HarvardX PH125.9x - Data Science: Capstone

Sebastien Pavailler

2022-11-22

Key points

- This project provides models to predict a sport activity duration based on my own activities log.
- The models include a distance and sport effect, as well as a positive elevation conditional effect.
- Four model versions are compared with a minimum average error around 10 minutes.

I. Introduction

For this project I chose to work on one of my passions: endurance sports and training. I thought of designing a project around the sports activity data log I have from my GPS watch, and came up with the following rationale. When planning my training sessions, I often use route planners such as **OpenRunner**. These route planners allow to create a route and obtain a GPS trace which also includes relevant summary data such as distance and positive elevation. The thing is, when I create such a route or plan any training session, I can only rely on my experience to estimate what the *duration* of the session will be. And here comes the data-led project from my GPS watch data: since I have a log of many past training sessions, I should be able to build a predictive algorithm of a particular session duration based on other summary data. Note here that my GPS data uses the term 'activity' to refer to a particular session, so I will also use this term 'activity' in the rest of this report. I selected a few summary data that I hypothesized could have an effect on activity duration:

- Sport - my average displacement speed is obviously different between sports, which will have an influence on duration.
- Distance - this is a no brainer, for any activity the higher the distance the longer the duration.
- Positive elevation - high positive elevation can lead to a decreased average speed and thus longer duration.
- Time of the session - depending on the period of the year and/or the time of the day, my physical shape may be different and so may be the session duration.

Note that I have talked about average speed but of course I will not include it in the predictive variables because speed includes the duration by itself.
Let's import the data from my activity log and build the dataset. The data is included in a *json* file and its structure provided a good data wrangling exercise presented below.

1. Data import

```r
# load relevant packages
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(data.table)) install.packages("data.table",
                                          repos = "http://cran.us.r-project.org")
if(!require(rjson)) install.packages("rjson",
                                     repos = "http://cran.us.r-project.org")
if(!require(lubridate)) install.packages("lubridate",
                                         repos = "http://cran.us.r-project.org")
if(!require(broom)) install.packages("broom", repos = "http://cran.us.r-project.org")
if(!require(gridExtra)) install.packages("gridExtra",
                                         repos = "http://cran.us.r-project.org")

# downloading the dataset
dl <- tempfile()
download.file(
  "https://raw.githubusercontent.com/SebPavailler/data-science-capstone-cyo/main/Activities.json"
  ,dl)

# opens data file - JSON format
tmp_dat <- fromJSON(file=dl)

# see data structure
summary(tmp_dat)
```

```
     Length Class  Mode
[1,] 1      -none- list
```

We can see the imported variable is a list of 1, however I know it contains all my activities. The data should be nested, let's see what is in this list of 1.

```r
summary(tmp_dat[[1]]) # show first element of the list
```

```
                           Length Class  Mode
summarizedActivitiesExport 1000   -none- list
```

Here we are: the first list of 1 contains another list of 1000 elements that correspond to all my activities. Let's unnest the data to get all my activities as this list of 1000.

```r
activities_list <- tmp_dat[[1]]$summarizedActivitiesExport # store the list of activities
rm(tmp_dat)

head(summary(activities_list))
```

```
     Length Class  Mode
[1,] 46     -none- list
[2,] 47     -none- list
[3,] 45     -none- list
[4,] 51     -none- list
[5,] 51     -none- list
[6,] 46     -none- list
```

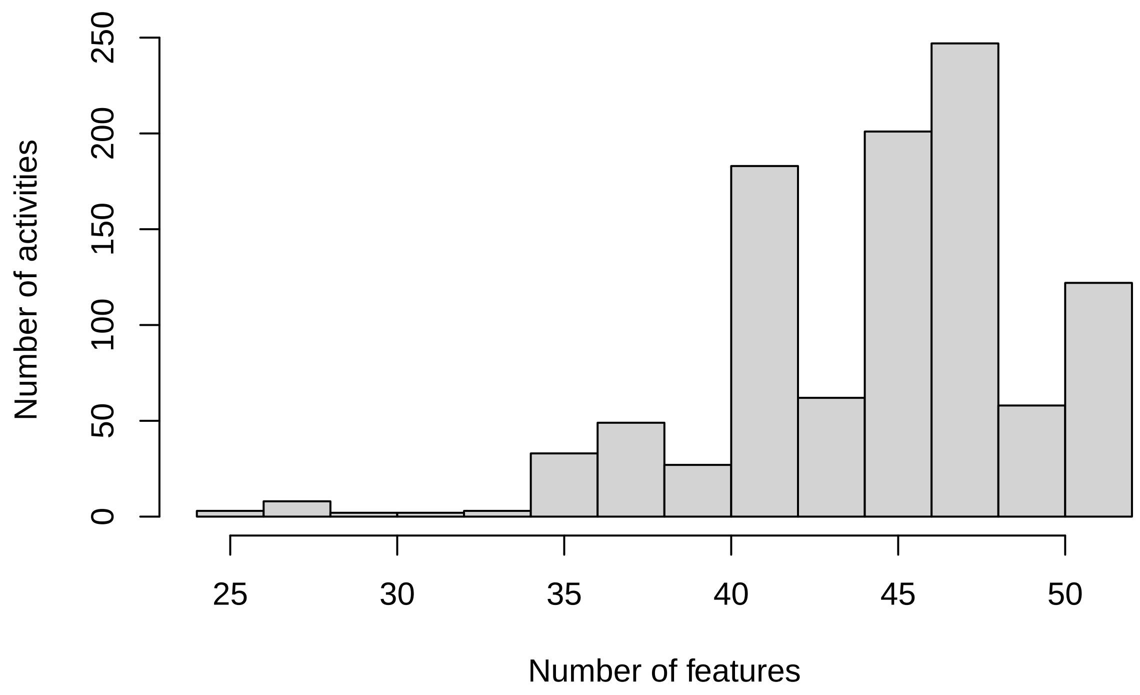
Each activity is a list itself, that looks to be of variable length. Let's take one example to see what is in these lists.

```r
head(activities_list[[1]]) # preview the first activity
```

```
$activityId
[1] 9623843142

$uuidMsb
[1] 1.78373e+18

$uuidLsb
[1] -8.016229e+18

$name
[1] "Champagnole Cyclisme"

$activityType
[1] "cycling"

$userProfileId
[1] 3533530
```

The features of my activities are here, each as one element of the list corresponding to a given activity. We can notice the lists corresponding to the activities are of variable length, let's plot this as a histogram.

```r
# extract number of features for each activity
list_lengths <- sapply(activities_list, function(x){length(x)})
# histogram plot
hist(list_lengths, xlab='Number of features', ylab='Number of activities', main=NULL)
```



This means not every activity has the same number of features. It may not be a big deal since I am going to use only a specific set of features as explained earlier.

Now I would like to have the features in a dataframe format to use them more easily. Let's write a function to do that and preview the dataframe.

```r
# unlist the features and build a dataframe
activities_df <- bind_rows(sapply(activities_list, function(x){
  as_tibble_row(unlist(x))
}))

head(activities_df) # preview of the dataframe
```

```
# A tibble: 6 x 77
  activityId uuidMsb uuidLsb name  activ~1 userP~2 timeZ~3 begin~4 event~5 rule
  <chr>      <chr>   <chr>   <chr> <chr>   <chr>   <chr>   <chr>   <chr>   <chr>
1 9623843142 178372~ -80162~ Cham~ cycling 3533530 124     166341~ 9       publ~
2 9623839838 -11717~ -57298~ Cham~ cycling 3533530 124     166339~ 9       publ~
3 9586411594 -41675~ -64773~ Epag~ cycling 3533530 124     166291~ 4       publ~
4 9623836269 -49558~ -64976~ Haut~ running 3533530 124     166228~ 9       publ~
5 9525333669 -55016~ -58581~ Sevi~ running 3533530 124     166196~ 9       publ~
6 9525333109 -90877~ -68000~ Epag~ cycling 3533530 124     166179~ 9       publ~
# ... with 67 more variables: sportType <chr>, startTimeGmt <chr>,
#   startTimeLocal <chr>, duration <chr>, distance <chr>, elevationGain <chr>,
#   elevationLoss <chr>, avgSpeed <chr>, maxSpeed <chr>, calories <chr>,
#   startLongitude <chr>, startLatitude <chr>, avgFractionalCadence <chr>,
#   maxFractionalCadence <chr>, elapsedDuration <chr>, movingDuration <chr>,
#   deviceId <chr>, minTemperature <chr>, maxTemperature <chr>,
#   minElevation <chr>, maxElevation <chr>, locationName <chr>, ...
```

We obtain a dataframe with 1000 rows, each corresponding to one activity, and 77 columns, each corresponding to one feature. After examining more precisely the dataframe and given the variables I want to work with (sport, distance, positive elevation, start time), I can select the relevant features for my project and transform them a bit to have the relevant classes, names, and units. I will also apply some data quality filters: remove activities with duration lower than 10 min (not relevant for this analysis), elevation gain higher than 5000 meters (clearly GPS tracking errors), remove activities of non relevant sports, remove some NAs.

```r
activities <- activities_df %>%
  # select relevant features
  select(activityId, activityType, startTimeLocal, duration, distance, elevationGain) %>%
  # change class/name/unit
  mutate(activity_id = as.numeric(activityId),
         activity_type = ifelse(activityType=="swimming", "lap_swimming",
                                            ifelse(activityType=="cross_country_skiing_ws",
                                                   "cross_country_skiing", activityType)),
         start_time = as.numeric(startTimeLocal)/1000,
         duration = as.numeric(duration)/1000,
         distance = as.numeric(distance)/100,
         elevation_gain = as.numeric(elevationGain)/100) %>%
  # select transformed features
  select(activity_id, activity_type, start_time, distance, elevation_gain, duration) %>%
  filter(activity_type %in% c("cycling",
                              "running",
                              "open_water_swimming",
                              "trail_running",
                              "cross_country_skiing",
                              "lap_swimming"),
         duration > 600,
         elevation_gain <5000 | is.na(elevation_gain)) %>% # data quality filters
  mutate(activity_type=as.factor(activity_type))

head(activities) # preview the new dataframe
```

```
# A tibble: 6 x 6
  activity_id activity_type start_time distance elevation_gain duration
        <dbl> <fct>              <dbl>    <dbl>          <dbl>    <dbl>
1  9623843142 cycling       1663424031   31070.            538    3688.
2  9623839838 cycling       1663404714   35296.            545    5485.
3  9586411594 cycling       1662918180   62036.           1114    9016
4  9623836269 running       1662292312    7502.             43    2686.
5  9525333669 running       1661974039    5777.             17    2213.
6  9525333109 cycling       1661797204   45311.            621    5876.
```

Here is a clean dataframe that includes 914 activities and the 6 features of interest I selected.

2. Dataset presentation and objective of the project

Let's review the content of each column:

- *activity_id*: a unique identification number of the activity
- *activity_type*: the sport practiced in the activity
- *start_time*: the timecode at which the activity started
- *distance*: the distance covered in the activity, in meters
- *elevation_gain*: the positive elevation in the activity, in meters
- *duration*: the duration of the activity, in seconds - which will be the variable to predict

> **The objective of this project is to give a prediction of activity duration based on simple summary data that can be obtained from a route planner when preparing for a training session.**

I believe this duration prediction could be a useful functionality for activity tracker websites who might propose it to their users.

To achieve the objective, I will develop several algorithms and compare how they perform using the Root Mean Squared Error (RMSE) as defined by :

$$RMSE = \sqrt{\frac{1}{N}\sum_{i}(\hat{y}_i - y_i)^2}$$

where:

- N is the number of activities
- y_i is the true duration of activity i
- $\hat{y}_i$ is the predicted duration

The methods employed to build the algortihm will be described in the **Analysis** section of this report. To perform a final validation of the algorithms, I will use a fraction of the dataset containing all my activities that I will not use in the algorithm development. Let's split the dataset into a **cyo** and a **validation** set. **Validation** will be 10% of the whole dataset.

```
# build validation set as 10% of dataset
set.seed(91, sample.kind="Rounding")
valid_index <- createDataPartition(y = activities$duration,
                                   times = 1, p = 0.1, list = FALSE)
cyo <- activities[-valid_index,]
validation <- activities[valid_index,]
rm(valid_index)
```

The prediction results and the final RMSE using the **validation** set will be included in the **Results** section of this report.

II. Analysis

1. Data preparation

As mentioned in the introduction, the performance of the algorithm will be assessed using the RMSE - let's first define a RMSE function.

```
RMSE <- function(true_values, predicted_values){
  sqrt(mean((true_values - predicted_values)^2))
}
```

Then, the **validation** set is going to be used only for the algorithm's final assessment in the **Results** section. To actually build the algorithm, it is relevant to split the **cyo** set into a train and a test set. The test set will be 20% of the **cyo** set.

```r
# build test set as 20% of cyo set
set.seed(2020, sample.kind="Rounding")
test_index <- createDataPartition(y = cyo$duration, times = 1, p = 0.2, list = FALSE)
train_set <- cyo[-test_index,]
test_set <- cyo[test_index,]
rm(test_index)
```

2. Baseline model

As a baseline model, I will use a very simple and naive model - using the average duration for all activities as a prediction for every activity duration.

$$Y_i = \mu + \varepsilon_i$$

where:

- Y_i is the duration of activity i
- μ is the overall average duration
- ε_i is an error term - the residual for activity i prediction

Let's build this model using the train set, make predictions on the test set and calculate the baseline RMSE.

```r
mu <- mean(train_set$duration) # average duration for all activities

baseline_rmse <- RMSE(test_set$duration, mu)
print(paste("Baseline RMSE is",round(baseline_rmse,0), "seconds"))
```

```
[1] "Baseline RMSE is 3616 seconds"
```

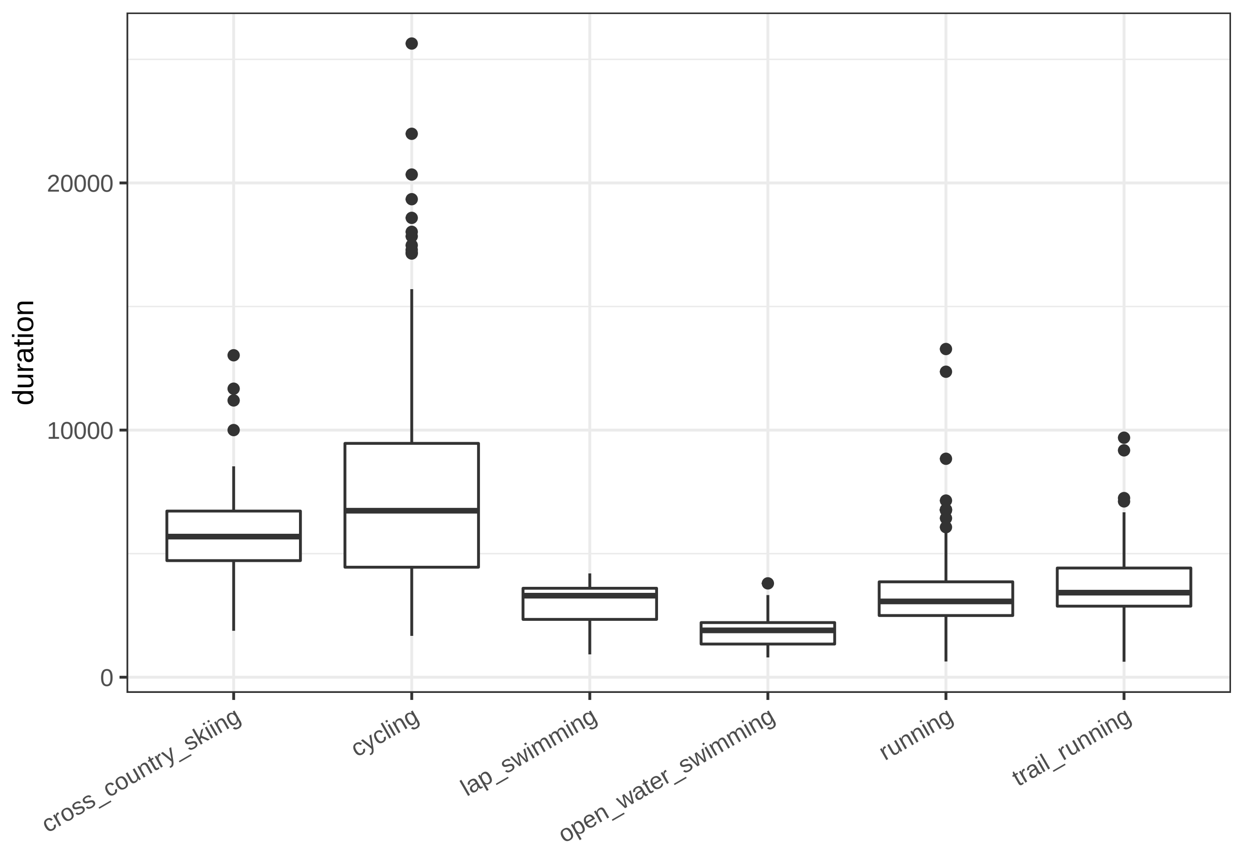
The RMSE is more than 1 hour, which is huge but expected for a model that simple. I will now study other effects and try to improve this RMSE.

3. Models development

3.1 Effect of sport

I will use the column *activity_type* to determine the effect of sport on activity duration. Let's plot duration as a function of sport in a boxplot.

```r
# effect of sport on duration
train_set %>%
  ggplot(aes(activity_type, duration)) +
  geom_boxplot() +
  theme_bw() +
  theme(axis.text.x = element_text(angle=30, hjust=1),
        axis.title.x = element_blank())
```



Sport appears to have an effect on duration, so I propose a first model to estimate activity duration based on a sport bias. With s_i as the sport for activity i, I will fit the following model:

$$Y_i = \mu + b_k + \varepsilon_i \text{ with } k = s_i$$

where b_k the average duration for sport k.

Let's build this model using the train set, make predictions on the test set and calculate this first model RMSE.

```r
# sport bias
b_k <- train_set %>%
  group_by(activity_type) %>%
  summarize(b_k = mean(duration - mu))

# make predictions on test set
pred_1 <- test_set %>%
  left_join(b_k, by='activity_type') %>%
  mutate(pred = mu + b_k) %>%
  pull(pred)

model_1_rmse <- RMSE(test_set$duration, pred_1) # calculate RMSE
print(paste("RMSE with model #1 is",round(model_1_rmse,0),"seconds")) # print results
```

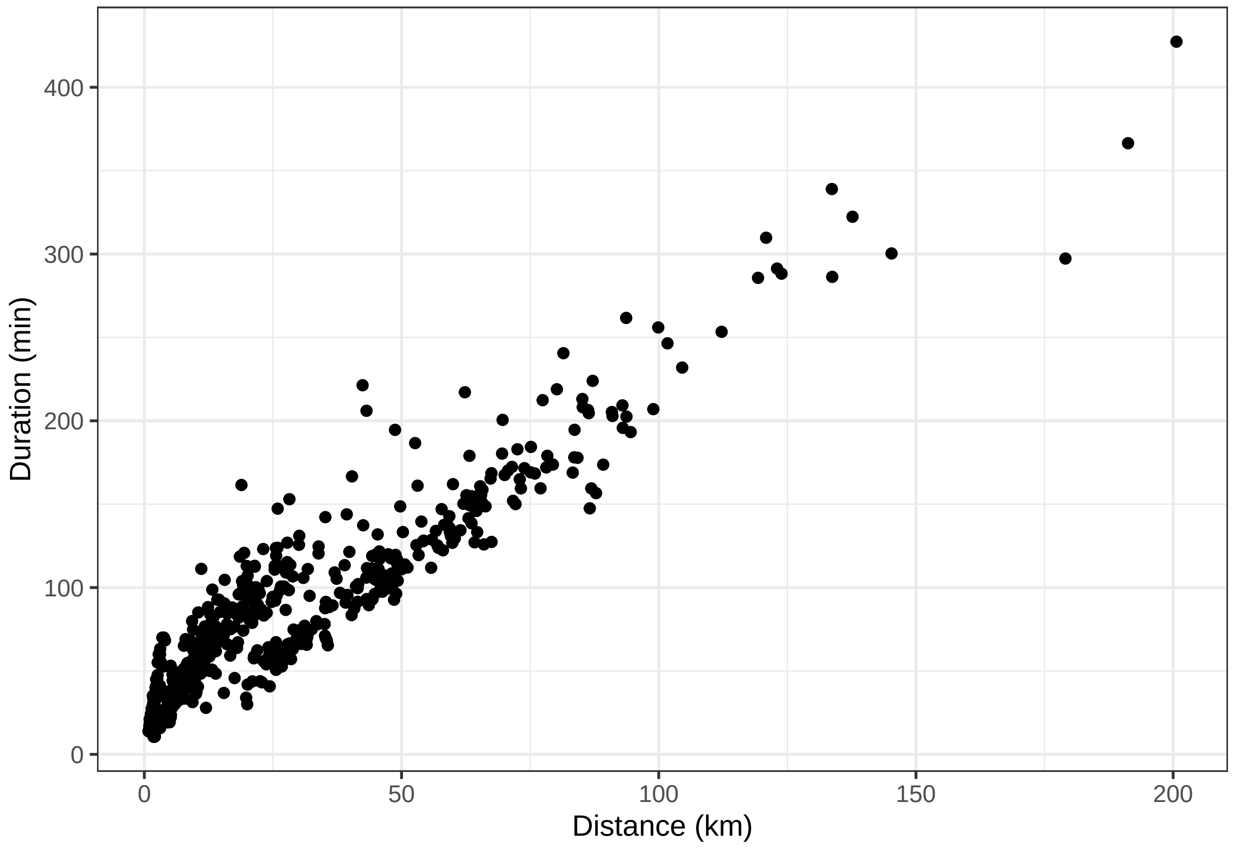
```
[1] "RMSE with model #1 is 2937 seconds"
```

The RMSE improved substantially compared to the baseline model, but the error is still quite high - more than 45 minutes.

3.2 Effect of distance

I will use the column *distance* to study the effect of distance on activity duration. Let's plot duration as a function of distance in a scatterplot. I'll convert distance to kilometers and duration to minutes to have a clearer read.

```r
# Effect of distance on duration
 train_set %>%
  ggplot(aes(distance/1000, duration/60)) + #convert distance to km and duration to min
  geom_point() +
  theme_bw() +
  xlab("Distance (km)") +
  ylab("Duration (min)")
```



From this visualization we can guess a linear effect of distance on duration. As stated in the introduction this is a no brainer, as activity distance and duration obviously increase simultaneously. Anyway, let's build a simple linear regression model to predict duration. With d_i the distance for activity i, the model is:

$$Y_i = p + m \times d_i + \varepsilon_i$$

where p the intercept and m the slope of the linear model.

Let's build this model using the train set, make predictions on the test set and calculate this second model RMSE.

```r
# fit a linear model
fit_dist <- train_set %>%
  lm(duration ~ distance, data=.)

# make predictions on the test set
pred_2 <- predict(fit_dist, test_set)

model_2_rmse <- RMSE(test_set$duration, pred_2) # calculate RMSE
print(paste("RMSE with model #2 is",round(model_2_rmse,0),"seconds")) # print results
```

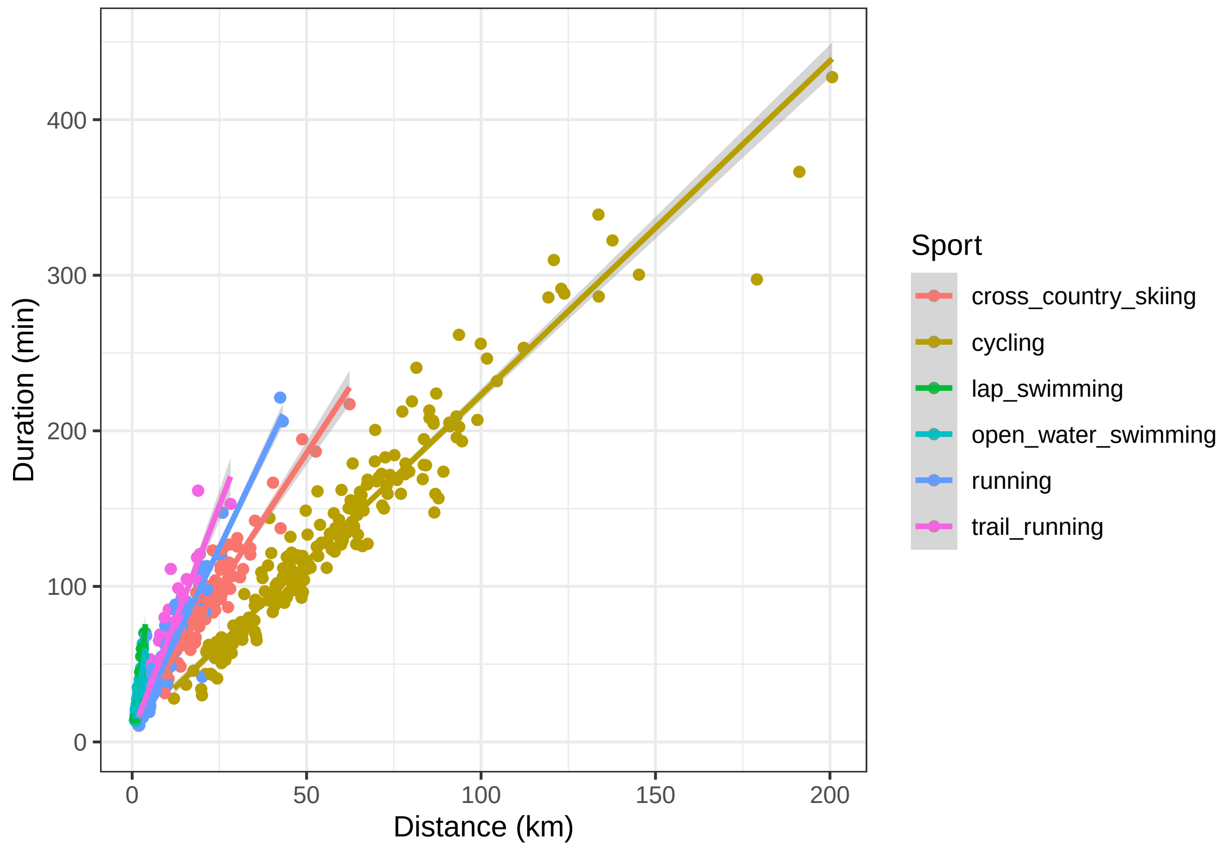
```
[1] "RMSE with model #2 is 1306 seconds"
```

The RMSE improved again greatly, the error is now closer to 20 min. Note that this linear model is equivalent to calculating an average speed for all activities and using this average speed to predict duration based on distance. To further improve the model, it seems reasonable to hypothesize that this average speed is dependent on the sport.

3.3 Interaction between distance and sport

I will now combine the information from the columns *activity_type* and *distance* to study the interaction between distance and sport effects. As stated above, this is equivalent to determine an average speed for each sport. Let's remake the plot of duration vs. distance but adding different colors for the different sports. I will also add a linear trend for each sport.

```r
# Interaction distance/sport
 train_set %>%
  ggplot(aes(distance/1000,
             duration/60, #convert distance to km and duration to min
             color=activity_type)) +
  geom_point() +
  geom_smooth(method = 'lm') + # add a linear tred
  theme_bw() +
  xlab("Distance (km)") +
  ylab("Duration (min)") +
  labs(color='Sport')
```



Different slopes are clearly visible for the different sports. This makes much sense as this slope is basically speed, and average speed is widely different between the sports shown here. Now let's write a model taking this into account. With d_i the distance and s_i the sport for activity i, the model is:

$$Y_i = \sum_{k=1}^{K} a_{i,k}(p_k + m_k \times d_i) + \varepsilon_i \text{ with } a_{i,k} = 1 \text{ if } k = s_i,\ 0 \text{ otherwise}$$

where p_k is the intercept and m_k the slope of the linear model for sport k.

Let's build this model using the train set - I'll build different linear regression models depending on sport. Then let's make predictions on the test set and calculate this third model RMSE.

```r
# extract a vector of sports
sports <- train_set %>%
  select(activity_type) %>%
  mutate(activity_type=as.character(activity_type)) %>%
  distinct(.) %>%
  pull(activity_type)

# fit linear models for each activity type
fit_d_k <- sapply(sports, function(x){
  fit <- train_set %>%
    filter(activity_type==x) %>%
    lm(duration ~ distance, data=.)
  c(fit$coefficients[1], fit$coefficients[2]) # extract coefficients
})
```

```r
# store the coefficients in a tibble
b_d_k <- tibble(activity_type = colnames(fit_d_k),
                     p_k=fit_d_k[1,],
                     m_k=fit_d_k[2,])

# make predictions on the test set
pred_3 <- test_set %>%
  left_join(b_d_k, by='activity_type') %>%
  mutate(pred = p_k+m_k*distance) %>%
  pull(pred)

model_3_rmse <- RMSE(test_set$duration, pred_3) # calculate RMSE
print(paste("RMSE with model #3 is",round(model_3_rmse,0),"seconds")) # print results
```

```
[1] "RMSE with model #3 is 819 seconds"
```

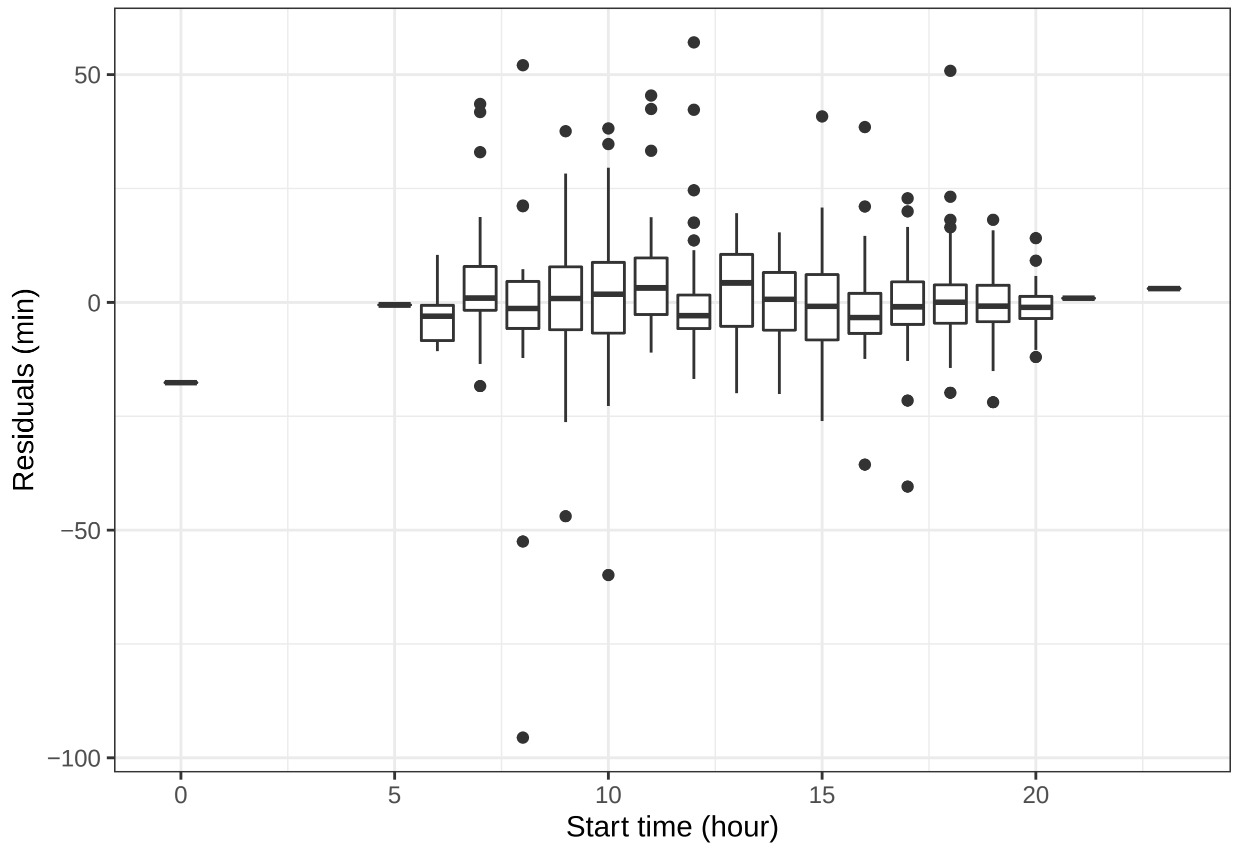
The RMSE improved again substantially, with the error being now lower than 15 minutes. To go further and explore the effect of other variables, I will calculate the residuals of this model #3.

3.4 Effect of date and time

I will use the column *start_time* to investigate the effect of date and time on the residuals of model #3. I thought of 2 possible effects: one linked to the time of the day which the activity happens at, and one linked to the month of the year which the activity happens at.

The hypothesis for the 'time of the day' effect is that I may perform better at certain times of the day and thus duration may be altered. For example, practicing sport in the early morning may be more challenging. Let's group the data on the hour of the day which the activity started at and plot the duration residuals against it.

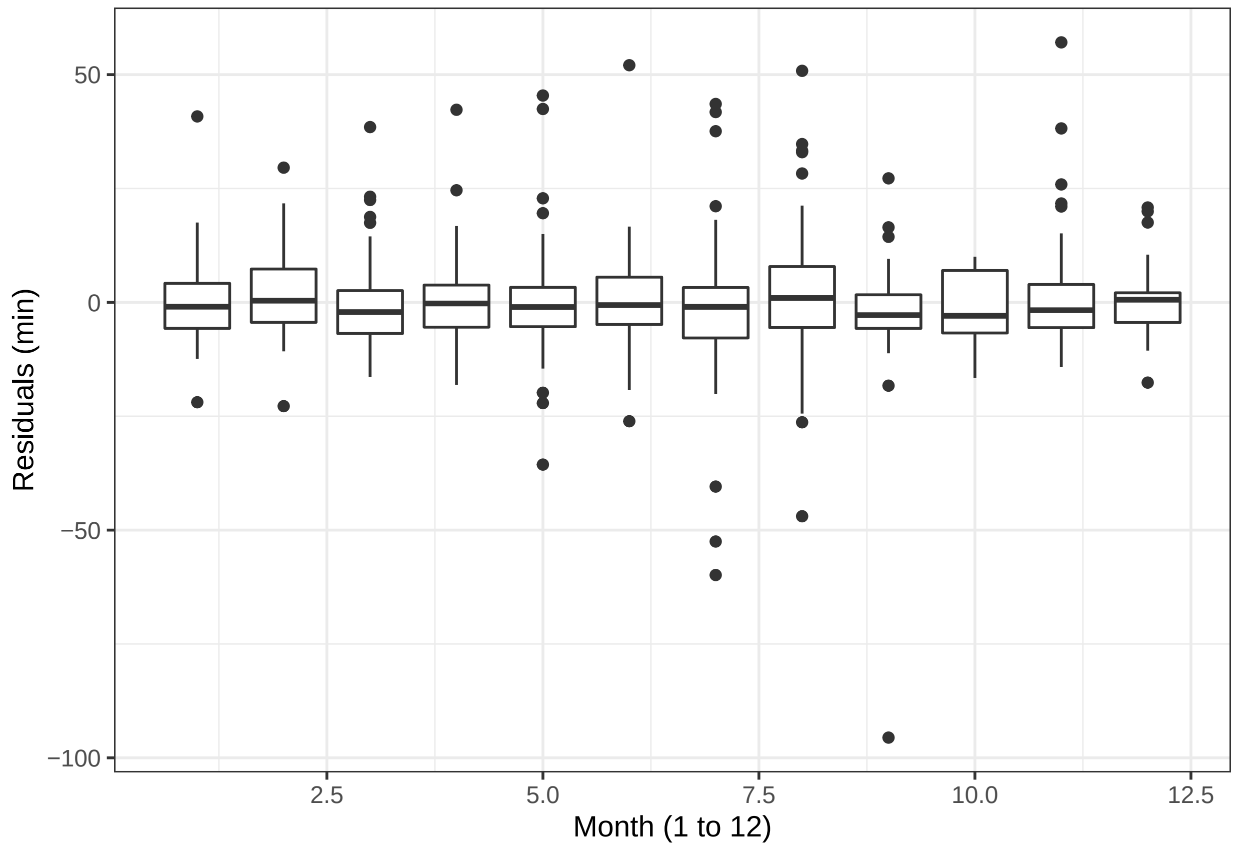
```r
train_set %>%
  left_join(b_d_k, by='activity_type') %>%
  mutate(residuals = duration - (p_k+m_k*distance), # residuals from model #3
         start_hour = hour(round_date(as_datetime(start_time)
                                      ,unit="hour"))) %>% # extract hour from the timecode
  ggplot(aes(start_hour, residuals/60, group=start_hour)) +
  geom_boxplot()+
  xlab("Start time (hour)") +
  ylab("Residuals (min)") +
  theme_bw()
```



The plot is not showing an obvious effect of the time of the day the activity started at. I will not include any effect of this kind in a further version of the model.

The hypothesis for the 'month of the year' effect is that I may have a better shape at certain periods of the year and thus duration may be altered. For example, I train less during late autumn and at easier pace. Let's group the data on the month of the year which the activity started at and plot the duration residuals against it.

```
train_set %>%
  left_join(b_d_k, by='activity_type') %>%
  mutate(residuals = duration - (p_k+m_k*distance), # residuals from model #3
         start_month = month(round_date(as_datetime(start_time),
                                        unit="month"))) %>% # extract month as 1 to 12
  ggplot(aes(start_month, residuals/60, group=start_month)) +
  geom_boxplot() +
  xlab("Month (1 to 12)") +
  ylab("Residuals (min)") +
  theme_bw()
```

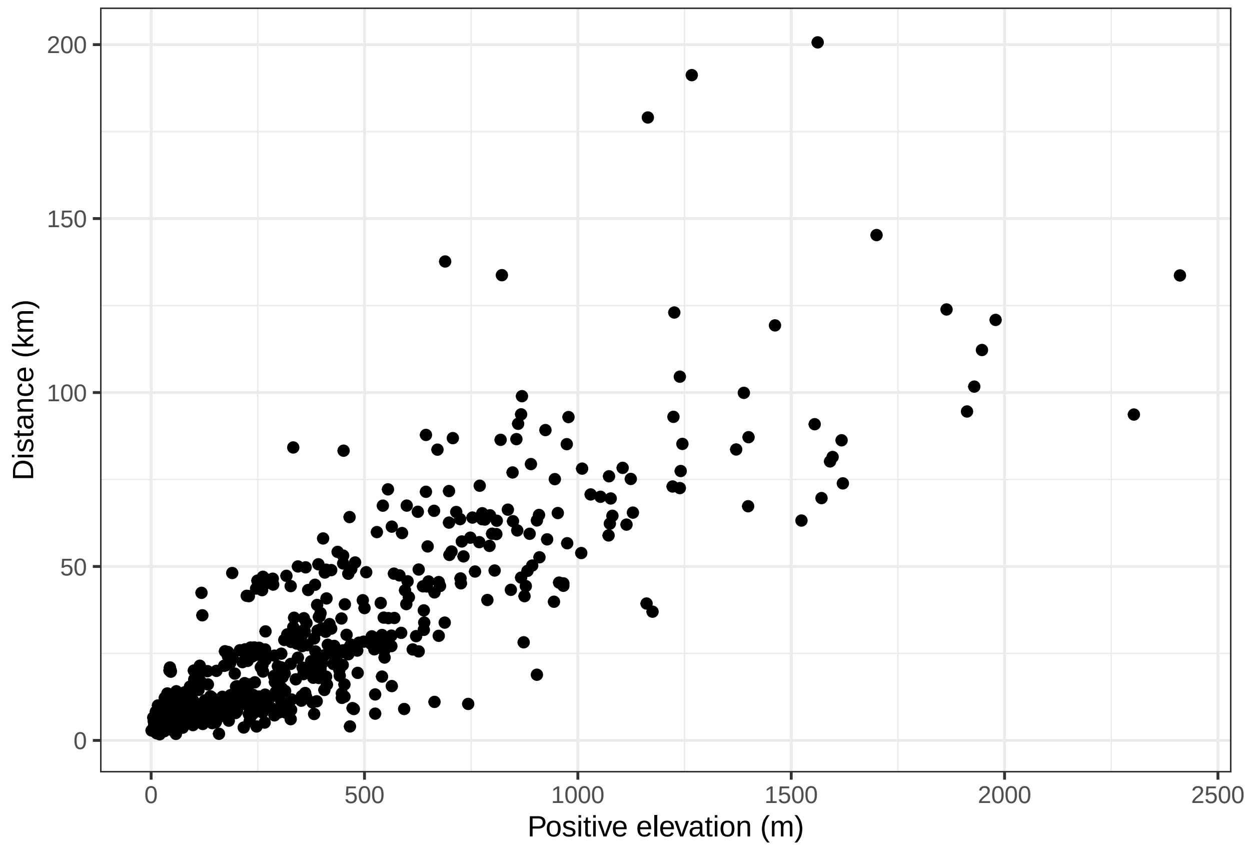


Similarly to the 'time of the day', the plot is not showing any effect of the month of the year which the activity started at. I will neither include any effect of this kind in a further version of the model.

3.5 Effect of elevation gain

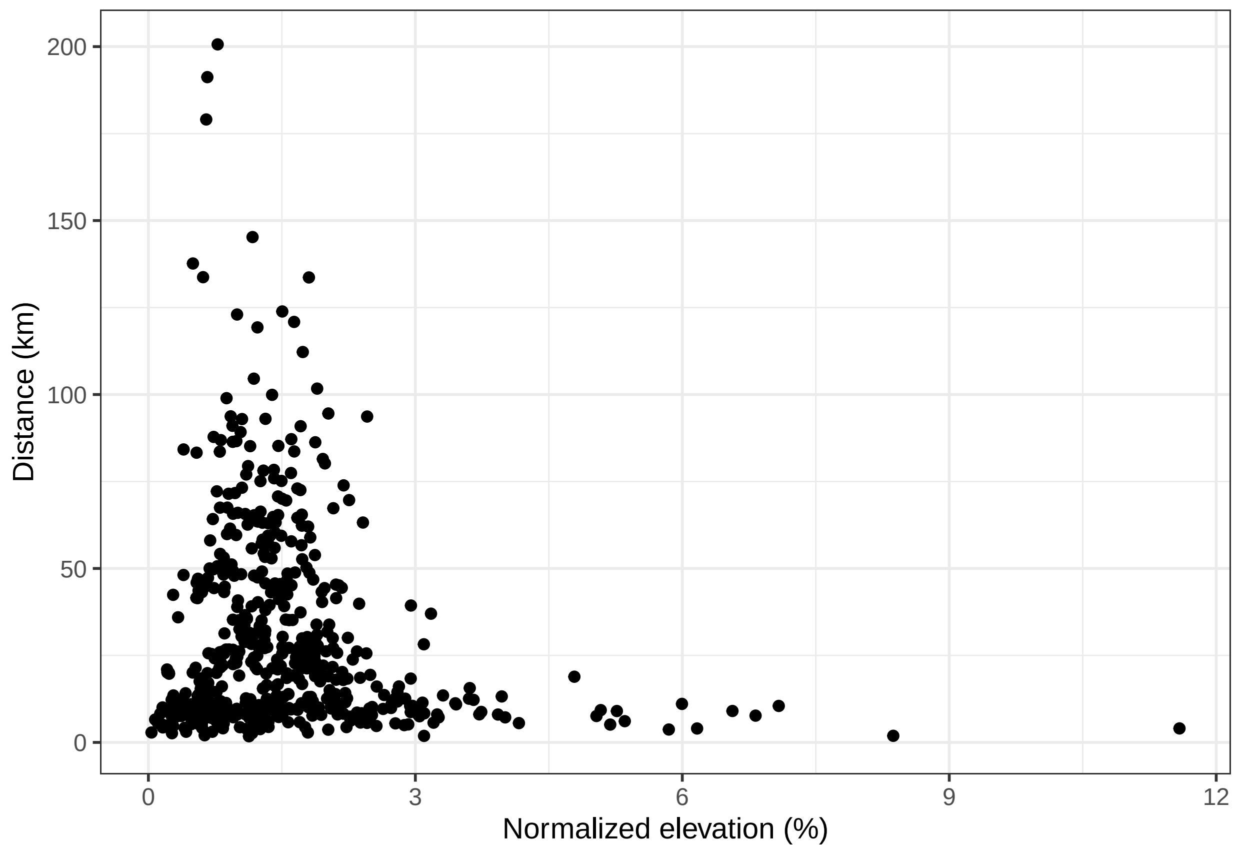
I will use the column *elevation_gain* to investigate the effect of elevation gain on the residuals of model #3. Elevation gain is the cumulative positive elevation recorded on a particular activity. Obviously, we can suppose the higher the distance the higher the elevation gain. This is confirmed by the graph below.

```r
train_set %>%
  ggplot(aes(elevation_gain, distance/1000)) +
  geom_point() +
  xlab("Positive elevation (m)") +
  ylab("Distance (km)") +
  theme_bw()
```



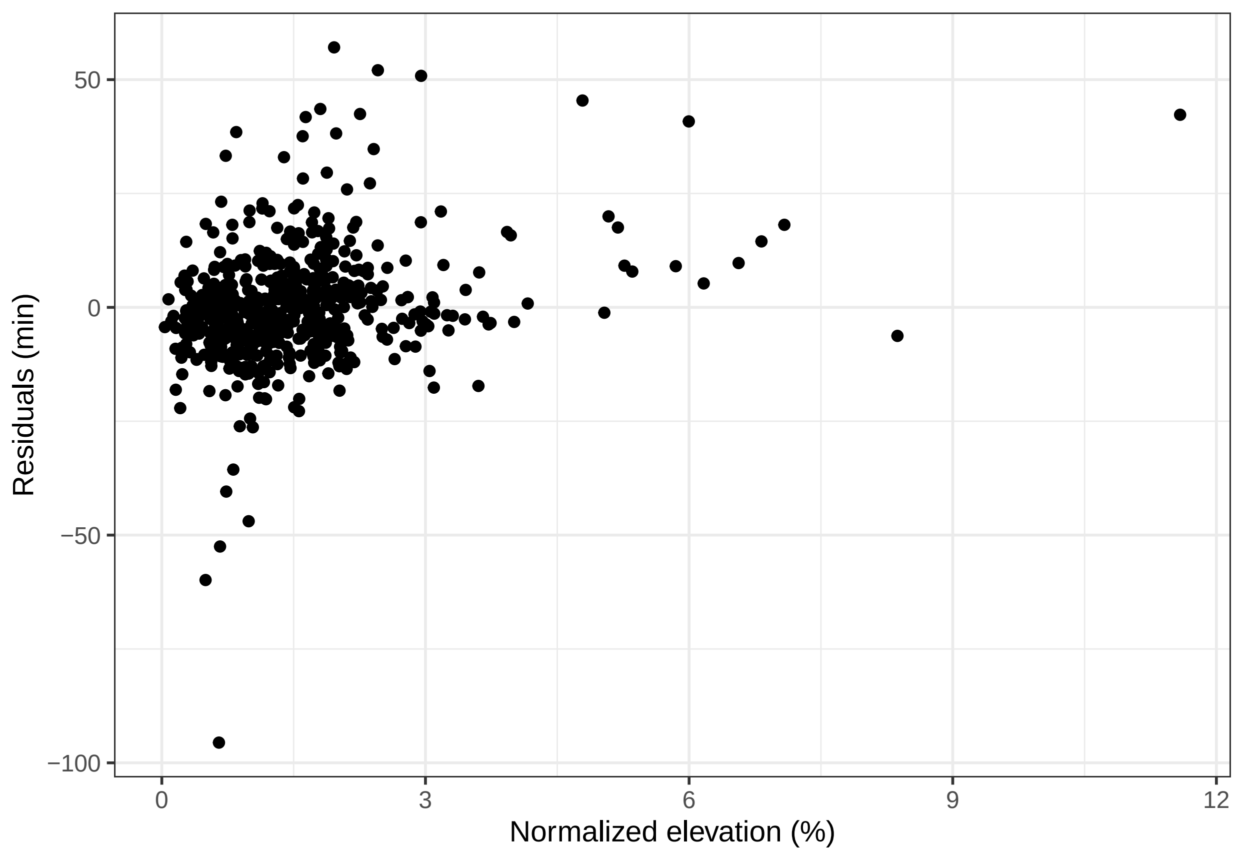
Thus, it makes sense to normalize the elevation gain by distance to get a variable corresponding to an average positive slope on the activity. Let's remake the previous plot with elevation gained normalized to distance.

```
train_set %>%
  mutate(elevation_norm = elevation_gain/distance) %>%
  ggplot(aes(elevation_norm*100, distance/1000)) +
  geom_point() +
  xlab("Normalized elevation (%)") +
  ylab("Distance (km)") +
  theme_bw()
```



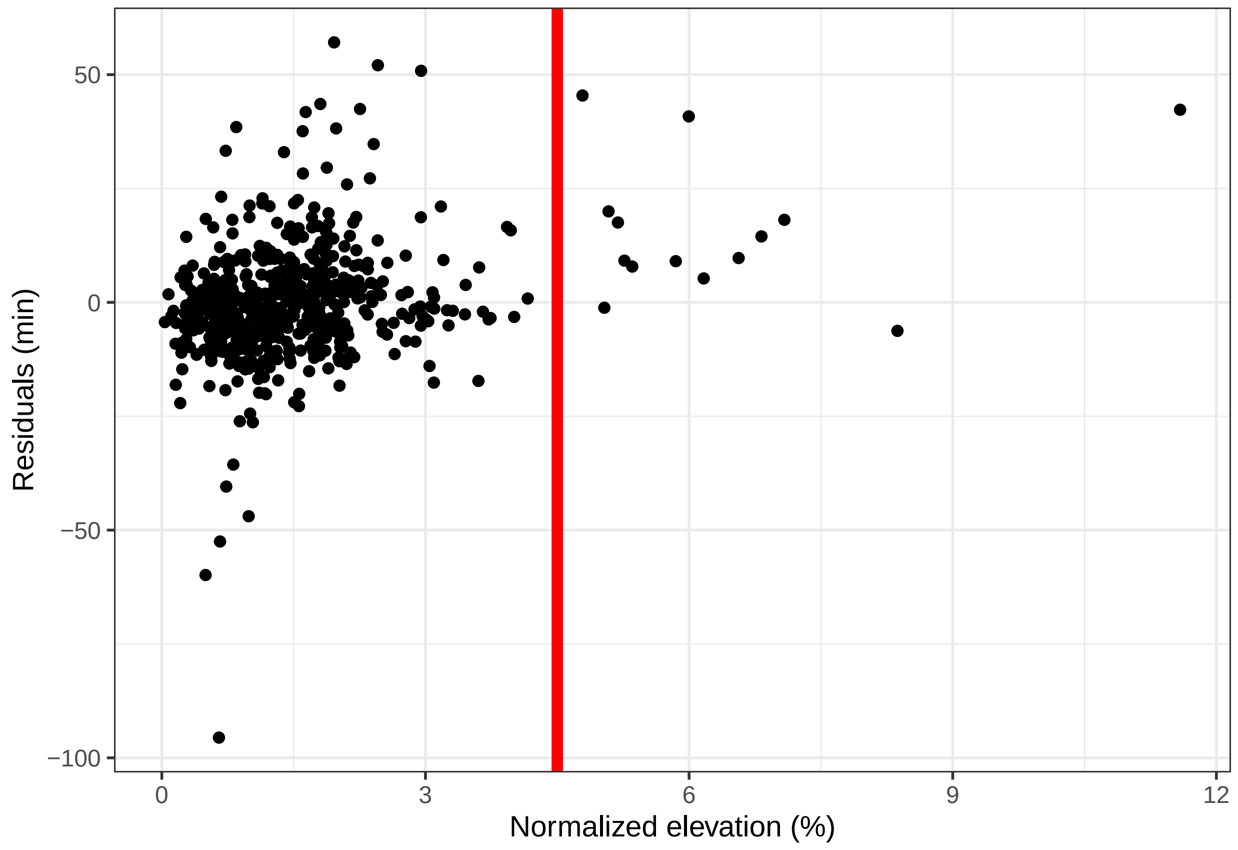
We can see a large variety of distances for similar normalized elevation gains, but also some really high normalized elevation gains for quite short distances, which is equivalent to a steep average positive slope on the activity. I hypothesize that the higher this average slope (ie. normalized elevation gain), the higher the duration since it will mean a particularly hilly and difficult route.
Let's plot the residuals of model #3 against normalized elevation gain.

```
train_set %>%
  left_join(b_d_k, by='activity_type') %>%
  mutate(residuals = duration - (p_k+m_k*distance)) %>%
  mutate(elevation_norm = elevation_gain/distance) %>%
  ggplot(aes(elevation_norm*100, residuals/60)) +
  geom_point() +
  xlab("Normalized elevation (%)") +
  ylab("Residuals (min)") +
  theme_bw()
```



From the graph above, it appears that there is no monotonous relationship between normalized elevation gain and the residuals from model #3, but rather a threshold elevation value above which the residuals looks to be more frequently positive.
Let's illustrate this threshold by a red line on the plot.

```
train_set %>%
  left_join(b_d_k, by='activity_type') %>%
  mutate(residuals = duration - (p_k+m_k*distance)) %>%
  mutate(elevation_norm = elevation_gain/distance) %>%
  ggplot(aes(elevation_norm*100, residuals/60)) +
  geom_point() +
  xlab("Normalized elevation (%)") +
  ylab("Residuals (min)") +
  theme_bw() +
  geom_vline(xintercept = 4.5, color='red', size = 2)
```



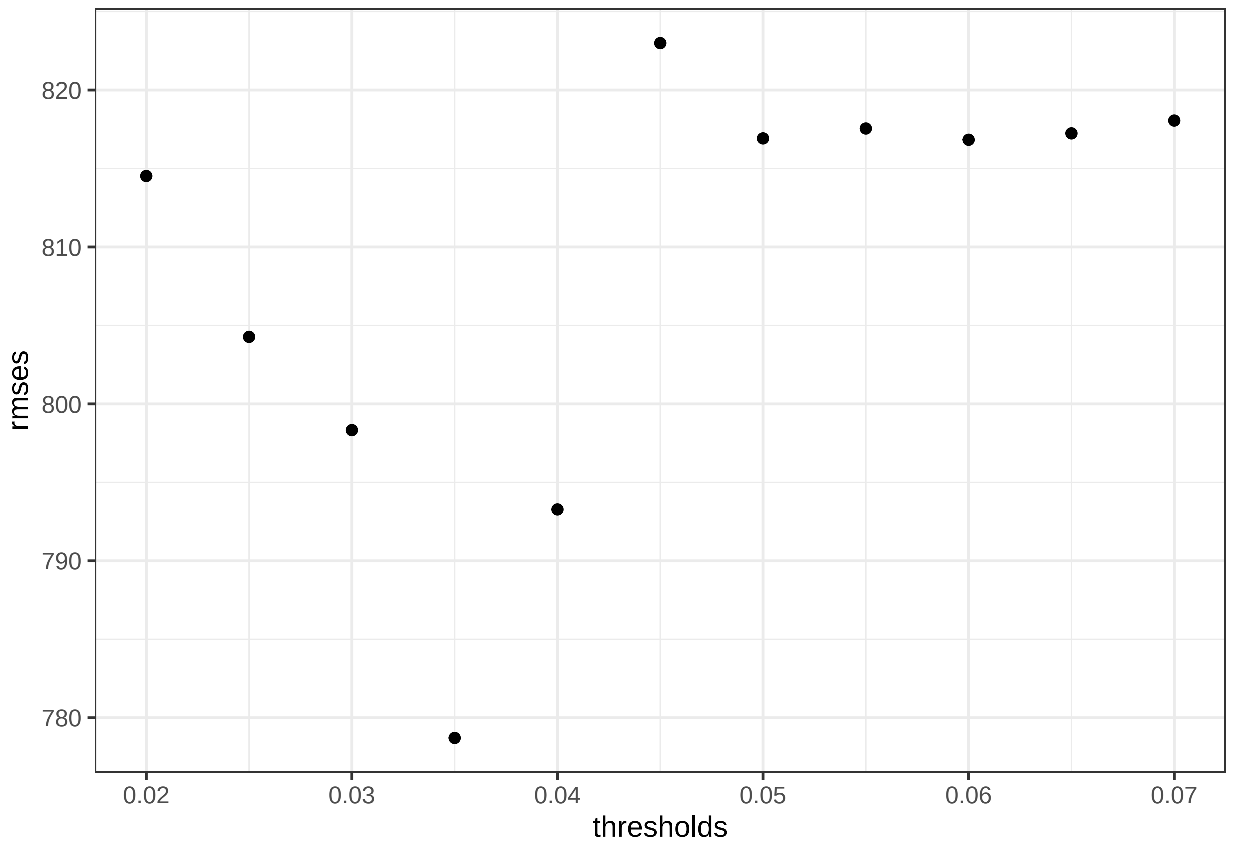
Let's add a conditional elevation effect to the previous model.
With g_i the normalized elevation gain for activity i, the model is:

$$Y_i = \sum_{k=1}^{K} a_{i,k}(p_k + m_k \times d_i) + h_{i,\tau} \times b_\tau + \varepsilon_i \text{ with } a_{i,k} = 1 \text{ if } k = s_i,\ 0 \text{ otherwise, and } h_{i,\tau} = 1 \text{ if } g_i > \tau,\ 0 \text{ otherwise}$$

where τ is the normalized elevation threshold above which it is relevant to apply the effect, and b_τ the average duration residual for all activities with a normalized elevation gain above τ.

This threshold τ is a tuning parameter, and I will use cross validation to select it as the value that minimizes the RMSE. Let's apply a range of τs on the calculation of b_τ and plot the obtained RMSEs.

```
taus <-  seq(0.02,0.07,0.005)

rmses <- sapply(taus, function(t){
b_tau <- train_set %>%
  left_join(b_d_k, by='activity_type') %>%
  mutate(elevation_norm = elevation_gain/distance) %>%
  filter(elevation_norm > t) %>% # conditional effect
  summarize(b_tau = mean(duration - (p_k+m_k*distance))) %>%
  pull(b_tau)

# make predictions
pred_4 <- test_set %>%
  left_join(b_d_k, by='activity_type') %>%
  mutate(elevation_norm = elevation_gain/distance,
         pred = ifelse(elevation_norm < t | is.na(elevation_norm),
                        p_k + m_k*distance,
                        p_k + m_k*distance + b_tau)) %>%
  pull(pred)

return(RMSE(test_set$duration, pred_4))
})

# plot RMSE vs tau
tibble(thresholds=taus, rmses=rmses) %>%
  ggplot(aes(thresholds, rmses)) +
  geom_point()+
  theme_bw()
```



We can then extract the value of τ that minimizes the RMSE and the minimum RMSE for this fourth model.

```
tau <- taus[which.min(rmses)] # optimal tau
model_4_rmse <- min(rmses) # minimum RMSE
print(paste("Optimal tau is", tau)) # print tau
```

```
[1] "Optimal tau is 0.035"
```

```
print(paste("RMSE with model #4 is",round(model_4_rmse,0),"seconds")) # print RMSE
```

```
[1] "RMSE with model #4 is 779 seconds"
```

The RMSE improved a bit further, the error is now around 13 minutes. This represents an error of about 15 % of the average activity duration in the train set.

Therefore, this analysis allowed me to design 4 models that I will now be able to apply to the **validation** set in order to assess their final performance.

III. Results

Let's apply all the models I studied in the **Analysis** section and compare them in a table.

```
# Baseline model - append _f suffix for 'final'
# Simple mean
mu_f <- mean(cyo$duration)
baseline_rmse_f <- RMSE(validation$duration, mu_f)


# Model 1 - append _f suffix for 'final'
# Sport bias
b_k_f <- cyo %>%
  group_by(activity_type) %>%
  summarize(b_k_f = mean(duration - mu_f))

pred_1_f <- validation %>%
  left_join(b_k_f, by='activity_type') %>%
  mutate(pred = mu + b_k_f) %>%
  pull(pred)

model_1_rmse_f <- RMSE(validation$duration, pred_1_f)

# Model 2 - append _f suffix for 'final'
# simple linear regression
fit_dist_f <- cyo %>%
  lm(duration ~ distance, data=.)

pred_2_f <- predict(fit_dist_f, validation)

model_2_rmse_f <- RMSE(validation$duration, pred_2_f)

# Model 3 - append _f suffix for 'final'
# linear regressions for each sport
fit_d_k_f <- sapply(sports, function(x){
  fit <- cyo %>%
    filter(activity_type==x) %>%
    lm(duration ~ distance, data=.)
  c(fit$coefficients[1], fit$coefficients[2])
})

b_d_k_f <- tibble(activity_type = colnames(fit_d_k_f),
                     p_k_f=fit_d_k_f[1,],
                     m_k_f=fit_d_k_f[2,])

pred_3_f <- validation %>%
  left_join(b_d_k_f, by='activity_type') %>%
  mutate(pred = p_k_f+m_k_f*distance) %>%
  pull(pred)

model_3_rmse_f <- RMSE(validation$duration, pred_3_f)

# Model 4 - append _f suffix for 'final'
# add a conditional elevation effect
b_tau_f <- cyo %>%
  left_join(b_d_k_f, by='activity_type') %>%
  mutate(elevation_norm = elevation_gain/distance) %>%
  filter(elevation_norm > tau) %>%
  summarize(b_tau_f = mean(duration - (p_k_f+m_k_f*distance))) %>%
  pull(b_tau_f)

pred_4_f <- validation %>%
  left_join(b_d_k_f, by='activity_type') %>%
  mutate(elevation_norm = elevation_gain/distance,
         pred = ifelse(elevation_norm < tau | is.na(elevation_norm),
                       p_k_f + m_k_f*distance,
                       p_k_f + m_k_f*distance + b_tau_f)) %>%
  pull(pred)

model_4_rmse_f <- RMSE(validation$duration, pred_4_f)

# Results in a table
rmse_results <- tibble(model = c("Baseline", "Model 1", "Model 2", "Model 3", "Model 4"),
                       method=c("Simply the mean",
                                "Sport effect",
                                "Simple distance linear regression",
                                "Distance linear regressions by sport",
                                "Distance by sport + elevation conditional effect"),
                           RMSE = c(round(baseline_rmse_f,0),
                                    round(model_1_rmse_f,0),
                                    round(model_2_rmse_f,0),
                                    round(model_3_rmse_f,0),
                                    round(model_4_rmse_f,0)))

rmse_results %>% knitr::kable()
```

model	method	RMSE
Baseline	Simply the mean	3808
Model 1	Sport effect	2901
Model 2	Simple distance linear regression	1041
Model 3	Distance linear regressions by sport	660
Model 4	Distance by sport + elevation conditional effect	632

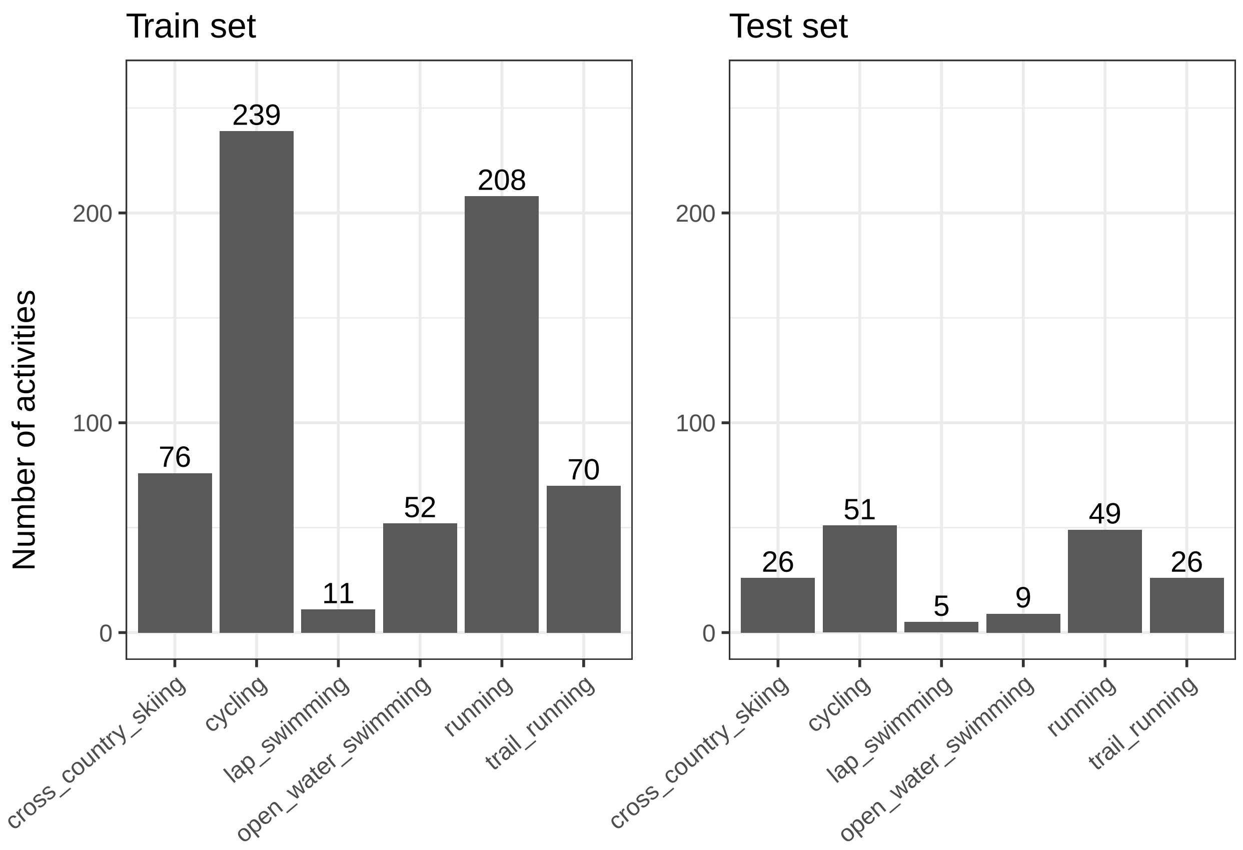
> **We can see the lowest achieved RMSE is 632 seconds which is about 10 minutes. It was achieved with Model 4 corresponding to Distance by sport + elevation conditional effect.**

This RMSE represents about 12 % of mean duration for all my activities. This looks like a quite acceptable rough estimation of a new activity duration when planning a training session.

What is interesting to note is that the improvement between the naive baseline model and the final most performing model was substantial - RMSE was divided by more than 6 - even though this model is quite simple. This means that most of the information needed to estimate an activity duration is actually included in a few simple features that can be extracted from a route planner software.
Let's mention 2 limitations for this project:

- I used only a one-fold cross-validation to check the performance of the linear regressions and select the optimal τ for the elevation effect. It may have been interesting to use multiple-folds cross-validation to obtain better tuned models.
- My activities log is not so large and the datasets I used were of somehow limited size. For example, some sports were poorly represented in the train and test sets when stratifying by sport - this is especially true for swimming activities (see graph below). This may lead to overtraining and more data would be needed to better tune the models.

```r
# plot activity count by sport for train_set
p1 <- train_set %>%
  group_by(activity_type) %>%
  summarize(count=n()) %>%
  ggplot(aes(activity_type, count)) +
  geom_col()+
  geom_text(aes(label=count), vjust=-.3)+
  theme_bw() +
  ggtitle("Train set") +
  ylim(c(0,260))+
  ylab(element_blank())+
  theme(axis.text.x = element_text(angle=40, hjust=1),
        axis.title.x = element_blank())

# plot activity count by sport for test_set
p2 <- test_set %>%
  group_by(activity_type) %>%
  summarize(count=n()) %>%
  ggplot(aes(activity_type, count)) +
  geom_col()+
  geom_text(aes(label=count), vjust=-.3)+
  theme_bw() +
  ggtitle("Test set")+
  ylim(c(0,260))+
  ylab(element_blank())+
  theme(axis.text.x = element_text(angle=40, hjust=1),
        axis.title.x = element_blank())

# plots next to each other
grid.arrange(p1,p2, nrow=1, left="Number of activities")
```



As perspectives to further improve the relevance of this project, a first idea is obviously to continue to log my activities and adjust the current models as more data flow in. Then, it may be worth looking into non-linear effects (especially for distance) as one hypothesis can be that the higher the distance, the lower the average speed and thus the relationship might have more of a logarithmic appearance. This was not obvious from the distance vs. duration plot (see the **Analysis** section, **3.2**), maybe due to the lack of data for higher distances. Finally, the models can be enriched from other users' data and collaborative filtering techniques may be used to make predictions based on a higher amount of data coming from users that are determined to be similar to each other.

IV. Conclusion

The objective of this project was to give predictions of the duration of future sport activities based on my personal activities log. The dataset analysis revealed that there were main effects of sport and distance on activity duration, and that elevation gain could be included as a conditional effect. I designed 4 models based on these effects with increasing complexity, and the best result obtained on the **validation** set is a RMSE of 632 seconds - around 10 minutes. I would think of gathering more data from my upcoming activities and also include activities from users similar to me (potentially determined by collaborative filtering) as further directions to improve the present models.