R-Bootcamp Analysis - Rental Bike Behaviour Seoul

Konstantinos Lessis, Pascal Himmelberger

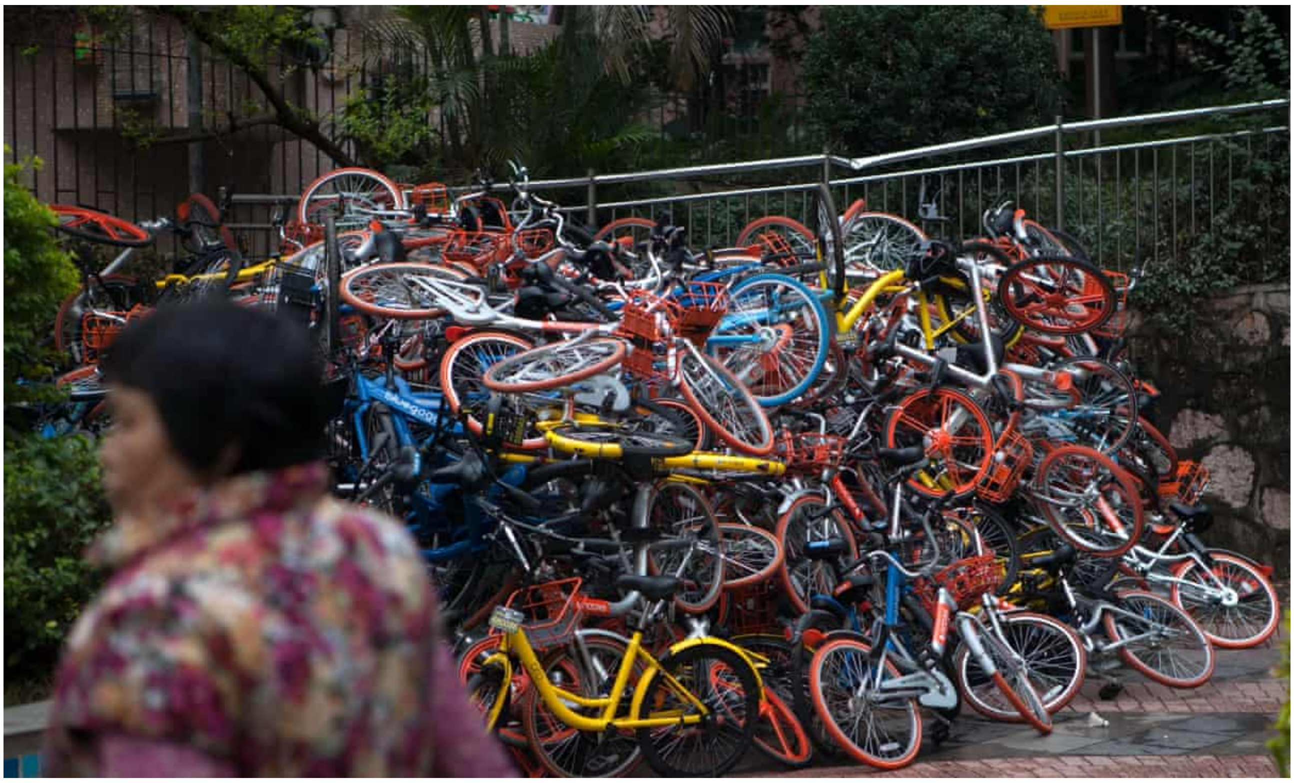
September 11, 2020

Background

The bike rental services business has exploded over the last 5 years and is scheduled to continue its significant growth over the next 5 years. *Hexa Research*, an independent research organisation estimates the global rental bike market to be valued at 1.37 billion USD as of 2017 and expects the growth until 2025 to continue at a pace of **14.3% compound annual growth rate (CAGR)** (Hexa Research, 2019). Other market intelligence and research firms present more conservative estimates. Mordor Intelligence for instance estimates a CAGR of the global bike sharing market of 6.5% for the 5 year period between 2020 and 2025 (Mordor Intelligence, 2019). Another market research firm puts the absolute global value of the rental bike (and scooter) market at 7.06 billion USD by 2026 in a more recent publication (Data Bridge Market Research, 2020).

No matter what the absolut numbers may be, it is clear that there is a huge and growing demand for more ecological transport modes and that the classic model of owning such transportation modes is dying. Rental models and on-demand availability of transport modes is clearly a promising avenue for the future of the personal transportation market.

While the initial promise of bike rental services was to offer an ecological alternative to cars and therefore cut emissions, experience in various cities have shown that these services lead to new problems. In the city of Shenzen (China) for instance, authorities and citizens struggle with 'pollution' of the city landscape with huge piles of discarded rental bikes (The Guardian, 2017). The below picture shows an example of that problem in Shenzen.


A gigantic pile of discarded rental bikes in Shenzen, China. Source: (source: Imaginechina/Rex/Shutterstock)

Predicting demand is essential not just to maximize profits but also to prevent waste and ensure bikes are available where ever and when ever needed. With this analysis, we aim to contribute to that by analysing a dataset of bike rentals in the South Korean capitol of Seoul. By considering a variety of predictors including weather conditions, time and seasonal components, holidays or airport arrivals we aim to create a simple, linear predictive model to predict the number of bike rentals across the city in a given hour on a given day.

In order to achieve the goal of predictive modelling the number of rentals, we sourced two main datasets:

- **Seoul Bike Rental Data 2017 - 2018** (source: https://archive.ics.uci.edu/ml/datasets/Seoul+Bike+Sharing+Demand (https://archive.ics.uci.edu/ml/datasets/Seoul+Bike+Sharing+Demand))
  - This dataset contains hourly rental counts for each day between December 2017 and November 2018 (so about 1 year)
  - Besides the rental counts, this dataset also includes information regarding weather, holidays and if the rental service was operational on that day/hour
- Arrivals and Departures at **Incheon Airport Seoul 2017 - 2018** (source: https://www.airport.kr/co/en/cpr/statisticCategoryOfTime.do (https://www.airport.kr/co/en/cpr/statisticCategoryOfTime.do))
  - This data was sourced from Incheon Airport Seoul directly and contains the aggregated arrivals / departures grouped by hour of the day
  - The data was chosen due to the suspected correlation between tourists/locals coming in at the airport and the rental count

Setup & Data Load

```
# Load packages
library(readxl)
library(ggplot2)
library(lubridate)
library(here)
library(scales)
library(dplyr)
library(plotly)
library(boot)
library(webshot)
library(DT)
#library(IRdisplay)
library(gifski) # used to create gif of animated graph
require(knitr)


# Setting locale
Sys.setlocale("LC_ALL", 'German_Switzerland')
```

```
## [1] "LC_COLLATE=German_Switzerland.1252;LC_CTYPE=German_Switzerland.1252;LC_MONETARY=German_Switzerland.1252;LC_NUMERIC=
C;LC_TIME=German_Switzerland.1252"
```

```
# Set working directory for R and knitr
#setwd(here('Group_Work'))
opts_knit$set(root.dir = getwd())

# Get rental data
bikedata.source <- read.csv2('../source/bikesharing/SeoulBikeData.csv', sep=',')
str(bikedata.source)
```

```
## 'data.frame':    8760 obs. of  14 variables:
##  $ Date                     : chr  "01/12/2017" "01/12/2017" "01/12/2017" "01/12/2017" ...
##  $ Rented.Bike.Count        : int  254 204 173 107 78 100 181 460 930 490 ...
##  $ Hour                     : int  0 1 2 3 4 5 6 7 8 9 ...
##  $ Temperature..C.          : chr  "-5.2" "-5.5" "-6" "-6.2" ...
##  $ Humidity...              : int  37 38 39 40 36 37 35 38 37 27 ...
##  $ Wind.speed..m.s.         : chr  "2.2" "0.8" "1" "0.9" ...
##  $ Visibility..10m.         : int  2000 2000 2000 2000 2000 2000 2000 2000 2000 1928 ...
##  $ Dew.point.temperature..C.: chr  "-17.6" "-17.6" "-17.7" "-17.6" ...
##  $ Solar.Radiation..MJ.m2.  : chr  "0" "0" "0" "0" ...
##  $ Rainfall.mm.             : chr  "0" "0" "0" "0" ...
##  $ Snowfall..cm.            : chr  "0" "0" "0" "0" ...
##  $ Seasons                  : chr  "Winter" "Winter" "Winter" "Winter" ...
##  $ Holiday                  : chr  "No Holiday" "No Holiday" "No Holiday" "No Holiday" ...
##  $ Functioning.Day          : chr  "Yes" "Yes" "Yes" "Yes" ...
```

As the output above shows, the **bike rental data from Seoul** (source: https://archive.ics.uci.edu/ml/datasets/Seoul+Bike+Sharing+Demand (https://archive.ics.uci.edu/ml/datasets/Seoul+Bike+Sharing+Demand)) contains over 8700 observations and 14 variables or attributes. This includes hourly number of bike rentals including meteorological information - such as humidity, temperature, rainfall etc. - as well as 'holiday' indicators. R tried to automatically determine the data types of our attributes but mis-categorized some. All attributes will be cast to their required data type in the next step (**Data Transformation**).

```
# Get airport arrival data 2017 & 2018
air17.source <- read_excel('../source/bikesharing/Airport_Statistics_Stat_by_Time_of_Day_201701_201712_edited.xls', sheet =
1)
air18.source <- read_excel('../source/bikesharing/Airport_Statistics_Stat_by_Time_of_Day_201801_201812_edited.xls', sheet =
1)
str(air17.source) #str(air18.source) # same as air17 but for year 2018
```

```
## tibble [25 x 4] (S3: tbl_df/tbl/data.frame)
##  $ time     : chr [1:25] "00:00 ~ 00:59" "01:00 ~ 01:59" "02:00 ~ 02:59" "03:00 ~ 03:59" ...
##  $ Arrival  : chr [1:25] "109,770" "28,682" "46,792" "179,620" ...
##  $ Departure: chr [1:25] "498,707" "254,771" "64,177" "18,400" ...
##  $ Total    : chr [1:25] "608,477" "283,453" "110,969" "198,020" ...
```

In addition to the above bike rental data set, we will employ data from the **Incheon Airport Seoul** (source: https://www.airport.kr/co/en/cpr/statisticCategoryOfTime.do (https://www.airport.kr/co/en/cpr/statisticCategoryOfTime.do)) which lists the total number of passengers (arrivals) for 2017 and 2018, grouped by hour of the day. This grouping was chosen due to the fact that the available bike data is also grouped on an hourly basis.

Data Transformation

In order to start analysing and building/validating models on this data, the available files and attributes need to be transformed into a suitable, standardise and combined form. Data transformation includes the following activities, depending on the specific dataset(s):

- casting data to correct types especially
  - factors
  - date/time
  - numeric values (int, double)
- remove invalid values / corrupted lines
- standardise missing values / NAs
- add computed / derived columns such as
  - hour extracted from field 'time' in air dataset to link to bike data
  - relative number of passengers per hour in air dataset
- combine both datasets

Let us start with the rental bike dataset. Below we will cast the attributes to their most appropriate type. Potential issues with missing / invalid data should be revealed during that process.

Missing Values

Before we continue to the further preparation steps, now would be a good time to check for missing values and handle them appropriately (before merging all datasets into one final dataframe). We use *is.na()* and a search for empty strings to identify potentially missing values. If feasible and necessary, further imputations will be made to replace those missing values. The output of the various function calls below is suppressed but did **not show any missing values or empty strings**.

```
# bike data set
bikedata.source[is.na(bikedata.source),]
# ==> no NA values!
bikedata.source[bikedata.source == '',]
# ==> no empty strings!

# air 2017 dataset
air17.source[is.na(air17.source)]
# ==> no NA values!
air17.source == ''
# ==> no empty strings!

# air 2018 dataset
is.na(air18.source)
# ==> no NA values!
air18.source == ''
# ==> no empty strings!
```

Casting of Data Types

First, let's cast the attributes of *bikedata* into their correct data types:

```
# creating cleaned version of source data with correctly cast data types for each attribute
bikedata.c <- data.frame(
    Date =          as.Date(bikedata.source$Date, format='%d/%m/%Y'),
    Year =          year(as.Date(bikedata.source$Date, format='%d/%m/%Y')),
    RentCount =     bikedata.source$Rented.Bike.Count,
    Hour =          bikedata.source$Hour,
    Temperature =   as.double(bikedata.source$Temperature..C., length = 1),
    Humidity =      bikedata.source$Humidity...,
    Windspeed =     as.double(bikedata.source$Wind.speed..m.s., length = 1),
    Visibility =    bikedata.source$Visibility..10m.,
    DewPointTemp =  as.double(bikedata.source$Dew.point.temperature..C., length = 2),
    SolarRadiation= as.double(bikedata.source$Solar.Radiation..MJ.m2., length = 1),
    Rainfall =      as.double(bikedata.source$Rainfall.mm., length = 1),
    Snowfall =      as.double(bikedata.source$Snowfall..cm., length = 1),
    Season =        factor(bikedata.source$Seasons),
    Holiday =       factor(bikedata.source$Holiday),
    Functional =    factor(bikedata.source$Functioning.Day)
)
# check
str(bikedata.c)
```

```
## 'data.frame':    8760 obs. of  15 variables:
##  $ Date          : Date, format: "2017-12-01" "2017-12-01" ...
##  $ Year          : num  2017 2017 2017 2017 2017 ...
##  $ RentCount     : int  254 204 173 107 78 100 181 460 930 490 ...
##  $ Hour          : int  0 1 2 3 4 5 6 7 8 9 ...
##  $ Temperature   : num  -5.2 -5.5 -6 -6.2 -6 -6.4 -6.6 -7.4 -7.6 -6.5 ...
##  $ Humidity      : int  37 38 39 40 36 37 35 38 37 27 ...
##  $ Windspeed     : num  2.2 0.8 1 0.9 2.3 1.5 1.3 0.9 1.1 0.5 ...
##  $ Visibility    : int  2000 2000 2000 2000 2000 2000 2000 2000 2000 1928 ...
##  $ DewPointTemp  : num  -17.6 -17.6 -17.7 -17.6 -18.6 -18.7 -19.5 -19.3 -19.8 -22.4 ...
##  $ SolarRadiation: num  0 0 0 0 0 0 0 0 0.01 0.23 ...
##  $ Rainfall      : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ Snowfall      : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ Season        : Factor w/ 4 levels "Autumn","Spring",..: 4 4 4 4 4 4 4 4 4 4 ...
##  $ Holiday       : Factor w/ 2 levels "Holiday","No Holiday": 2 2 2 2 2 2 2 2 2 2 ...
##  $ Functional    : Factor w/ 2 levels "No","Yes": 2 2 2 2 2 2 2 2 2 2 ...
```

The casting of data types did not show any problems with invalid data / missing values so no removal / cleanup of that dataset seems to be necessary at this point. In the next step we will continue with the two air passenger datasets for **Incheon Airport**. This dataset is split by year. After casting all attributes to appropriate data types, the two data sets (2017 and 2018) will be combined into one dataframe with the addition of a *year* attribute.

```
# before going further, we need to remove the 'TOTAL' row from the data set
air17.edit <- air17.source[air17.source$time != 'Total',]
air18.edit <- air18.source[air18.source$time != 'Total',]

# 2017 air passenger data Seoul
air17.c <- data.frame(
    Year =          2017,
    Hour =          as.integer(substring(air17.edit$time, 1, 2)),
    Arrival =       as.integer(gsub(',', '', air17.edit$Arrival)),
    Departure =     as.integer(gsub(',', '', air17.edit$Departure)),
    TotalArrival =  as.integer(gsub(',', '', air17.source$Arrival[length(air17.source$Arrival)])), # get total arrivals for
 2017
    TotalDeparture= as.integer(gsub(',', '', air17.source$Departure[length(air17.source$Departure)])) # get total departures
for 2017
)
# 2018 air passenger data Seoul
air18.c <- data.frame(
    Year =          2018,
    Hour =          as.integer(substring(air18.edit$time, 1, 2)),
    Arrival =       as.integer(gsub(',', '', air18.edit$Arrival)),
    Departure =     as.integer(gsub(',', '', air18.edit$Departure)),
    TotalArrival =  as.integer(gsub(',', '', air18.source$Arrival[length(air18.source$Arrival)])), # get total arrivals for
 2018
    TotalDeparture= as.integer(gsub(',', '', air18.source$Departure[length(air18.source$Departure)])) # get total departures
for 2018
)
```

Data Merge

As a last step in the transformation phase, the currently separate datasets *air.17*, *air.18* and *bikedata* will be merged together to create one single data frame to be used for the further analysis. In a first step, the two *air** datasets will be merged, before then joining them to the *bikedata* dataset.

Below we then use the *datatable()* function from the *library(DT)* to show an interactive, filterable and searchable version of the datatable in the HTML knitr output document. This allows for quick preliminary exploratory analysis of the source data, directly in the output document.

```r
# combine air17 and air 18 into air.c
air.c <- rbind(air17.c, air18.c)
# adding relative calculations (percentage of total yearly arrivals/departures per hour)
ArrivalPct <- air.c$Arrival / air.c$TotalArrival
DeparturePct <- air.c$Departure / air.c$TotalDeparture
air.c <- cbind(air.c, ArrivalPct, DeparturePct)
# result check
datatable(air.c, rownames = FALSE, filter='top', options = list(pageLength = 10, scrollX= TRUE))
```

Show 10 entries Search:

Year	Hour	Arrival	Departure	TotalArrival	TotalDeparture	ArrivalPct	Dep
				A	All	All	All
2017	0	109770	498707	31071662	31010370	0.00353280104553146	0.01608194
2017	1	28682	254771	31071662	31010370	0.000923091915714068	0.00821567
2017	2	46792	64177	31071662	31010370	0.00150593811171092	0.00206953
2017	3	179620	18400	31071662	31010370	0.00578083013390143	0.000593349
2017	4	1038334	43739	31071662	31010370	0.0334173949240308	0.00141046
2017	5	1785259	18774	31071662	31010370	0.0574561798464466	0.000605410
2017	6	2225334	113316	31071662	31010370	0.0716194067765027	0.00365413
2017	7	1544489	825870	31071662	31010370	0.0497073185206507	0.02663205
2017	8	1322593	2098728	31071662	31010370	0.0425658917118756	0.06767826
2017	9	1104749	2577750	31071662	31010370	0.0355548731187923	0.08312541

Showing 1 to 10 of 48 entries Previous 1 2 3 4 5 Next

The air passenger data sets required a bit more attribute engineering. We first had to remove the *TOTAL* row from both files (2017 & 2018). After that, we added a *Year* attribute to enable us to latter merge the two data sets. We then isolated a substring of the time description (used to be in the form of "00:00 ~ 00:59") to standardise the values to the same values found in the bike sharing data set (integers from 0 to 23). We then had to replace "," characters found in the total *Arrival* and *Departure* counts before being able to cast them to integer. Further, two new attributes *TotalArrival* and *TotalDeparture* were added to both the 2017 and 2018 data which hold the total arrivals and departures per year respectively. This information will be used to calculate the **percentage** of arrivals/departures per hour compared to the total arrivals/departures per year. The last step combined the two year 2017 and 2018 into one data frame *air.c*.

The next step will merge the air passenger information with the bike sharing data to create one data frame as the basis for further analysis and modelling.

The last step will then serialise the resulting object (the *air.c* data frame) so that the data preparation steps do not have to be run everytime the analysis and/or the R markdown file are run. This allows us to simply load the created and exported data frame via the *readRDS()* function before starting with the analysis.

```r
# merging the two data frames based on 'Year' and 'Hour)'
data.m <- merge(bikedata.c, air.c, by = c('Year','Hour'), all = TRUE)
# sorting the result by 'Date' and 'Hour'
data.m <- data.m[order(data.m$Date, data.m$Hour),]
# persist resulting (basis) dataframe for further analysis via serialisation
saveRDS(data.m, file='../db/data.RDS')
```

Exploratory Analysis

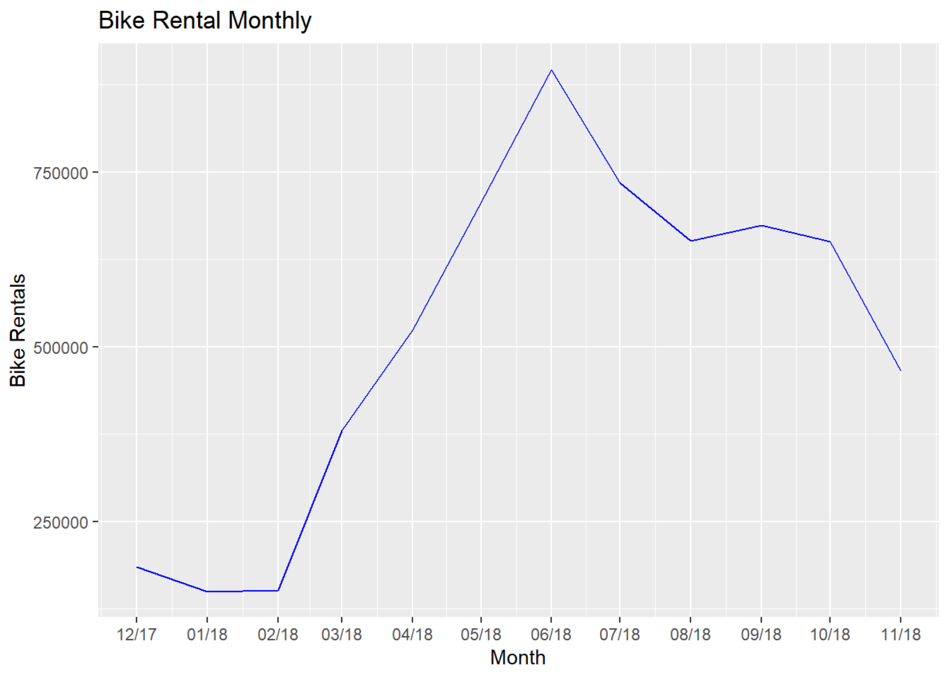
```
#setwd(here())
data <- readRDS('../db/data.RDS')
head(data)
```

	Year <dbl>	Hour <int>	Date <date>	RentCount <int>	Temperature <dbl>	Humidity <int>	Windspeed <dbl>	Visibility <int>	DewPointTemp <dbl>
1	2017	0	2017-12-01	254	-5.2	37	2.2	2000	-17.6
41	2017	1	2017-12-01	204	-5.5	38	0.8	2000	-17.6
379	2017	2	2017-12-01	173	-6.0	39	1.0	2000	-17.7
530	2017	3	2017-12-01	107	-6.2	40	0.9	2000	-17.6
571	2017	4	2017-12-01	78	-6.0	36	2.3	2000	-18.6
598	2017	5	2017-12-01	100	-6.4	37	1.5	2000	-18.7

6 rows | 1-10 of 22 columns

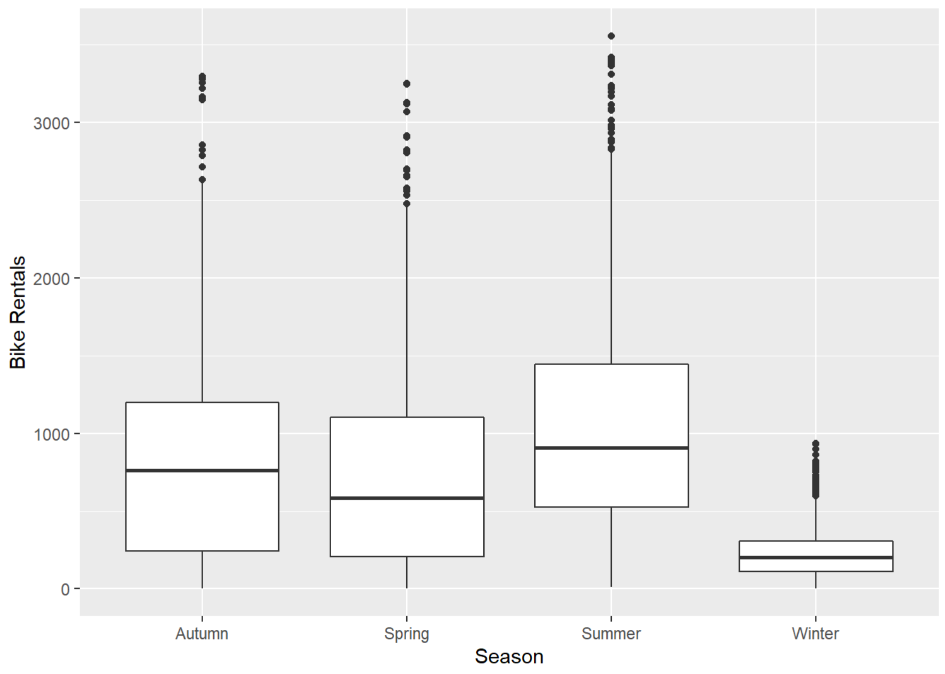
Let us now create various graphs to get insights to the data. We are interested to see how the bike rental count developed in the months from end of 2017 to end of 2018.

```
#head(data)

data$Month <- as.Date(cut(data$Date,
  breaks = "month"))

ggplot(data = data, mapping = aes(x=Month, y=RentCount))+
  stat_summary(fun = sum, geom = "line", color="blue")+
  labs(title="Bike Rental Monthly", y="Bike Rentals")+
  scale_x_date(labels =  date_format("%m/%y"), breaks = "1 month")
```



Clearly a peak of bike rentals is visible in the transition from spring to summer whereas the minimum is in the winter months. Let’s compare the different seasons by creating boxplots for each season.

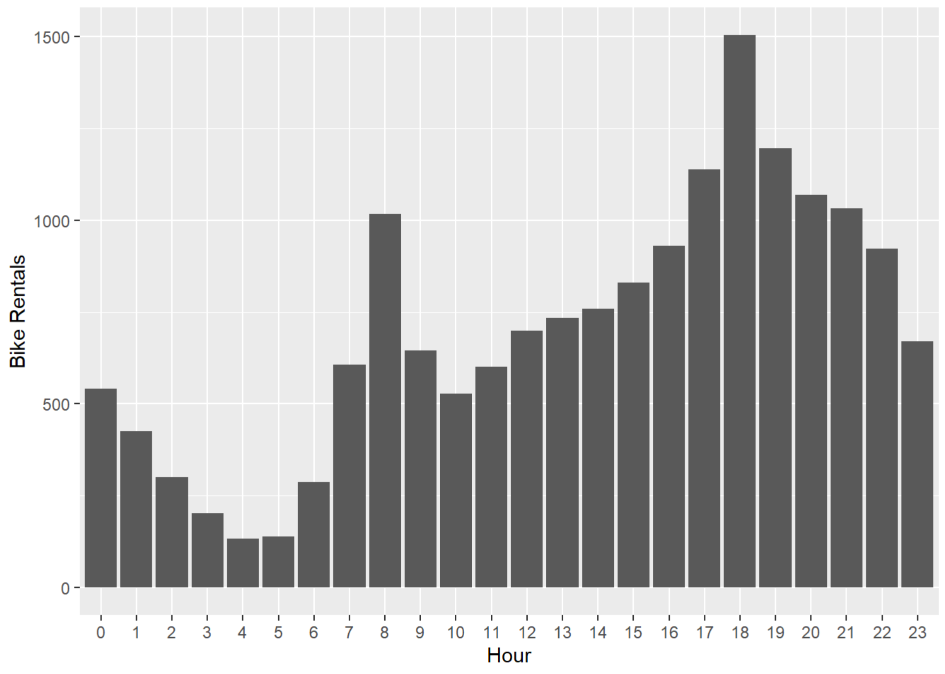
```
#This shows the hourly data not daily
ggplot(data = data, mapping = aes(x=Season, y = RentCount)) +
  geom_boxplot()+
  labs(y = "Bike Rentals")
```



As expected the season of winter differs from the warmer months.More bike rentals occur from spring to summer compared to winter. Autumn and spring show a similar distribution.

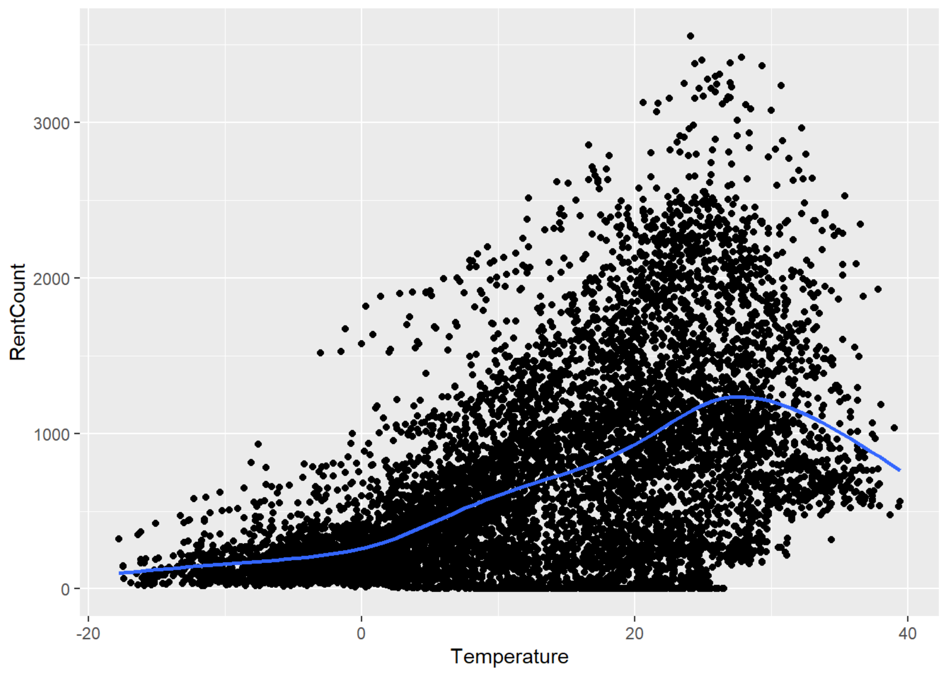
Let’s see how the rental count behaves during a day. Are certain hours preferred for renting a bike? In order to get a first answer to the question we will average over the hours.

```
ggplot(data = data, mapping = aes(x=as.factor(Hour), y=RentCount))+
  stat_summary(fun="mean", geom="bar")+
  labs(y = "Bike Rentals", x = "Hour")
```



We assume that the temperature influences the number of bikes rented. Let’s create a scatterplot comparing those two variables.

```
#Plot the hourly count values based on recorded temperatures (in °C)
ggplot(data = data, mapping = aes(y = RentCount, x = Temperature)) +
  geom_point()+
  geom_smooth(se = FALSE)
```



The resulting plot shows an interesting relationship between temperature and the hourly rental count:

- There seems to be a strongly positive trend between -20 °C (almost 0 rentals) up to around +27/28 °C (average of around 1’000 rentals)
- After that, a clear decline in the average hourly rental count is visible beyond +27/28 °C

This seems inherently explainable since people are less likely to use a bicycle in full summer heat and to prefer alternative modes of transport (preferably with A/C).

Linear Modelling

In this chapter, the linear relationship between various predictors and the target variable *RentCount* will be considered. In a first step, a simple linear model with 1 predictor will be considered whereas the second step will implement a more complex model. Both models are based on **generalised linear models (GLM)** as we deal with a **count** of rentals as the target variable.

Implementing first (simple) Model

As we are looking at count data it makes sense to use a generalised linear model (GLM) in order to specify the distribution of our data as a poisson and apply the appropriate link function (logarithm). In a first, simple model we will look into the effect of the **season** categorical variable (*factor* in R) onto the target variable **rental count**.

```
glm.bike <- glm(RentCount ~ Season,
                family = "poisson",
                data = data)

glm.bike
```

```
##
## Call:  glm(formula = RentCount ~ Season, family = "poisson", data = data)
##
## Coefficients:
##  (Intercept)  SeasonSpring  SeasonSummer  SeasonWinter
##       6.7088       -0.1157        0.2324       -1.2903
##
## Degrees of Freedom: 8759 Total (i.e. Null);  8756 Residual
## Null Deviance:       4979000
## Residual Deviance: 3682000   AIC: 3749000
```

```
# Absolute count numbers per model
print(paste(  'Intercept (Autumn) = ', exp(coef(glm.bike))[1],
        '\nSpring Absolute Count = ', round(exp(coef(glm.bike))[1] * exp(coef(glm.bike))[2],0),
        '\nSummer Absolute Count = ', round(exp(coef(glm.bike))[1] * exp(coef(glm.bike))[3], 0),
        '\nWinter Absolute Count = ', round(exp(coef(glm.bike))[1] * exp(coef(glm.bike))[4], 0)
))
```

```
## [1] "Intercept (Autumn) =  819.597985348541 \nSpring Absolute Count =  730 \nSummer Absolute Count =  1034 \nWinter Absol
ute Count =  226"
```

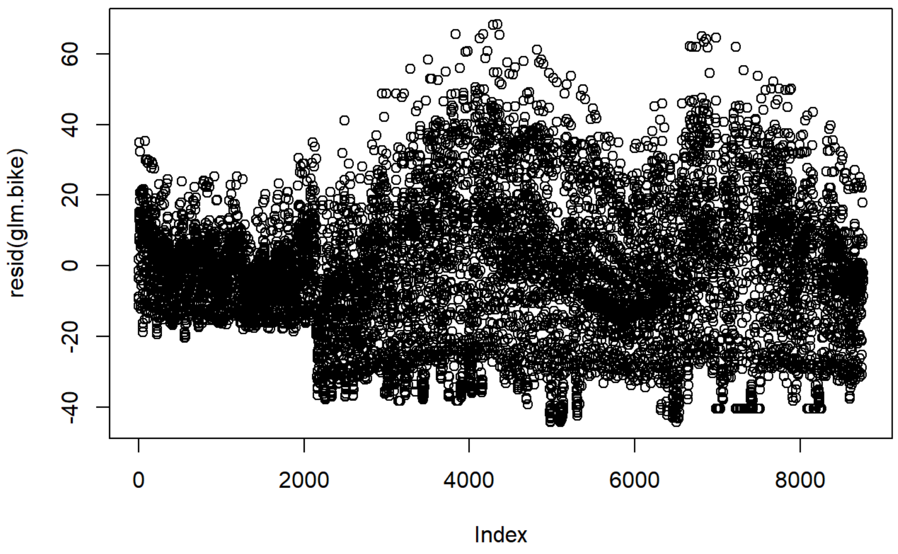
As suspected when investigating the boxplots based on season, temperature obviously has a significant impact on the bike rental count. However, as also evident in these results, there seems to be only a relatively minor difference between autumn, spring and summer (819, $0.9 * 819 = 730, 1.26 * 810 = 1034$). The only relatively large differences occurs between these three seasons and winter ($0.28 * 819 = 226$).

Model Performance

In order to be able to compare the model performance of our first simple model to later, improved versions we will look into model performance by analysing the **mean squared error(MSE)** of our current model.

Let's start by visualising the residuals (errors) per sample point. After that the MSE will be calculated.

```
plot(resid(glm.bike))
```



```
# calculating the MSE using the residuals (errors) of the simple model
glm.bike.mse <- mean((data$RentCount-fitted(glm.bike))^2)

#data.frame(fitted(glm.bike), data$RentCount, fitted(glm.bike) - data$RentCount)
glm.bike.mse
```

```
## [1] 328564.5
```

The plot of residuals shows a few interesting characteristics:

- the error for the first ~2000 samples seems to be significantly smaller than for the remaining 6000 samples
- the model tends to underestimate the actual value since residuals are largely positive and have a larger positive extreme (around 60) compared to the negative extreme (around -40)

The calculated **MSE is 328564.5** ($MSE = 1/n * \sum(Y - \hat{Y})^2 = 328564.5$)

In order to get a more reliable measure of fit, we will further compute the test set MSE using **10-fold cross-validation** in the next part:

```
# split dataset into 10 parts
ids <- sample(1:10, nrow(data), replace=TRUE  )
df.test <- data.frame(data, id = round(ids,0))

# fit model to each training set and compute test MSE
mse <- c()
for(i in 1:10){
  fit.updated <- update(glm.bike,
                        data = df.test[df.test['id']!=i,])
  # predict using test set
  fit.test <- predict(fit.updated, newdata = df.test[df.test['id']==i,])
  # store results in df
  pred.test <- data.frame( pred = exp(fit.test), obs = df.test[df.test['id']==i,]$RentCount)

  # compute MSE
  mse[i] <- mean((pred.test$obs - pred.test$pred)^2)
}
mean(mse)
```

```
## [1] 329106.6
```

This yields a 10-fold cross validated testing MSE of **329106.6**, slightly worse than the in-sample / training MSE.

Extended Modelling

After creating our first simple model, let us now implement a more complex one by adding new predictors. In the exploratory analysis we assumed that the variables “Hour” and “Temperature” have an impact on the number of bike rentals. We presume that the variables “Holiday” and “Arrival” might also influence the bike rentals. In order to test our assumptions we will use a GLM containing those variables.

```
glm.bike.complex <- glm(RentCount ~ Holiday + Hour + Temperature  + Arrival,
                family = "poisson",
                data = data)

summary(glm.bike.complex)
```

```
##
## Call:
## glm(formula = RentCount ~ Holiday + Hour + Temperature + Arrival,
##     family = "poisson", data = data)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -52.830  -12.692   -3.151    8.658   60.733
##
## Coefficients:
##                   Estimate Std. Error z value Pr(>|z|)
## (Intercept)      4.983e+00  2.466e-03  2021.0   <2e-16 ***
## HolidayNo Holiday 2.198e-01  2.195e-03   100.2   <2e-16 ***
## Hour             4.823e-02  6.510e-05   740.8   <2e-16 ***
## Temperature      4.063e-02  3.780e-05  1074.7   <2e-16 ***
## Arrival          6.477e-08  5.668e-10   114.3   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for poisson family taken to be 1)
##
##     Null deviance: 4979261  on 8759  degrees of freedom
## Residual deviance: 2732524  on 8755  degrees of freedom
## AIC: 2799628
##
## Number of Fisher Scoring iterations: 5
```

There is strong evidence that the variables included in the model have an effect on the number of bikes rented. In order to interpret the coefficients we need to take the exponential as we did in the simple model. Let's for example interpret the coefficient for the variable "Temperature".

```
exp(coef(glm.bike.complex)["Temperature"])
```

```
## Temperature
##    1.041465
```

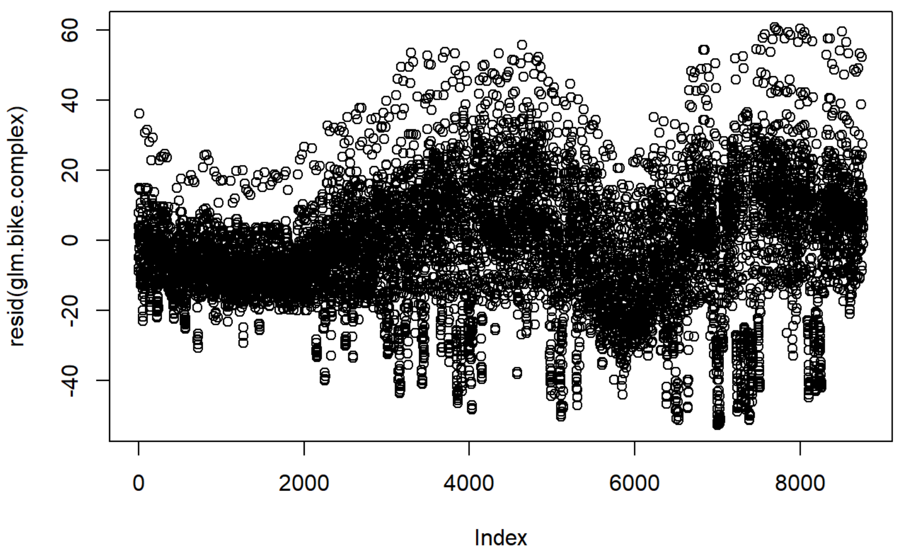
Interpretation: If the temperature increases by 1 the number of bikes rented increases on average by 4.14%.

Complex Model Performance

In order to be able to compare the model performance of our complex model to the simple one, we will look into model performance by analysing the **mean squared error(MSE)** of our complex model.

Let's start by visualising the residuals (errors) per sample point. After that the MSE will be calculated.

```
plot(resid(glm.bike.complex))
```



```
# calculating the MSE using the residuals (errors) of the simple model

glm.bike.complex.mse <- mean((data$RentCount-fitted(glm.bike.complex))^2)
glm.bike.complex.mse
```

```
## [1] 239356
```

The plot of residuals shows a few interesting characteristics:

- In comparison to the residual plot of the simple model, the data points seem to be more concentrated and nearer to the y-value 0, implying a better prediction of the complex model.
- Similar to the residual plot of the simple model, the residuals vary in the range of -40 to 60.

The calculated **MSE is 239356** ($MSE = 1/n * \sum(Y - \hat{Y})^2 = 239356$) and therefore smaller than the MSE of the simple model.

In order to get a more reliable measure of fit, we will further compute the test set MSE using **10-fold cross-validation** in the next part:

```
set.seed(21)
# split dataset into 10 parts
ids <- sample(1:10, nrow(data), replace=TRUE  )
df.test <- data.frame(data, id = round(ids,0))

# fit model to each training set and compute test MSE
mse <- c()
for(i in 1:10){
  fit.updated <- update(glm.bike.complex,
                        data = df.test[df.test['id']!=i,])
  # predict using test set
  fit.test <- predict(fit.updated, newdata = df.test[df.test['id']==i,])
  # store results in df
  pred.test <- data.frame( pred = exp(fit.test), obs = df.test[df.test['id']==i,]$RentCount)

  # compute MSE
  mse[i] <- mean((pred.test$obs - pred.test$pred)^2)
}
mean(mse)
```

```
## [1] 239671.1
```

This yields a 10-fold cross validated testing MSE of **239671**, slightly worse than the in-sample / training MSE.

In the course of creating the code for the 10-fold cross validation we found that there is an extra implemented function for this purpose. As one of our goal is to find and learn about new functions in R we will redo the 10-fold cross validation with that specific function.

```
set.seed(12)
cv.glm.complex <- cv.glm(data,glm.bike.complex, K=10)
cv.glm.complex$delta
```

```
## [1] 239823.2 239799.1
```

This yields a 10-fold cross validated testing MSE of **239823.2** and an adjusted MSE of **239799.1**

[CHAPTER OF CHOICE] - Exploring ***plotly***

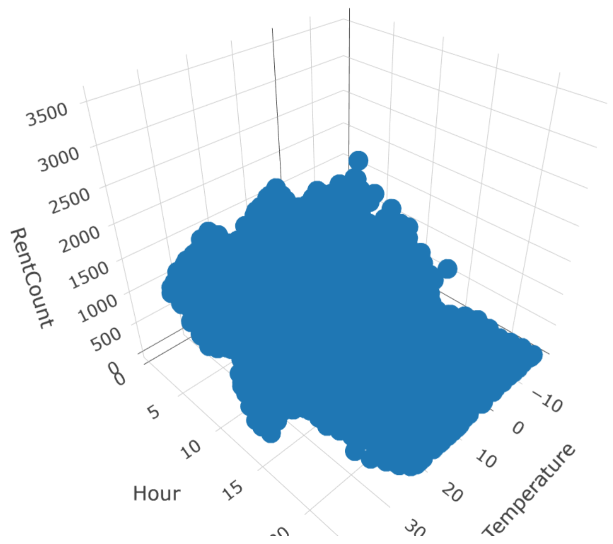
NOTE: The following plots are meant to be interactive and are therefore best enjoyed in the .HTML markdown version!!

For the chapter of choice - with the aim to use a package not mentioned in the course - we will explore the **plotly** library. This library allows to embed advanced, interactive charts in markdown by using the *plotly.js* JavaScript library. The documentation for this library can be found here: https://plotly.com/r/ (https://plotly.com/r/)

Interactive 3D Plot

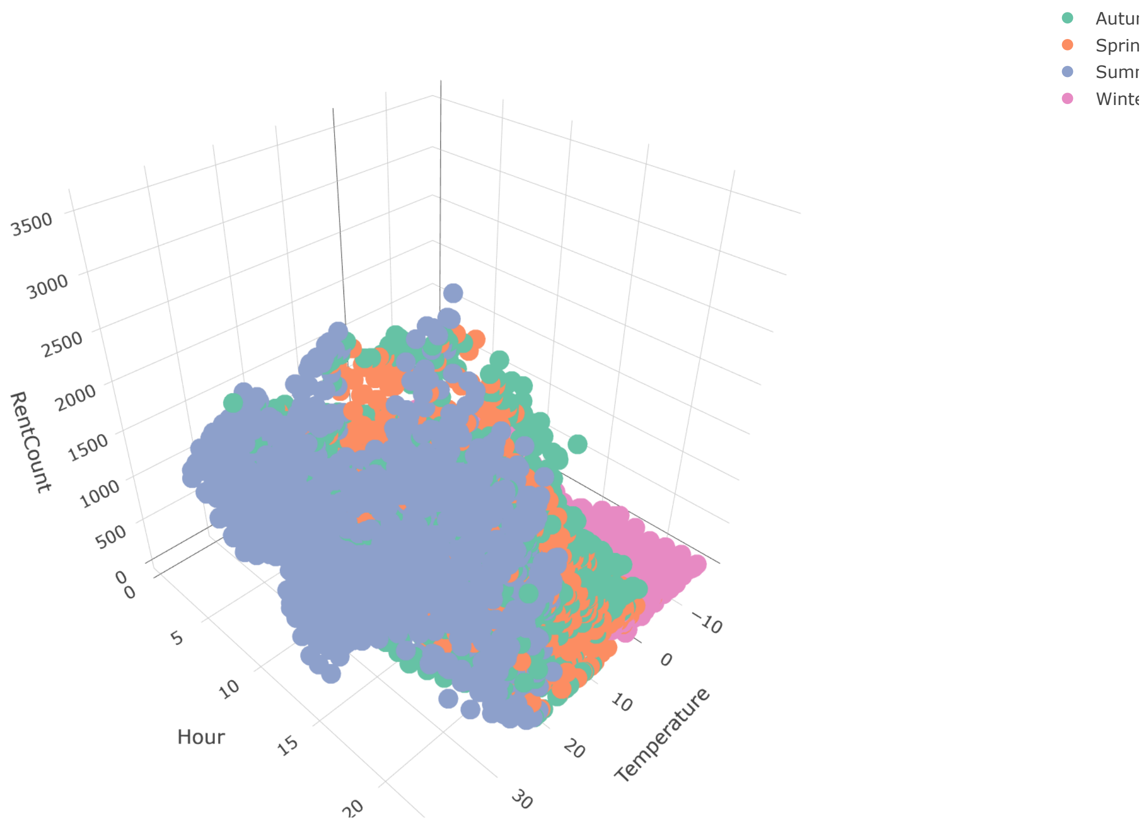
Lets start with a simple, multi-dimensional plot using the *plot_ly()* function. Per default, plotly graphs allow for interactive selections and filtering, which is very nice!

```
#library(plotly) #install.packages('plotly')

fig <- plot_ly(data=data,
        x=~Temperature,
        y=~Hour,
        z=~RentCount,
        type="scatter3d", mode="markers")
fig
```



That does not seem too bad! We have quickly created an **interactive 3D plot** which allows us to change the perspective and hover over single data points. However, the plot area is rather small and there are many data points to be displayed. In the current form, its hard to distinguish the separate points. Let's see if we can adjust the plot area accordingly and create more room for the 3D plot to shine. In order to show case interactive capabilities of plotly, we will further add a categorical variable and use this to color the different data points. By doing so, plotly will automatically add a filter and allow to select different levels of the factor variable.

```
fig2 <- plot_ly(data=data,
        x=~Temperature,
        y=~Hour,
        z=~RentCount,
        type="scatter3d", mode="markers", color=~Season #here, we added the color argument and set it to the 'Season' factor
)
# in order to adjust the plot area, we have to adjust the width, height and margins of the graph as well as deactivate autos
ize
fig2 %>% layout(autosize = FALSE, width = 1000, height = 600, margin = list(l=50, r=10, b=10, t=10, pad=1))
```



Now this seems a lot better! In this version, we are able to select/deselect one ore several seasons and only show the necessary observations belonging to those seasons. The graph also has more room so the single observations become apparent and we can see all axis and their labels.

Interactive, Animated Line Plot

The plotly package offers a wide variety of possibilities. One of which is interactive and **animated** plots which we will explore in the next part.

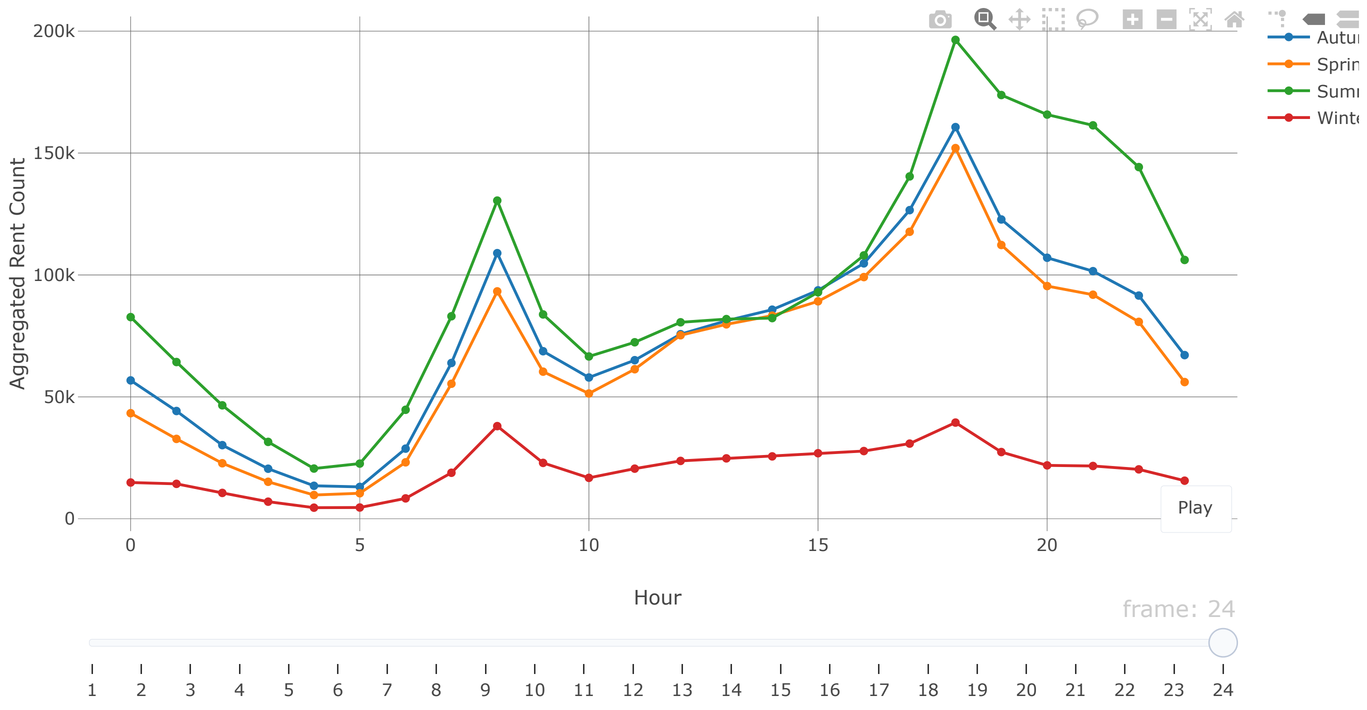
The goal is to show an animated plot which tracks the development of the total rent count per hour, split by season. To do that, we first need to aggregate the data accordingly and add a frame index to each set of values (count per hour and season). In order to show a moving line, each frame needs to contain both the information for itself, as well as the previous frames. All this transformation and plotting is done in the next chunk:

```
# creating an aggregated dataframe with rental counts per hour and season
data.acc <- data %>%
    group_by(Hour, Season) %>%
    summarise(SumRentCount = sum(RentCount))
head(data.acc)
```

Hour	Season	SumRentCount
<int>	<fctr>	<int>
0	Autumn	56755
0	Spring	43298
0	Summer	82714
0	Winter	14866
1	Autumn	44200
1	Spring	32755

6 rows

```r
# adding frame information to each element in the data frame. Each frame needs cumalitive information (so info for this fram
e and all previous frames)
lvls <- plotly:::getLevels(data.acc$Hour)
seq <- seq_along(lvls)

data.acc$frame <- seq[data.acc$Hour+1]
data.acc2 <- data.acc[0,]

for (i in 1:max(seq)){
  data.update <- data.acc[data.acc$frame %in% 1:i,]
  data.update$frame <- i
  data.acc2 <- rbind(data.acc2, data.update)
}

# implementing an interactive, animated line plot showing the development per hour over the different seasons

fig3 <- data.acc2 %>%
  plot_ly(
    x = ~Hour,
    y = ~SumRentCount,
    frame = ~frame,
    #color = ~Season,
    split = ~Season,
    type = 'scatter',
    #mode = 'lines',
    line = list(simplyfy = FALSE)
  )
fig3 <- fig3 %>% layout(
  xaxis = list(
    title = "Hour",
    zeroline = FALSE
  ),
  yaxis = list(
    title = "Aggregated Rent Count",
    zeroline = FALSE
  )
)
fig3 <- fig3 %>% animation_opts(
  frame = 100,
  transition = 0,
  redraw = FALSE
)
fig3 <- fig3 %>% animation_slider(
  hide = FALSE
)
fig3 <- fig3 %>% animation_button(
  x = 1, xanchor = "right", y = 0, yanchor = "bottom"
)

fig3 <- fig3 %>% layout(autosize = FALSE, width = 1000, height = 500, margin = list(l=50, r=10, b=10, t=10, pad=1))
#htmlwidgets::saveWidget(fig3, 'fig3.html')

fig3
```



References

Data Bridge Market Research (2020). *Bike and Scooter Rental Market Increasing Demand, Industry Share, Revenue Analysis by 2026*. Published 27.05.2020, accessed on 10.09.2020 via https://www.globenewswire.com/news-release/2020/05/27/2039660/0/en/Bike-and-Scooter-Rental-Market-Increasing-Demand-Industry-Share-Revenue-Analysis-by-2026-Top-Leaders-Cityscoot-LIME-Uber-Coup-Mobility-nextbike-GmbH-Mobike-%E6%91%A9%E6%8B%9C%E5%8D%95%E8%BD%A6-eCooltra-Yulu-emmy-.html (https://www.globenewswire.com/news-release/2020/05/27/2039660/0/en/Bike-and-Scooter-Rental-Market-Increasing-Demand-Industry-Share-Revenue-Analysis-by-2026-Top-Leaders-Cityscoot-LIME-Uber-Coup-Mobility-nextbike-GmbH-Mobike-%E6%91%A9%E6%8B%9C%E5%8D%95%E8%BD%A6-eCooltra-Yulu-emmy-.html)

Hexa Research (2019). *Bike Rental Market Size And Forecast, By Product (Docked, Dockless), By Region (North America, Europe, Asia Pacific, Rest of the World) And Trend Analysis, 2019 - 2025*. Published February, 2019, accessed on 10.09.2020 via https://www.hexaresearch.com/research-report/bike-rental-market (https://www.hexaresearch.com/research-report/bike-rental-market)

Mordor Intelligence (2019). *Bike Sharing Market - Growth, Trends, And Forecasts (2020 - 2025)*. Published 2019, accessed on 11.09.2020 via https://www.mordorintelligence.com/industry-reports/bike-sharing-market (https://www.mordorintelligence.com/industry-reports/bike-sharing-market)

The Guardian (2017). *Chinese discard hundreds of cycles-for-hire in giant piles*. published 17.01.2017, accessed on 11.09.2020 via https://www.theguardian.com/world/2017/jan/17/chinese-discard-hundreds-of-cycles-for-hire-in-giant-pile (https://www.theguardian.com/world/2017/jan/17/chinese-discard-hundreds-of-cycles-for-hire-in-giant-pile)