Edinburgh Bicycles

Felix Wiesner

07/01/2021

Abstract

This document details efforts to source and better understand trip data from a publicly available bike scheme in Edinburgh. The data origin and how it was obtained (scraped) is explained, as are key underlying features of the data set. Machine learning principles are applied to the data to predict, with limited success, the expected duration of a trip if the start station is known. In addition the anticipated net starts from any station are estimated from machine learning. For the latter, multiple approaches are assessed, the most succesfull being a generalized boosted regression model with an RMSE of 2.35. Overall, this project gives further insight into a popular bike sharing scheme and explains how data science principles can help to optimise bike availabilty.

Contents

1 Introduction

In cities, cycling is often considered as a healthier, more environmentally friendly alternative to driving. However, bikes themselves cost money, require maintenance, and a secure storage space. For casual bike users, the purchase and maintenance of a bike might constitute efforts that outweigh the benefits of having regular access to a bike. This gap is bridged by bike sharing schemes, that exist in multiple cities. These schemes usually consist of stations of bikes that allow time-limited use for a bike for a small fee.

Edinburgh is the capital of Scotland, with a population of roughly half a million. The city's bike-share scheme is sponsored by Just-Eat, so the scheme is known as the Just Eat Cycles. Users can either pay a one off fee per one-hour trip taken with a bike or purchase a subscription that allows free trips for an hour. Trips that last beyond an hour cost an additional £0.05 per minute.

This project is part of the HarvardX: PH125.9x Data Science: Capstone course, hosted on the EdX platform. The aim of the project is to use a publicly available data set to solve the problem of my choice.

2 The data

The Edinburgh Just Eat cycle scheme was chosen for this project as it offers its data for open access at: https://edinburghcyclehire.com/open-data.

This project will have two main tasks, achieved using R: (1) sourcing the data, and (2) analysing the data to assess if it is suitable for machine learning.

2.1 Sourcing the data

The data is freely available and can be downloaded either as .csv files, or in json format. The most straightforward solution would be to download the csv files for each month individually and to then read those files into R. However, this is would not be feasible for large datasets and would not fully utilise the capabilities that the R environment offers. Instead, the data will be scraped directly from the website. For this the *rvest* package will be utilised, alongside the stringr package and the tidyverse environment.

```
if(!require(rvest)) install.packages("rvest",
repos = "http://cran.us.r-project.org")
if(!require(tidyverse)) install.packages("tidyverse",
repos = "http://cran.us.r-project.org")
if(!require(caret)) install.packages("caret",
repos = "http://cran.us.r-project.org")
if(!require(stringr)) install.packages("stringr",
repos = "http://cran.us.r-project.org")
if(!require(lubridate)) install.packages("lubridate",
repos = "http://cran.us.r-project.org")
if(!require(randomForest)) install.packages("randomForest",
repos = "http://cran.us.r-project.org")


library(rvest)
library(stringr)
library(tidyverse)
library(lubridate)
library(caret)
library(randomForest)
```

The web address can be defined as a string.

```
targetpage<-'https://edinburghcyclehire.com/open-data/historical'
```

Within the target page the required nodes can be extracted as shown below. This will return a vector containing strings with the links to all available .csv files, which correspond to data from each month on record.

```
targetnodes<-read_html(targetpage) %>%
  html_nodes(".hHEtRQ") %>%
  html_attr(name='content')

head(targetnodes)
```

```
## [1] "https://data.urbansharing.com/edinburghcyclehire.com/trips/v1/2021/01.csv"
## [2] "https://data.urbansharing.com/edinburghcyclehire.com/trips/v1/2020/12.csv"
## [3] "https://data.urbansharing.com/edinburghcyclehire.com/trips/v1/2020/11.csv"
## [4] "https://data.urbansharing.com/edinburghcyclehire.com/trips/v1/2020/10.csv"
## [5] "https://data.urbansharing.com/edinburghcyclehire.com/trips/v1/2020/09.csv"
## [6] "https://data.urbansharing.com/edinburghcyclehire.com/trips/v1/2020/08.csv"
```

After all links to files are sourced, these can now be used to read in the data from each file. For this the lapply function is used to *read.csv()* on all links in *targetnodes*, the output from each of these calls is bound together into a global data frame containing all the data.

Since this call can take a while, this code snippet is not evaluated for this document; the output is provided as a separate file instead.

```
# now we can read each of the links. this may take a while
cycleframe<-do.call(rbind,lapply(targetnodes,read.csv))

save(cycleframe,file='cycleframe.Rda')
```

The data contains a multitude of variables: the ids and names of the start and end stations for each trip, respectively, as well as the time of the trip and the duration. The data already contains the geographical locations for both start and end station. To add to the data, we will also add the elevation data for each *start_station_id*. To do this we will use the free API from https://open-elevation.com. This open data web page allows determination of elevation based on latitude and longitude data. The requests can be slow, so rather than sending individual requests for each station, we will lump all *station_ids* into one json request. This code can take a while and can sometimes time out, so as for the .csv files above, we will only run this once and then store the data for further analysis.

```
library(httr)
library(jsonlite)


# define a function to read elevation data from open-elevation
scrape_elev <- function (indat)
{
  u <- "https://api.open-elevation.com/api/v1/lookup"
  h <- c ("Accept" = "application/json",
          "Content-Type" = "application/json")
  dat <- indat %>%
    jsonlite::toJSON ()
  b <- paste0 ('{"locations":', dat, '}')
  res <- httr::POST (u, httr::add_headers (.headers = h), body = b)
  httr::content (res, as = 'text', encoding = "UTF-8",
                 type = "application/xml") %>%
    jsonlite::fromJSON()

}
```

To use the *scrape_elev()* function that we defined above, we need format our geographical data into a data frame that only consists of the required values.

```
# here the data is loaded from the local respository to avoid a repeat of the download.
# This also allows analysis when temporary without internet access.
load('cycleframe.Rda')
eledata<-cycleframe %>%distinct(start_station_id,.keep_all = TRUE)%>%
  select(start_station_latitude,start_station_longitude)
colnames(eledata)<-c('latitude','longitude')
head(eledata)
```

```
##   latitude longitude
## 1 55.93651 -3.180166
## 2 55.93981 -3.182739
## 3 55.93643 -3.194150
## 4 55.93532 -3.198763
```

```
## 5 55.93971 -3.220589
## 6 55.90841 -3.328784
```

Now we can use the created data frame with our JSON API function that we defined above. Within the function each row in the data frame will be formatted into a json entry. All the json entries are then sent to the API which returns the elevation data for each entry (i.e. each station). As for the download of the original csv data, this process can take a while, and is therefore only run twice, once of the start_station_ids and once for the end stations. The code has been checked and worked, however, as *open-elevation* is open source, it can experience time outs for large requests, it is therefore recommended to run the scraping operations separately, they are shown here for completeness, but with options *'eval=F'*, meaning the code chunks are shown but not run.

```
# get the data for each end station
out<-scrape_elev(eledata)
head(out$results)

# The elevation data for the start stations can now be stored in a seperate dataframe.
startele<-data.frame(start_station_id=cycleframe %>%distinct(start_station_id),
                     start_station_elevation=out$results$elevation)


# now add the elevations to the original dataframe
new<-left_join(cycleframe,startele,by="start_station_id")
```

We now repeat this procedure for the end stations, then the data can, again, be merged with the original *cycleframe* data frame.

```
# now the process can be repeated for the end stations
# create a dataframe that can be used with the required format
eledata<-cycleframe %>%distinct(end_station_id,.keep_all = TRUE)%>%
  select(start_station_latitude,start_station_longitude)
colnames(eledata)<-c('latitude','longitude')

# get the data for each end station
out<-scrape_elev(eledata)

endele<-data.frame(end_station_id=cycleframe %>%distinct(end_station_id),
                   end_station_elevation=out$results$elevation)

# now add the end elevations to the original dataframe
new<-left_join(new,endele,by="end_station_id")
```

```
# now store it all as a dataframe locally

cycleframe_plus<-new

save(cycleframe_plus,file='cycleframe_plus.Rda')
```

2.2 Exploration of data

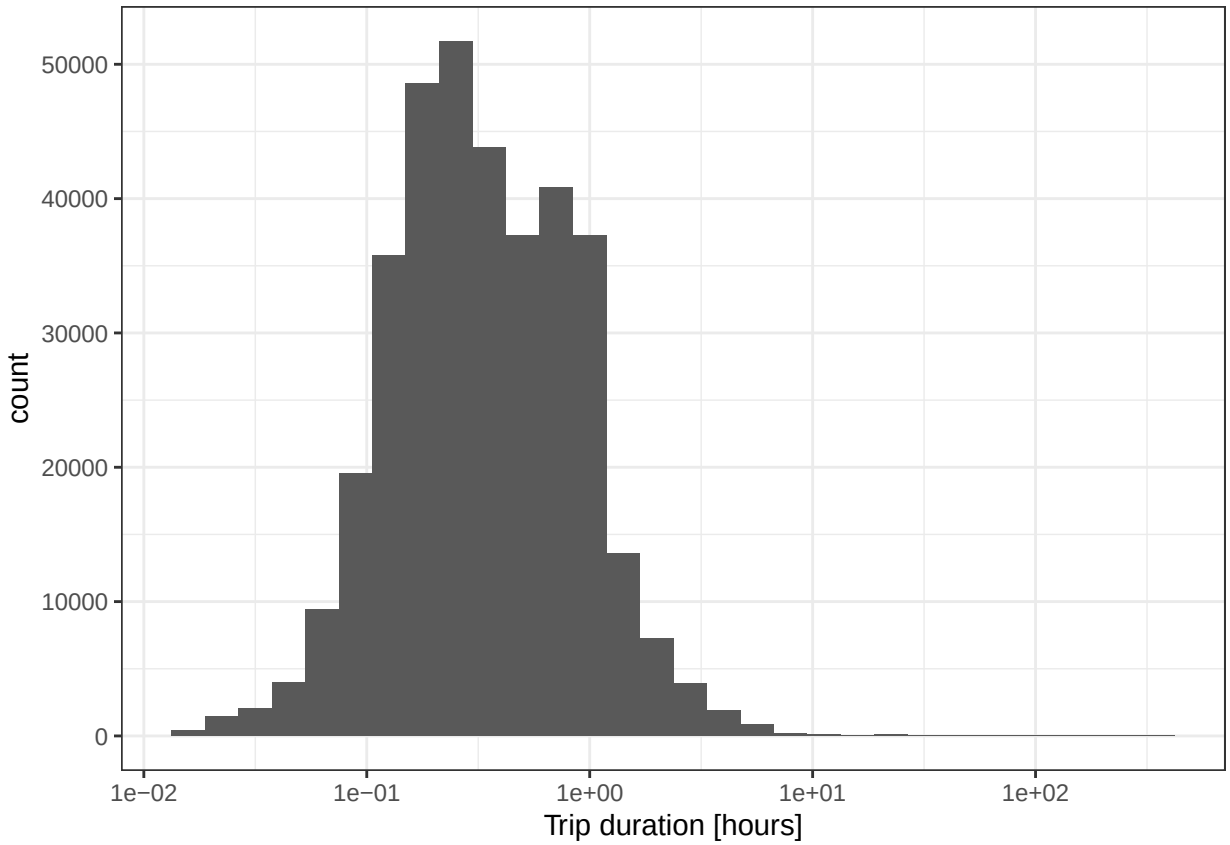
Now that we have sourced the data, and added elevation data for each entry, we can explore the data a bit.

```
# load the data from the previsouly generated file
load('cycleframe_plus.Rda')
```

2.2.1 Trip duration

One variable we can look at is the duration of each trip and the distribution of trip size. Note that we have to use a log scale here to properly visualise the data; this is caused by extremely long trip durations, some of which take over 1000 hours. The trip duration appears to follow a normal distribution.

```
cycleframe_plus%>%
  ggplot(aes(x=duration/(60*60)))+
  geom_histogram()+ scale_x_log10()+
  xlab('Trip duration [hours]')+
  theme_bw()
```



We can look at the top 10 trip durations, these are shown in **days** below. It can be observed that the top trip durations take more than 15 days. Considering that any additional hour beyond a trip costs £0.05 per minute, a 15 day trip would cost a user around £1,000. It is unlikely that anyone would voluntarily pay this amount of money for a three gear rental bike. Instead we can assume that these trip durations arise from errors with bike placement. Maybe sometimes a bike does not correctly connect to its return station, or maybe the bike was discarded somewhere. For this reason trips that appear abnormally long or short are removed from the dataset.

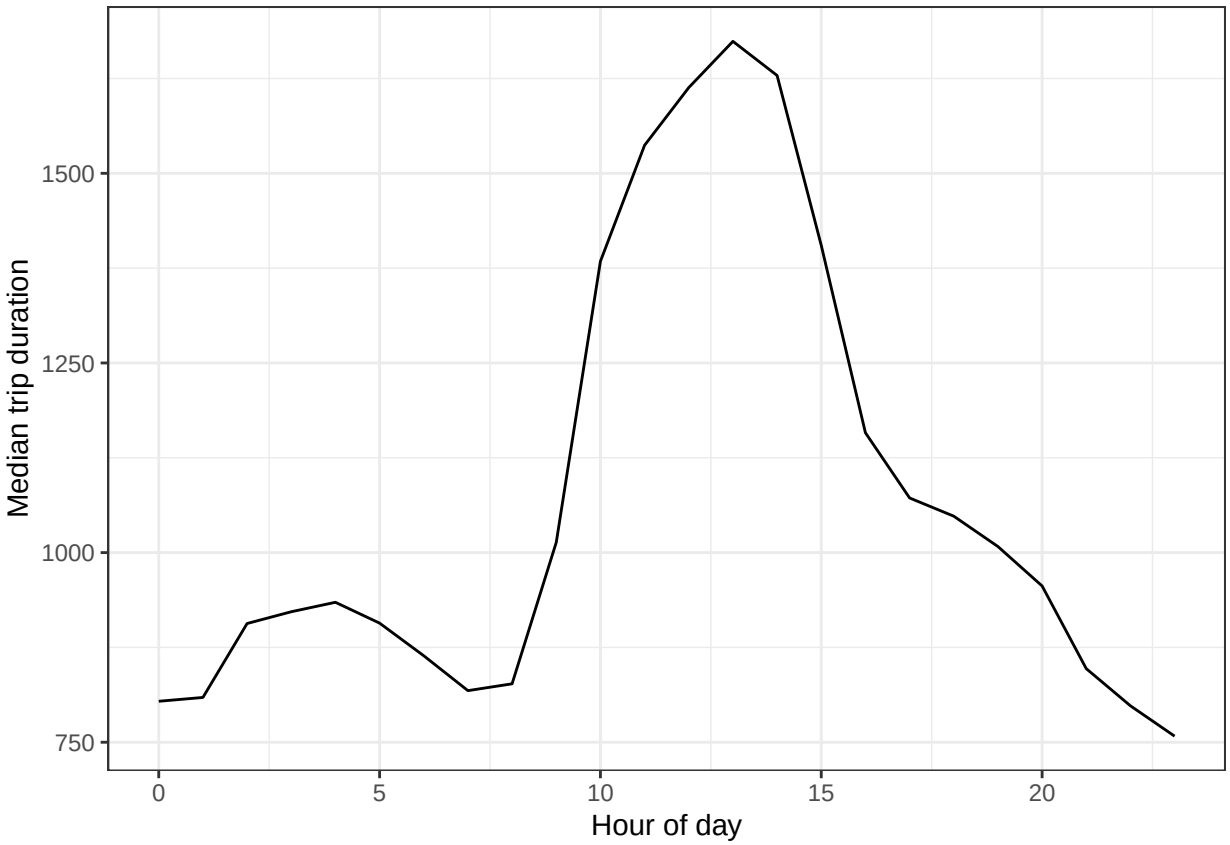
```
# let's get the top 10 durations
cycleframe_plus%>%
  select(duration) %>% '/'(60*60*24) %>%
  slice_max(duration , n = 10) %>%
  `colnames<-`(c('trip duration [days]'))%>%
  knitr::kable()
```

trip duration [days]
15.891701
15.437639
15.019768
13.643866
13.013264
11.954144
11.624051
11.303715
9.973275
9.833357

```
# let's remove trips that are longer than 12 hours and shorter than 2 minutes
cycleframe_plus<-cycleframe_plus%>%filter(duration<12*60*60,duration>60*2)
```

The data also contains the time when each trip starts. This can be seperated into the type of day (i.e. workday or weekend), the time, the hour of the day and the month. We can then assess how trip durations change throughout the day. As would be expected, users take longer trips during the day.

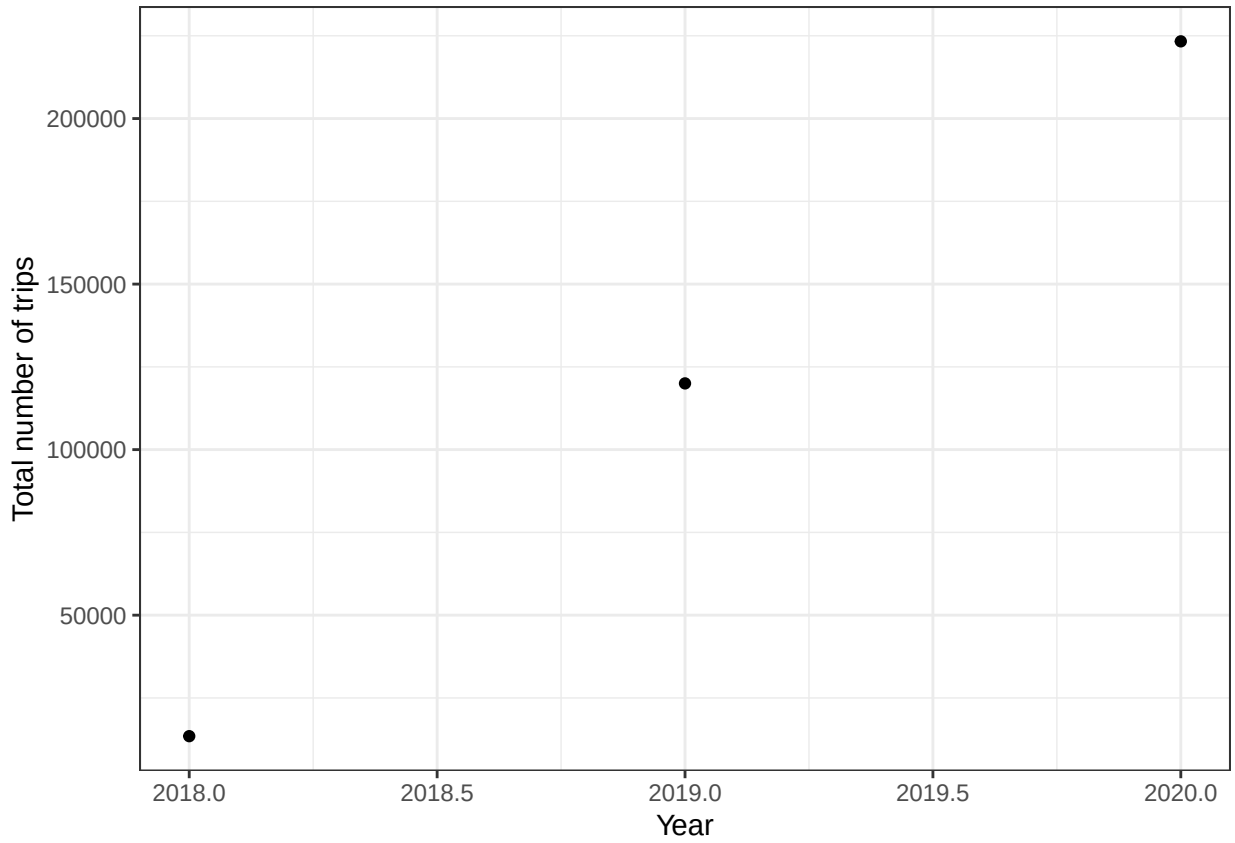
```
# define a vector with all workdays
workdays<-c('Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday')

new<-cycleframe_plus  %>%
  mutate(day=weekdays(as.Date(started_at)),
                      date=as.Date(started_at),
                      time=format(as.POSIXct(started_at), format = "%H:%M:%S"),
                      daytype=as.factor(c('weekend', 'weekday')
                                        [(weekdays(as.Date(started_at)) %in%
                                            workdays)+1L]),
                      hour=hour(hms(as.character(time))),
                      month= as.factor(month.abb[month(date)]),
                      year=year(date),
                      day=weekdays(as.Date(started_at)))


new%>% group_by(hour) %>%
  summarize(md=median(duration)) %>%
  ggplot(aes(x=hour,y=md))+
  geom_line()+
  xlab('Hour of day')+
  ylab('Median trip duration')+
  theme_bw()
```



2.2.2 No of trips

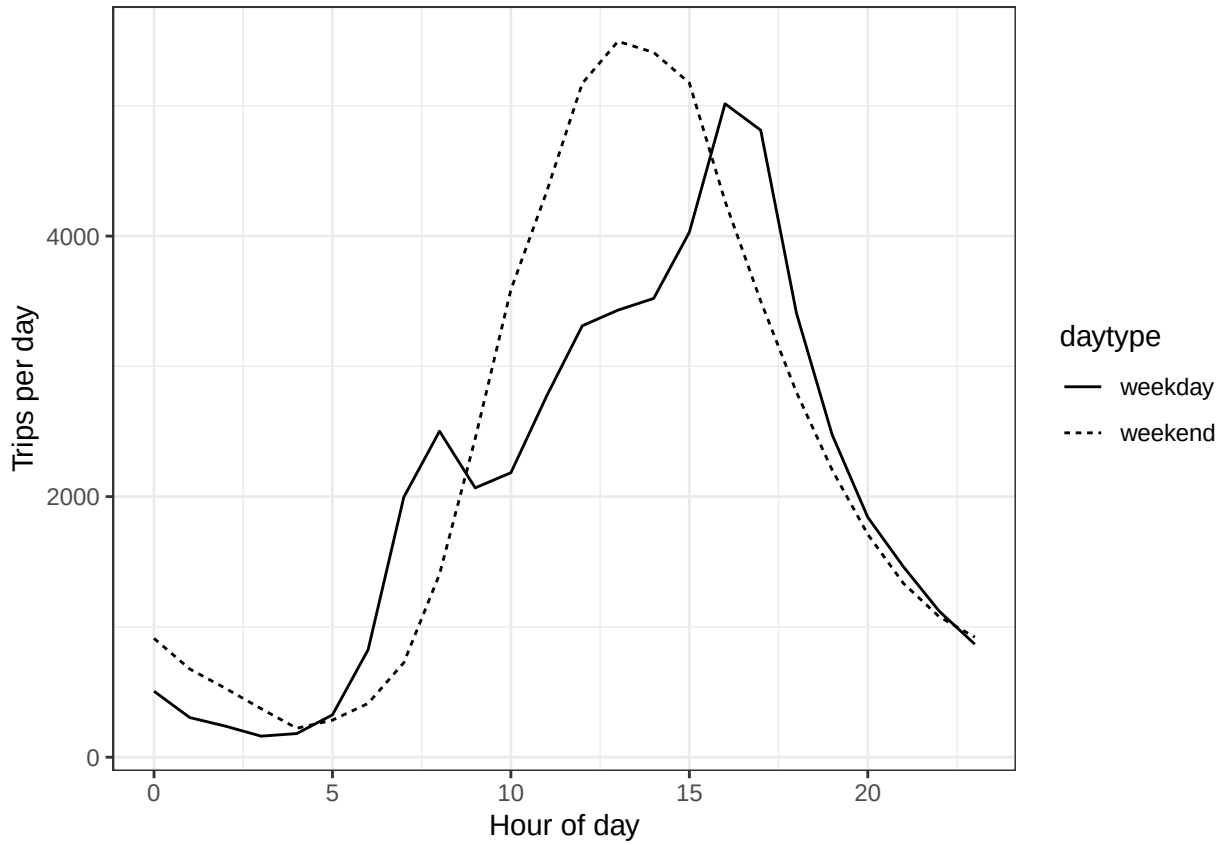
Another potentially interesting variable is the number of trips taken. First, let's consider the total number of trips in any given year. We can see the there is a linear increase in total trips every year since the cycle scheme started. This could be due to an expanded user base, or the increase in stations.

```
new%>%group_by(year)%>%
  summarise(c=n())%>%
  ggplot(aes(x=year,y=c))+geom_point()+
  xlab('Year')+
  ylab('Total number of trips')+
  theme_bw()
```



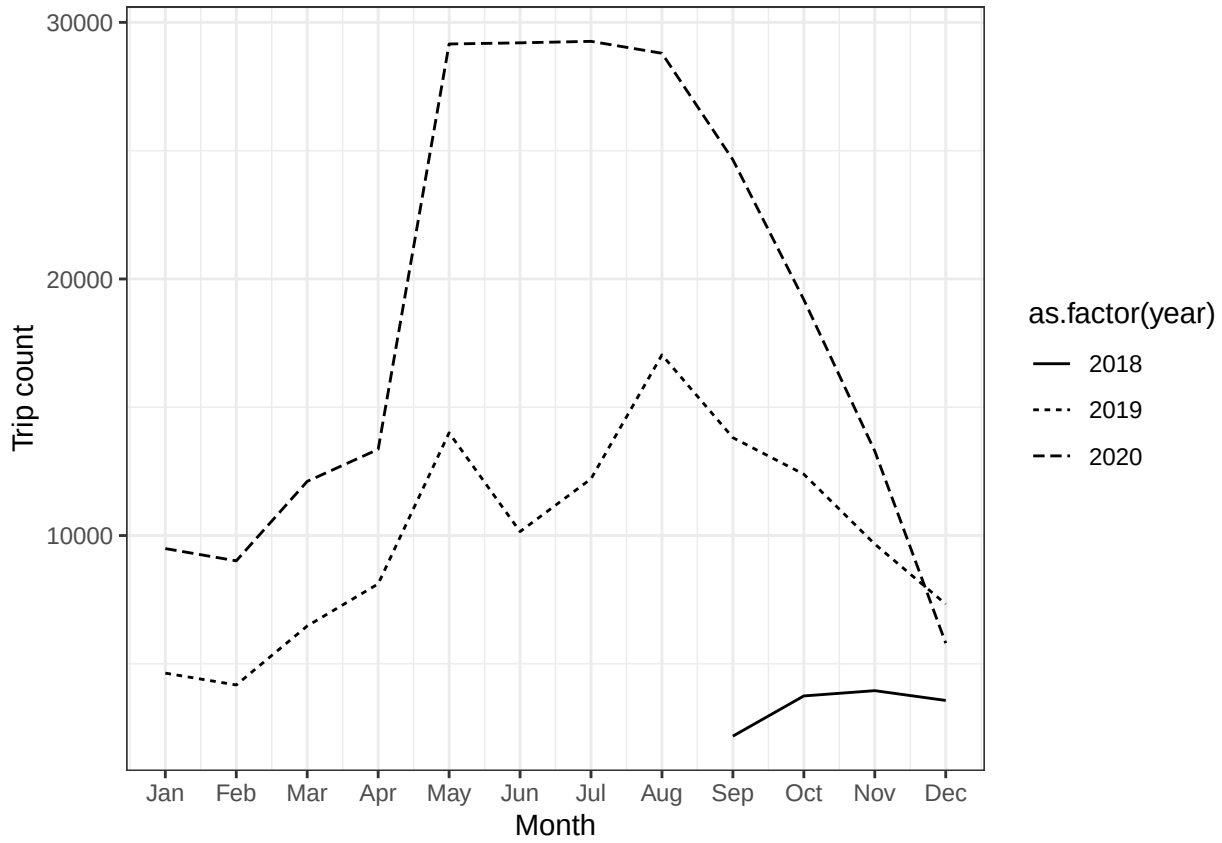
We can also assess the number of trips that occur at all stations per day and plot them against the hour of the day for each daytype (i.e. weekend or workday). For this we have to specify the number of days in the weekend and workdays, otherwise the values for workdays would be much larger, as unfortunately there are five workdays and only two days of weekend. By dividing the number of trips by eitehr 2 or 5 days we ensure that the values are normalised per day. We can see that, as expected, more trips occur during the day than at night. We can also see a slight difference in trips during week and the weekend. On workdays, two peaks exists, which could be related to commuters.

```
new %>% mutate(hour=hour(hms(as.character(time))),
               no_of_days=c(2,5)[(day %in% workdays)+1L])%>%
  group_by(hour,daytype) %>%
  summarize(mcount=n()/no_of_days,daytype=daytype) %>%
  ggplot(aes(x=hour,y=mcount,linetype=daytype)) + geom_line()+
  xlab('Hour of day')+
  ylab('Trips per day')+
  theme_bw()
```



The data also gives us the opportunity to assess seasonal variation in trips for each year on record. It can be observed that many more trips occur in the summer months, compared to winter. Two peaks can be observed for 2019, one coinciding with the onset of spring/summer, and another peak in August, when the city hosts multiple international festivals and the population increases significantly. In 2020 more trips can be observed, as discussed, this is likely related to the overall grwoth of the cycle scheme. No peaks can be observed for 2020; this can be attributed to the cancellation of the festival due to the global COVID-19 pandemic. The trip count is consistently high in the warm months of 2020. During the pandemic cycling estabished itself as an attractive mode of transport compared to public transport in cramped conditions (e.g. busses).

```
new %>% mutate(month=month(date)) %>%
  group_by(month,year) %>% summarize (Mcount=n()) %>%
  ggplot(aes(x=month,y=Mcount)) +geom_line(aes(linetype=as.factor(year)))+
  xlab('Month')+
  ylab('Trip count')+
  scale_x_continuous(breaks=1:12, labels=month.abb)+
  theme_bw()
```



Let's look at the top and bottom number of trips per station. We can see that some stations have a very low number of trips. These seem to be related to specific events, which might have been one-offs. To avoid the influence of these unique entries we will remove any stations with 100 trips or less from our data.

```
cycleframe_plus%>%group_by(start_station_name) %>%
summarize(count=n()) %>%
  slice_max(count , n = 5) %>%
   `colnames<-`(c('Station name','top trip count'))%>%
  knitr::kable()
```

Station name	top trip count
Meadows East	13619
Bristo Square	11321
St Andrew Square	10617
Meadow Place	9245
Portobello - Kings Road	8332

```
cycleframe_plus%>%group_by(start_station_name) %>%
summarize(count=n()) %>%
  slice_min(count , n = 5) %>%
   `colnames<-`(c('Station name','bottom trip count')) %>%
knitr::kable()
```

Station name	bottom trip count
City Chambers Launch Station	1

Station name	bottom trip count
Cycling Scotland Conference	1
Picady Place	1
Royal Highland Show - West Gate (19th to 23rd June)	1
Pleasance - Edinburgh University Sports Fair	2

```
new<-new%>%group_by(start_station_name) %>% filter(n()>100) %>% ungroup()
```

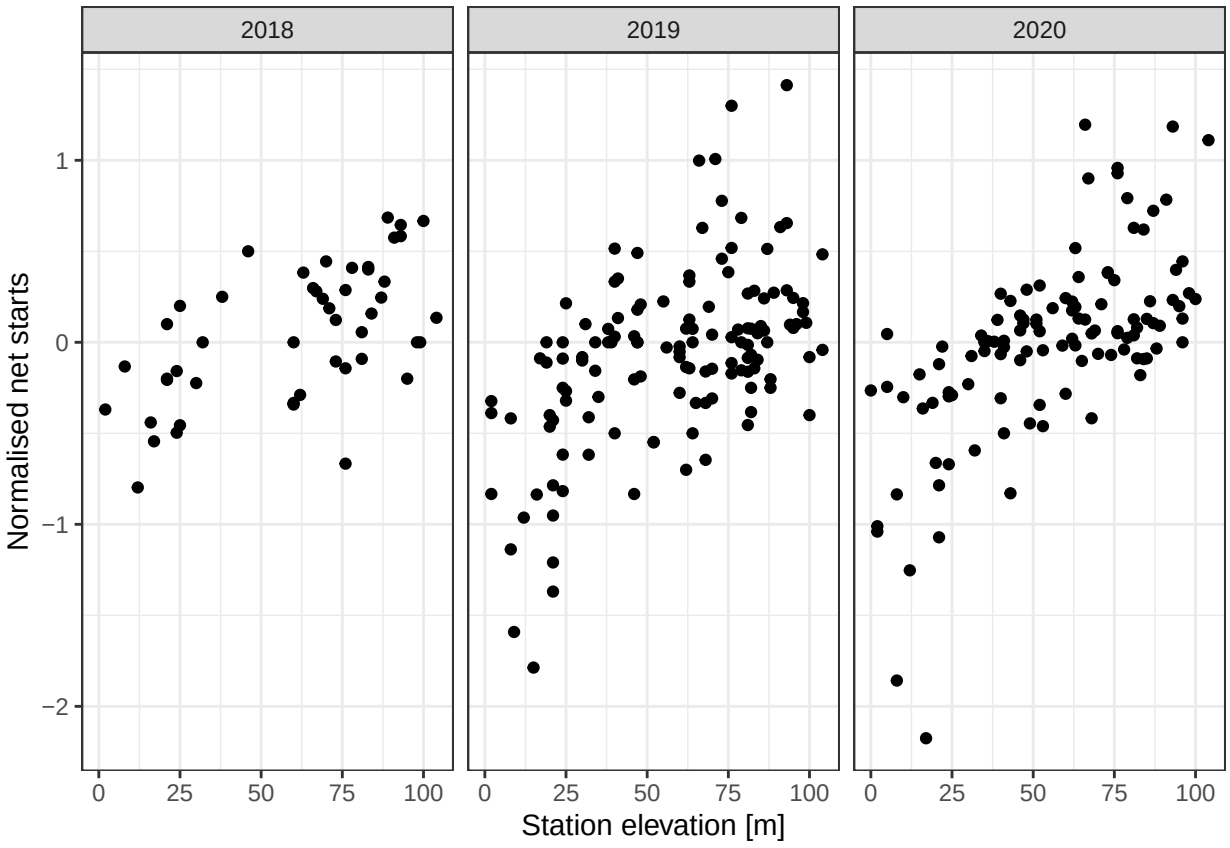
2.2.3 Net starts

Of possible interest are the net starts that occur at each station. This number will be the difference of starts and endings at each station, either in total or within a defined timeframe.

```
starts<-new%>%group_by(as.factor(start_station_id),month,daytype,hour,year,day)%>%
summarise(starts=(n()),
elevation=mean(start_station_elevation)) %>%
`colnames<-`(c('station_id','month','daytype','hour','year','day','starts','elevation'))

ends<-new%>%group_by(as.factor(end_station_id),month,daytype,hour,year,day) %>%
summarise(ends=n()) %>%
`colnames<-`(c('station_id','month','daytype','hour','year','day','ends')) %>% ungroup()


net<-inner_join(starts,ends,by=c('station_id','hour','month','daytype','year','day')) %>%
  mutate(net_starts=starts-ends)
```

Lets look at the total net starts divided by total trips for each station (otherwise stations with a lot of trips will dominate the plot) plotted against elevation. There is a clear positive correlation between the elevation of a station and the number of net starts as a proportion of all station trips. Cycling uphill can be hard, so it is clear that scheme users prefer to take a bike from an elevated station and return it to a station that is situated at a lower elevation. This data vindicates our decision to go through the effort of obtaining the elevation data.

```
net %>% group_by(station_id, year)%>%
  summarise(ns=sum(net_starts/n()),elevation=mean(elevation)) %>%
  ggplot(aes(x=elevation,y=ns))+geom_point()+
  facet_grid(cols=vars(year))+
  xlab('Station elevation [m]')+
  ylab('Normalised net starts')+
  theme_bw()
```



3 Data Analysis

So far this project has detailed how cycle data from Edinburgh can be accessed and how additional elevation data can be obtained for each station, based on latitude and longitude. This next section will investigate if machine learning principles can be applied to predict soem of the aspects that have been investigated. Two outcomes will be attempted to be predicted: (1) the duration of a trip, and (2) the net starts of a station for a particular hour or time period. Both of these parameters are potentially useful for the scheme owners/operators. Knowing the duration of a trip can help to predict how long a bike will be unavailable for, and knowledge about the expected net starts helps to anticipate at which stations bikes will be needed or where too many bikes will accumualte. As we have seen, the net starts differ between stations, bike schemes usually solve this by bulk moving bikes between stations.

3.1 Duration

Each trip has a unique duration, so if we consider the whole data set we have more than 360,000 rows to consider. In order to speed up the computation we will create a smaller dataset from the available data and will only select variables that are potentially useful in the analysis. Any features related to the end stations are removed, sicne we don't know where a customer will venture once they pick up their bike.

```
set.seed(1984)

new<-cycleframe_plus%>% mutate(day=weekdays(as.Date(started_at)),
                               date=as.Date(started_at),
                               time=format(as.POSIXct(started_at), format = "%H:%M:%S"),
                               daytype=as.factor(c('weekend', 'weekday')
                                                 [(weekdays(as.Date(started_at)) %in%
```

```
                                                    workdays)+1L]),
                                hour=hour(hms(as.character(time))),
                                month= month(date),
                                year=year(date)) %>%
  select(-c('start_station_name','end_station_name','start_station_description',
            'end_station_description','end_station_id','end_station_elevation',
            'end_station_longitude','end_station_latitude',
            'started_at','ended_at','time','date')) # removed cols



# now select a smaller subset from the trimmed down dataset.

new_s<-new[sample(nrow(new),size=25000),]
```

Now we can try to attempt to predict trip duration based on the retained variables. The first attempt will be the to simply use the mean duration of each trip. Since the outcome is continuous we will use the root mean squared error (RMSE) to measure the quality of our prediction.

```
ti <- createDataPartition(y=new_s$duration, times = 1, p = 0.5, list = FALSE)

test_set <- new_s[ti, ]
train_set <- new_s[-ti, ]

# we define a function to get the Root measn squared error in minutes
RMSE <- function(true_values, pred_values){
  sqrt(mean((true_values - pred_values)^2,na.rm = T))/60
}


y_h_mean<-mean(train_set$duration)

rmse_mean<-RMSE(test_set$duration,y_h_mean)
rmse_mean
```

```
## [1] 38.56999
```

We obtain an rmse of around 29 minutes. This does not sound too bad, although if we compare it against the mean trip duration of 32 minutes, it appears to be not a very good prediction. Our expected error more than doubles or halves the predicted outcome. Let's see if we can do better using a linear regression fit, to account for the individual variables.

```
linfit<-lm(duration~.,data=train_set)

y_h_lin<-predict(linfit,test_set)


rmse_lin<-RMSE(test_set$duration,y_h_lin)
rmse_lin
```

```
## [1] 37.31711
```

The RMSE barely improves, indicating that either our model, or the underlying data are not suitable for this type of prediction. Instead of trying to predict the duration as a continous variable we can try to classify how long a duration will be once a bike is picked up. To do this we *cut()* the duration variable into a factor with three levels, short, medium and long trips.

```
new_c<-new_s
# cut the duration into three classes.
new_c$duration<-cut(new_c$duration,3,
                    labels=c('short','medium','long'))

test_set <- new_c[ti, ]
train_set <- new_c[-ti, ]

train_rf <- randomForest(duration ~., data = train_set)
y_h_rf<-predict(train_rf,test_set)

confusionMatrix(y_h_rf,test_set$duration)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction short medium  long
##     short  12420     72     6
##     medium     3      0     0
##     long       0      0     0
##
## Overall Statistics
##
##                Accuracy : 0.9935
##                  95% CI : (0.992, 0.9949)
##     No Information Rate : 0.9938
##     P-Value [Acc > NIR] : 0.6603
##
##                   Kappa : -4e-04
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: short Class: medium Class: long
## Sensitivity                0.9998       0.00000     0.00000
## Specificity                0.0000       0.99976     1.00000
## Pos Pred Value             0.9938       0.00000         NaN
## Neg Pred Value             0.0000       0.99424     0.99952
## Prevalence                 0.9938       0.00576     0.00048
## Detection Rate             0.9935       0.00000     0.00000
## Detection Prevalence       0.9998       0.00024     0.00000
## Balanced Accuracy          0.4999       0.49988     0.50000
```

From the confusion matrix we can see that we achieve a very high accuracy, but we can also see an issue with our data. The prevalence of medium and long trips is very low, which is why our accuracy is so high for short trips. Our balanced accuracy is actually relatively low. One way around this is to cut the duration into quantiles, so that each quantile contains the same amount of data points. This can be achieved with the *cut2()* function from the Hmisc package.

```
library(Hmisc)

new_c<-new_s
# cut the duration into three classes.
new_c$duration<-cut2(new_c$duration,g=3)
```

```r
# note that cut2() does not support levels or labels

test_set <- new_c[ti, ]
train_set <- new_c[-ti, ]

train_rf <- randomForest(duration ~., data = train_set)
y_h_rf<-predict(train_rf,test_set)

confusionMatrix(y_h_rf,test_set$duration)
```

```
## Confusion Matrix and Statistics
##
##               Reference
## Prediction     [ 121,  769) [ 769, 1883) [1883,41639]
##   [ 121,  769)         2347         1388          802
##   [ 769, 1883)         1157         1554          991
##   [1883,41639]          690         1195         2377
##
## Overall Statistics
##
##                Accuracy : 0.5022
##                  95% CI : (0.4934, 0.511)
##     No Information Rate : 0.3355
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.2531
##
##  Mcnemar's Test P-Value : 1.74e-10
##
## Statistics by Class:
##
##                      Class: [ 121,  769) Class: [ 769, 1883)
## Sensitivity                       0.5596              0.3756
## Specificity                       0.7364              0.7432
## Pos Pred Value                    0.5173              0.4198
## Neg Pred Value                    0.7681              0.7064
## Prevalence                        0.3355              0.3309
## Detection Rate                    0.1877              0.1243
## Detection Prevalence              0.3629              0.2961
## Balanced Accuracy                 0.6480              0.5594
##                      Class: [1883,41639]
## Sensitivity                       0.5700
## Specificity                       0.7737
## Pos Pred Value                    0.5577
## Neg Pred Value                    0.7824
## Prevalence                        0.3336
## Detection Rate                    0.1901
## Detection Prevalence              0.3409
## Balanced Accuracy                 0.6719
```

Now the overall accuracy is lower but our balanced accuracy is higher. Still this outcome is not really useful to predict the duration of a trip. The existence of few very long trips complicates the prediction of the majority of trips signifcantly. We must ask ourselves if cutting of trip duration at (somewhat arbritarily) 12 hours was perhaps too generous. We will try to rerun our prediction excerises by using only the lower 95 %

of our duration data; the 95 % quantile corresponds to a trip duration of 90 minutes.

```
new <- new %>% filter(duration < quantile(.$duration, 0.95))


new_s<-new[sample(nrow(new),size=25000),]

ti <- createDataPartition(y=new_s$duration, times = 1, p = 0.5, list = FALSE)

test_set <- new_s[ti, ]
train_set <- new_s[-ti, ]

linfit<-lm(duration~.,data=train_set)

y_h_lin<-predict(linfit,test_set)


rmse_lin<-RMSE(test_set$duration,y_h_lin)
rmse_lin
```

```
## [1] 18.35156
```

We can see that we improved our RMSE, but considering the reduced mean trip duration of 25 minutes, our RMSE is still far from perfect. There is certainly room for improvement. It would be intersting to know if user data would be helpful to predict trip duration. Another potential helpful variable that we could introduce would be distance to the traffic centre or a measure of traffic density w.r.t. the start station id, althugh these measures are out of scope for this epxloratory report.

3.2 Net starts

To assess the net starts that a station encounters we will initially use the *net* data frame that we created earlier for data exploration. We also redefine our RMSE function, since we are not working with minutes but absolute net starts.

```
# take out starts and ends,
# if we include them as features we will get perfect predictions
net<-net%>%select(-c('starts','ends'))


# we also redefine our RMSE function, as we are not predicting
# minutes anymore
RMSE <- function(true_values, pred_values){
  sqrt(mean((true_values - pred_values)^2,na.rm = T))
}



net_s<-net[sample(nrow(net),size=40000),]


ti <- createDataPartition(y=net_s$net_starts, times = 1, p = 0.5, list = FALSE)


test_set <- net_s[ti, ]
train_set <- net_s[-ti, ]
```

Firstly we will try to use the mean net starts from the train set to estimate net starts in the test.

```
y_h_mean=mean(train_set$net_starts)

RMSE(test_set$net_starts,y_h_mean)
```

```
## [1] 2.807096
```

We obtain an RMSE of 2.81.

The data exploration showed that there are some correlations in our data set, especially w.r.t to elevation. So let's try a linear fit.

We can see that our RMSE improves only marginally. One issue is that we define station_id as a factor with many levels, this means computationally each level returns a variable. If we check *summary(linfit)* we get a very long list of coefficients, some of these have no statistical significance.

```
linfit<-lm(net_starts~.,data=train_set)

y_hat_lin<-predict(linfit,test_set)

RMSE(test_set$net_starts,y_hat_lin)
```

```
## [1] 2.74971
```

```
summary(linfit)%>%.$coefficients%>%head()
```

```
##                    Estimate Std. Error    t value     Pr(>|t|)
## (Intercept)   -209.6707160 93.4422097 -2.2438544 2.485277e-02
## station_id183    0.0954369  0.2016990  0.4731651 6.361006e-01
## station_id189    0.7976554  0.2472392  3.2262501 1.256273e-03
## station_id225    0.2305023  0.2867483  0.8038489 4.214939e-01
## station_id246    0.4250674  0.2288866  1.8571090 6.331048e-02
## station_id247    1.1044771  0.2146765  5.1448430 2.702976e-07
```

To organise our data better we will redefine the *new* and *net* data frames to reduce the number of factors used. This will involve changing *months* to the numeric class and rather than using the station_id, we will retain the latitude and longitude of the stations as numeric features.

```
# let's redefine the net starts,

new<-cycleframe_plus  %>%
  mutate(day=weekdays(as.Date(started_at)),
                      date=as.Date(started_at),
                      time=format(as.POSIXct(started_at), format = "%H:%M:%S"),
                      daytype=as.factor(c('weekend', 'weekday')
                                        [(weekdays(as.Date(started_at)) %in%
                                            workdays)+1L]),
                      hour=hour(hms(as.character(time))),
                      month= (month(date)),
                      year=year(date))

starts<-new%>%group_by(as.factor(start_station_id),month,daytype,hour,year,day,start_station_latitude,s
summarise(starts=(n()),
elevation=mean(start_station_elevation)) %>%
`colnames<-`(c('station_id','month','daytype','hour','year','day','starts','elevation','station_latitud

ends<-new%>%group_by(as.factor(end_station_id),month,daytype,hour,year,day) %>%
summarise(ends=n()) %>%
```

```
`colnames<-`(c('station_id','month','daytype','hour','year','day','ends')) %>% ungroup()


net<-inner_join(starts,ends,by=c('station_id','hour','month','daytype','year','day')) %>%
  mutate(net_starts=starts-ends)


net<-subset(net,select=-c(station_id,starts,ends))


net_s<-net[sample(nrow(net),size=40000),]


ti <- createDataPartition(y=net_s$net_starts, times = 1, p = 0.5, list = FALSE)


test_set <- net_s[ti, ]
train_set <- net_s[-ti, ]
```

With this redefinition we can again use a linear fit, and we can see that the RMSE has reduced.

```
linfit<-lm(net_starts~.,data=train_set)
summary(linfit)
```

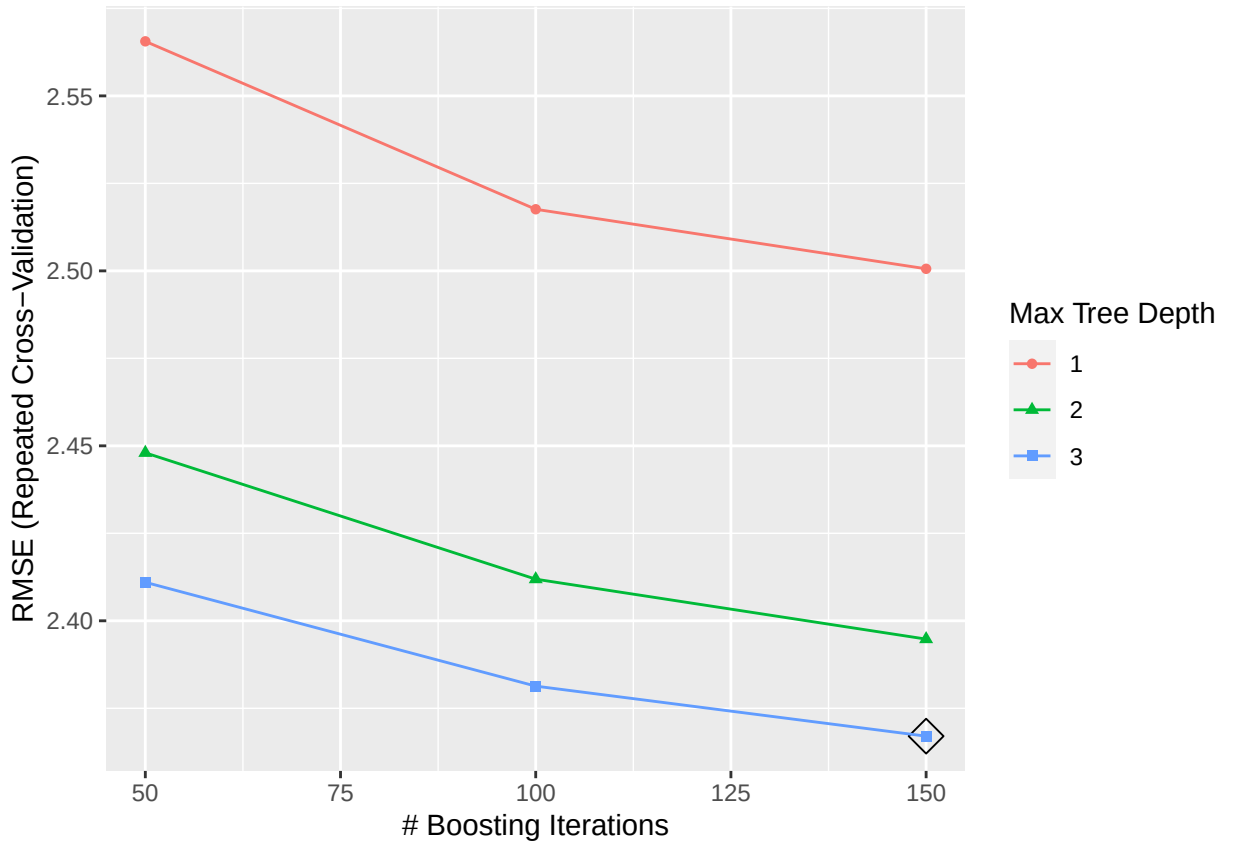
```
##
## Call:
## lm(formula = net_starts ~ ., data = train_set)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -31.3019  -0.8834   0.4156   1.3815  17.3013
##
## Coefficients: (1 not defined because of singularities)
##                     Estimate Std. Error t value Pr(>|t|)
## (Intercept)        5.072e+02  6.519e+01   7.781 7.57e-15 ***
## month             -4.967e-03  6.247e-03  -0.795 0.426598
## daytypeweekend    -9.352e-02  6.677e-02  -1.401 0.161333
## hour              -1.426e-02  3.939e-03  -3.620 0.000295 ***
## year              -2.332e-01  3.236e-02  -7.207 5.92e-13 ***
## dayMonday          4.008e-02  6.769e-02   0.592 0.553786
## daySaturday       -9.666e-02  6.692e-02  -1.444 0.148677
## daySunday                 NA         NA      NA       NA
## dayThursday       -3.882e-02  6.651e-02  -0.584 0.559427
## dayTuesday        -1.363e-02  6.738e-02  -0.202 0.839720
## dayWednesday      -4.394e-02  6.683e-02  -0.657 0.510927
## elevation         -5.498e+00  4.575e-01 -12.018  < 2e-16 ***
## station_latitude  -5.942e-01  6.078e-03 -97.775  < 2e-16 ***
## station_longitude  2.026e-02  6.266e-04  32.330  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 2.553 on 19986 degrees of freedom
## Multiple R-squared:  0.3814, Adjusted R-squared:  0.381
## F-statistic:  1027 on 12 and 19986 DF,  p-value: < 2.2e-16
```

```
y_hat_lin<-predict(linfit,test_set)

RMSE(test_set$net_starts,y_hat_lin)
```

```
## [1] 2.572865
```

In addition to the linear fit we can also try a more advanced algorithm. We are using a Generalized Boosted Regression model from the *gbm* package. The boosting procedure is based on the idea that weak learner outcomes can be boosted by addition of new learning algorithms (e.g. regression trees), whilst removing poorly performing ones from prior boost stages.

```
library(gbm)

fitControl <- trainControl(## 5-fold CV
  method = "repeatedcv",
  number = 5,
  ## repeated two times
  repeats = 2)

train_gbm <- train(net_starts ~ ., data = train_set,
                   method = "gbm",
                   trControl = fitControl,
                   ## This last option is actually one
                   ## for gbm() that passes through
                   verbose = FALSE)


ggplot(train_gbm, highlight = TRUE)
```




```
y_hat_gbm<-predict(train_gbm,test_set)

RMSE(test_set$net_starts,y_hat_gbm)
```

```
## [1] 2.353519
```

We can see that we have further reduced the RMSE to 2.35 to predict net starts for any station, given that we know the date, time, elevation, latitude, and longitude of the station. In practice this means we can try to anticipate how many bikes or free docking spaces will be required at a station each hour. Considering our RMSE we can keep two bikes and two free stations to ensure we have margin for error. In reality the scheme operators also have access to live data, which can help to make decisions on short term bike requirements for a station.

4 Conclusion

This data science project investigated the cycling habits of rental bike users in the city of Edinburgh, Scotland. The project demonstrated how data can be obtained through establishing a connection to the internet and using html tags to find key files for direct reading into R. The project further successfully demonstrated how API requests can be utilised to gather additional data. With the gathered data, this report highlighted some key patterns in how often and how long rental bicycles were accessed in the last three years. Finally, the data was used to predict the expected duration of a trip and the anticipated net starts at a station based on station features and date data. For the latter it was demonstrated that a change from a large categorical feature to a numerical one enabled a reduced RMSE and better predictions.