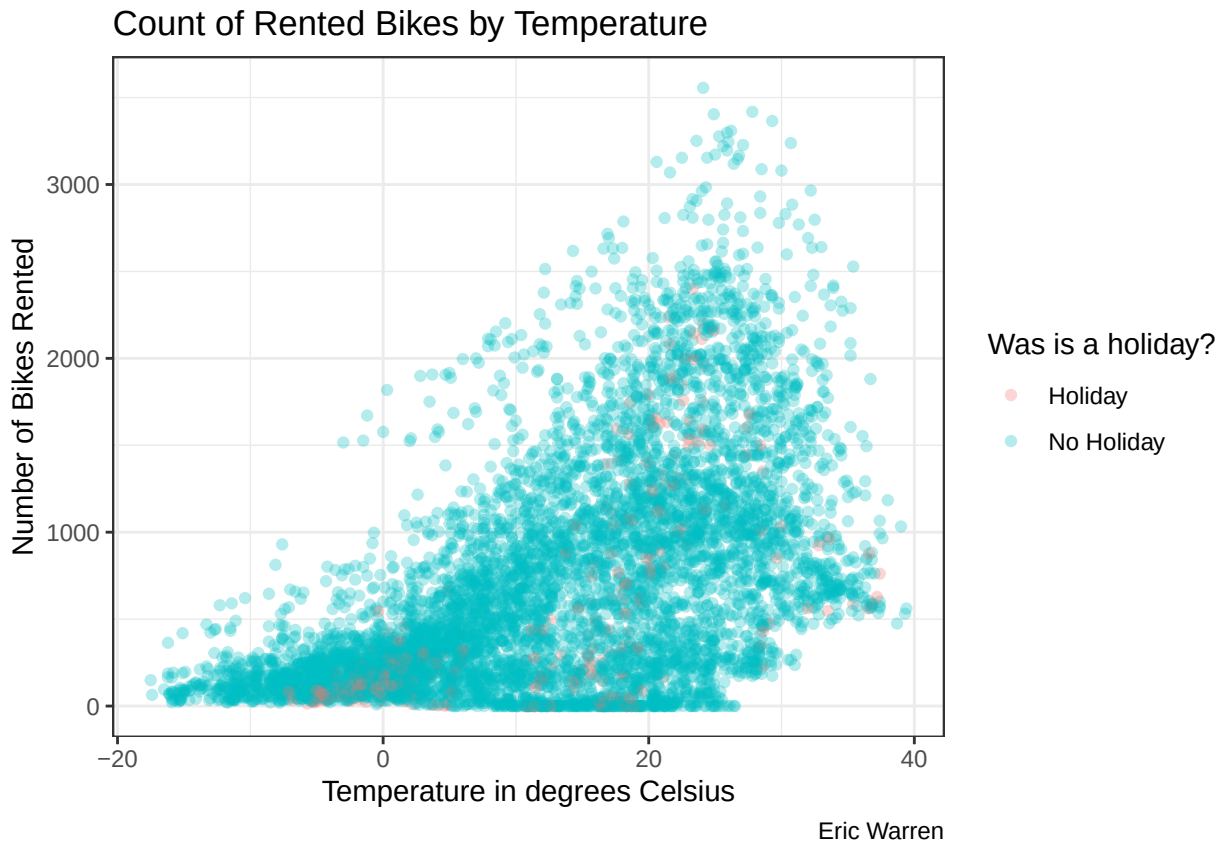


There is not a lot of correlation that exists between the numerical variables as really only the dew point temperatures and the actual temperature are the only pair that show high correlation (from studying weather we know this makes sense since the dew point and temperatures do have correlation). The only two pairs that show some moderate correlation is the dew point temperature and the humidity (also makes some sense with what we know with weather) and the visibility and humidity (which might make some sense since humidity could potentially cause fog which makes it harder to see).

Now we are going to take a look at a scatter plot seeing if the temperature causes more bikes to be rented. We are also going to group it by the holiday variable to see if maybe holidays on nicer temperature days cause more people to rent bikes.

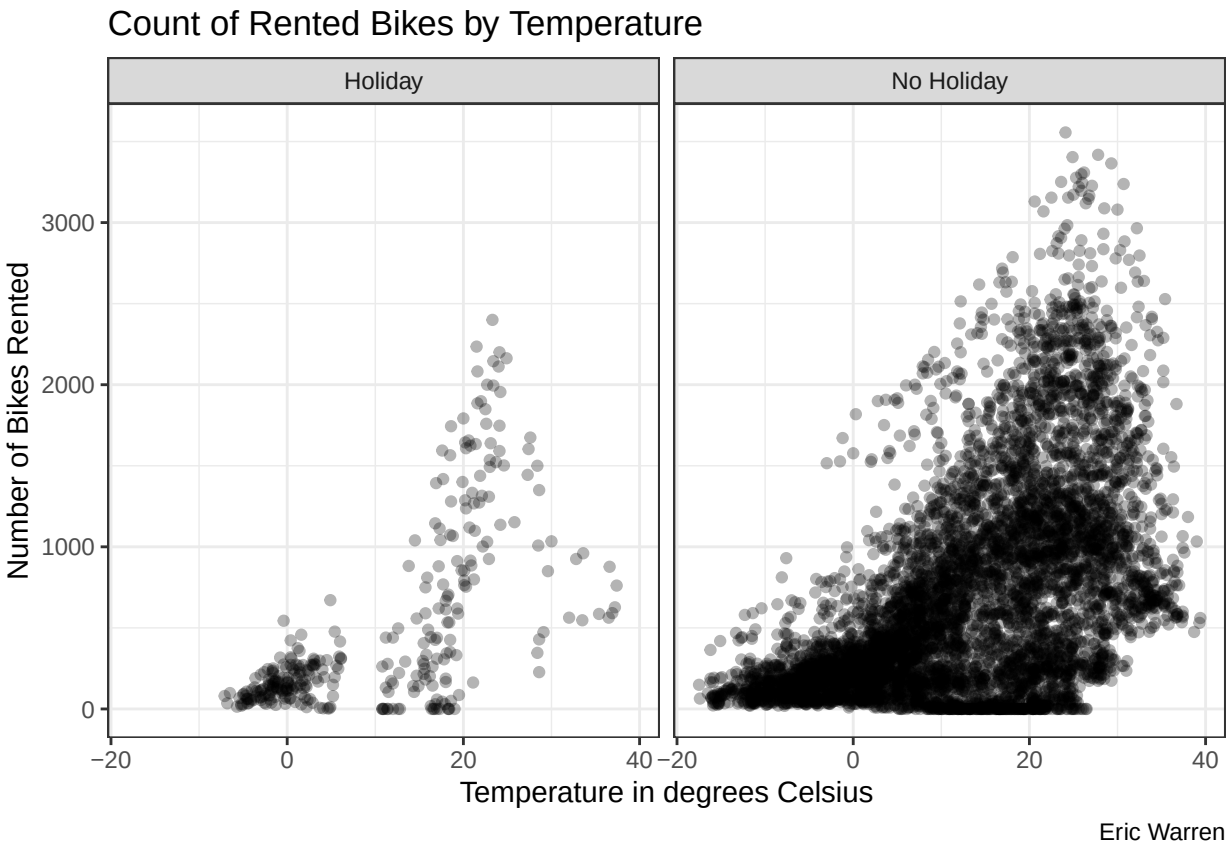
```
bikeTrain %>%
    ggplot(aes(x = temperature_c, y = rented_bike_count)) + geom_point(alpha = 0.3,
    aes(color = holiday)) + labs(x = "Temperature in degrees Celsius",
    y = "Number of Bikes Rented", color = "Was is a holiday?",
```

```
    title = "Count of Rented Bikes by Temperature", caption = "Eric Warren") +
    theme_bw()
```



So it looks like there is an increasing trend with temperature causing more rented bikes until it gets "too hot" for people and then it starts to decrease around 25-30 degrees Celsius (around 77 to 86 degrees Fahrenheit). It does not seem holiday makes a difference on the number of rented bikes but we can make a side by side scatterplot faceting off of holiday to double check.

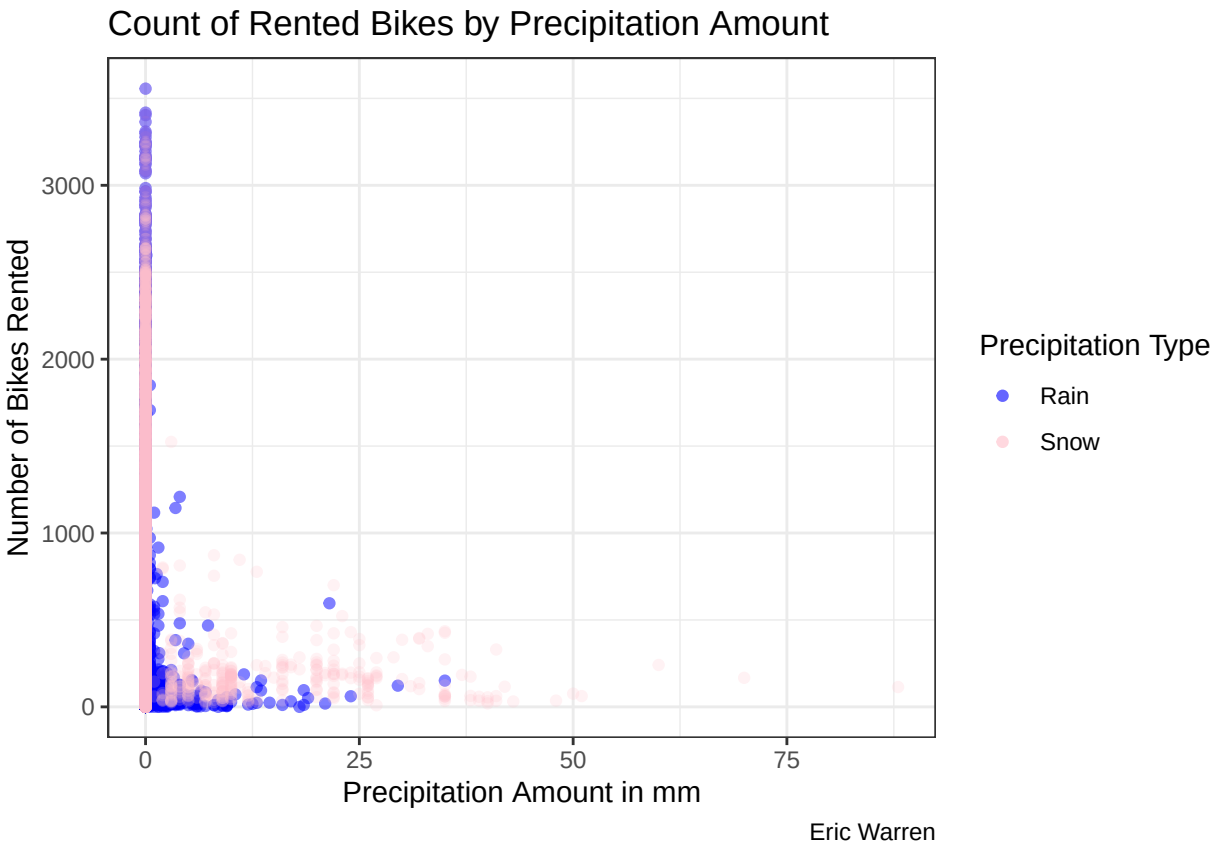
```
bikeTrain %>%
    ggplot(aes(x = temperature_c, y = rented_bike_count)) + geom_point(alpha = 0.3) +
    labs(x = "Temperature in degrees Celsius", y = "Number of Bikes Rented",
        title = "Count of Rented Bikes by Temperature", caption = "Eric Warren") +
    facet_wrap(~holiday) + theme_bw()
```



It seems that it being a holiday does not have an effect on the number of bikes rented but the temperature seems to have a quadratic effect on rented bikes.

Does rain or snow just mean the same thing in terms of precipitation? Let us look and see if this is the case. If so, we can create only one column (precipitation) later on.

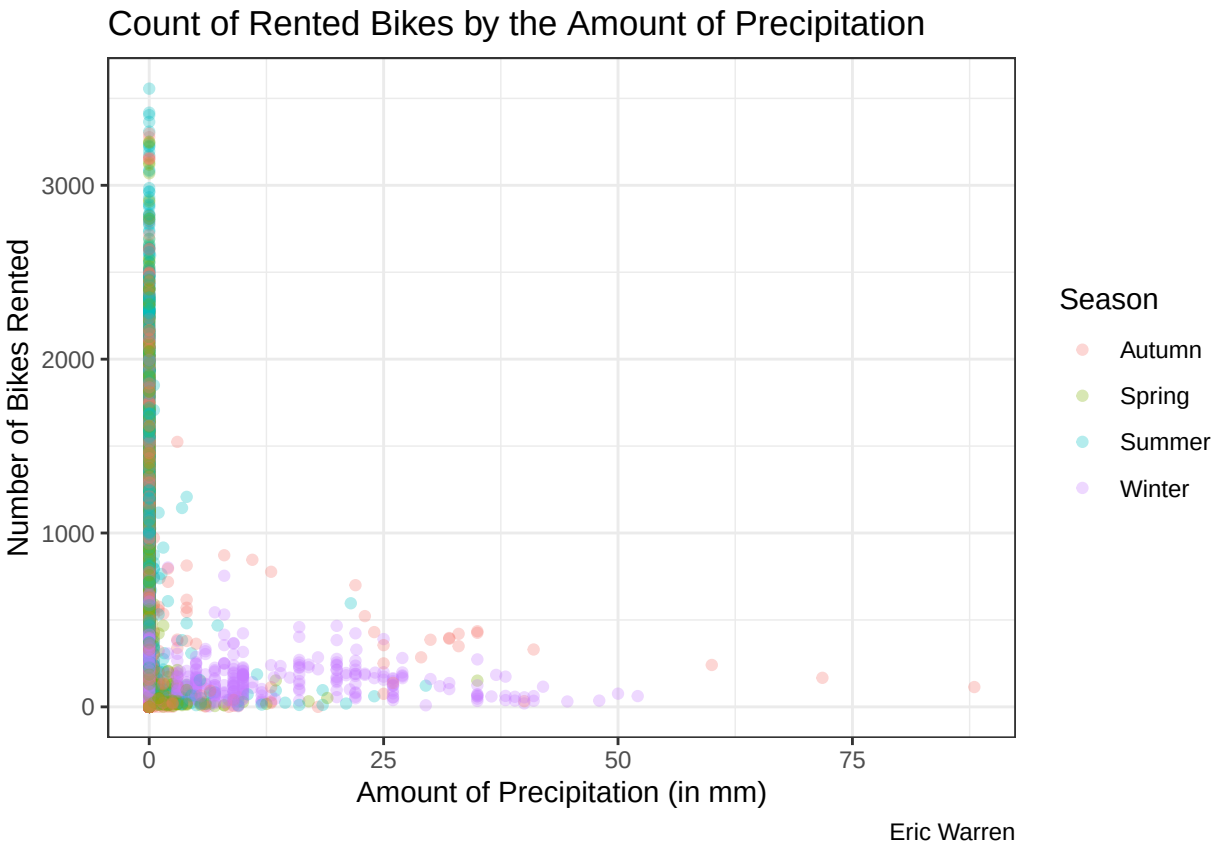
```
bikeTrain %>%
    ggplot(aes(y = rented_bike_count)) + geom_point(alpha = 0.5,
    aes(x = rainfall_mm, color = "Rain")) + geom_point(alpha = 0.2,
    aes(x = snowfall_cm * 10, color = "Snow")) + scale_color_manual(name = "Precipitation Type",
    breaks = c("Rain", "Snow"), values = c(Rain = "blue", Snow = "pink")) +
    labs(x = "Precipitation Amount in mm", y = "Number of Bikes Rented",
        title = "Count of Rented Bikes by Precipitation Amount",
        caption = "Eric Warren") + theme_bw()
```



As we can see, rain and snow really do not matter in terms of bike sales so we can analyze them together as one variable. Also rain and snow cannot happen at the same time (so they are independent) which is why we can do this.

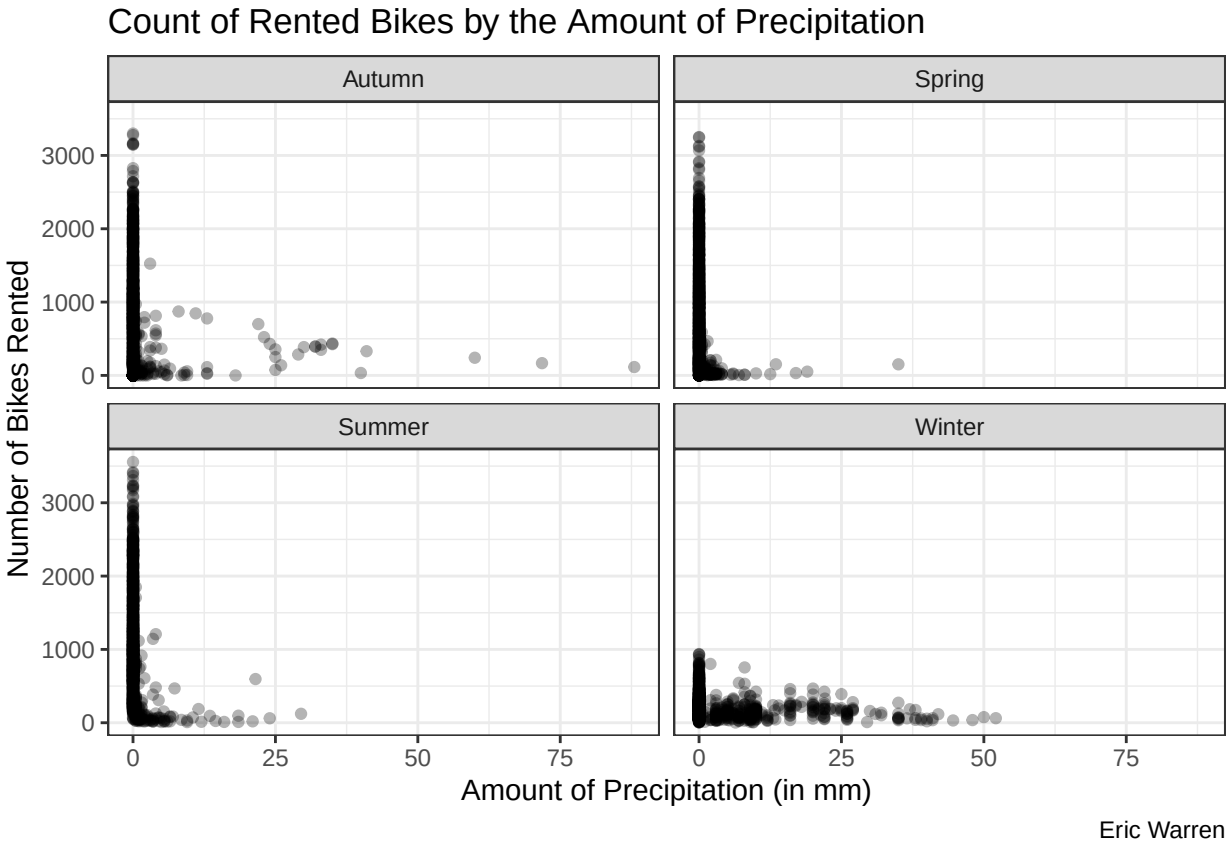
Does precipitation matter? What about by season? We are going to look at the precipitation amounts by adding snow and rainfall together (since snow is in cm and rain in mm we are going to say 10*snow + rain to keep it in mm). We are going to have a grouping variable by season to see if the amount of precipitation has a correlation with season or the amount of bikes rented.

```
bikeTrain %>%
    ggplot(aes(x = rainfall_mm + 10 * snowfall_cm, y = rented_bike_count)) +
    geom_point(alpha = 0.3, aes(color = seasons)) + labs(x = "Amount of Precipitation (in mm)",
    y = "Number of Bikes Rented", color = "Season", title = "Count of Rented Bikes by the Amount of Pre
    caption = "Eric Warren") + theme_bw()
```



It looks like there is more precipitation in winter and autumn and people tend to rent more bikes when there is not precipitation which makes sense. We are going to now facet by season to look at each season individually.

```
bikeTrain %>%
    ggplot(aes(x = rainfall_mm + 10 * snowfall_cm, y = rented_bike_count)) +
    geom_point(alpha = 0.3) + labs(x = "Amount of Precipitation (in mm)",
    y = "Number of Bikes Rented", title = "Count of Rented Bikes by the Amount of Precipitation",
    caption = "Eric Warren") + facet_wrap(~seasons) + theme_bw()
```



Again we can see more precipitation in autumn and winter. The number of bikes rented and the precipitation levels seems to be an exponential decay function. We can also see the seasons of summer and spring tend to do better (with autumn not too far behind) of months where people rent more bikes but this is probably due to the precipitation amounts being lower at those times.

After looking at some thing that might have been connected I wanted to do a summary of the numeric variables to see how they are summarized.

```
summary(bikeTrain[sapply(bikeTrain, is.numeric)])
```

```
##  rented_bike_count      hour       temperature_c    humidity_percent
##  Min.   :   0.0    Min.   : 0.00   Min.   :-17.50   Min.   : 0.00
##  1st Qu.: 193.0    1st Qu.: 5.00   1st Qu.:  3.50   1st Qu.:42.00
##  Median : 509.0    Median :11.00   Median : 13.80   Median :57.00
##  Mean   : 708.7    Mean   :11.45   Mean   : 12.93   Mean   :58.15
##  3rd Qu.:1063.8    3rd Qu.:17.00   3rd Qu.: 22.60   3rd Qu.:74.00
##  Max.   :3556.0    Max.   :23.00   Max.   : 39.40   Max.   :98.00
##  wind_speed_m_s  visibility_10m dew_point_temperature_c solar_radiation_mj_m2
##  Min.   :0.000   Min.   :  27   Min.   :-30.600         Min.   :0.0000
##  1st Qu.:0.900   1st Qu.: 953   1st Qu.: -4.800         1st Qu.:0.0000
##  Median :1.500   Median :1708   Median :  5.200         Median :0.0100
##  Mean   :1.729   Mean   :1443   Mean   :  4.099         Mean   :0.5689
##  3rd Qu.:2.400   3rd Qu.:2000   3rd Qu.: 14.900         3rd Qu.:0.9200
##  Max.   :7.300   Max.   :2000   Max.   : 27.200         Max.   :3.5200
##   rainfall_mm      snowfall_cm
##  Min.   : 0.000   Min.   :0.00000
```

```
##  1st Qu.: 0.000   1st Qu.:0.00000
##  Median : 0.000   Median :0.00000
##  Mean   : 0.162   Mean   :0.07227
##  3rd Qu.: 0.000   3rd Qu.:0.00000
##  Max.   :35.000   Max.   :8.80000
```

We should note some of the numeric variables where we should point out some important thing. First let us look at those where the median and mean are not close to matching. The first is our rented bike count which we already determined is skewed left and not normally distributed. This summary reaffirms that by the median being so much smaller than the mean (causing high outliers to drive up the average number of bikes rented). The visibility is something that looks to be skewed left as the mean is much lower than the median. Thus there are some outliers where the visibility is low. Lastly, if we look at both precipitation levels (rain and snow) there are at least 75% of observations where it does not precipitate (since the 3^{rd} quartile is 0). So we can say that it does not precipitate a ton but when it does it could potentially change the bike rented amount (which we showed earlier it does).

Now we might want to look at the categorical variables to see what the breakdown of those are and see how even things are. Things that are not close to what we would expect would definitely affect our results since it wouldn't be a "normal situation" most of the time.

```
# Get the character columns
character_columns <- colnames(bikeTrain[sapply(bikeTrain, is.character)])
# Do not want to look at date column
character_columns <- character_columns[character_columns != "date"]

# Create list of tables
character_df <- bikeTrain %>%
    dplyr::select(one_of(character_columns))  # Select only character columns we want

(myListTables <- mapply(table, character_df))  # Show list of tables
```

```
## $seasons
##
## Autumn Spring Summer Winter
##   1643   1652   1660   1615
##
## $holiday
##
##    Holiday No Holiday
##         323       6247
##
## $functioning_day
##
##   No  Yes
##  208 6362
```

From all of this, we can see that the seasons are broken down fairly evenly which is good since we have seasons with roughly equal lengths. Holidays are less common than no holiday days so this makes sense to see it largely in favor of no holiday days and we would expect a lot more functioning days than non-functioning days. So our breakdown of the character columns make sense too. It seems we have a pretty good idea of what our data looks like so we can start with manipulating our data before modeling.

3.2 Changing Variables

In this section we are going to make new variables while also manipulating the ones we have already. This will be done before we get into our modeling stage. Some variables we are changing include creating a binary variable for the count of rented bikes if it is greater than or equal to 650 (1) or not (0). The other thing I am going to do is turn all precipitation into one measurement (combine rain and snow) which will be in mm. Personally I believe that rain and snowfalls will not affect how people feel about wanting to rent a bike and I am going to remove the snow and rainfall columns once I create this now one to avoid collinearity. I am also removing the `date` since we said we do not need it. I would also like to try to scale the number of rented bikes since there are many large outliers and these might mess up our model but we can do this later since we want the train and test data to have the same mean and variance it is being scaled on.

```
# Do this on training data first Note precipitation is in
# mm and snow is in cm so 10*snowfall_cm = snowfall_mm
# Remove rainfall and snowfall
bikeTrain <- bikeTrain %>%
    mutate(large_rental_day = ifelse(rented_bike_count >= 650,
        1, 0), precipitation_mm = rainfall_mm + 10 * snowfall_cm) %>%
    dplyr::select(-c(rainfall_mm, snowfall_cm, date)) %>%
    dplyr::select(rented_bike_count, large_rental_day, everything())

# Do this on testing data now Note precipitation is in mm
# and snow is in cm so 10*snowfall_cm = snowfall_mm Remove
# rainfall and snowfall
bikeTest <- bikeTest %>%
    mutate(large_rental_day = ifelse(rented_bike_count >= 650,
        1, 0), precipitation_mm = rainfall_mm + 10 * snowfall_cm) %>%
    dplyr::select(-c(rainfall_mm, snowfall_cm, date)) %>%
    dplyr::select(rented_bike_count, large_rental_day, everything())
```

We have manipulated the variables we wanted to do so now we can go onto our modeling component.

4 Fitting MLR Models

We are now ready to fit some models on the training set and compare them on the test set. Choosing variables for a regression model is a pretty difficult task, especially when you aren't an expert in the data being used. That being said, there are some methods we can use. One group of methods is forward, backward, and best subset regression (see chapter 6 from the ISLR book, note that best subset regression isn't implemented in `caret`). These aren't theoretically sound methods and are generally not recommended, but they are still widely used. We should be familiar with them! These models can be fit with the `caret` package.

Construct three regression models:

- One via forward selection
  - Including interaction terms and higher-order polynomial terms can be odd here so you may want to leave those out
- One via backward selection (no need to change any of the settings on the methods (like entry/exit criteria))
  - Including interaction terms and higher-order polynomial terms can be odd here so you may want to leave those out
- Fit a LASSO model using cross-validation to determine the tuning parameter

- Again, using polynomial terms and interaction terms can be problematic here so I'd leave those out
- Getting the final model coefficients out is a little tougher than it should be! This will give you the info you need: `predict(lasso$finalModel, type="coef")` if you combine it with the `bestTune` result

The first thing we will do is get a model using forward selection.

```
library(caret)
library(leaps)
library(MASS)

# Set seed for reproducibility
set.seed(999)

# Train the model using forward selection
forward_model <- train(rented_bike_count ~ . - large_rental_day,
    data = bikeTrain, method = "leapForward", preProcess = c("scale",
        "center"), trControl = trainControl(method = "cv", number = 10))

# Coefficients of best model
coef(forward_model$finalModel, id = forward_model$bestTune[[1]])
```

```
##        (Intercept)               hour      temperature_c   humidity_percent
##           708.7409          188.1279           355.3422          -139.8391
## functioning_dayYes
##           140.0213
```

Now we are going to do the same thing with a backwards selection model.

```
# Set seed for reproducibility
set.seed(999)

# Train the model using forward selection
backward_model <- train(rented_bike_count ~ . - large_rental_day,
    data = bikeTrain, method = "leapBackward", preProcess = c("scale",
        "center"), trControl = trainControl(method = "cv", number = 10))

# Coefficients of best model
coef(backward_model$finalModel, id = backward_model$bestTune[[1]])
```

```
##        (Intercept)               hour      temperature_c   humidity_percent
##           708.7409          188.1279           355.3422          -139.8391
## functioning_dayYes
##           140.0213
```

Something to note is the forwards and backwards selection gave us the same essential best model.

Now we are going to do it for LASSO.

```r
# Set seed for reproducibility
set.seed(999)

# Train the model using forward selection
lasso_model <- train(rented_bike_count ~ . - large_rental_day,
    data = bikeTrain, method = "glmnet", preProcess = c("scale",
        "center"), trControl = trainControl(method = "cv", number = 10))

# Coefficients of best model
lambda_use <- min(lasso_model$finalModel$lambda[lasso_model$finalModel$lambda >=
    lasso_model$bestTune$lambda])
position <- which(lasso_model$finalModel$lambda == lambda_use)
data.frame(coef(lasso_model$finalModel)[, position])
```

```
##                         coef.lasso_model.finalModel....position.
## (Intercept)                                            708.740944
## hour                                                   190.416141
## temperature_c                                          207.930163
## humidity_percent                                      -224.491364
## wind_speed_m_s                                         13.334138
## visibility_10m                                          9.469240
## dew_point_temperature_c                                117.516513
## solar_radiation_mj_m2                                  -65.908329
## seasonsSpring                                          -56.773826
## seasonsSummer                                          -57.728069
## seasonsWinter                                         -153.368747
## holidayNo Holiday                                      30.264226
## functioning_dayYes                                     160.179638
## precipitation_mm                                        -4.519959
```

As we can see the Lasso model is different than that of the first two models we looked at.

Now, we are going to create three models that include interaction terms and polynomial terms of your choosing. Use cross-validation to compare these models and select a 'best' model from the three. We are going to first make a model with all the possible interaction terms from the backwards selection variables. The second model is going to be a quadratic model with all the backwards selection variables. The last model is going to be a model of what I found to be interested in which is including hour, functioning_day, temperature_c, humidity, and interaction of humidity and temperature. Let us see how these models compare.

```r
# Set seed for reproducibility
set.seed(999)

# Train the first model we create
made_lm_model1 <- train(rented_bike_count ~ (hour + temperature_c +
    humidity_percent + functioning_day)^2, data = bikeTrain,
    method = "lm", preProcess = c("scale", "center"), trControl = trainControl(method = "cv",
        number = 10))

# Show coefficients
made_lm_model1$finalModel$coefficients
```

```
##                         (Intercept)                                hour
##                           708.74094                           66.83102
```

```
##                          temperature_c                      humidity_percent
##                              -148.68971                            209.10543
##                     functioning_dayYes                  `hour:temperature_c`
##                               87.37936                            366.90912
##                `hour:humidity_percent`             `hour:functioning_dayYes`
##                              -266.23556                            227.88039
##      `temperature_c:humidity_percent`    `temperature_c:functioning_dayYes`
##                              -120.18419                            335.61430
## `humidity_percent:functioning_dayYes`
##                              -180.49337
```

```
# Train the second model we create
made_lm_model2 <- train(rented_bike_count ~ poly(hour, 2) + poly(temperature_c,
    2) + poly(humidity_percent, 2) + functioning_day, data = bikeTrain,
    method = "lm", preProcess = c("scale", "center"), trControl = trainControl(method = "cv",
        number = 10))

# Show coefficients
made_lm_model2$finalModel$coefficients
```

```
##                   (Intercept)             `poly(hour, 2)1`
##                     708.74094                   187.15122
##             `poly(hour, 2)2`    `poly(temperature_c, 2)1`
##                      14.39840                   352.02054
##    `poly(temperature_c, 2)2` `poly(humidity_percent, 2)1`
##                     -66.10613                  -157.27667
## `poly(humidity_percent, 2)2`           functioning_dayYes
##                    -114.22310                   154.81396
```

```
# Train the third model we create
made_lm_model3 <- train(rented_bike_count ~ hour + temperature_c +
    humidity_percent + functioning_day + temperature_c:humidity_percent,
    data = bikeTrain, method = "lm", preProcess = c("scale",
        "center"), trControl = trainControl(method = "cv", number = 10))

# Show coefficients
made_lm_model3$finalModel$coefficients
```

```
##                      (Intercept)                             hour
##                        708.74094                        187.44217
##                      temperature_c                   humidity_percent
##                         564.85078                         -54.16285
##                functioning_dayYes `temperature_c:humidity_percent`
##                         143.39654                        -248.97317
```

```
# Pick the best model from training data
made_lm_model1$results$RMSE
```

```
## [1] 423.5102
```

```
made_lm_model2$results$RMSE
```

```
## [1] 439.8596
```

```
made_lm_model3$results$RMSE
```

```
## [1] 452.0043
```

```
# Pick the best model from test data
postResample(predict(made_lm_model1, bikeTest), obs = bikeTest$rented_bike_count)
```

```
##        RMSE    Rsquared         MAE
## 416.1257786   0.5697054 297.1369626
```

```
postResample(predict(made_lm_model2, bikeTest), obs = bikeTest$rented_bike_count)
```

```
##        RMSE    Rsquared         MAE
## 429.1445702   0.5424066 324.9359378
```

```
postResample(predict(made_lm_model3, bikeTest), obs = bikeTest$rented_bike_count)
```

```
##        RMSE    Rsquared         MAE
## 442.1905873   0.5139978 329.6459081
```

As we can see the best model is the first model we made. We can use this to compare to the backwards, forwards, and lasso models we made earlier.

```
# Look at train RMSE (just curious)
min(forward_model$results$RMSE)
```

```
## [1] 457.2616
```

```
min(backward_model$results$RMSE)
```

```
## [1] 457.7295
```

```
min(lasso_model$results$RMSE)
```

```
## [1] 440.7947
```

```
made_lm_model1$results$RMSE
```

```
## [1] 423.5102
```

```
# Compare best model with test RMSE
postResample(predict(forward_model, bikeTest), obs = bikeTest$rented_bike_count)
```

```
##        RMSE    Rsquared         MAE
## 448.1379310   0.5008417 337.4295216
```

```
postResample(predict(backward_model, bikeTest), obs = bikeTest$rented_bike_count)
```

```
##        RMSE    Rsquared         MAE
## 448.1379310   0.5008417 337.4295216
```

```
postResample(predict(lasso_model, bikeTest), obs = bikeTest$rented_bike_count)
```

```
##        RMSE    Rsquared         MAE
## 428.8261304   0.5428342 326.3629809
```

```
postResample(predict(made_lm_model1, bikeTest), obs = bikeTest$rented_bike_count)
```

```
##        RMSE    Rsquared         MAE
## 416.1257786   0.5697054 297.1369626
```

As we can see for our model that was created using the interaction terms along with the columns selected from our backwards selection model turned out to be the best model we could create using linear regression. So if we had to make and pick a model that `made_lm_model1` is the best model out of the ones we have looked at.

5 Logistic Regression

We are going to do the same thing we did in the **Linear Regression** section. Start off by constructing three regression models:

- One via forward selection
  - Including interaction terms and higher-order polynomial terms can be odd here so you may want to leave those out
- One via backward selection (no need to change any of the settings on the methods (like entry/exit criteria))
  - Including interaction terms and higher-order polynomial terms can be odd here so you may want to leave those out
- Fit a LASSO model using cross-validation to determine the tuning parameter
  - Again, using polynomial terms and interaction terms can be problematic here so I'd leave those out
  - Getting the final model coefficients out is a little tougher than it should be! This will give you the info you need: `predict(lasso$finalModel, type="coef")` if you combine it with the `bestTune` result

The first thing we will do is get a model using forward selection.

```
library(caret)
library(leaps)
library(MASS)

# Set seed for reproducibility
set.seed(999)

# Train the model using forward selection
forward_model_log <- train(large_rental_day ~ . - rented_bike_count,
    data = bikeTrain, method = "leapForward", family = "binomial",
    preProcess = c("scale", "center"), trControl = trainControl(method = "cv",
        number = 10))

# Coefficients of best model
coef(forward_model_log$finalModel, id = forward_model_log$bestTune[[1]])
```

```
##      (Intercept)             hour    temperature_c humidity_percent
##        0.4307458        0.1183774        0.1736226       -0.1280133
##    seasonsWinter
##       -0.1368876
```

Now we are going to do the same thing with a backwards selection model.

```
# Set seed for reproducibility
set.seed(999)

# Train the model using forward selection
backward_model_log <- train(large_rental_day ~ . - rented_bike_count,
    data = bikeTrain, method = "leapBackward", family = "binomial",
    preProcess = c("scale", "center"), trControl = trainControl(method = "cv",
        number = 10))

# Coefficients of best model
coef(backward_model_log$finalModel, id = backward_model_log$bestTune[[1]])
```

```
##             (Intercept)                    hour        humidity_percent
##               0.4307458               0.1201208              -0.2085352
## dew_point_temperature_c           seasonsWinter
##               0.2013149              -0.1392080
```

Unlike with linear regression, the forward and backward selection techniques produce different results

Now we are going to do it for LASSO.

```
# Set seed for reproducibility
set.seed(999)

# Train the model using forward selection
lasso_model_log <- train(large_rental_day ~ . - rented_bike_count,
    data = bikeTrain, method = "glmnet", family = "binomial",
    preProcess = c("scale", "center"), trControl = trainControl(method = "cv",
        number = 10))
```

```
# Coefficients of best model
lambda_use <- min(lasso_model_log$finalModel$lambda[lasso_model_log$finalModel$lambda >=
    lasso_model_log$bestTune$lambda])
position <- which(lasso_model_log$finalModel$lambda == lambda_use)
data.frame(coef(lasso_model_log$finalModel)[, position])
```

```
##                          coef.lasso_model_log.finalModel....position.
## (Intercept)                                                -0.50011821
## hour                                                        0.57181402
## temperature_c                                               0.53391831
## humidity_percent                                           -0.40056662
## wind_speed_m_s                                              0.00000000
## visibility_10m                                              0.11085269
## dew_point_temperature_c                                     0.24798925
## solar_radiation_mj_m2                                       0.32096339
## seasonsSpring                                              -0.07282229
## seasonsSummer                                               0.00000000
## seasonsWinter                                              -0.72374939
## holidayNo Holiday                                           0.08540699
## functioning_dayYes                                          0.41583356
## precipitation_mm                                           -0.18420623
```

As we can see Lasso uses more predictors to make its model.

Now, we are going to create three models that include interaction terms and polynomial terms of your choosing. Use cross-validation to compare these models and select a 'best' model from the three. We are going to first make a model with all the possible interaction terms from the forwards and backwards selection variables that appeared. The second model is going to be a quadratic model with all the forwards and backwards selection variables. The last model is going to be a model of what I found to be interested in which is including hour, functioning_day, temperature_c, season, humidity, and interaction of humidity and temperature. Let us see how these models compare.

```
# Set seed for reproducibility
set.seed(999)

# Train the first model we create
made_log_model1 <- train(large_rental_day ~ (hour + temperature_c +
    humidity_percent + seasons + dew_point_temperature_c)^2,
    data = bikeTrain, method = "glm", family = "binomial", preProcess = c("scale",
        "center"), trControl = trainControl(method = "cv", number = 10))

# Show coefficients
made_log_model1$finalModel$coefficients
```

```
##                                (Intercept)
##                                -0.70654469
##                                       hour
##                                2.60205661
##                             temperature_c
##                                -0.47710134
##                           humidity_percent
##                                -0.29009125
```

```
##                               seasonsSpring
##                                  -1.48157325
##                               seasonsSummer
##                                   4.49015050
##                               seasonsWinter
##                                  -2.55311322
##                     dew_point_temperature_c
##                                   0.86796284
##                        `hour:temperature_c`
##                                  -0.57731469
##                     `hour:humidity_percent`
##                                  -1.68208386
##                        `hour:seasonsSpring`
##                                  -0.02398382
##                        `hour:seasonsSummer`
##                                   0.23124725
##                        `hour:seasonsWinter`
##                                  -0.77125528
##              `hour:dew_point_temperature_c`
##                                   0.82901171
##            `temperature_c:humidity_percent`
##                                   4.67234904
##               `temperature_c:seasonsSpring`
##                                   1.31032687
##               `temperature_c:seasonsSummer`
##                                  -4.58765682
##               `temperature_c:seasonsWinter`
##                                  -0.21294619
##     `temperature_c:dew_point_temperature_c`
##                                  -1.29977442
##            `humidity_percent:seasonsSpring`
##                                   0.17625191
##            `humidity_percent:seasonsSummer`
##                                  -3.33415468
##            `humidity_percent:seasonsWinter`
##                                   1.59123095
## `humidity_percent:dew_point_temperature_c`
##                                  -3.48005560
##     `seasonsSpring:dew_point_temperature_c`
##                                  -0.16072138
##     `seasonsSummer:dew_point_temperature_c`
##                                   3.53930984
##     `seasonsWinter:dew_point_temperature_c`
##                                  -0.59982390
```

```r
# Train the second model we create
made_log_model2 <- train(large_rental_day ~ poly(hour, 2) + poly(temperature_c,
    2) + poly(humidity_percent, 2) + poly(dew_point_temperature_c) +
    seasons, data = bikeTrain, method = "glm", family = "binomial",
    preProcess = c("scale", "center"), trControl = trainControl(method = "cv",
        number = 10))

# Show coefficients
made_log_model2$finalModel$coefficients
```

```
##                       (Intercept)                `poly(hour, 2)1`
##                        -1.1059068                       0.9018358
##                `poly(hour, 2)2`       `poly(temperature_c, 2)1`
##                        -0.0583308                       3.2024061
##       `poly(temperature_c, 2)2`    `poly(humidity_percent, 2)1`
##                        -0.8353099                      -0.1149364
##    `poly(humidity_percent, 2)2` `poly(dew_point_temperature_c)`
##                        -0.5363148                      -2.0163364
##                     seasonsSpring                    seasonsSummer
##                        -0.2506069                       0.1559896
##                     seasonsWinter
##                        -1.3097577
```

```
# Train the third model we create
made_log_model3 <- train(large_rental_day ~ hour + temperature_c +
    humidity_percent + seasons + functioning_day + temperature_c:humidity_percent,
    data = bikeTrain, method = "glm", family = "binomial", preProcess = c("scale",
        "center"), trControl = trainControl(method = "cv", number = 10))

# Show coefficients
made_log_model3$finalModel$coefficients
```

```
##                       (Intercept)                              hour
##                         -1.292655                         0.9150662
##                       temperature_c                  humidity_percent
##                          2.119592                        -0.6200273
##                       seasonsSpring                     seasonsSummer
##                         -0.469881                        -0.3592424
##                       seasonsWinter                functioning_dayYes
##                         -1.762365                         3.3965000
## `temperature_c:humidity_percent`
##                         -0.838400
```

```
# Pick the best model from train data (just to see and
# compare)
made_model1_predictions <- predict(made_log_model1, bikeTrain)
accuracy_made_model1 <- table(bikeTrain$large_rental_day, made_model1_predictions >=
    0.5)
(accuracy_made_model1[2, 2] + accuracy_made_model1[1, 1])/sum(accuracy_made_model1)
```

```
## [1] 0.8534247
```

```
made_model2_predictions <- predict(made_log_model2, bikeTrain)
accuracy_made_model2 <- table(bikeTrain$large_rental_day, made_model2_predictions >=
    0.5)
(accuracy_made_model2[2, 2] + accuracy_made_model2[1, 1])/sum(accuracy_made_model2)
```

```
## [1] 0.8515982
```

```
made_model3_predictions <- predict(made_log_model3, bikeTrain)
accuracy_made_model3 <- table(bikeTrain$large_rental_day, made_model3_predictions >=
    0.5)
(accuracy_made_model3[2, 2] + accuracy_made_model3[1, 1])/sum(accuracy_made_model3)
```

```
## [1] 0.8601218
```

```
# Pick the best model from test data
made_model1_predictions <- predict(made_log_model1, bikeTest)
accuracy_made_model1 <- table(bikeTest$large_rental_day, made_model1_predictions >=
    0.5)
(accuracy_made_model1[2, 2] + accuracy_made_model1[1, 1])/sum(accuracy_made_model1)
```

```
## [1] 0.8438356
```

```
made_model2_predictions <- predict(made_log_model2, bikeTest)
accuracy_made_model2 <- table(bikeTest$large_rental_day, made_model2_predictions >=
    0.5)
(accuracy_made_model2[2, 2] + accuracy_made_model2[1, 1])/sum(accuracy_made_model2)
```

```
## [1] 0.8374429
```

```
made_model3_predictions <- predict(made_log_model3, bikeTest)
accuracy_made_model3 <- table(bikeTest$large_rental_day, made_model3_predictions >=
    0.5)
(accuracy_made_model3[2, 2] + accuracy_made_model3[1, 1])/sum(accuracy_made_model3)
```

```
## [1] 0.8589041
```

As we can see the best model is the third model we made since its accuracy is the highest comparing to the test data (also better for training data but that does not matter). We can use this to compare to the backwards, forwards, and lasso models we made earlier.

```
# Look at train accuracy (just curious)
forward_predictions <- predict(forward_model_log, bikeTrain)
accuracy_forward <- table(bikeTrain$large_rental_day, forward_predictions >=
    0.5)
(accuracy_forward[2, 2] + accuracy_forward[1, 1])/sum(accuracy_forward)
```

```
## [1] 0.8383562
```

```
backward_predictions <- predict(backward_model_log, bikeTrain)
accuracy_backward <- table(bikeTrain$large_rental_day, backward_predictions >=
    0.5)
(accuracy_backward[2, 2] + accuracy_backward[1, 1])/sum(accuracy_backward)
```

```
## [1] 0.8391172
```

```
lasso_predictions <- predict(lasso_model_log, bikeTrain)
accuracy_lasso <- table(bikeTrain$large_rental_day, lasso_predictions >=
    0.5)
(accuracy_lasso[2, 2] + accuracy_lasso[1, 1])/sum(accuracy_lasso)
```

```
## [1] 0.8584475
```

```
made_model3_predictions <- predict(made_log_model3, bikeTrain)
accuracy_made_model3 <- table(bikeTrain$large_rental_day, made_model3_predictions >=
    0.5)
(accuracy_made_model3[2, 2] + accuracy_made_model3[1, 1])/sum(accuracy_made_model3)
```

```
## [1] 0.8601218
```

```
# Compare best model with test accuracy
forward_predictions <- predict(forward_model_log, bikeTest)
accuracy_forward <- table(bikeTest$large_rental_day, forward_predictions >=
    0.5)
(accuracy_forward[2, 2] + accuracy_forward[1, 1])/sum(accuracy_forward)
```

```
## [1] 0.8260274
```

```
backward_predictions <- predict(backward_model_log, bikeTest)
accuracy_backward <- table(bikeTest$large_rental_day, backward_predictions >=
    0.5)
(accuracy_backward[2, 2] + accuracy_backward[1, 1])/sum(accuracy_backward)
```

```
## [1] 0.8278539
```

```
lasso_predictions <- predict(lasso_model_log, bikeTest)
accuracy_lasso <- table(bikeTest$large_rental_day, lasso_predictions >=
    0.5)
(accuracy_lasso[2, 2] + accuracy_lasso[1, 1])/sum(accuracy_lasso)
```

```
## [1] 0.8534247
```

```
made_model3_predictions <- predict(made_log_model3, bikeTest)
accuracy_made_model3 <- table(bikeTest$large_rental_day, made_model3_predictions >=
    0.5)
(accuracy_made_model3[2, 2] + accuracy_made_model3[1, 1])/sum(accuracy_made_model3)
```

```
## [1] 0.8589041
```

As we can see for our model that was created using the varaibles that seemed to be important along with the interaction of temperature and humidity turned out to be the best model we could create using logistic regression. So if we had to make and pick a model that `made_log_model3` is the best model out of the ones we have looked at.