The Effect of Weather on citiBike Usage, and a Geographic Analysis of Bike Flow

Charley Ferrari

Sunday, December 14, 2014

```
#Packages used:

library(rvest)
library(dplyr)
library(plyr)
library(zoo)
library(RPostgreSQL)
library(ggmap)
library(ggplot2)
library(ggthemes)
library(coefplot)
library(scales)
```

Introduction

The CitiBike bikeshare system in New York City produces detailed ride level data for research purposes. This data details each trip taken by a CitiBike, including information about the bike, the rider, timing, and the start and end station. Inspired by the Kaggle competition's bike share challenge, which included aggregate hourly data joined with weather data, I decided to rebuild this framework using the raw rides data coming from CitiBike.

The weather data was scraped directly from the WUnderground website, looping through daily pages. Weekends are expressed as a dummy variable, on an hourly basis. The CitiBike data was just obtained from the CitiBike site as raw CSVs

For analysis, I created a regression model that predicts hourly CitiBike usage based on weather and weekend. Because I had the trip level data, I also created a map with station aggregated data animated by hour, to get a look at an average day's flow of bikes. This data is divided into both a weekday and weekend average.

Data was brought into PostgreSQL as soon as possible. The majority of aggregations were done in PostgreSQL, and tables were then imported into R.

Data Profile

Weather data:

Weather data was scraped using the following code. The for loops were complicated due to the unclean nature of the weather data, and the entire code is reproduced in Appendix A. The loops recreate the html addresses, an example of which can be found here. The table at the bottom of the page was scraped.

Please note, this code is a bit long…

```
monthlookup <- data.frame(
  monthnum = 1:12,
  numdays = c(31,28,31,30,31,30,31,31,30,31,30,31))
```

```r
df <- data.frame(year = numeric(0),
                 month = numeric(0),
                 day = numeric(0),
                 hour = numeric(0),
                 minute = numeric(0),
                 temp = numeric(0),
                 windchill = numeric(0),
                 heatindex = numeric(0),
                 dewpoint = numeric(0),
                 humidity = numeric(0),
                 pressure = numeric(0),
                 visibility = numeric(0),
                 winddir = character(0),
                 windspeed = numeric(0),
                 gustspeed = numeric(0),
                 precip = numeric(0),
                 events = character(0),
                 conditions = character(0),
                 stringsAsFactors = FALSE)


for(monthindex in 7:12){


for(date in 1:filter(monthlookup, monthnum == monthindex)$numdays){

  weatherpage <- html(paste("http://www.wunderground.com/history/airport/KNYC/2013/",monthindex,"/",dat
                      "/DailyHistory.html", sep = ""))

  #####
  # The columns change. The third column could either be Windchill, Heat Index
  # Or Dew Point (which should be the fourth column)
  #####

  weathertest <- weatherpage %>%
    html_nodes("#obsTable > thead > tr > th:nth-child(3)") %>%
    html_text()

  time <- weatherpage %>%
    html_nodes("#obsTable > tbody > tr > td:nth-child(1)") %>%
    html_text()
  time <- paste(monthindex,"/",date,"/2013 ", time, sep="")
  time <- strptime(time, format="%m/%d/%Y %I:%M %p")

  year <- time$year+1900
  month <- time$mon+1
  day <- time$mday
  hour <- time$hour
  minute <- time$min

  temp <- weatherpage %>%
    html_nodes("#obsTable > tbody > tr > td:nth-child(2) > span > span.wx-value") %>%
```

```r
  html_text() %>%
  as.numeric()

if(weathertest == "Windchill"){

    windchill <- weatherpage %>%
      html_nodes("#obsTable > tbody > tr > td:nth-child(3)") %>%
      html_text()
    windchill[windchill == "\n  -\n"] <- NA
    windchill[!is.na(windchill)] <- substr(windchill[!is.na(windchill)], 4,
                                        nchar(windchill[!is.na(windchill)])-4)
    windchill <- as.numeric(windchill)

    heatindex <- rep(NA, each=length(temp))

    print("Windchill")

}

if(weathertest == "Heat Index"){

    windchill <- rep(NA, each=length(temp))

    heatindex <- weatherpage %>%
      html_nodes("#obsTable > tbody > tr > td:nth-child(3)") %>%
      html_text()
    heatindex[heatindex == "\n  -\n"] <- NA
    heatindex[!is.na(heatindex)] <- substr(heatindex[!is.na(heatindex)], 4,
                                        nchar(heatindex[!is.na(heatindex)])-4)
    heatindex <- as.numeric(heatindex)

    print("heatindex")
}

if(weathertest == "Dew Point"){

    windchill <- rep(NA, each=length(temp))

    heatindex <- rep(NA, each=length(temp))

    print("dew point")
}


if(weathertest == "Windchill" | weathertest == "Heat Index"){

    dewpoint <- weatherpage %>%
      html_nodes("#obsTable > tbody > tr > td:nth-child(4) > span > span.wx-value") %>%
      html_text() %>%
      as.numeric()

    humidity <- weatherpage %>%
      html_nodes("#obsTable > tbody > tr > td:nth-child(5)") %>%
```

```r
  html_text()
humidity <- as.numeric(substr(humidity, 1, 2))



#pressure <- weatherpage %>%
#  html_nodes("#obsTable > tbody > tr > td:nth-child(6) > span > span.wx-value") %>%
#  html_text() %>%
#  as.numeric()

pressure <- weatherpage %>%
  html_nodes("#obsTable > tbody > tr > td:nth-child(6)") %>%
  html_text()
pressure[pressure == "\n  -\n"] <- NA
pressure[!is.na(pressure)] <- substr(pressure[!is.na(pressure)],4,
                                     nchar(pressure[!is.na(pressure)])-4)
pressure <- as.numeric(pressure)

visibility <- weatherpage %>%
  html_nodes("#obsTable > tbody > tr > td:nth-child(7)") %>%
  html_text()
visibility[visibility == "\n  -\n"] <- NA
visibility[!is.na(visibility)] <- substr(visibility[!is.na(visibility)],4,
                                     nchar(visibility[!is.na(visibility)])-4)
visibility <- as.numeric(visibility)

winddir <- weatherpage %>%
  html_nodes("#obsTable > tbody > tr > td:nth-child(8)") %>%
  html_text()

windspeed <- weatherpage %>%
  html_nodes("#obsTable > tbody > tr > td:nth-child(9)") %>%
  html_text()
windspeed[windspeed == "\n  -\n"] <- NA
windspeed[windspeed == "Calm"] <- NA
windspeed[!is.na(windspeed)] <- substr(windspeed[!is.na(windspeed)], 4,
       nchar(windspeed[!is.na(windspeed)])-5)
windspeed <- as.numeric(windspeed)

#heatindex <- rep(NA, each=length(windspeed))

gustspeed <- weatherpage %>%
  html_nodes("#obsTable > tbody > tr > td:nth-child(10)") %>%
  html_text()
gustspeed[gustspeed == "\n  -\n"] <- NA
gustspeed[!is.na(gustspeed)] <- as.numeric(substr(gustspeed[!is.na(gustspeed)],4,7))

precip <- weatherpage %>%
  html_nodes("#obsTable > tbody > tr > td:nth-child(11)") %>%
  html_text()
precip[precip == "N/A"] <- NA
precip[!is.na(precip)] <- as.numeric(substr(precip[!is.na(precip)],4,7))
```

```
    eventTestPage <- html("http://www.wunderground.com/history/airport/KNYC/2014/12/3/DailyHistory.htm
    eventTest <- eventTestPage %>%
      html_nodes("#obsTable > tbody > tr:nth-child(1) > td:nth-child(12)") %>%
      html_text()

    events <- weatherpage %>%
      html_nodes("#obsTable > tbody > tr > td:nth-child(12)") %>%
      html_text()
    events[events == eventTest] <- NA
    events[!is.na(events)] <- substr(events[!is.na(events)], 2, nchar(events[!is.na(events)])-1)

    conditions <- weatherpage %>%
      html_nodes("#obsTable > tbody > tr > td:nth-child(13)") %>%
      html_text()

}

if(weathertest == "Dew Point"){

  dewpoint <- weatherpage %>%
    html_nodes("#obsTable > tbody > tr > td:nth-child(3) > span > span.wx-value") %>%
    html_text() %>%
    as.numeric()

  humidity <- weatherpage %>%
    html_nodes("#obsTable > tbody > tr > td:nth-child(4)") %>%
    html_text()
  humidity <- as.numeric(substr(humidity, 1, 2))

  pressure <- weatherpage %>%
    html_nodes("#obsTable > tbody > tr > td:nth-child(5)") %>%
    html_text()
  pressure[pressure == "\n  -\n"] <- NA
  pressure[!is.na(pressure)] <- substr(pressure[!is.na(pressure)],4,
                                       nchar(pressure[!is.na(pressure)])-4)
  pressure <- as.numeric(pressure)

  visibility <- weatherpage %>%
    html_nodes("#obsTable > tbody > tr > td:nth-child(6)") %>%
    html_text()
  visibility[visibility == "\n  -\n"] <- NA
  visibility[!is.na(visibility)] <- substr(visibility[!is.na(visibility)],4,
                                       nchar(visibility[!is.na(visibility)])-4)
  visibility <- as.numeric(visibility)

  winddir <- weatherpage %>%
    html_nodes("#obsTable > tbody > tr > td:nth-child(7)") %>%
    html_text()

  windspeed <- weatherpage %>%
    html_nodes("#obsTable > tbody > tr > td:nth-child(8)") %>%
    html_text()
  windspeed[windspeed == "\n  -\n"] <- NA
```

```r
    windspeed[windspeed == "Calm"] <- NA
    windspeed[!is.na(windspeed)] <- substr(windspeed[!is.na(windspeed)], 4,
                                            nchar(windspeed[!is.na(windspeed)])-5)
    windspeed <- as.numeric(windspeed)

    heatindex <- rep(NA, each=length(windspeed))

    gustspeed <- weatherpage %>%
      html_nodes("#obsTable > tbody > tr > td:nth-child(9)") %>%
      html_text()
    gustspeed[gustspeed == "\n  -\n"] <- NA
    gustspeed[!is.na(gustspeed)] <- as.numeric(substr(gustspeed[!is.na(gustspeed)],4,7))

    precip <- weatherpage %>%
      html_nodes("#obsTable > tbody > tr > td:nth-child(10)") %>%
      html_text()
    precip[precip == "N/A"] <- NA
    precip[!is.na(precip)] <- as.numeric(substr(precip[!is.na(precip)],4,7))

    eventTestPage <- html("http://www.wunderground.com/history/airport/KNYC/2014/12/3/DailyHistory.html
    eventTest <- eventTestPage %>%
      html_nodes("#obsTable > tbody > tr:nth-child(1) > td:nth-child(12)") %>%
      html_text()

    events <- weatherpage %>%
      html_nodes("#obsTable > tbody > tr > td:nth-child(11)") %>%
      html_text()
    events[events == eventTest] <- NA
    events[!is.na(events)] <- substr(events[!is.na(events)], 2, nchar(events[!is.na(events)])-1)

    conditions <- weatherpage %>%
      html_nodes("#obsTable > tbody > tr > td:nth-child(12)") %>%
      html_text()

  }

  dfadd <- data.frame(cbind(
    year, month, day, hour, minute, temp, windchill, heatindex, dewpoint,
    humidity,pressure, visibility, winddir, windspeed, gustspeed, precip,
    events, conditions), stringsAsFactors = FALSE)

  df <- rbind( df,dfadd)

  rm(dfadd)


}



}

for(monthindex in 1:8){
```

```
for(date in 1:filter(monthlookup, monthnum == monthindex)$numdays){

  weatherpage <- html(paste("http://www.wunderground.com/history/airport/KNYC/2014/",monthindex,"/",da
                            "/DailyHistory.html", sep = ""))

  #####
  # The columns change. The third column could either be Windchill, Heat Index
  # Or Dew Point (which should be the fourth column)
  #####

  weathertest <- weatherpage %>%
    html_nodes("#obsTable > thead > tr > th:nth-child(3)") %>%
    html_text()

  time <- weatherpage %>%
    html_nodes("#obsTable > tbody > tr > td:nth-child(1)") %>%
    html_text()
  time <- paste(monthindex,"/",date,"/2014 ", time, sep="")
  time <- strptime(time, format="%m/%d/%Y %I:%M %p")

  year <- time$year+1900
  month <- time$mon+1
  day <- time$mday
  hour <- time$hour
  minute <- time$min

  temp <- weatherpage %>%
    html_nodes("#obsTable > tbody > tr > td:nth-child(2) > span > span.wx-value") %>%
    html_text() %>%
    as.numeric()

  if(weathertest == "Windchill"){

    windchill <- weatherpage %>%
      html_nodes("#obsTable > tbody > tr > td:nth-child(3)") %>%
      html_text()
    windchill[windchill == "\n  -\n"] <- NA
    windchill[!is.na(windchill)] <- substr(windchill[!is.na(windchill)], 4,
                                           nchar(windchill[!is.na(windchill)])-4)
    windchill <- as.numeric(windchill)

    heatindex <- rep(NA, each=length(temp))

    print("Windchill")

  }

  if(weathertest == "Heat Index"){

    windchill <- rep(NA, each=length(temp))

    heatindex <- weatherpage %>%
```

```r
    html_nodes("#obsTable > tbody > tr > td:nth-child(3)") %>%
    html_text()
  heatindex[heatindex == "\n  -\n"] <- NA
  heatindex[!is.na(heatindex)] <- substr(heatindex[!is.na(heatindex)], 4,
                                         nchar(heatindex[!is.na(heatindex)])-4)
  heatindex <- as.numeric(heatindex)

  print("heatindex")
}

if(weathertest == "Dew Point"){

  windchill <- rep(NA, each=length(temp))

  heatindex <- rep(NA, each=length(temp))

  print("dew point")
}


if(weathertest == "Windchill" | weathertest == "Heat Index"){

  dewpoint <- weatherpage %>%
    html_nodes("#obsTable > tbody > tr > td:nth-child(4) > span > span.wx-value") %>%
    html_text() %>%
    as.numeric()

  humidity <- weatherpage %>%
    html_nodes("#obsTable > tbody > tr > td:nth-child(5)") %>%
    html_text()
  humidity <- as.numeric(substr(humidity, 1, 2))



  #pressure <- weatherpage %>%
  #  html_nodes("#obsTable > tbody > tr > td:nth-child(6) > span > span.wx-value") %>%
  #  html_text() %>%
  #  as.numeric()

  pressure <- weatherpage %>%
    html_nodes("#obsTable > tbody > tr > td:nth-child(6)") %>%
    html_text()
  pressure[pressure == "\n  -\n"] <- NA
  pressure[!is.na(pressure)] <- substr(pressure[!is.na(pressure)],4,
                                       nchar(pressure[!is.na(pressure)])-4)
  pressure <- as.numeric(pressure)

  visibility <- weatherpage %>%
    html_nodes("#obsTable > tbody > tr > td:nth-child(7)") %>%
    html_text()
  visibility[visibility == "\n  -\n"] <- NA
  visibility[!is.na(visibility)] <- substr(visibility[!is.na(visibility)],4,
                                           nchar(visibility[!is.na(visibility)])-4)
```

```
  visibility <- as.numeric(visibility)

  winddir <- weatherpage %>%
    html_nodes("#obsTable > tbody > tr > td:nth-child(8)") %>%
    html_text()

  windspeed <- weatherpage %>%
    html_nodes("#obsTable > tbody > tr > td:nth-child(9)") %>%
    html_text()
  windspeed[windspeed == "\n  -\n"] <- NA
  windspeed[windspeed == "Calm"] <- NA
  windspeed[!is.na(windspeed)] <- substr(windspeed[!is.na(windspeed)], 4,
                                        nchar(windspeed[!is.na(windspeed)])-5)
  windspeed <- as.numeric(windspeed)

  #heatindex <- rep(NA, each=length(windspeed))

  gustspeed <- weatherpage %>%
    html_nodes("#obsTable > tbody > tr > td:nth-child(10)") %>%
    html_text()
  gustspeed[gustspeed == "\n  -\n"] <- NA
  gustspeed[!is.na(gustspeed)] <- as.numeric(substr(gustspeed[!is.na(gustspeed)],4,7))

  precip <- weatherpage %>%
    html_nodes("#obsTable > tbody > tr > td:nth-child(11)") %>%
    html_text()
  precip[precip == "N/A"] <- NA
  precip[!is.na(precip)] <- as.numeric(substr(precip[!is.na(precip)],4,7))

  eventTestPage <- html("http://www.wunderground.com/history/airport/KNYC/2014/12/3/DailyHistory.htm
  eventTest <- eventTestPage %>%
    html_nodes("#obsTable > tbody > tr:nth-child(1) > td:nth-child(12)") %>%
    html_text()

  events <- weatherpage %>%
    html_nodes("#obsTable > tbody > tr > td:nth-child(12)") %>%
    html_text()
  events[events == eventTest] <- NA
  events[!is.na(events)] <- substr(events[!is.na(events)], 2, nchar(events[!is.na(events)])-1)

  conditions <- weatherpage %>%
    html_nodes("#obsTable > tbody > tr > td:nth-child(13)") %>%
    html_text()

}

if(weathertest == "Dew Point"){

  dewpoint <- weatherpage %>%
    html_nodes("#obsTable > tbody > tr > td:nth-child(3) > span > span.wx-value") %>%
    html_text() %>%
    as.numeric()
```

```
humidity <- weatherpage %>%
  html_nodes("#obsTable > tbody > tr > td:nth-child(4)") %>%
  html_text()
humidity <- as.numeric(substr(humidity, 1, 2))

pressure <- weatherpage %>%
  html_nodes("#obsTable > tbody > tr > td:nth-child(5)") %>%
  html_text()
pressure[pressure == "\n  -\n"] <- NA
pressure[!is.na(pressure)] <- substr(pressure[!is.na(pressure)],4,
                                     nchar(pressure[!is.na(pressure)])-4)
pressure <- as.numeric(pressure)

visibility <- weatherpage %>%
  html_nodes("#obsTable > tbody > tr > td:nth-child(6)") %>%
  html_text()
visibility[visibility == "\n  -\n"] <- NA
visibility[!is.na(visibility)] <- substr(visibility[!is.na(visibility)],4,
                                         nchar(visibility[!is.na(visibility)])-4)
visibility <- as.numeric(visibility)

winddir <- weatherpage %>%
  html_nodes("#obsTable > tbody > tr > td:nth-child(7)") %>%
  html_text()

windspeed <- weatherpage %>%
  html_nodes("#obsTable > tbody > tr > td:nth-child(8)") %>%
  html_text()
windspeed[windspeed == "\n  -\n"] <- NA
windspeed[windspeed == "Calm"] <- NA
windspeed[!is.na(windspeed)] <- substr(windspeed[!is.na(windspeed)], 4,
                                       nchar(windspeed[!is.na(windspeed)])-5)
windspeed <- as.numeric(windspeed)

heatindex <- rep(NA, each=length(windspeed))

gustspeed <- weatherpage %>%
  html_nodes("#obsTable > tbody > tr > td:nth-child(9)") %>%
  html_text()
gustspeed[gustspeed == "\n  -\n"] <- NA
gustspeed[!is.na(gustspeed)] <- as.numeric(substr(gustspeed[!is.na(gustspeed)],4,7))

precip <- weatherpage %>%
  html_nodes("#obsTable > tbody > tr > td:nth-child(10)") %>%
  html_text()
precip[precip == "N/A"] <- NA
precip[!is.na(precip)] <- as.numeric(substr(precip[!is.na(precip)],4,7))

eventTestPage <- html("http://www.wunderground.com/history/airport/KNYC/2014/12/3/DailyHistory.htm
eventTest <- eventTestPage %>%
  html_nodes("#obsTable > tbody > tr:nth-child(1) > td:nth-child(12)") %>%
  html_text()
```

```
      events <- weatherpage %>%
        html_nodes("#obsTable > tbody > tr > td:nth-child(11)") %>%
        html_text()
      events[events == eventTest] <- NA
      events[!is.na(events)] <- substr(events[!is.na(events)], 2, nchar(events[!is.na(events)])-1)

      conditions <- weatherpage %>%
        html_nodes("#obsTable > tbody > tr > td:nth-child(12)") %>%
        html_text()

    }

    dfadd <- data.frame(cbind(
      year, month, day, hour, minute, temp, windchill, heatindex, dewpoint,
      humidity,pressure, visibility, winddir, windspeed, gustspeed, precip,
      events, conditions), stringsAsFactors = FALSE)

    df <- rbind( df,dfadd)

    rm(dfadd)


  }



}

df$year <- as.numeric(df$year)
df$month <- as.numeric(df$month)
df$day <- as.numeric(df$day)
df$hour <- as.numeric(df$hour)
df$minute <- as.numeric(df$minute)
df$temp <- as.numeric(df$temp)
df$windchill <- as.numeric(df$windchill)
df$heatindex <- as.numeric(df$heatindex)
df$dewpoint <- as.numeric(df$dewpoint)
df$humidity <- as.numeric(df$humidity)
df$pressure <- as.numeric(df$pressure)
df$visibility <- as.numeric(df$visibility)
df$windspeed <- as.numeric(df$windspeed)
df$gustspeed <- as.numeric(df$gustspeed)
df$precip <- as.numeric(df$precip)
```

Trip Data:

The trip data from the source CSV included the following columns:

- Trip Duration (seconds): num
- Start Time and Date: character, converted to POSIXlt(varchar in PostgreSQL though)
- stop Time and Date: same as above
- Start Station Name: character

- End Station Name: character
- Station ID: num
- Station Latitude: num
- Station Longitude: num
- Bike ID: num
- User Type: character
- Gender: num
- Year of Birth: num

In addition to this, I parsed out the date to get separate columns for years, months, days, hours, minutes, and seconds. Except for minutes and seconds, I'll be using these columns for aggregation purposes.

```
workingdirectory <- "C:/Users/Charley/Downloads/CUNY/IS 607 Data Acquisition and Management/Semester Pr
setwd(workingdirectory)

years2013 <- c("08","09","10","11","12")
years2014 <- c("01","02","03","04","05","06","07","08")
csvs <- paste("2013-",years2013," - Citi Bike trip data.csv",sep="")
csvs <- c(csvs,paste("2014-",years2014," - Citi Bike trip data.csv",sep=""))

bikesharedf <- read.csv("2013-07 - Citi Bike trip data.csv", stringsAsFactors=FALSE)

for(filename in csvs){
  bikesharedf <- rbind(bikesharedf, read.csv(filename))
}

bikesharedf$birth.year[bikesharedf$birth.year == "\\N"] <- NA
bikesharedf$birth.year <- as.numeric(bikesharedf$birth.year)
```

Using this data, I took out the stations and made a separate stations table for my PostgreSQL database. I was then able to remove the station information columns from the trips table. I noticed a few duplicate stations, which seemed to be the result of switching the locations slightly. For these, I just erased the duplicates, as a spot check revealed they were trivially close to their original locations.

```
stationlist <- unique(select(bikesharedf, start.station.id, start.station.name,
                             start.station.latitude, start.station.longitude))
row.names(stationlist) <- NULL
colnames(stationlist) <- c("station.id", "station.name", "station.latitude",
                           "station.longitude")

## Checking the duplicates to make sure they're not too far apart

stationlist %>%
  filter(station.id %in% stationlist$station.id[duplicated(stationlist$station.id)]) %>%
  arrange(station.id)

## Close enough to remove the duplicates

stationlist <- stationlist[!duplicated(stationlist$station.id),]

bikesharedf <- bikesharedf %>%
  select(-(start.station.name:start.station.longitude),
         -(end.station.name:end.station.longitude))
```

```r
bikesharedf$starttime <- strptime(bikesharedf$starttime,
                                  format = "%Y-%m-%d %H:%M:%S")

bikesharedf$stoptime <- strptime(bikesharedf$stoptime,
                                 format = "%Y-%m-%d %H:%M:%S")



bikesharedf$starthour <- bikesharedf$starttime$hour
bikesharedf$startday <- bikesharedf$starttime$mday
bikesharedf$startmonth <- bikesharedf$starttime$mon+1
bikesharedf$startyear <- bikesharedf$starttime$year+1900
bikesharedf$startminute <- bikesharedf$starttime$min
bikesharedf$startsecond <- bikesharedf$starttime$sec

bikesharedf$endhour <- bikesharedf$stoptime$hour
bikesharedf$endday <- bikesharedf$stoptime$mday
bikesharedf$endmonth <- bikesharedf$stoptime$mon+1
bikesharedf$endyear <- bikesharedf$stoptime$year+1900
bikesharedf$endminute <- bikesharedf$stoptime$min
bikesharedf$endsecond <- bikesharedf$stoptime$sec
```

I also created a quick daily weekend file, and made the dataframe hourly for use in the hourly comparisons:

```r
weekends <- read.csv("weekends.csv")

hours <- 0:23

addhours <- function(vec)
{
  weekends$hours <- vec
  return(weekends)
}

weekends <- adply(hours, .margins = 1, .fun=addhours)

weekends <- weekends %>%
  select(-X1)

colnames(weekends) <- c("date", "day", "month", "year", "weekend", "hour")
```

Regression Methodology

Once the preliminary data was brought into PostgreSQL, tables were created there for further analysis. Below is the R code that brought this data into PostgreSQL:

```r
library(RPostgreSQL)

con <- dbConnect(RPostgreSQL::PostgreSQL(), user="postgres", password="insertepasswordhere",
                 dbname="bikeshare")

```

```
con

dbWriteTable(con, "hourlyweather", df, row.names=FALSE)

dbWriteTable(con, "trips", bikesharedf, row.names=TRUE)

dbWriteTable(con, "stations", stationlist, row.names=FALSE)

dbWriteTable(con, "weekends", weekends, row.names=FALSE)
```

I kept the row.names TRUE for trips to use as my primary key.

Once in PostgreSQL I created a few primary and foreign keys:

```
# -- Constraint: stations_pkey
#
# -- ALTER TABLE stations DROP CONSTRAINT stations_pkey;
#
# ALTER TABLE stations
# ADD CONSTRAINT stations_pkey PRIMARY KEY("station.id");
#
# -- Constraint: hourlyweather_pkey
#
# -- ALTER TABLE hourlyweather DROP CONSTRAINT hourlyweather_pkey;
#
# ALTER TABLE hourlyweather
#   ADD CONSTRAINT hourlyweather_pkey PRIMARY KEY(year, month, day, hour);
#
# -- Constraint: trips_pkey
#
# -- ALTER TABLE trips DROP CONSTRAINT trips_pkey;
#
# ALTER TABLE trips
#   ADD CONSTRAINT trips_pkey PRIMARY KEY("row.names");
#
# -- Foreign Key: endstation_fkey
#
# -- ALTER TABLE trips DROP CONSTRAINT endstation_fkey;
#
# ALTER TABLE trips
#   ADD CONSTRAINT endstation_fkey FOREIGN KEY ("end.station.id")
#       REFERENCES stations ("station.id") MATCH SIMPLE
#       ON UPDATE NO ACTION ON DELETE NO ACTION;
#
# -- Foreign Key: endweather_fkey
#
# -- ALTER TABLE trips DROP CONSTRAINT endweather_fkey;
#
# ALTER TABLE trips
#   ADD CONSTRAINT endweather_fkey FOREIGN KEY (endyear, endmonth, endday, endhour)
#       REFERENCES hourlyweather (year, month, day, hour) MATCH SIMPLE
#       ON UPDATE NO ACTION ON DELETE NO ACTION;
#
# -- Foreign Key: endweekend_fkey
```

```
#
# -- ALTER TABLE trips DROP CONSTRAINT endweekend_fkey;
#
# ALTER TABLE trips
#   ADD CONSTRAINT endweekend_fkey FOREIGN KEY (endyear, endmonth, endday, endhour)
#       REFERENCES weekends (year, month, day, hour) MATCH SIMPLE
#       ON UPDATE NO ACTION ON DELETE NO ACTION;
#
# -- Foreign Key: startstation_fkey
#
# -- ALTER TABLE trips DROP CONSTRAINT startstation_fkey;
#
# ALTER TABLE trips
#   ADD CONSTRAINT startstation_fkey FOREIGN KEY ("start.station.id")
#       REFERENCES stations ("station.id") MATCH SIMPLE
#       ON UPDATE NO ACTION ON DELETE NO ACTION;
#
#
# -- Foreign Key: startweather_fkey
#
# -- ALTER TABLE trips DROP CONSTRAINT startweather_fkey;
#
# ALTER TABLE trips
#   ADD CONSTRAINT startweather_fkey FOREIGN KEY (startyear, startmonth, startday, starthour)
#       REFERENCES hourlyweather (year, month, day, hour) MATCH SIMPLE
#       ON UPDATE NO ACTION ON DELETE NO ACTION;
#
# -- Foreign Key: startweekend_fkey
#
# -- ALTER TABLE trips DROP CONSTRAINT startweekend_fkey;
#
# ALTER TABLE trips
#   ADD CONSTRAINT startweekend_fkey FOREIGN KEY (startyear, startmonth, startday, starthour)
#       REFERENCES weekends (year, month, day, hour) MATCH SIMPLE
#       ON UPDATE NO ACTION ON DELETE NO ACTION;
```

To perform my hourly analysis, I aggregated the trips table by hour, and then joined this with the weather and weekend data to create a summary table:

```
# CREATE TABLE hourlyrides AS
#
# SELECT
#   startyear AS year,
#   startmonth AS month,
#   startday AS day,
#   starthour AS hour,
#   count(*)
# FROM
#   trips
# GROUP BY
#   startyear,
#   startmonth,
#   startday,
```

```
#   starthour;
#
# CREATE TABLE hourlyrideswithweather AS
#
# SELECT
#   hourlyrides.year,
#   hourlyrides.month,
#   hourlyrides.day,
#   hourlyrides.hour,
#   count AS ridescount
#   hourlyweather.temp,
#   hourlyweather.dewpoint,
#   hourlyweather.humidity,
#   hourlyweather.pressure,
#   hourlyweather.visibility,
#   hourlyweather.windspeed,
#   hourlyweather.conditions,
#   weekends.weekend
#
# FROM
#   hourlyrides
# INNER JOIN
#   hourlyweather on hourlyweather.year = hourlyrides.year
#   AND hourlyweather.month = hourlyrides.month
#   AND hourlyweather.day = hourlyrides.day
#   AND hourlyweather.hour = hourlyrides.hour
# INNER JOIN
#   weekends ON weekends.year = hourlyrides.year
#   AND weekends.month = hourlyrides.month
#   AND weekends.day = hourlyrides.day
#   AND weekends.hour = hourlyrides.hour;
```

Now the data is ready for a regression model.

```
con <- dbConnect(RPostgreSQL::PostgreSQL(), user="postgres", password="is607",
                 dbname="bikeshare")

con
```

```
## <PostgreSQLConnection:(43684,0)>
```

```
fulldata <- dbReadTable(con, "hourlyrideswithweather")
```

First lets take a look at some of the long term trends. I'll create a monthly aggregate to look at how CitiBike average usage evolved over the available time period:

```
# monthlyagg <- fulldata %>%
#   group_by(year, month) %>%
#   summarise(ridescount = sum(ridescount))
#
# monthlyagg$date <- ISOdatetime(monthlyagg$year, monthlyagg$month, 1, 0, 0, 0)
#
# ggplot(monthlyagg, aes(x=date, y=ridescount)) + geom_line()
```

There's an expected dip during the winter (with a polar vortex) but usage appears to be flat comparing summer 2013 to 2014. So it looks like month (and season) is a factor, but there isn't a general growth trend to account for.

For explanatory purposes, I'll look at the scatter plots of the numeric variables, and use colour and facet to display the categorical variables (focusing on hour and weather conditions.).

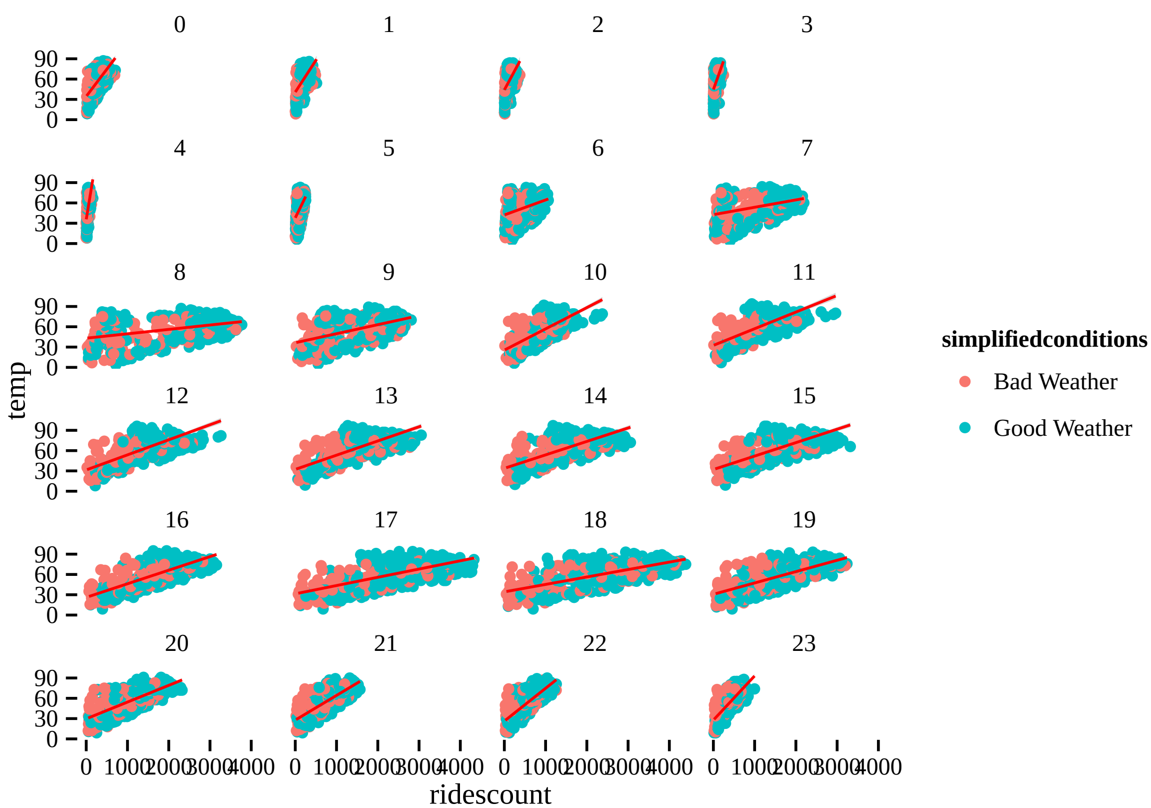
There were too many weather conditions to show on a graph, so I created a "simplified weather conditions" variable, divided into "Good Weather" and "Bad Weather", the definitions of which are below in the code:

```
goodweather <- c("Clear", "Fog", "Haze", "Mostly Cloudy", "Overcase", "Partly Cloudy", "Scattered Cloud
badweather <- c("Heavy Rain", "Heavy Snow", "Light Freezing Rain", "Light Rain", "Light Snow",
                "Rain", "Snow", "Unknown")

fulldata$simplifiedconditions <- ifelse(fulldata$conditions %in% goodweather, "Good Weather", "Bad Weat
```

The graphic results are shown below.

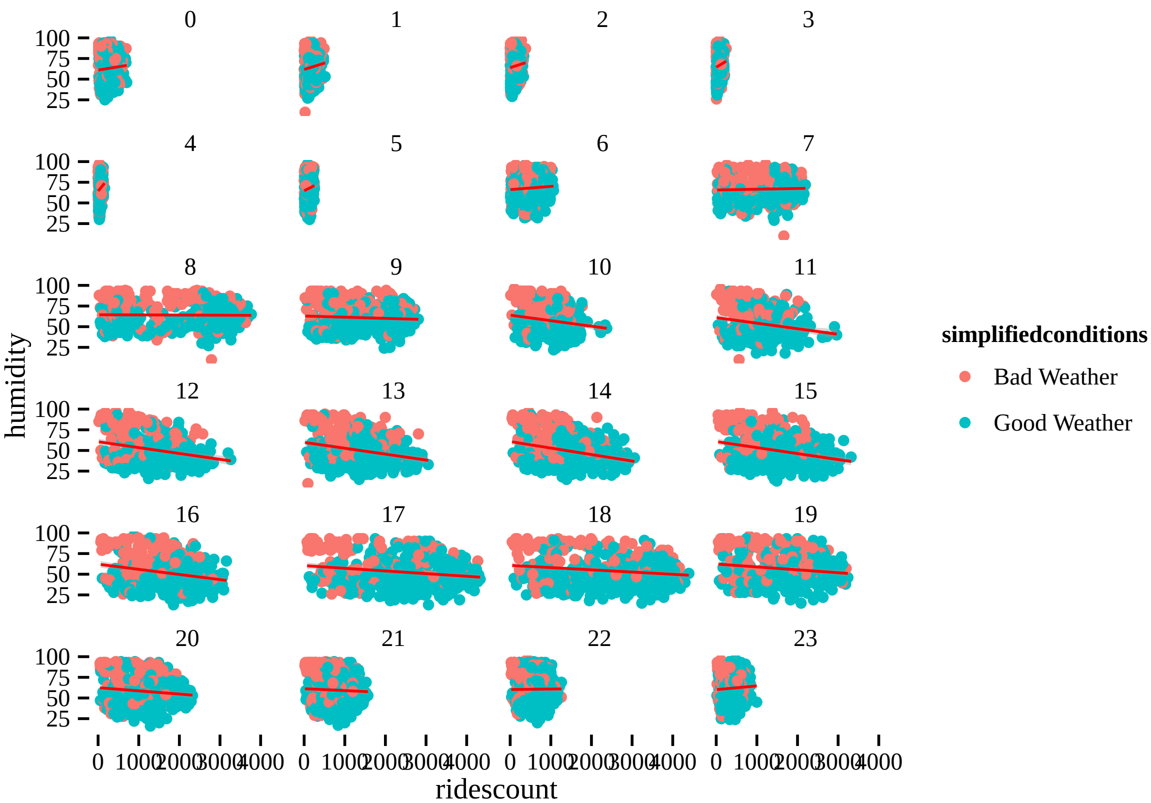
Temperature:

```
ggplot(fulldata, aes(x=ridescount, y=temp, colour = simplifiedconditions)) +
  geom_point() +
  geom_smooth(method="lm", color="red") +
  facet_wrap( ~ hour, nrow=6, ncol=4) +
  theme_tufte()
```



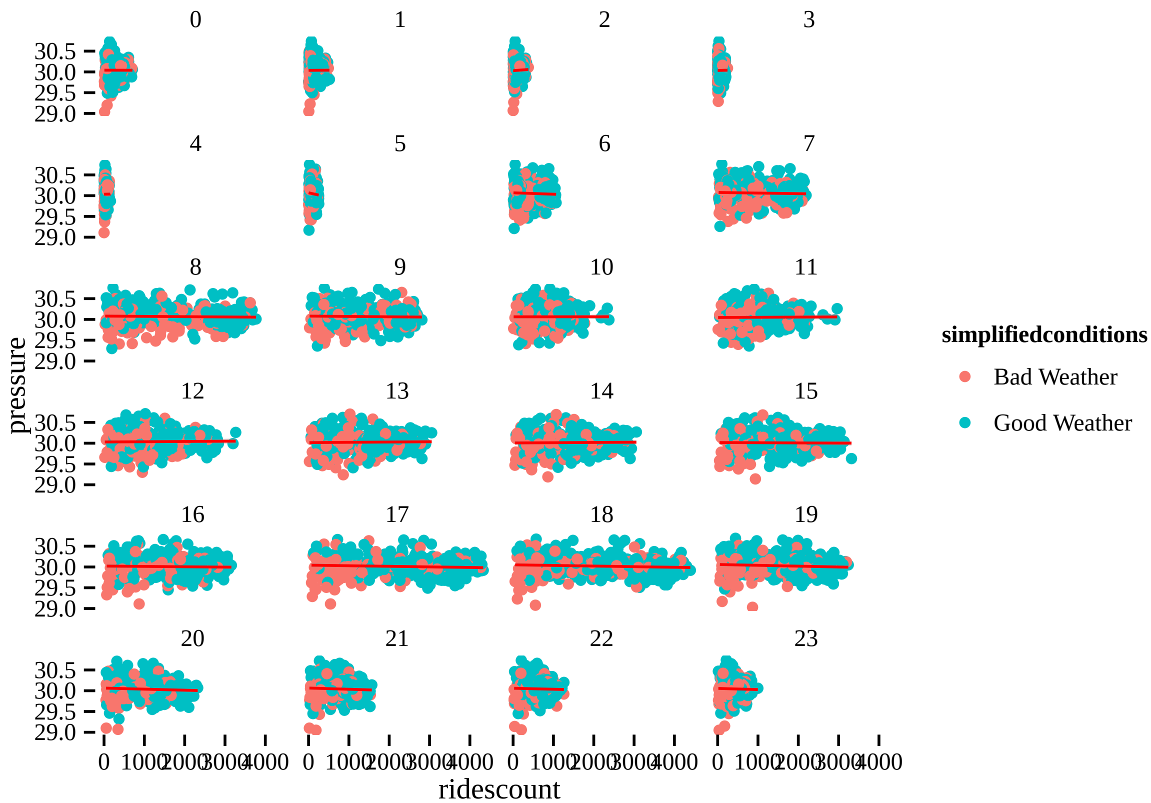
###Humidity:###

```
ggplot(fulldata, aes(x=ridescount, y=humidity, colour = simplifiedconditions)) +
  geom_point() +
  geom_smooth(method="lm", color="red") +
  facet_wrap( ~ hour, nrow=6, ncol=4) +
  theme_tufte()
```



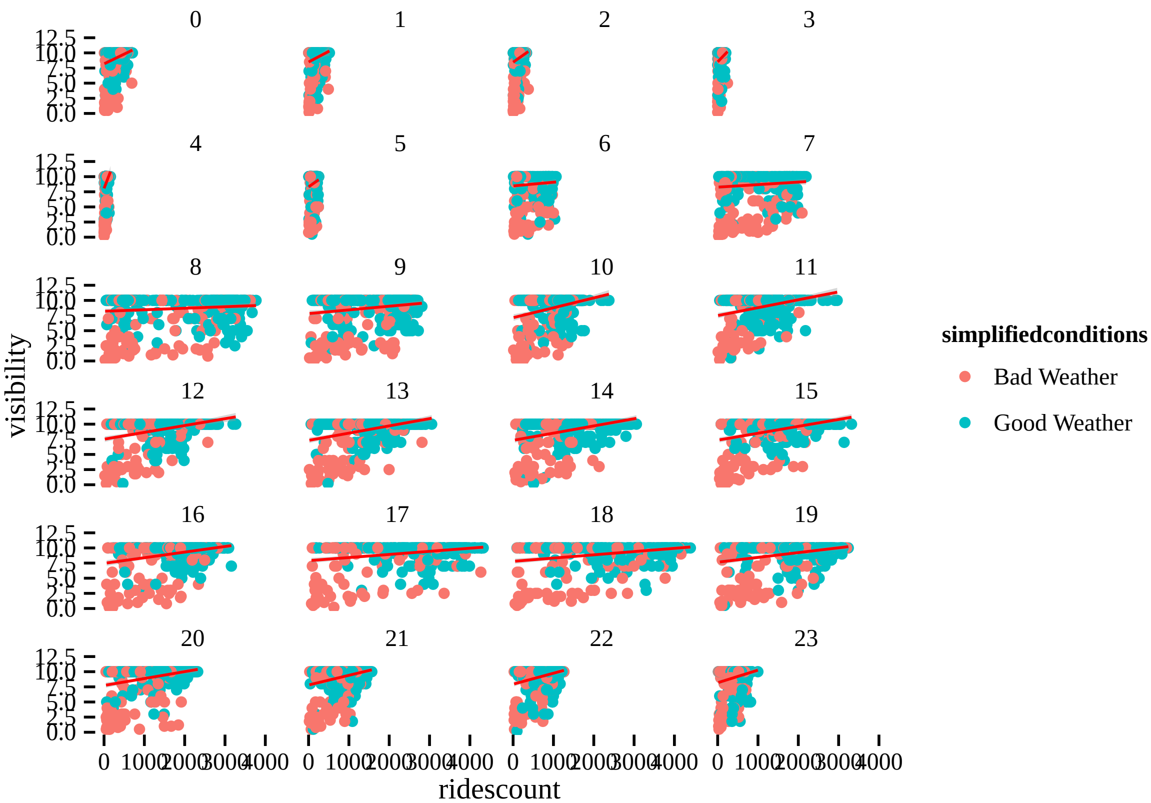
###Pressure:###

```r
ggplot(fulldata, aes(x=ridescount, y=pressure, colour = simplifiedconditions)) +
  geom_point() +
  geom_smooth(method="lm", color="red") +
  facet_wrap( ~ hour, nrow=6, ncol=4) +
  theme_tufte()
```



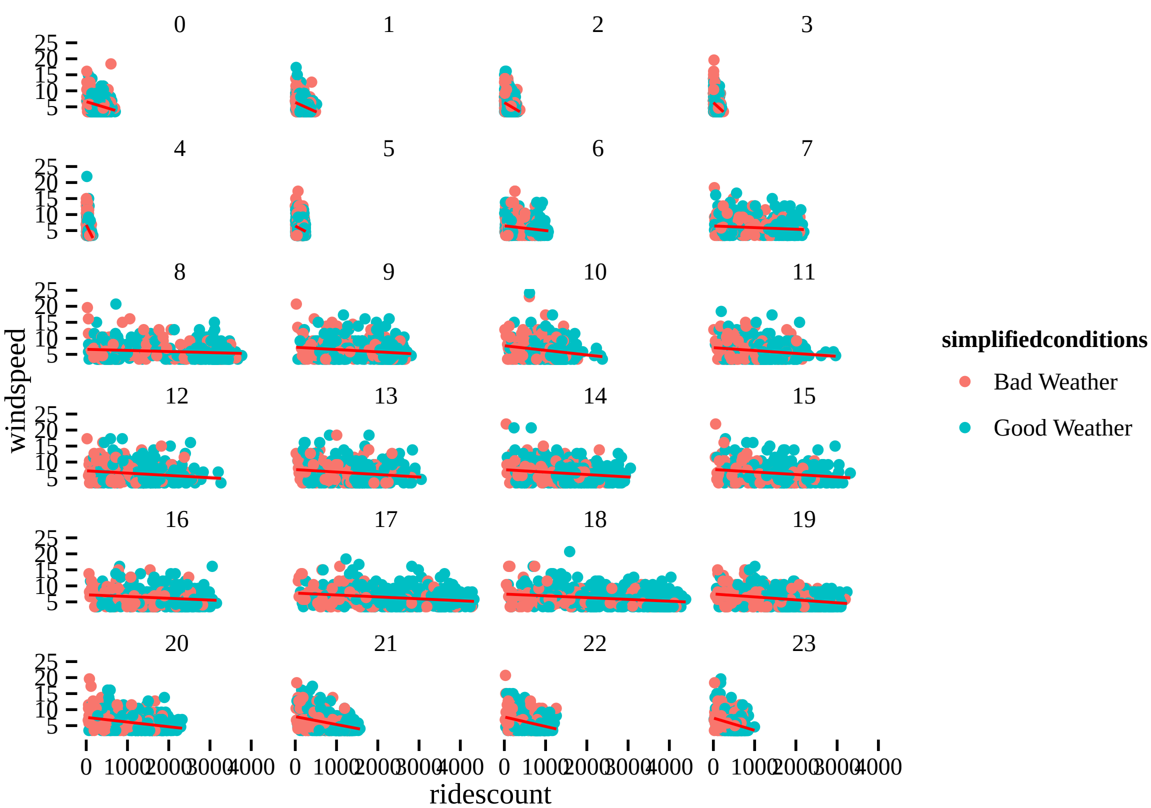
###Visibility:###

```r
ggplot(fulldata, aes(x=ridescount, y=visibility, colour = simplifiedconditions)) +
  geom_point() +
  geom_smooth(method="lm", color="red") +
  facet_wrap( ~ hour, nrow=6, ncol=4) +
  theme_tufte()
```



###Windspeed:###

```
ggplot(fulldata, aes(x=ridescount, y=windspeed, colour = simplifiedconditions)) +
  geom_point() +
  geom_smooth(method="lm", color="red") +
  facet_wrap( ~ hour, nrow=6, ncol=4) +
  theme_tufte()
```




```
# Along with some code to print out a series of pictures for an animated gif:

# for(var in c("temp", "humidity", "pressure", "visibility", "windspeed")){
#
#   png(filename=paste("analysis",var,".png",sep=""), width=600)
#
#   print(ggplot(fulldata, aes(x=ridescount, y=get(var), colour = simplifiedconditions)) +
#     geom_point() +
#     geom_smooth(method="lm", color="red") +
#     facet_wrap( ~ hour, nrow=6, ncol=4) +
#     theme_tufte() +
#     ggtitle(paste(toupper(substr(var,1,1)),substr(var,2,nchar(var)),
#                  " Colored by Weather Condition, Facet by Hour", sep="")))
#
#   dev.off()
#
# }
```

It looks like a few variables don't have an effect. Pressure, for example, seems pretty flat (which is surprising, because pressure should be correlated with bad weather…) Hour seems to have a clear effect on ridership, as does month from the monthly aggregate shown earlier.

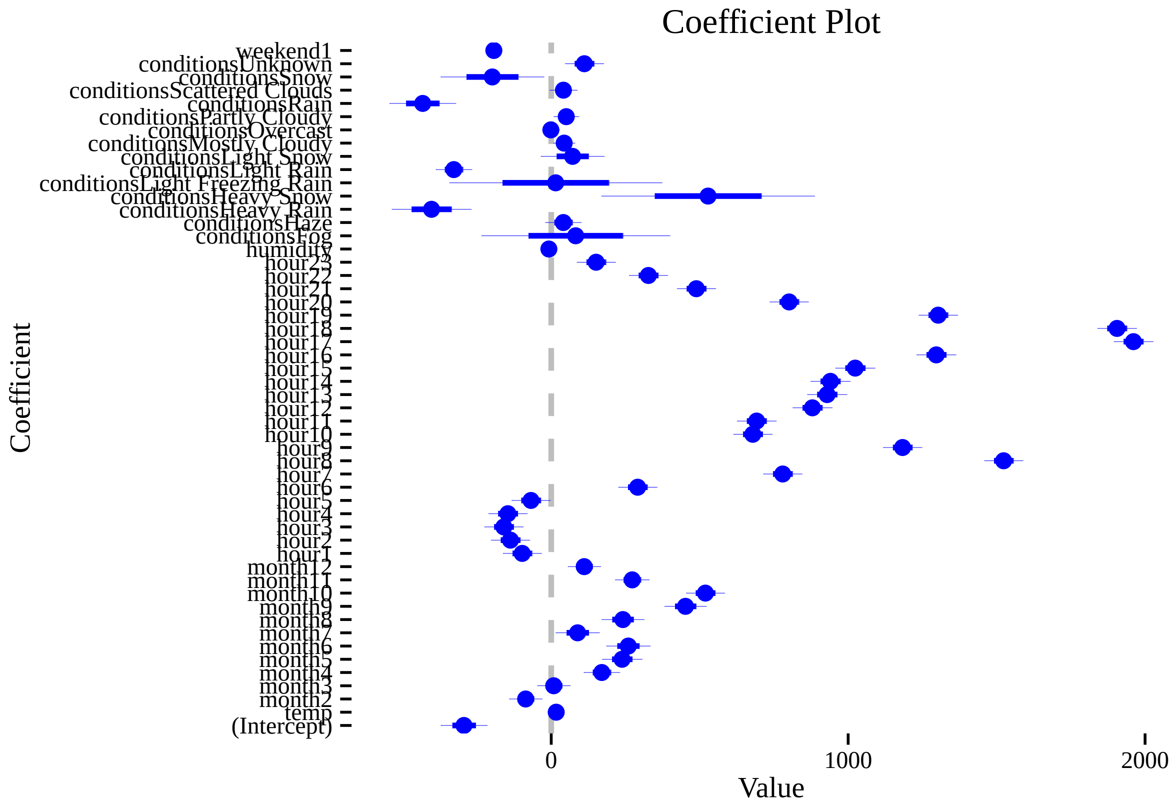
For my model, I'll choose the numeric variables temp and humidity, the categorical variables month, hour, conditions, and weekend.

```
fulldata$month <- as.factor(fulldata$month)
fulldata$hour <- as.factor(fulldata$hour)
fulldata$weekend <- as.factor(fulldata$weekend)
fulldata$conditions <- as.factor(fulldata$conditions)

model <- lm(ridescount ~ temp+month+hour+humidity+conditions+weekend, data=fulldata)

summary(model)
```

```
##
## Call:
## lm(formula = ridescount ~ temp + month + hour + humidity + conditions +
##     weekend, data = fulldata)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -2028.9  -284.9   -33.1   272.4  1695.5
##
## Coefficients:
##                              Estimate Std. Error t value Pr(>|t|)
## (Intercept)                  -293.938     38.807   -7.57  3.9e-14 ***
## temp                           16.756      0.595   28.16  < 2e-16 ***
## month2                        -86.124     27.443   -3.14  0.00170 **
## month3                          8.335     27.606    0.30  0.76270
## month4                        170.335     30.171    5.65  1.7e-08 ***
## month5                        238.604     33.532    7.12  1.2e-12 ***
## month6                        259.539     36.830    7.05  1.9e-12 ***
## month7                         88.804     36.697    2.42  0.01554 *
## month8                        241.294     35.448    6.81  1.1e-11 ***
## month9                        452.132     35.064   12.89  < 2e-16 ***
## month10                       519.200     32.257   16.10  < 2e-16 ***
## month11                       272.757     28.611    9.53  < 2e-16 ***
## month12                       111.086     27.410    4.05  5.1e-05 ***
## hour1                         -97.666     32.322   -3.02  0.00252 **
## hour2                        -137.325     32.399   -4.24  2.3e-05 ***
## hour3                        -159.654     32.365   -4.93  8.2e-07 ***
## hour4                        -145.969     32.449   -4.50  6.9e-06 ***
## hour5                         -68.153     32.433   -2.10  0.03563 *
## hour6                         291.104     32.421    8.98  < 2e-16 ***
## hour7                         779.612     32.407   24.06  < 2e-16 ***
## hour8                        1523.643     32.380   47.05  < 2e-16 ***
## hour9                        1183.266     32.438   36.48  < 2e-16 ***
## hour10                        678.863     32.521   20.87  < 2e-16 ***
## hour11                        691.851     32.779   21.11  < 2e-16 ***
## hour12                        879.611     33.043   26.62  < 2e-16 ***
## hour13                        929.111     33.181   28.00  < 2e-16 ***
## hour14                        940.292     33.167   28.35  < 2e-16 ***
```

```
## hour15                         1023.706     33.077   30.95  < 2e-16 ***
## hour16                         1296.813     32.967   39.34  < 2e-16 ***
## hour17                         1960.811     32.818   59.75  < 2e-16 ***
## hour18                         1905.491     32.667   58.33  < 2e-16 ***
## hour19                         1303.124     32.539   40.05  < 2e-16 ***
## hour20                          801.179     32.450   24.69  < 2e-16 ***
## hour21                          488.769     32.374   15.10  < 2e-16 ***
## hour22                          327.204     32.361   10.11  < 2e-16 ***
## hour23                          151.226     32.425    4.66  3.1e-06 ***
## humidity                         -7.921      0.362  -21.86  < 2e-16 ***
## conditionsFog                    82.296    158.604    0.52  0.60386
## conditionsHaze                   40.828     30.202    1.35  0.17646
## conditionsHeavy Rain           -403.115     66.721   -6.04  1.6e-09 ***
## conditionsHeavy Snow            528.168    179.194    2.95  0.00321 **
## conditionsLight Freezing Rain    14.979    178.850    0.08  0.93325
## conditionsLight Rain           -328.055     30.126  -10.89  < 2e-16 ***
## conditionsLight Snow             71.907     53.290    1.35  0.17725
## conditionsMostly Cloudy          43.541     17.865    2.44  0.01482 *
## conditionsOvercast               -1.237     14.242   -0.09  0.93077
## conditionsPartly Cloudy          50.599     21.045    2.40  0.01622 *
## conditionsRain                 -432.856     55.563   -7.79  7.3e-15 ***
## conditionsScattered Clouds       41.025     22.930    1.79  0.07362 .
## conditionsSnow                 -198.439     86.740   -2.29  0.02217 *
## conditionsUnknown               111.521     32.206    3.46  0.00054 ***
## weekend1                       -193.247     10.414  -18.56  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 470 on 10128 degrees of freedom
##   (66 observations deleted due to missingness)
## Multiple R-squared:  0.753,  Adjusted R-squared:  0.752
## F-statistic:  607 on 51 and 10128 DF,  p-value: <2e-16
```

```r
coefplot(model) + theme_tufte()
```



Mapping Methodology

With the trip level data, I'm able to do more than recreate aggregates of rides over time. Since the trip level data included station information, I wanted to display a geographic analysis of these trips.

I decided to take the trips data and aggregate it by only hour. This way, you can get an average "day in the life" of the CitiBike system.

I also wanted to break apart weekends and weekdays. The first few steps took place in PostgreSQL, joining the weekends data with the trips data, and aggregating based off of that. This had to be done twice, once for the start stations and once fore the end stations. The aggregations took place in two steps. First, I would count the trips on a daily basis, to get the number of trips per hour per day. Then, I averaged this hourly count across all days, to get a representative weekday and weekend:

```
# Start trip tables ##################################

# CREATE TABLE weekendstarttrips AS
#
# SELECT
#   "row.names", tripduration, starttime,
#   stoptime, "start.station.id", "end.station.id",
#   bikeid, usertype, "birth.year", gender, starthour,
#   startday, startmonth, startyear, startminute,
#   startsecond,
#
#   weekends.weekend
```

```
# FROM
#   trips
# INNER JOIN
#   weekends ON
#   trips.startyear = weekends.year
#   AND trips.startmonth = weekends.month
#   AND trips.startday = weekends.day
#   AND trips.starthour = weekends.hour;
#
# CREATE TABLE firststartstationweekenddailyhourly AS
#
# SELECT
#   starthour AS hour,
#   startday AS day,
#   startmonth AS month,
#   startyear AS year,
#   "start.station.id",
#   weekend,
#   count(*)
# FROM
#   weekendstarttrips
# GROUP BY
#   starthour,
#   startday,
#   startmonth,
#   startyear,
#   "start.station.id",
#   weekend;
#
# CREATE TABLE firststartstationweekendhourly AS
#
# SELECT
#   hour,
#   "start.station.id",
#   weekend,
#   avg(count) AS count
# FROM
#   firststartstationweekenddailyhourly
# GROUP BY
#   hour,
#   "start.station.id",
#   weekend;
#
#
# CREATE TABLE startstationweekendhourly AS
#
# SELECT
#   hour,
#   "start.station.id",
#   stations."station.name",
#   stations."station.latitude",
#   stations."station.longitude",
#   weekend,
```

```
#   count
# FROM
#   firststartstationweekendhourly
# INNER JOIN
#   stations ON firststartstationweekendhourly."start.station.id" = stations."station.id";

######################################################

# End trip tables   ##################################

# CREATE TABLE weekendendtrips AS
#
# SELECT
#   "row.names", tripduration, starttime,
#   stoptime, "start.station.id", "end.station.id",
#   bikeid, usertype, "birth.year", gender, endhour,
#   endday, endmonth, endyear, endminute,
#   endsecond,
#
#   weekends.weekend
# FROM
#   trips
# INNER JOIN
#   weekends ON
#   trips.endyear = weekends.year
#   AND trips.endmonth = weekends.month
#   AND trips.endday = weekends.day
#   AND trips.endhour = weekends.hour;
#
# CREATE TABLE firstendstationweekenddailyhourly AS
#
# SELECT
#   endhour AS hour,
#   endday AS day,
#   endmonth AS month,
#   endyear AS year,
#   "end.station.id",
#   weekend,
#   count(*)
# FROM
#   weekendendtrips
# GROUP BY
#   endhour,
#   endday,
#   endmonth,
#   endyear,
#   "end.station.id",
#   weekend
#
# CREATE TABLE firstendstationweekendhourly AS
#
# SELECT
#   hour,
```

```
#   "end.station.id",
#   weekend,
#   avg(count) as count
# FROM
#   firstendstationweekenddailyhourly
# GROUP BY
#   hour,
#   "end.station.id",
#   weekend
#
#
#
# CREATE TABLE endstationweekendhourly AS
#
# SELECT
#   hour,
#   "end.station.id",
#   stations."station.name",
#   stations."station.latitude",
#   stations."station.longitude",
#   weekend,
#   count
# FROM
#   firstendstationweekendhourly
# INNER JOIN
#   stations ON firstendstationweekendhourly."end.station.id" = stations."station.id";

#####################################################
```

After creating these tables, I was able to bring them into R. There were a few hours where stations recorded zero usage, which I had to account for.

```
con <- dbConnect(RPostgreSQL::PostgreSQL(), user="postgres", password="is607",
                 dbname="bikeshare")

con
```

```
## <PostgreSQLConnection:(43684,1)>
```

```
start.station.pop <- dbReadTable(con, "startstationweekendhourly")

end.station.pop <- dbReadTable(con, "endstationweekendhourly")

colnames(start.station.pop)[2] <- "station.id"
colnames(end.station.pop)[2] <- "station.id"


# There are a few hours when no one borrowed from a station,
# lets find those and add 0's
```

```
addzerosweekend <- function(data)
{
  comboaz <- filter(combo, weekend == 1)
  vec <- unique(comboaz$station.id[!(comboaz$station.id %in% data$station.id)])
  return(vec)
}

addzerosweekday <- function(data)
{
  comboaz <- filter(combo, weekend == 0)
  vec <- unique(comboaz$station.id[!(comboaz$station.id %in% data$station.id)])
  return(vec)
}


makedf <- function(ec)
{
  df <- data.frame(station.id = ec)
}

combo <- expand.grid(unique(start.station.pop$station.id),0:23,0:1)
colnames(combo) <- c("station.id", "hour", "weekend")

stationagg <- start.station.pop %>%
  select(-count, -hour, -weekend)

stationagg <- unique(stationagg)

start.station.weekend <- filter(start.station.pop, weekend==1)
start.station.weekend <- select(start.station.weekend, -weekend)
emptycheckss <- dlply(start.station.weekend, .variables="hour", .fun=addzerosweekend)
missingss <- ldply(emptycheckss, .fun = makedf)
missingss <- merge(missingss, stationagg, by="station.id")
missingss$count <- 0
start.station.weekend <- rbind(start.station.weekend, missingss)

start.station.weekday <- filter(start.station.pop, weekend==0)
start.station.weekday <- select(start.station.weekday, -weekend)
emptycheckss <- dlply(start.station.weekday, .variables="hour", .fun=addzerosweekday)
missingss <- ldply(emptycheckss, .fun = makedf)
missingss <- merge(missingss, stationagg, by="station.id")
missingss$count <- 0
start.station.weekday <- rbind(start.station.weekday, missingss)

end.station.weekend <- filter(end.station.pop, weekend==1)
end.station.weekend <- select(end.station.weekend, -weekend)
emptycheckes <- dlply(end.station.weekend, .variables="hour", .fun=addzerosweekend)
missinges <- ldply(emptycheckes, .fun=makedf)
missinges <- merge(missinges, stationagg, by="station.id")
missinges$count <- 0
end.station.weekend <- rbind(end.station.weekend, missinges)

end.station.weekday <- filter(end.station.pop, weekend==0)
```

```r
end.station.weekday <- select(end.station.weekday, -weekend)
emptycheckes <- dlply(end.station.weekday, .variables="hour", .fun=addzerosweekday)
missinges <- ldply(emptycheckes, .fun=makedf)
missinges <- merge(missinges, stationagg, by="station.id")
missinges$count <- 0
end.station.weekday <- rbind(end.station.weekday, missinges)


colnames(start.station.weekend)[6] <- "startcount"
colnames(end.station.weekend)[6] <- "endcount"
colnames(start.station.weekday)[6] <- "startcount"
colnames(end.station.weekday)[6] <- "endcount"
```

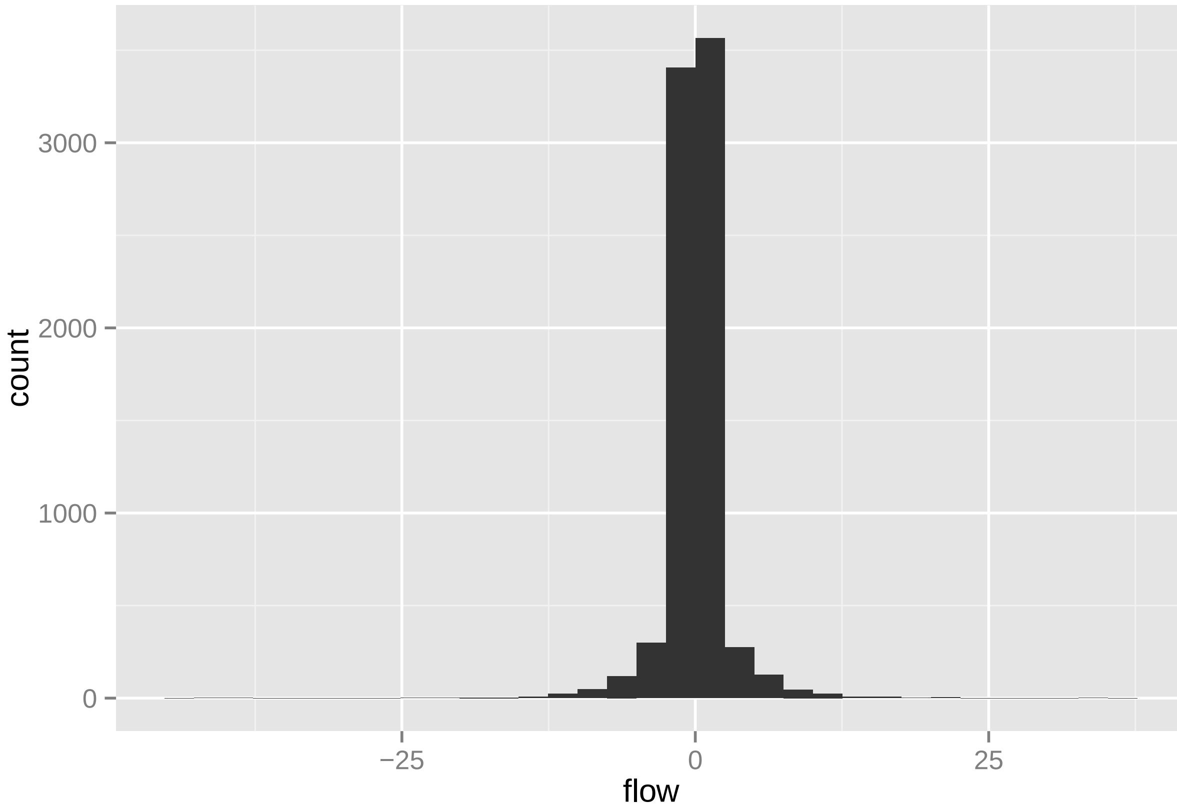
I subtracted the start counts and end counts at each station to arrive at a "flow" of bikes to and from each station. Using the ggmap package, I mapped out the stations by latitude and longitude. First, I'll merge the stations to create a flow.

```r
mergedweekend <- merge(start.station.weekend, end.station.weekend,
                       by=c("hour", "station.id", "station.name",
                            "station.latitude", "station.longitude"))
mergedweekend <- mergedweekend %>%
  mutate(flow = endcount - startcount)

mergedweekday <- merge(start.station.weekday, end.station.weekday,
                       by=c("hour", "station.id", "station.name",
                            "station.latitude", "station.longitude"))
mergedweekday <- mergedweekday %>%
  mutate(flow = endcount - startcount)
```

Checking a histogram of the flow by station lead to some skewed distributions:

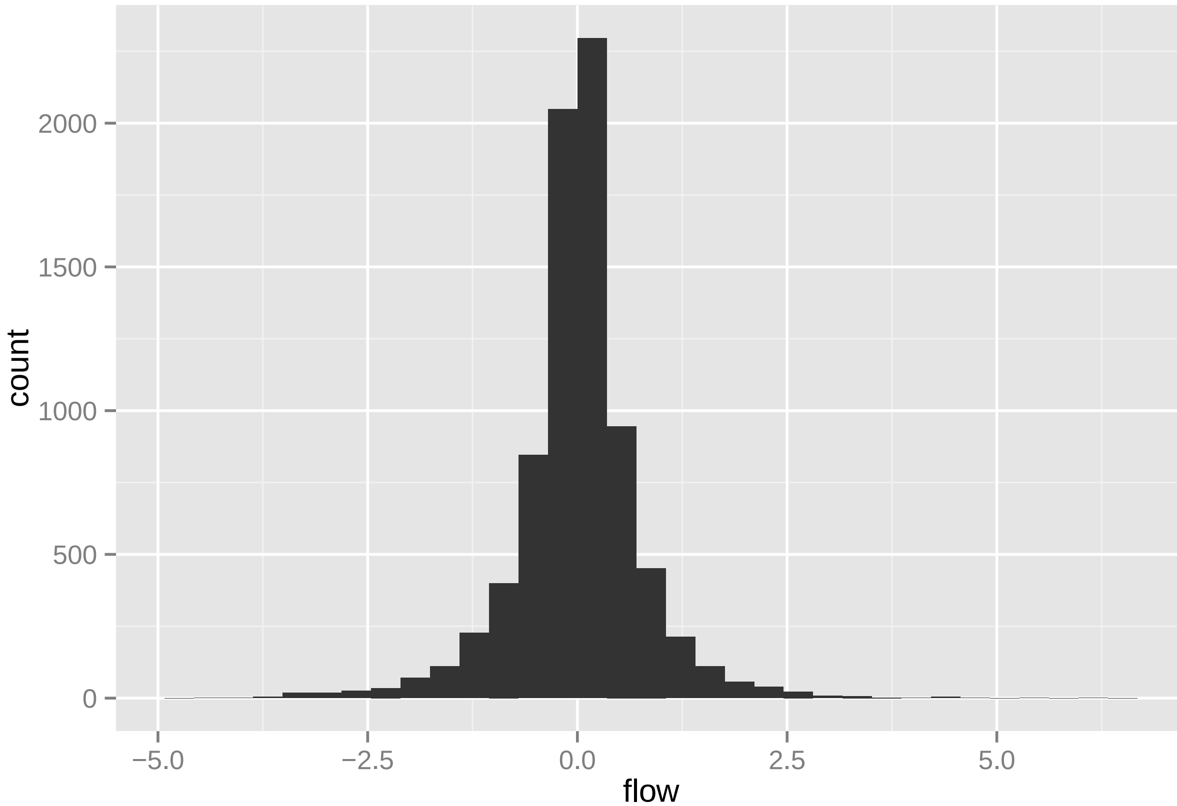
```r
ggplot(mergedweekday, aes(x=flow)) + geom_histogram()
```

```
## stat_bin: binwidth defaulted to range/30. Use 'binwidth = x' to adjust this.
```




```r
ggplot(mergedweekend, aes(x=flow)) + geom_histogram()
```

```
## stat_bin: binwidth defaulted to range/30. Use 'binwidth = x' to adjust this.
```



For the purposes of this map, in order to see how the detail in the vast majority of stations, I'm going to take care of outliers. Instead of simply deleting them, I'm going to cap all counts at 3 standard deviations out from the mean. I'll lose a bit of detail with the actual numbers of those larger stations, but the tradeoff is worth seeing the detail in the majority of stations where it matters (the alternative, adding another color to the color scale, might be a bit misleading visually.)

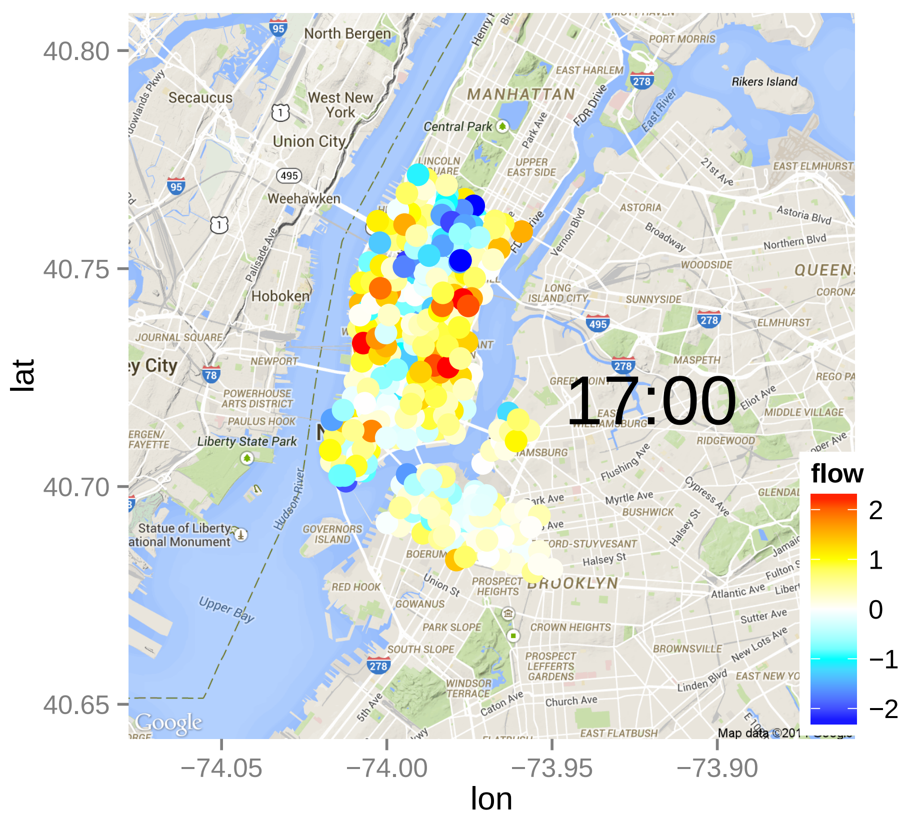
```
upperweekdaycap <- mean(mergedweekday$flow) + 3*sd(mergedweekday$flow)
lowerweekdaycap <- mean(mergedweekday$flow) - 3*sd(mergedweekday$flow)
upperweekendcap <- mean(mergedweekend$flow) + 3*sd(mergedweekend$flow)
lowerweekendcap <- mean(mergedweekend$flow) - 3*sd(mergedweekend$flow)

mergedweekday$flow[mergedweekday$flow > upperweekdaycap] <- upperweekdaycap
mergedweekday$flow[mergedweekday$flow < lowerweekdaycap] <- lowerweekdaycap
mergedweekend$flow[mergedweekend$flow > upperweekendcap] <- upperweekendcap
mergedweekend$flow[mergedweekend$flow < lowerweekendcap] <- lowerweekendcap
```

With this done, I can map out the flow of weekday and weekend hours. As an example, below is a graph of the flow of bikes on a weekday at 5pm.

```
nyc <- get_map(location = c(lon=-73.968410, lat=40.725496), zoom = 12)
```

```
## Map from URL : http://maps.googleapis.com/maps/api/staticmap?center=40.725496,-73.96841&zoom=12&size=
## Google Maps API Terms of Service : http://developers.google.com/maps/terms
```

```
upperweekendlimit <- max(mergedweekend$flow)
lowerweekendlimit <- min(mergedweekend$flow)
upperweekdaylimit <- max(mergedweekday$flow)
lowerweekdaylimit <- min(mergedweekday$flow)

nyc <- ggmap(nyc)

i = 17

mergedstationsfilter <- mergedweekend %>%
  filter(hour == i)

nyc <- nyc +
  geom_point(data = mergedstationsfilter,
             aes(x = station.longitude, y = station.latitude, colour=flow), size = 4) +
  scale_colour_gradientn(colours=c("blue","cyan","white", "yellow","red"),
                         values=rescale(c(lowerweekendlimit,-1,0,1,upperweekendlimit)),
                         limits=c(lowerweekendlimit,upperweekendlimit)) +
  theme(legend.position = c(1,0.2)) +
  annotate("text", x=-73.92, y=40.72, label=paste(i,":00",sep=""), size=9)

print(nyc)
```



I ran this code in a for loop for both weekends and weekdays:

```r
nyc2 <- ggmap(nyc)

for(i in 0:23){
  mergedstationsfilter <- mergedweekend %>%
    filter(hour == i)

  png(filename=paste("weekendhour",i,".png",sep=""), width = 600)

  nyc <- nyc2

  nyc <- nyc +
    geom_point(data = mergedstationsfilter,
               aes(x = station.longitude, y = station.latitude, colour=flow), size = 4) +
    scale_colour_gradientn(colours=c("blue","cyan","white", "yellow","red"),
                           values=rescale(c(lowerweekendlimit,-1,0,1,upperweekendlimit)),
                           limits=c(lowerweekendlimit,upperweekendlimit)) +
    theme(legend.position = c(1,0.2)) +
    annotate("text", x=-73.92, y=40.72, label=paste(i,":00",sep=""), size=9)

  print(nyc)

  dev.off()

}

for(i in 0:23){
  mergedstationsfilter <- mergedweekday %>%
    filter(hour == i)

  png(filename=paste("weekdayhour",i,".png",sep=""), width = 600)

  nyc <- nyc2

  nyc <- nyc +
    geom_point(data = mergedstationsfilter,
               aes(x = station.longitude, y = station.latitude, colour=flow), size = 4) +
    scale_colour_gradientn(colours=c("blue","cyan","white", "yellow","red"),
                           values=rescale(c(lowerweekdaylimit,-1,0,1,upperweekdaylimit)),
                           limits=c(lowerweekdaylimit,upperweekdaylimit)) +
    theme(legend.position = c(1,0.2)) +
    annotate("text", x=-73.92, y=40.72, label=paste(i,":00",sep=""), size=9)

  print(nyc)

  dev.off()

}
```

These pictures then had to be uploaded to an external service to create an animated gif. Originally, I wanted to use the animate package to do this, but unfortunately it seems the dependent packages that animate uses are not able to work with current versions of R. The weekday animated patterns can be found here and the weekend animated patterns can be found here

Conclusion and Discussion

The correlation analysis and regression served to highlight what sorts of factors can be used to predict ridership on the CitiBike system. Weather definitely had a marked effect, but not as much as hour itself. Rush hour seemed to have the highest predictive effect on CitiBike usage. For planning purposes, one could imagine using more sophisticated statistical techniques to separate the model by hour, and use weather to predict what happens at each hour knowing the underlying temporal pattern.

The geographic analysis led to some very interesting patterns. In general, there was a clear rush hour pattern coming in from the far east and west sides of Manhattan, and ending in the center of the island. I was expecting a more marked influx into Midtown and the Financial District, but the distribution was actually more uniform across the center of Manhattan, including a swath from Soho, up through eastern Chelsea and Union Square. This area has been a new commercial center. While more traditional businesses stay in Midtown and the Financial district, this new emerging business district has been a center for start ups, tech, and fashion.

I'd argue the workers in these companies trend younger, and are more likely to use a service like CitiBike, which is why this business district becomes pronounced when looking at the data. Another whimsical conclusion comes out when looking at the morning rush hour. Young employees at trendy companies show up to work later than those in Midtown or in the Financial District.

There was also a confirmation of my personal CitiBike strategy. I work in Midtown East, and instead of using the station closer to my office, I use one closer to Grand Central. Further away from this station, rush hour trends dominate. By Grand Central, however, this is moderated by commuters coming in from the Northern suburbs and biking from Grand Central to their final destination. You'll notice this anomaly looking at the stations around Grand Central and Penn Station.

There were a few important lessons working on this project taught me. This was larger than the other datasets we worked with in this class. My programming flow tends to involve a lot of trial and error, and I was forced to divide my work into chunks to get large jobs out of the way. For example, while originally when I was scraping the weather data, I was debugging the entire code that pulled the data from each individual website, a long process. When it became clear I was wasting time (and I stopped being stubborn thinking "this time there's no way it can't work!") I changed my code to pull in the raw data once, and then separated out the code that massages that raw data into the dataframe I needed.