Predicting Heart Rates of Sport Activities Using Machine Learning Models

Zhuoyi Jin[1], **Yuqi Yang**[2], **Yukai Ma**[3]

[1,3] Development Engineering Department, UC Berkeley
[2] Industrial Engineering & Operations Research Department, UC Berkeley

Abstract

Abstract Activity logs collected from wearable devices provide promising data source for various fields of study. In this paper, we introduce machine learning methods to analyze and predict the max heart rates from the activity logs. We have performed the Linear Regression and Random Forest model to monitor and forecast the user's maximum heart rate, and have examined the risk group derived from a K-means Clustering model. We have discovered that using Random Forest classification yielded us significantly higher accuracy score compared to using Linear Regression. In terms of the user research, we have found that lack of physical activities is correlated to maximum heart rate anomalies.

Keywords: *Heart Rate; Prediction; Machine Learning; User Research*

1. Introduction

Endomondo is a social fitness network that allowed users to track their fitness and health statistics with a mobile application and website. This is done based on both user variables and riding circumstances.[2] The data set from Endomondo collect various types of data, including health-related measurements (e.g. heart rate) and contextual measurements (e.g. location, altitude, activity type). The goal of this research is to detect and predict the abnormal heart rate during exercises that can be further developed into a system delivering heart rate alarms and also providing users with route recommendation.

In this paper, we can define two problems. The first is the problem of the general method to conduct heart rate prediction. The second is to study the behavior and the contexts of user groups. This study, therefore, incorporates both supervised and unsupervised learning tasks. We will use linear regression, random forest, k-means clustering, and PCA to find a solution to these problems.

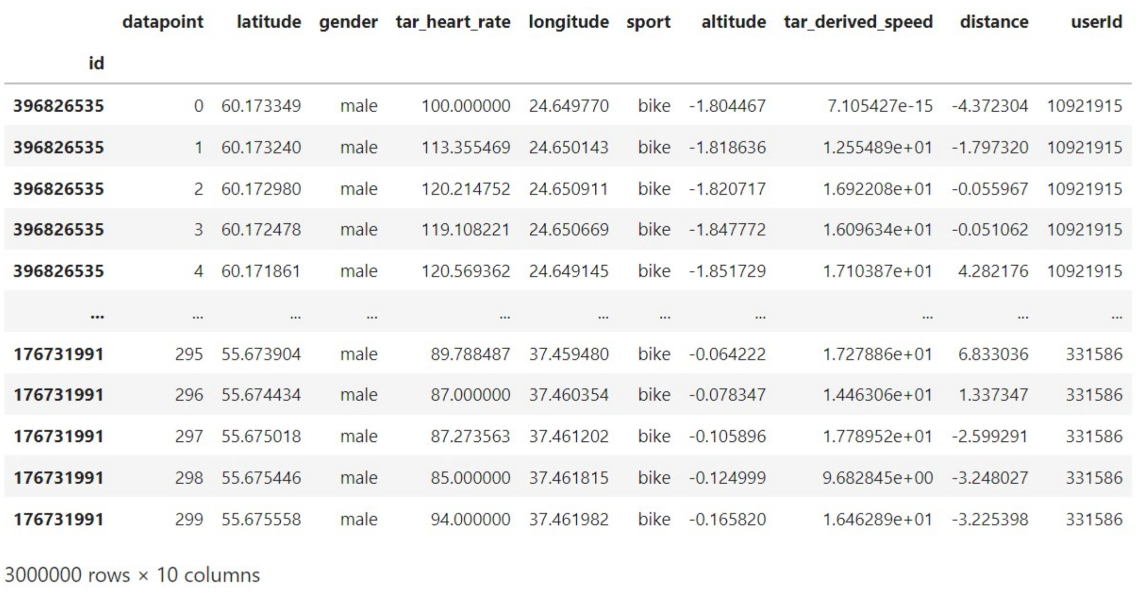
The dependent variables we are interested in is the maximum heart rate. The variables consist of both internal variables, such as gender, and external variables such as altitude difference which is the average slope of a ride or ride segment. First, there will be an exploration of background information and related works. After this, the methodology is defined. This is followed by the experiments and their analysis. Finally, there is a discussion of the applications and limitations.

2. EDA and Data Wrangling

The section includes descriptive statistics, distribution, and simple relationships among different variables, in terms of tables, scatter plots, violin plots, and box plots etc. This section is aimed at providing an overall understanding of the dataset and exploratory data analysis.

2.1. Descriptive statistics

This part includes descriptive statistics. In order to facilitate the exploration, the index and several columns of the original data are adjusted.Figure 1 displays partial columns of the original data. The "id" column is set as the index, which represent a unique sports record. The index in original data is renamed as "data point", each of which represents a time node and is not uniformly distributed. The data includes 3000000 in total, includes 100000 sports records, namely 10000 unique "id". Each "id" has 300 data points, recording multiple information at that moment, such as coordinate, speed, heart rate, gender, sports category and user ID.



	datapoint	latitude	gender	tar_heart_rate	longitude	sport	altitude	tar_derived_speed	distance	userId
id										
396826535	0	60.173349	male	100.000000	24.649770	bike	-1.804467	7.105427e-15	-4.372304	10921915
396826535	1	60.173240	male	113.355469	24.650143	bike	-1.818636	1.255489e+01	-1.797320	10921915
396826535	2	60.172980	male	120.214752	24.650911	bike	-1.820717	1.692208e+01	-0.055967	10921915
396826535	3	60.172478	male	119.108221	24.650669	bike	-1.847772	1.609634e+01	-0.051062	10921915
396826535	4	60.171861	male	120.569362	24.649145	bike	-1.851729	1.710387e+01	4.282176	10921915
...	...	...	...	...	...	...	...	...	...	...
176731991	295	55.673904	male	89.788487	37.459480	bike	-0.064222	1.727886e+01	6.833036	331586
176731991	296	55.674434	male	87.000000	37.460354	bike	-0.078347	1.446306e+01	1.337347	331586
176731991	297	55.675018	male	87.273563	37.461202	bike	-0.105896	1.778952e+01	-2.599291	331586
176731991	298	55.675446	male	85.000000	37.461815	bike	-0.124999	9.682845e+00	-3.248027	331586
176731991	299	55.675558	male	94.000000	37.461982	bike	-0.165820	1.646289e+01	-3.225398	331586

3000000 rows × 10 columns

Figure 1. Partial data after adjustment

In terms of users' information, the data includes 100 different user ID, the numbers of male and female are 90 and 10 respectively, while 3 users' gender is unknown.

Table 1. Number of users by gender

Gender	Number of users
Male	90
Female	10
unknown	3

The data includes 18 different kinds of sports among 10000 records. Table 2 displays their occurrence, where bike, run and mountain bike are three sports with the highest occurrence. In the follow-up EDA, the sports with poor representation will be eliminated for convenience, namely "basketball"," skate"," soccer"," tennis" and "weight training" are dropped since their occurrence is less than 3.

Table 2. Number of records by sports

Sport	Number of records	Sport	Number of records
basketball	1	bike	4766
bike (transport)	209	circuit training	8
core stability training	14	cross-country skiing	16
hiking	10	indoor cycling	27
kayaking	8	mountain bike	712
orienteering	163	rowing	11
run	4013	skate	1
soccer	2	tennis	1
walk	36	weight training	2

2.2. Maximum heart rate and Qualitative data

This part explores the most explanatory qualitative data to predict maximum heart rate. Initial, we plot the distribution of our target prediction variable. The Figure 2 shows the distribution of maximum heart rate of all sports records. The distribution is right-skewed, while the largest data point lies on 200, the smallest one lies on 74, and the median is around 160.

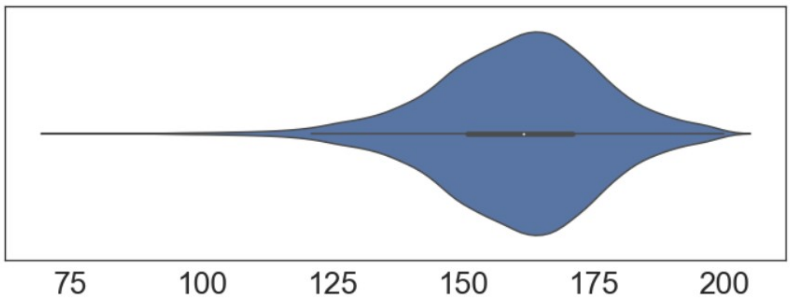


Figure 2. Distribution of Max heart rate

To explore the relationship between user and maximum heart rate, Figure 3 shows the distribution of each users' average maximum heart rate of all his/her sports records. Unlike Figure 2, this figure is symmetric and more centralized, whose maximum, median and minimum are 193, 164 and 129 respectively.

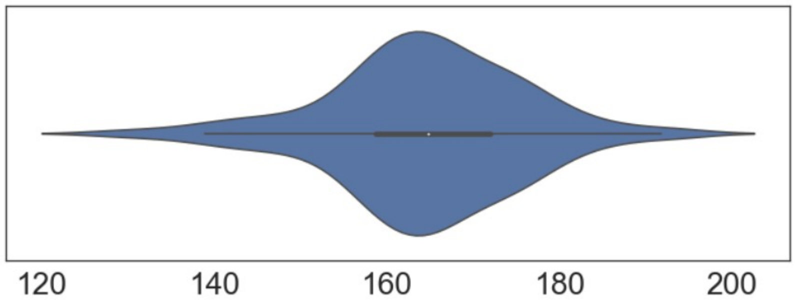


Figure 3. Distribution of Average Max heart rate

To explore the relationship between sports category and maximum heart rate, Figure 4 shows the distributions of maximum heart rate in different sports. It is obvious that different sports have different range and distribution, while walk has the lowest median and kayaking has the highest median.

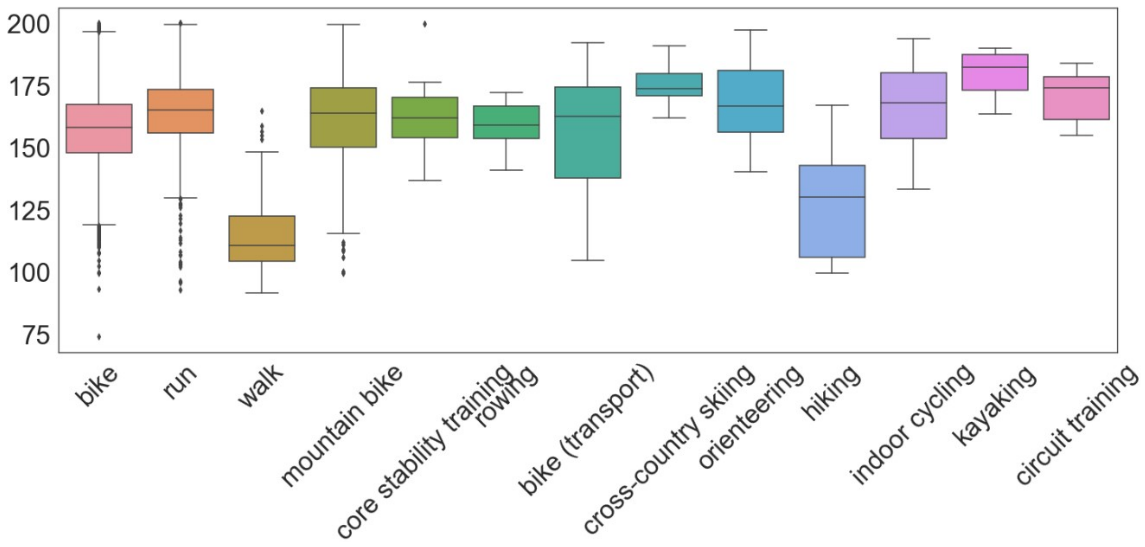


Figure 4. Distribution of Max heart rate by sports

Intuitively, we infer there are great differences between males and females in sports, since competitive sports are separated by genders. However, distinct from our expectation, Figure 5 shows that distributions of male's and female's maximum heart rate are quite consistent, both of which centered at 160. The female's distribution has a tiny left "long tail", this may relate with "yoga", a low intensity aerobic exercise. In short, gender could not explain the variance among maximum heart rate.

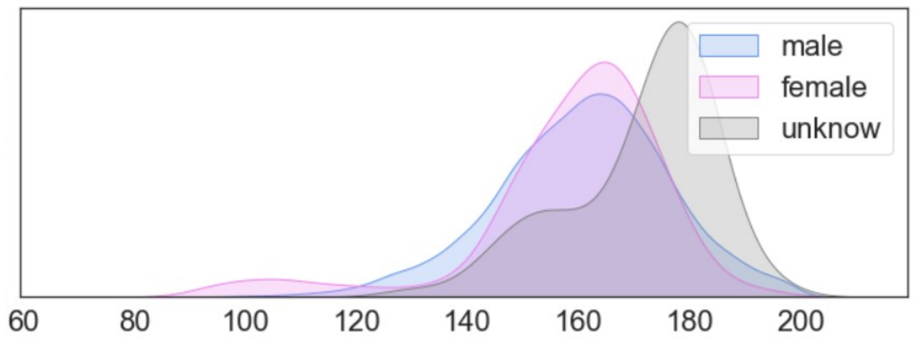


Figure 5. Distribution of Max heart rate by sports

Even though we can not rule out the possibility that there is collinearity between "user ID" and "sports", the difference between Figure 2 and Figure 3, as well as distinct distribution in Figure 4 and similar distribution in Figure 5 imply that maximum heart rate is more related with sports category.

2.3. Maximum heart rate and Motion information

This part involves the relationships between maximum heart rate and motion information, such as speed, coordinate and time. Speed is probably an important factor influencing the maximum heart rate. This part explore their relationship using two partial data. To eliminate the disturbance, "circuit training" and "core stability training", two in situ sports, are dropped. Figure 6 is a scatter plot between max speed and maximum heart rate among all the sports record in adjusted data. Although the figure has been scaled and numerous outliers have been discarded, it still does not show any pattern.

To eliminate the disturbance, here we only includes three most frequent sports:"run"," bike","

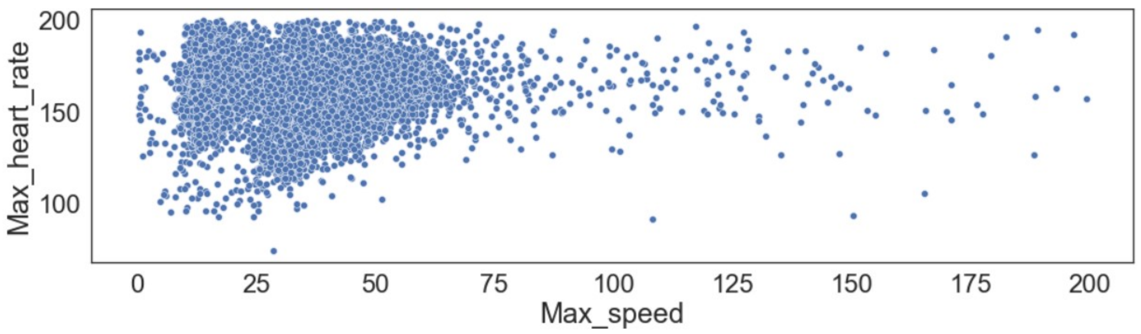


Figure 6. Scatter plot of max speed and max heart rate

mountain bike". But Figure 7 still does not show any pattern, so we imply that in given data, "speed" is not a dominant factor to predict maximum heart rate.

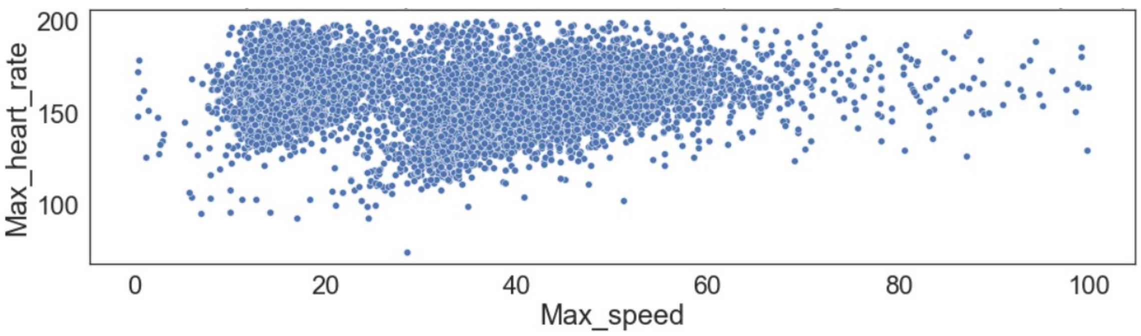


Figure 7. Scatter plot of max speed and max heart rate (Top 3 frequent sports only)

The relation between drop and maximum heart rate is similar. Figure 8 is a scatter plot between drop, namely max altitude minus min altitude, and maximum heart rate. Even though only includes three most frequent sports, there is no pattern between two variables.

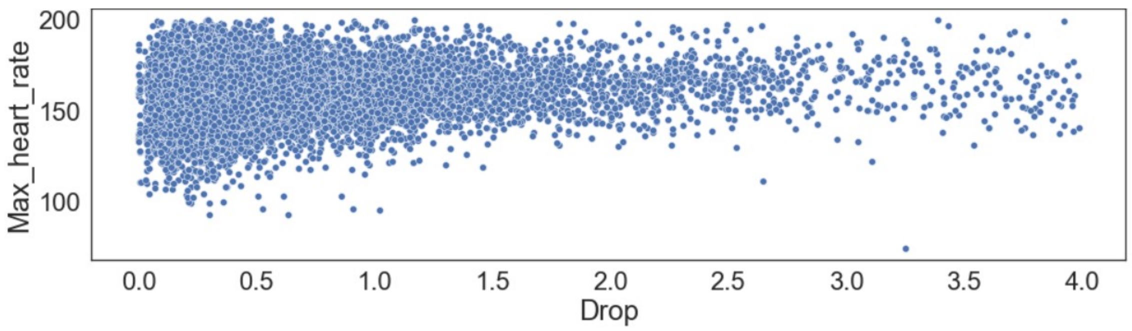


Figure 8. Scatter plot of drop and max heart rate

Figure 9 shows 5 most frequent sports' average heart rates of all the records, along the data point. The five sports share the similar patter that the heart rates rise rapidly and peak at the 30th data point, then maintain that rate until the end. Since among most time of a sport record, the heart rate is maintained at the highest level, it is rational to pay no regard to time variable when predict max heart rate.

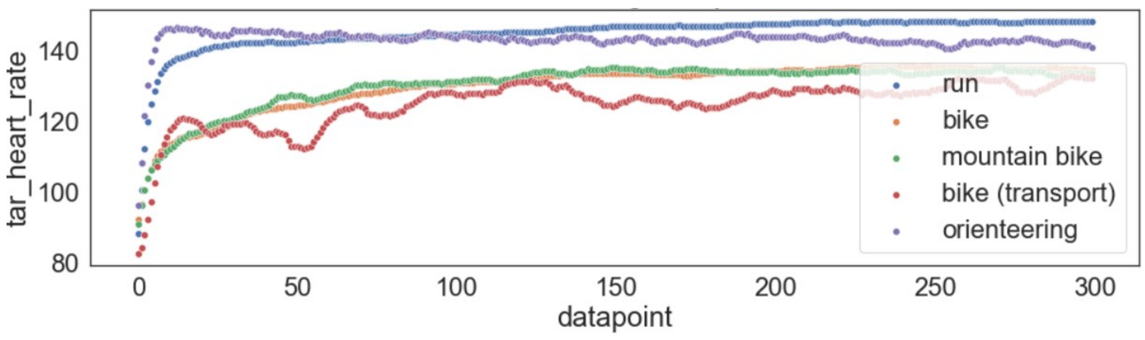


Figure 9. Average heart rate along data point by sports

3. Methods

We apply several machine learning models to further analyze the data set after completing EDA, including both supervised and unsupervised models. In this section we will introduce the methods of the models.

3.1. Supervised Models

We first try to fit supervised models to make predictions on the maximum heart rate during a single workout using other features in the data set. Specifically, we fit one linear regression model and two random forest models of different types (regresstion and classification).

For supervised models, we apply the `train_test_split` method from the `sklearn.model_selection` package to randomly split the data set into a training set and a test set, with a size of 67% and 33% of the original data set respectively. All models will be trained only using the training set, and we will evaluate their performance using the test set.

3.1.1. Linear Regression

By fitting a linear regression model, we try to predict the heart rate with the Equation 1.

$$Max\ Heart\ Rate = c + \sum_{j=1}^{n} a_j x_j \tag{1}$$

where c is the constant term, a_j's are the coefficients, and x_j's are the features for $j = 1, 2, ..., n$.

To perform feature selection to avoid collinearity or overfitting, we will first calculate the variance inflation factors (VIF) for all the numerical features. VIF is an indicator of how well one particular feature can be explained by a linear combination of other features given a set of features. A VIF value of > 10 indicates that the corresponding feature can be explained by a linear combination of other features to a high extent, i.e. it has a high collinearity. We should exclude those features with high VIF values until we obtain a subset of features whose VIFs are all below 10, or ideally, below 5.

After eliminating collinearity, we will also exclude any feature that has a p-value higher than 0.05, which means that we are not able to reject the null hypothesis that the coefficient of that feature is equal to 0 with $\alpha = 0.05$. In other words, the feature might be insignificant to the model.

To evaluate the performance of the linear regression model, we calculate the out-of-sample R^2 (OSR2) using the predicted covariant values and observed covariant values of the test set and compare that to the training R^2.

3.1.2. Random Forest Regression

We adopt a 5-fold cross validation to find the optimal `max_features` parameter in terms of MSE for the random forest regression model. Note that since $R^2 = 1 - \frac{SSE}{SST} = 1 - \frac{\frac{1}{n}SSE}{\frac{1}{n}SST} = 1 - \frac{MSE}{Var(y)}$, where $Var(y)$ is the variance of the observed covariant values and thus a constant, optimizing MSE is equivalent to optimizing R^2.

Similarly to the linear regression model, we calculate the OSR2 to evaluate the performance of the random forest regressor. Since there is no explicit coefficients given by random forest models like those given by linear regression models, we calculate the importance scores of the features to interpret their influence on the model and their relationship with the covariant.

3.1.3. Random Forest Classification

To further improve the performance, we train a random forest classification model as well.

According to CDC, the maximum heart rate can be calculated with the formula $\textit{Max Heart Rate} = 220 - \textit{Age}$ [1]. For example, for an 30-year-old person, the maximum heart rate would be $220 - 30 = 190$. For vigorous-intensity physical activity, the target heart rate should be between 77% and 93% of the maximum heart rate. In other words, a heart rate above 93% of the maximum heart rate could signify an overwhelming intensity of the sport. We can use a random forest classifier to predict the sports that could raise such problems.

According to this information, we label the Max Heart Rate with 1 and 0 respectively for Warning and No Warning. Then we can train a random forest classification model with the same features. We also use a 5-fold cross validation to find the optimal `max_features`, but since this is a classification model, we use accuracy instead of R^2 as the scoring criterion for choosing the optimal parameter.

To access the performance of the random forest classification model, we will calculate the following metrics:

$$\textit{Accuracy} = (TP + TN)/n \tag{2}$$

$$\textit{Precision} = TP/(TP + FP) \tag{3}$$

$$\textit{Recall} = TP/(TP + FN) \tag{4}$$

$$\textit{False Alarm Rate} = FP/(TN + FP) \tag{5}$$

where n is the number of test entries, and compare with those of the baseline predictor (or zero predictor, which simply predict all covariant to be zero).

To understand how the features affect the model, we calculate the importance score similarly as the random forest regression model.

3.2. Unsupervised Model

3.2.1. K-Means Clustering

The K-means algorithm is one of the algorithms with partition, since K-Means is based on determining the initial number of groups by defining the initial centroid value. The K-Means algorithm requires precise numbers in determining the number of clusters k, since the initial cluster centre may change so that this event may result in unstable grouping of data. The output of K-Means depends on the selected centre values on clustering. This algorithm the initial value of the cluster's centre point becomes the basis for the cluster determination. The initial cluster centroid cluster randomly assigns an impact to the performance of the cluster. K-Means Clustering algorithm is one of the clustering methods by partitioning from set data into cluster K. It is a distance-based clustering algorithm that divides data into a number of clusters in numerical attributes. The algorithm has the following steps:

1. Start by randomly choosing k data points to serve as initial centroids

2. Until there is no change in cluster assignments (or max iteration is reached):(re)assign each data point to the cluster centroid to which the data point is closest to in euclidean space. Update the cluster centroid values to represent the means of the data points of each cluster

3. Return the cluster membership for all data points

Determining the number of clusters in a data set is important, not only because some clustering algorithms like k-means require such a parameter, but also because the appropriate number of clusters controls the proper granularity of cluster analysis. It can be regarded as finding a good balance between compressibility and accuracy in cluster analysis.

The elbow method is often used to choose the number of clusters, the parameter k. It is based on the observation that increasing the number of clusters can help to reduce the sum of within-cluster variance of each cluster. This is because having more clusters allows one to capture finer groups of data objects that are more similar to each other. However, the marginal effect of reducing the sum of within-cluster variances may drop if too many clusters are formed, because splitting a cohesive cluster into two gives only a small reduction. Consequently, a heuristic for selecting the right number of clusters is to use the turning point in the curve of the sum of within-cluster variances with respect to the number of clusters.

4. Results

4.1. Supervised Models

4.1.1. Linear Regression

After the initial train of the linear regression model, we obtained a model of $R^2 = 0.106$, which means the model already had poor performance on the training set. Besides, there were several coefficients with p-values significantly greater than 0.05.

Just in case, we still calculated the VIFs for all numerical features:

Table 3. VIF scores

Feature	VIF
latitude	1.267990
longitude	1.305368
altitude	1.047534
derived_speed	1.002630
distance	1.001181

It can be seen that all features have a VIF score below 5, so there is no significant collinearity between the features. Therefore, we only removed features with p-values greater than 0.05, and obtained an adjusted model with $R^2 = 0.104$, with the following linear regression equation:

$$\begin{aligned} \textit{Max Heart Rate} = {} & 167.0133 + 0.0144 \times \textit{longitude} + 1.7844 \times \textit{altitude} + 0.0372 \times \textit{derived speed} && (6)\\ & -2.6135 \times \textit{bike} + 11.7381 \times \textit{circuit training} + 15.0344 \times \textit{cross-country skiing} && (7)\\ & -33.7487 \times \textit{hiking} + 19.5240 \times \textit{kayaking} + 9.7821 \times \textit{orienteering} + 4.7575 \times \textit{run} && (8)\\ & +25.2735 \times \textit{soccer} - 44.4671 \times \textit{walk} - 7.9560 \times female - 9.1267 \times \textit{male} && (9) \end{aligned}$$

We can observe that Max Heart Rate is positively correlated to altitude. Some sports are positively correlated with Heart Rate, such as skiing, kayaking, and soccer, indicating that they might be more vigorous, while others are negatively correlated, such as bike and walk, indicating that they might be milder. Noticeably, both genders have negative coefficients, which agrees with what we have seen during EDA that users with unknown gender had a higher Max Heart Rate.

The OSR2 of the linear regression model is 0.089, which means the linear regression model performs poorly on the test set as well. However, since it is not too far away from the training R^2 (0.104), we may conclude that the problem of this model is not overfitting. We are either underfitting the data, which means we need more features, or linear models are simply not suitable for the data.

The residuals against the observed variant values in the test set are shown in Figure 10. We can see that in most part the residuals are greater than 0 for higher observed heart rates, and less than 0 for lower observed heart rates, which means the model is over-predicting low heart rates and under-predicting high heart rates. It can be concluded that the relationship between heart rates and other features could not be well-explained linearly.

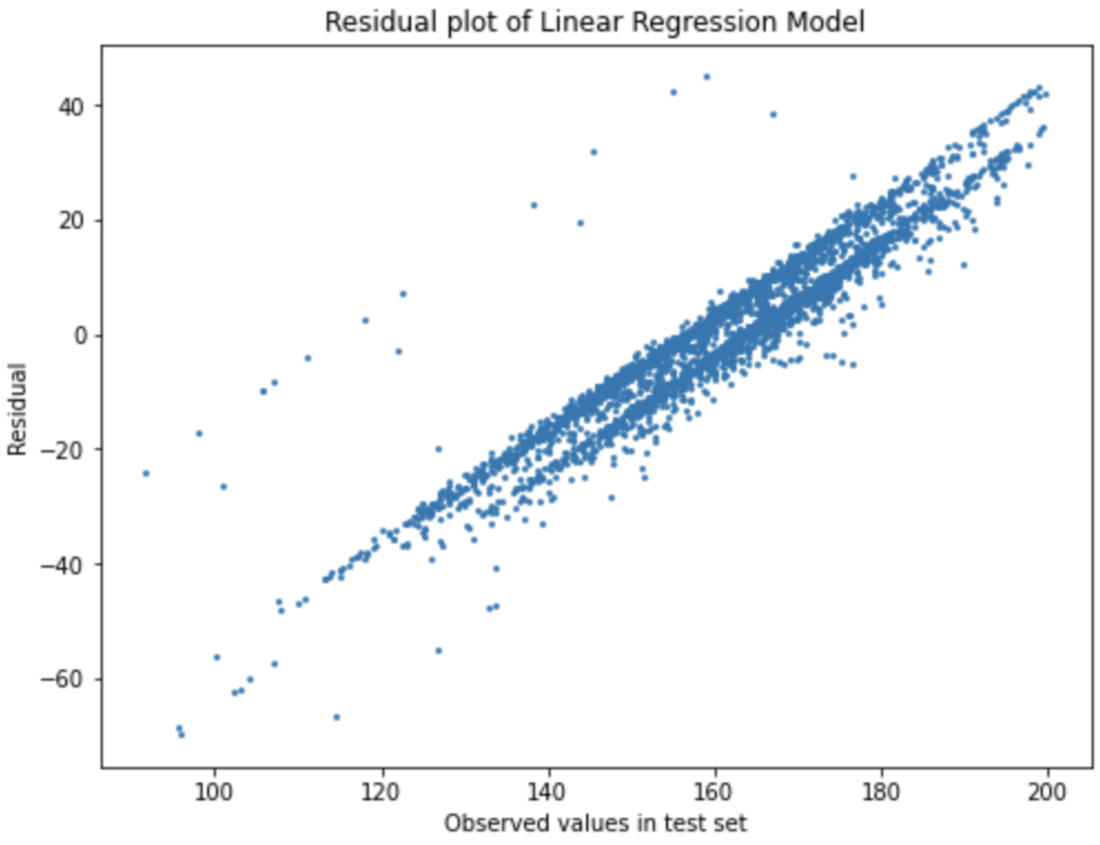


Figure 10. Residual plot of linear regression model

4.1.2. Random Forest Regression

Figure 11 shows the cross validation result of different values of `max_features`. The optimal value of `max_features` is 10 with a training R^2 of 0.4605, and the OSR2 is 0.4820. Although the R^2 and OSR2 scores are already much higher that those of the linear regression model, models with scores under 0.5 are still not considered to be good enough. Figure 12 shows the residuals against the observed variant values for the random forest regression model. We can see more points that are closer to 0, meaning that the residuals are smaller, i.e. the random forest model performs better on the test set than the linear regression model, yet there are still extremely large residuals at around -60 and +40.

Table 4 shows the importance score for each feature in the random forest regression model. Surprisingly, latitude and longitude have the highest scores among all features. Speed and altitude are the next most important features. The type of sports and gender seems to have little influence in the model.

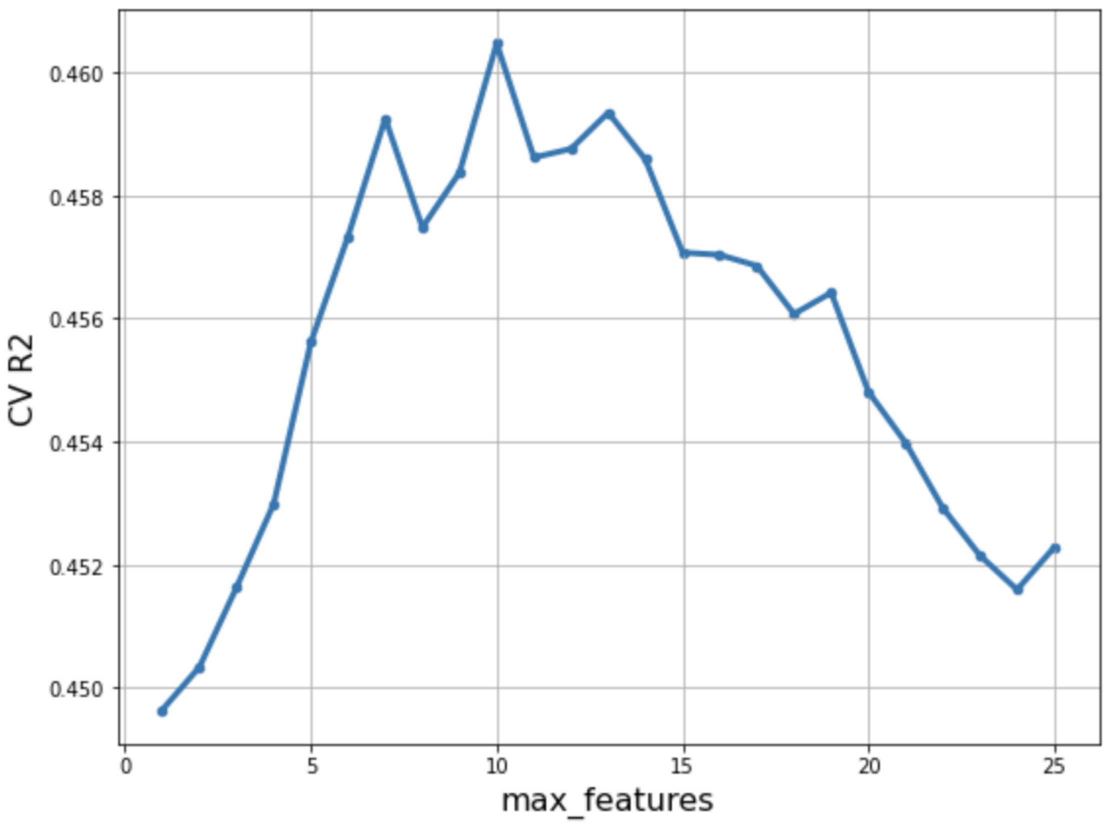


Figure 11. Training R2 vs value of max_features

Table 4. Importance Score of Random Forest Regression

Feature	Importance Score
latitude	23.88
longitude	19.34
altitude	16.84
derived speed	17.25
distance	12.95
basketball	0.00
bike	1.94
bike (transport)	0.70
circuit training	0.01
core stability training	0.02
cross-country skiing	0.02
hiking	0.38
indoor cycling	0.02
kayaking	0.05
mountain bike	0.48
orienteering	0.24
rowing	0.01
run	2.39
skate	0.01
soccer	0.02
tennis	0.00
walk	2.40
weight training	0.00
female	0.51
male	0.53

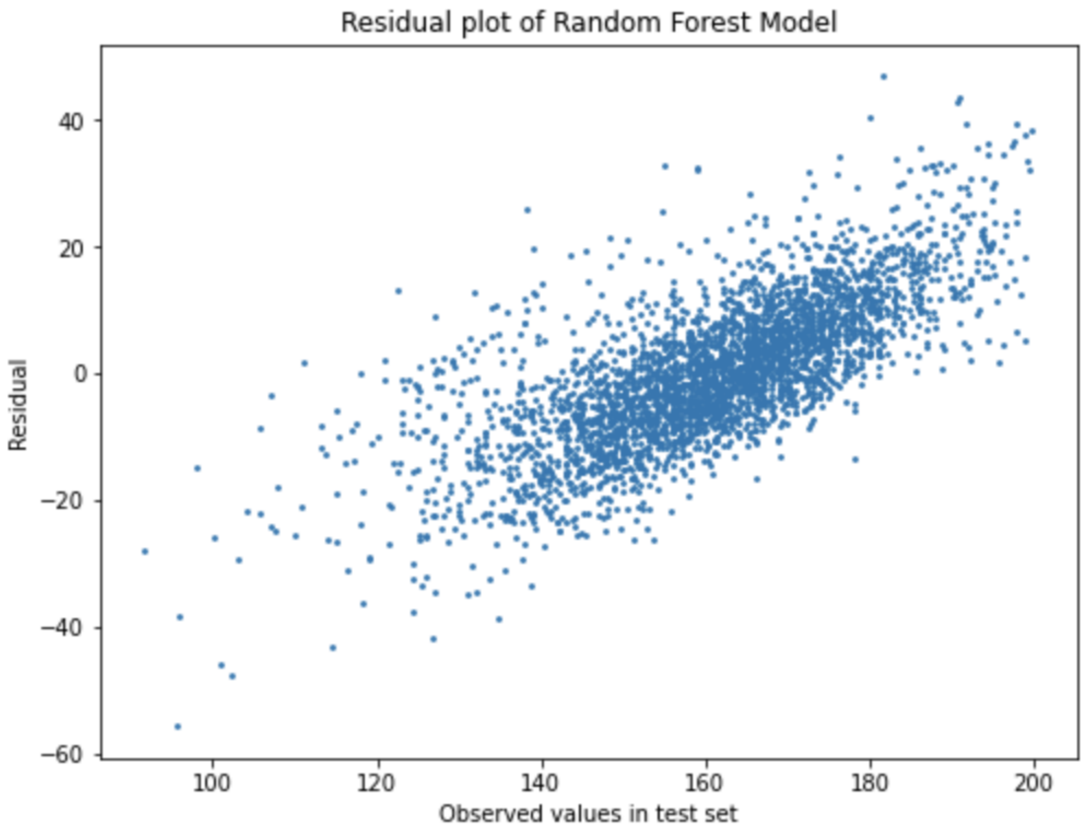


Figure 12. Residual plot of random forest regression model

4.1.3. Random Forest Classification

Figure 13 shows the cross validation result of different values of `max_features`. The optimal value of `max_features` is 16 with a training accuracy of 0.8754. The confusion matrix is:

Table 5. Confusion Matrix of Random Forest Classification

	Predicted 0	**Predicted 1**
Observed 0	2798	45
Observed 1	357	100

From the confusion matrix, we calculated the assessing metrics and compare with the zero estimator. Table 6 shows the comparison.

Table 6. Assessing Metrics of the Random Forest Classifier and the Zero Estimator

	Random Forest	**Zero Estimator**
Accuracy	0.8782	0.8615
Precision	0.6897	N/A
Recall	0.2188	0
False Alarm Rate	0.0158	0

In terms of accuracy, the random forest model did not actually improve much from the baseline model. We can also notice that the recall of the random forest model is only 0.2188, which means

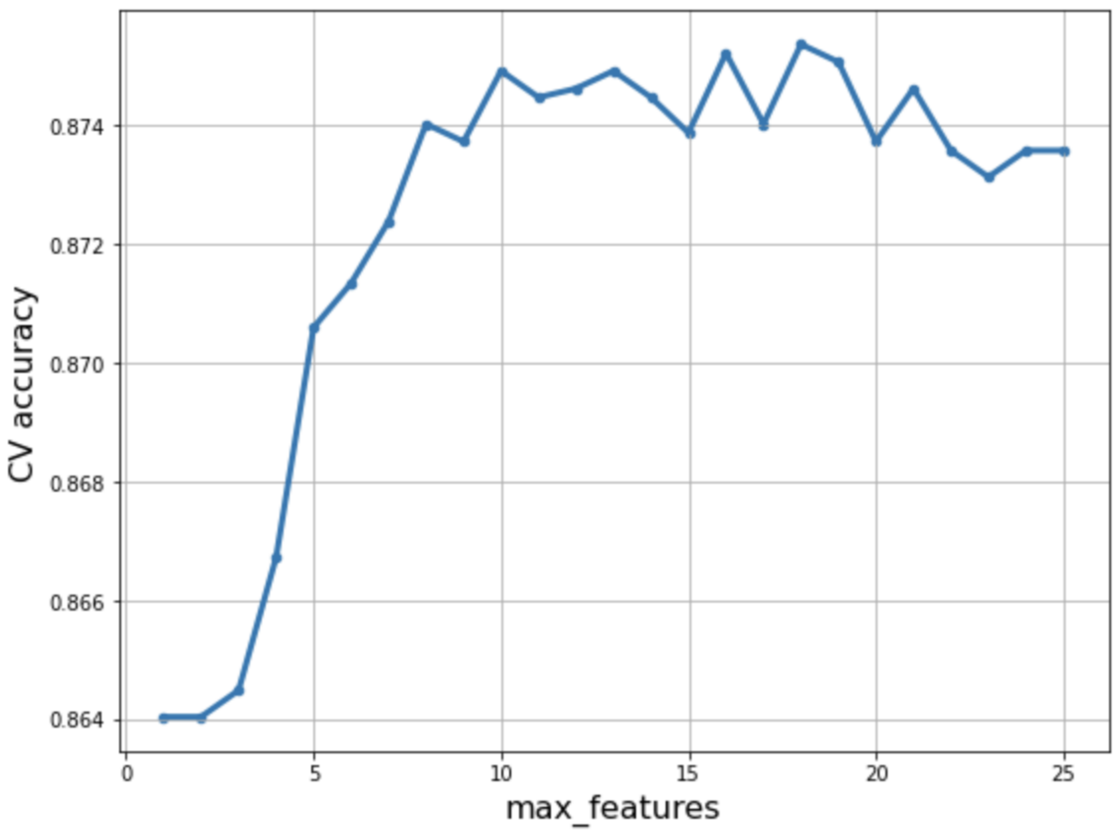


Figure 13. Training accuracy vs value of max_features

around 80% of all heart rates that should raise a warning are still undetected. However, the false alarm rate of the random forest model is very low at the same time. Given that it can increase the recall by 0.2188 and only increase the false alarm rate by 0.0158, the model is still useful in detecting potential warnings.

Table 7 shows the importance score for each feature in the random forest classification model. We can observe a similar pattern as the importance scores of the random forest regression model, with latitude and longitude having the highest score, and altitude, speed, and distance next.

Table 7. Importance Score of Random Forest Classification

Feature	Importance Score
latitude	28.51
longitude	21.38
altitude	13.29
derived speed	17.12
distance	13.66
basketball	0.00
bike	2.05
bike (transport)	0.10
circuit training	0.01
core stability training	0.00
cross-country skiing	0.06
hiking	0.00
indoor cycling	0.04
kayaking	0.08
mountain bike	1.31
orienteering	0.93
rowing	0.00
run	1.00
skate	0.00
soccer	0.00
tennis	0.00
walk	0.01
weight training	0.00
female	0.12
male	0.33

4.2. Unsupervised Models

4.2.1. K-means Clustering

In this sector, we leverage clustering method to segment biking users. The reason of we choosing bike is that the data set has over 140,000 data points of bike records, which far exceeds all other types of sport. So, we aggregate the those timestamp data points into a new table of user as granularity. Finally, we obtain the bike data set of 69 users and each column records the median of the corresponding variable in all their exercises. For example, the column `speed_mean` refers to the median of the each exercise's average speed, that is $median(avg(speed\,column))$ grouped by user.

To determine the optimal number of clusters, we have to select the value of k at the "elbow". The idea is that we want a small SSE, but that the SSE tends to decrease toward 0 as we increase k. The SSE is 0 when k is equal to the number of data points in the dataset, because then each data point is its own cluster, and there is no error between it and the center of its cluster. So our goal is to choose a small value of k that still has a low SSE, and the elbow usually represents where we start to have diminishing returns by increasing k. Figure 14 shows for the given data, we conclude that the optimal number of clusters for the data is 4.

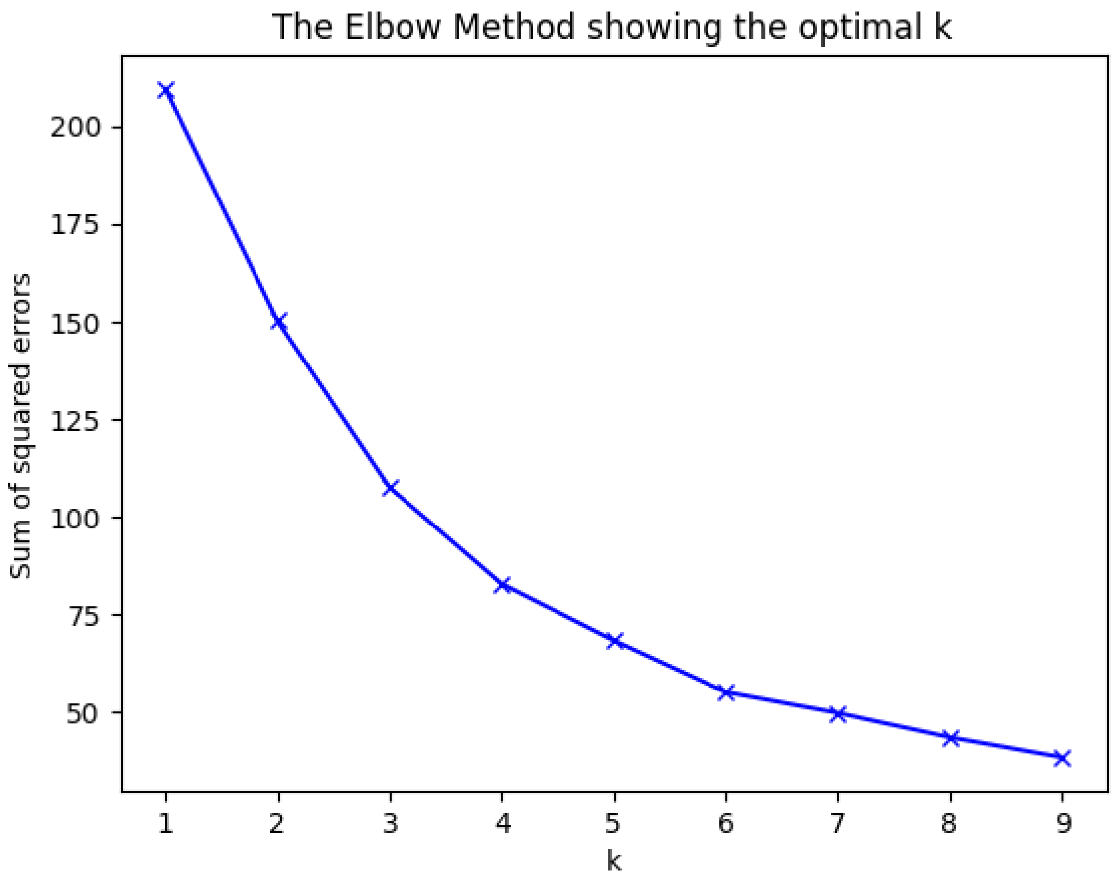


Figure 14. Elbow plot of clustering model

Table 8. Centroids of clusters

	Gender	**Speed**	**Max heart rate**	**Altitude difference**
Group 0	0.9	-1	0.8	-0.3
Group 1	0.9	0.7	0.2	-0.1
Group 3	1.0	-1.4	0.1	3.7
Group 4	0.9	-0.5	-1.4	-0.3

Table 8 presents centroids of each group when we pick the k equal to 4. The group 0 consists of users with low speed, high maximum heart rate and moderate altitude difference. This group is identified as the risk group that indicates probable heart anomaly and entails careful analysis. Group 1 is considered as road biker, who is used to ride at high speed with moderate maximum heart rate and average altitude difference. Group 2 has low speed, moderate maximum heart rate, and significantly high altitude difference, so we hypothesize they are the advanced mountain off-road biker. The users in the Group 3 are likely to be beginners with exercise habits because they has moderate speed, low maximum heart rate and moderate altitude difference.

4.2.2. Visualizing High Dimensional Clusters with PCA

As our clustering model is of 4 dimensions, we have to use dimension reduction techniques for the convenience of visualization. So, we use Principal Component Analysis (PCA). PCA is an algorithm that is used for dimensionality reduction meaning. We will use these principal components to help us visualize our clusters in 2-D and 3-D space.

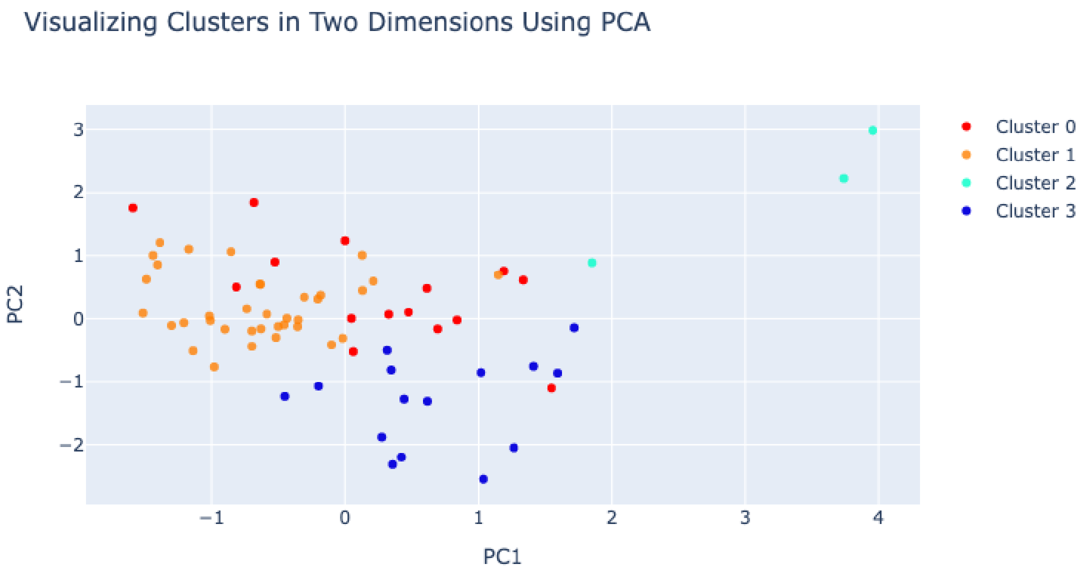


Figure 15. Visualizing Clusters in Two Dimensions

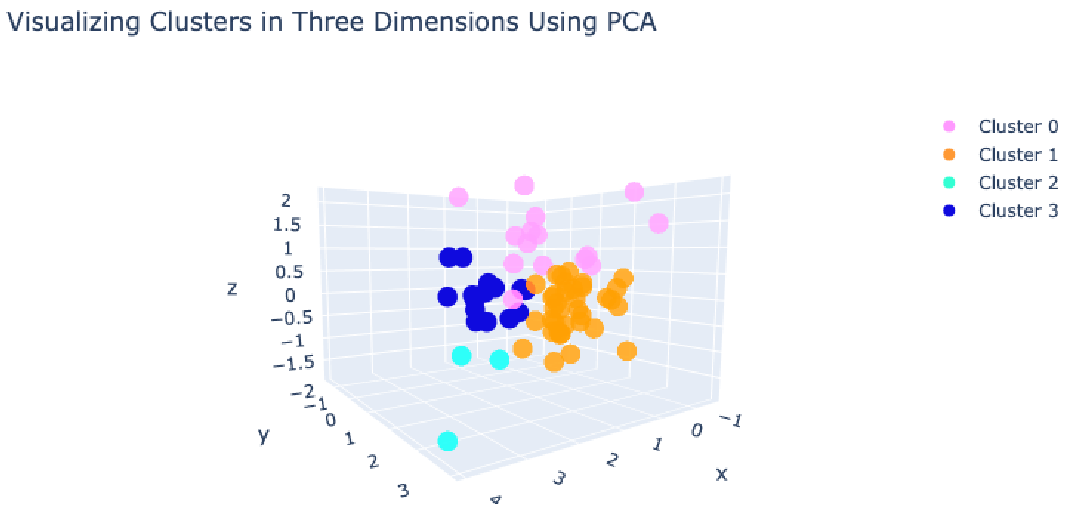


Figure 16. Visualizing Clusters in Three Dimensions

In Figure 15 and Figure 16, we are able to see how each group scatters away from others. The 3-D visualization is better in visualizing the risk group, that is group 0, which distributes on the top of the figure, and is relatively distant from the rest of groups. The advantages of PCA is that it removes correlated features and transforms a high dimensional data to low dimensional data so that it can be visualized easily. However, the limitations of PCA lay in the assumption that the principal components are orthogonal which might not be true in every case. Also, PCA transformed the data into lower dimension vectors that is virtually interpretable. In our case, we are unable to strictly interpret the the meaning of each coordinate.

4.2.3. Examine the Risk Group

The last step of our clustering analysis is to examine the risk group. We have discovered they prone to ride at low speed, and appear to be with high maximum heart rate and moderate altitude difference. We tentatively explore that reason for their unique activity logs by comparing them to the non-risk group.

Table 9. Comparison of the risk and the non-risk group

	Risk Group	**Non-risk Group**
Gender	0.92	0.95
Speed	-2.3	-0.4
Max heart rate	165.6	160.0
Std heart rate	13.8	12.8
Altitude diff	0.61	0.92
Avg records	64.9	105.3

Table 9 compares the difference of the risk group to all other groups. We have found that the gender of two groups is almost the same. Maximum heart rate of the risk group is higher than this of the non-risk group, that is 165 vs 160. Simultaneously, altitude difference again validates the risk group tends to ride on flatter routes, and their riding speed is fairly slow. We also noticed that the standard deviation of heart rate of the risk group is slightly higher than the normal group, indicating the risk group experiences a more drastic heart rate fluctuation during biking.

It is also noteworthy that the risk group has significantly fewer sport records compared to the non-risk group, which can imply a lack of experience in scheduling and physical distribution, but can also come from the mismatched, difficult biking routes. We assume that the insufficient sport experience is essentially a reason for the intense maximum heart rate of the risk group.

5. Discussion

5.1. Supervised Models

In general, we can see that the supervised models did not have satisfactory performance. One limitation is the lack of relevant features, especially in the linear regression model where we discovered a severe underfitting problem. To solve this problem, we could collect data for more features in the future. For example, the age, ethnicity, and body weight can all be potentially correlated with the heart rate.

Another main limitation is the sample size. We found in both random forest models that longitude and latitude were the two most important features, which was rather counter-intuitive. One possible explanation for this is that the workouts done at the same (or very similar) geographical location are actually done by the same person, so what we were really basing our prediction on was the individual person who did workouts at that position, and it makes sense that the same person tends to have similar heart rate at each of her workout. Yet basing the prediction on individual person might not be helpful if we have new users. To eliminate this problem, we could either exclude the longitude and latitude in training the models, or increase the sample size so that the focus on individual person may be diluted and other features may take more importance.

The recall of the random forest classifier is only 0.2188, which is not convincing. One strategy to improve the recall is to add a class weight during training so that the model can be more sensitive to 1's (e.g. set `class_weight = {0:1, 1:3}`). However, altering the class weight like this usually improves the recall while sacrificing some accuracy and false alarm rate. Further research can be done in tuning with the class weights.

5.2. Unsupervised Models

The clustering analysis shows how we can leverage an unsupervised machine learning model to detect the heart anomaly and identify the risk group. This result is particularly useful when people decide to design an alert system that provides the user with a heart health caveat on wearable devices. And once combined with geographical data like latitude deviation and route length, we are able to construct a route recommendation system that matches the level of physical ability and exercise habits of each user.

However, this analysis is not flawless. It suffers a lot from the constraints of the data set. One concern is the data insufficiency that many common correlated features, for example, age and race, and personal information, for example, medical history and exercise frequency are inaccessible for this analysis; whereas they are likely to be essential in explaining disparities among groups. Another concern is a technical one, in that we are not sure what kind of matrices to assess the clustering model. Also, we would be glad to improve our analysis and insights if we receive support for more clinical knowledge of cardiology in the future.

References

[1] "Target Heart Rate and Estimated Maximum Heart Rate," *Centers of Diseases Control and Prevention*, June 3, 2022. [Online]. Available: https://www.cdc.gov/physicalactivity/basics/measuring/heartrate.htm. Accessed on December 7, 2022

[2] "Modeling Heart Rate and Activity Data for Personalized Fitness Recommendation," Jianmo Ni, Larry Muhlstein, and Julian McAuley, WWW '19: The World Wide Web Conference 2019, https://doi.org/10.1145/3308558.3313643

.

In [1]:
```
import seaborn as sns
import csv
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from pathlib import Path
%matplotlib inline
```

In [2]:
```
data = pd.read_csv(r"C:\Users\Joy Jin\Downloads\full_data.csv")
```

In [3]:
```
data=data.rename(columns={"Unnamed: 0":"datapoint"})
datadisply=data.copy().set_index("id")
datadisply.drop(["since_begin","time_elapsed","timestamp","since_last"],axis=1)
```

Out[3]:

	datapoint	latitude	gender	tar_heart_rate	longitude	heart_rate	sport	altitude	derived_speed	tar_derived_sp
id										
396826535	0	60.173349	male	100.000000	24.649770	-8.197369	bike	-1.804467	-7.082944	7.105427e
396826535	1	60.173240	male	113.355469	24.650143	-5.369012	bike	-1.818636	-2.088780	1.255489e-
396826535	2	60.172980	male	120.214752	24.650911	-3.916386	bike	-1.820717	-0.351569	1.692208e-
396826535	3	60.172478	male	119.108221	24.650669	-4.150721	bike	-1.847772	-0.680039	1.609634e-
396826535	4	60.171861	male	120.569362	24.649145	-3.841288	bike	-1.851729	-0.279256	1.710387e-
...	...	...	...	...	...	...	...	...	...	
176731991	295	55.673904	male	89.788487	37.459480	-10.359914	bike	-0.064222	-0.209647	1.727886e-
176731991	296	55.674434	male	87.000000	37.460354	-10.950447	bike	-0.078347	-1.329734	1.446306e-
176731991	297	55.675018	male	87.273563	37.461202	-10.892513	bike	-0.105896	-0.006515	1.778952e-
176731991	298	55.675446	male	85.000000	37.461815	-11.373997	bike	-0.124999	-3.231240	9.682845e-
176731991	299	55.675558	male	94.000000	37.461982	-9.468020	bike	-0.165820	-0.534229	1.646289e-

3000000 rows × 12 columns

In [4]:
```
number_of_records_by_gender_soprts=(data.groupby("sport").size()/300)\
                                    .to_frame("Number of records").astype(int)
number_of_records_by_gender_soprts.transpose()
```

Out[4]:

sport	basketball	bike	bike (transport)	circuit training	core stability training	cross-country skiing	hiking	indoor cycling	kayaking	mountain bike	orienteering	rowing
Number of records	1	4766	209	8	14	16	10	27	8	712	163	11

In [5]:
```
number_of_users_by_gender=data.groupby("gender")['userId']\
                         .nunique().to_frame("Numbe of users")
number_of_users_by_gender.transpose()
```

Out[5]:

gender	female	male	unknown
Numbe of users	10	90	3

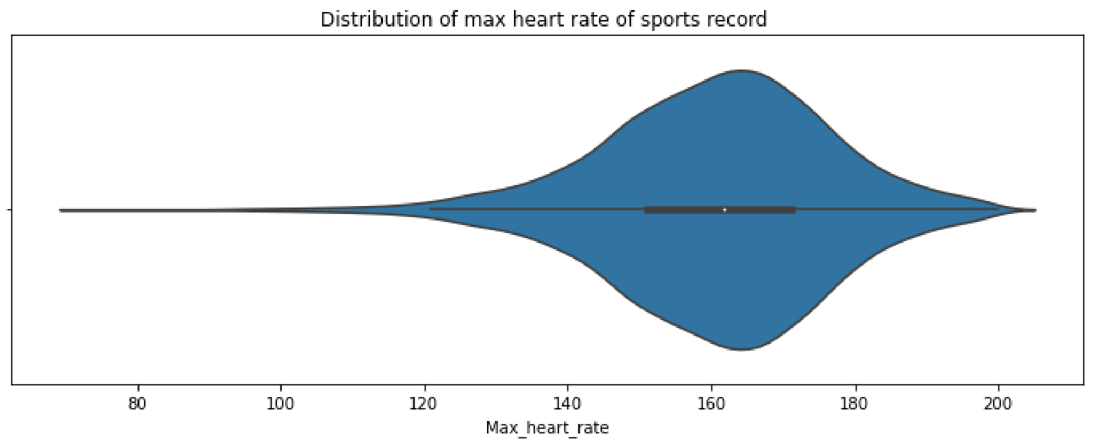
In [6]:
```
data_enoughsample= data[(data["sport"] != "basketball") &
(data["sport"] != "skate") &
(data["sport"] != "soccer") &
(data["sport"] != "tennis") &
(data["sport"] != "weight training")]
```

In [7]:
```
(data_enoughsample.groupby("sport").size()/300).astype(int).sort_values
```

Out[7]:
```
<bound method Series.sort_values of sport
bike                          4766
bike (transport)               209
circuit training                 8
core stability training         14
cross-country skiing            16
hiking                          10
indoor cycling                  27
kayaking                         8
mountain bike                  712
orienteering                   163
rowing                          11
run                           4013
walk                            36
dtype: int32>
```

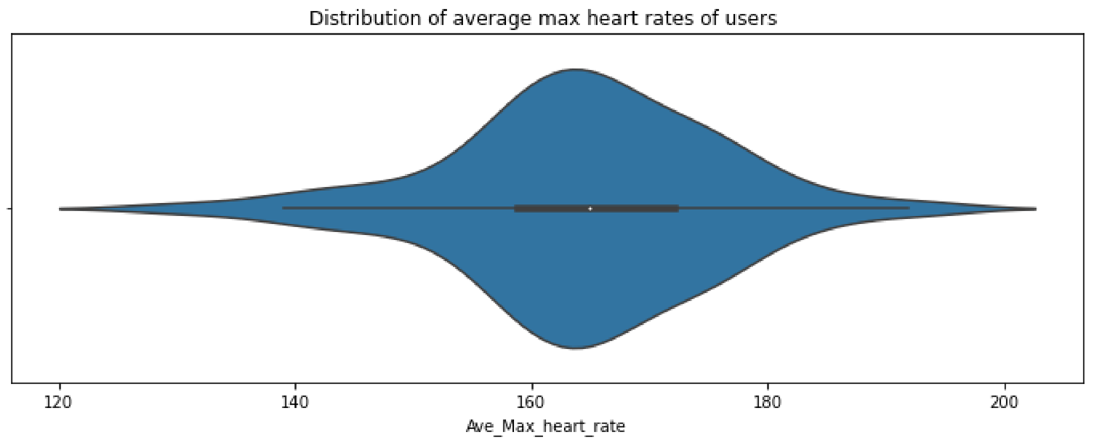
In [8]:
```
max_heartrate=data_enoughsample.groupby("id")["tar_heart_rate"].max().to_frame()\
            .rename(columns = {'tar_heart_rate':'Max_heart_rate'})
plt.figure(figsize=(12, 4))
sns.violinplot(data=max_heartrate, x="Max_heart_rate")
plt.title("Distribution of max heart rate of sports record ")
# white, dark, whitegrid, darkgrid, ticks
```

Out[8]:
```
Text(0.5, 1.0, 'Distribution of max heart rate of sports record ')
```



In [9]:
```
max_heartrate_byuser=data_enoughsample.groupby(["id","userId"])["tar_heart_rate"].max()\
                    .to_frame().groupby(["userId"]).mean()\
                    .rename(columns = {'tar_heart_rate':'Ave_Max_heart_rate'})

plt.figure(figsize=(12, 4))
sns.violinplot(data=max_heartrate_byuser,x="Ave_Max_heart_rate")
plt.title("Distribution of average max heart rates of users ")
```

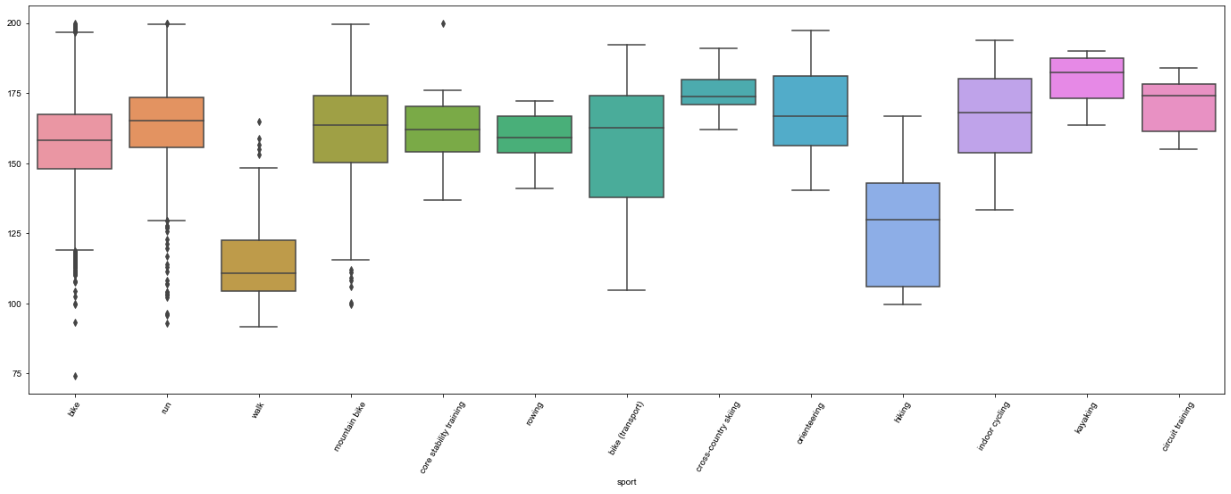
Out[9]:
```
Text(0.5, 1.0, 'Distribution of average max heart rates of users ')
```



In [10]:
```
max_heartrate_bysports=data_enoughsample.groupby(["id","sport"])["tar_heart_rate"].max()\
                      .to_frame().rename(columns = {'tar_heart_rate':' '}).reset_index()
```

In [11]:
```
plt.figure(figsize=(24, 8))
sns.boxplot(data=max_heartrate_bysports, x="sport",y=" ")
sns.set_theme(font_scale=3)
plt.xticks(rotation=60)
```

Out[11]:
```
(array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12]),
 [Text(0, 0, 'bike'),
  Text(1, 0, 'run'),
  Text(2, 0, 'walk'),
  Text(3, 0, 'mountain bike'),
  Text(4, 0, 'core stability training'),
  Text(5, 0, 'rowing'),
  Text(6, 0, 'bike (transport)'),
  Text(7, 0, 'cross-country skiing'),
  Text(8, 0, 'orienteering'),
  Text(9, 0, 'hiking'),
  Text(10, 0, 'indoor cycling'),
  Text(11, 0, 'kayaking'),
  Text(12, 0, 'circuit training')])
```



In [12]:
```
max_heartrate_bygender=(data_enoughsample.groupby(["id","gender"])["tar_heart_rate"].max()\
                        .to_frame()).rename(columns = {'tar_heart_rate':'Max_heart_rate'}).reset_index()
max_heartrate_bygender
```

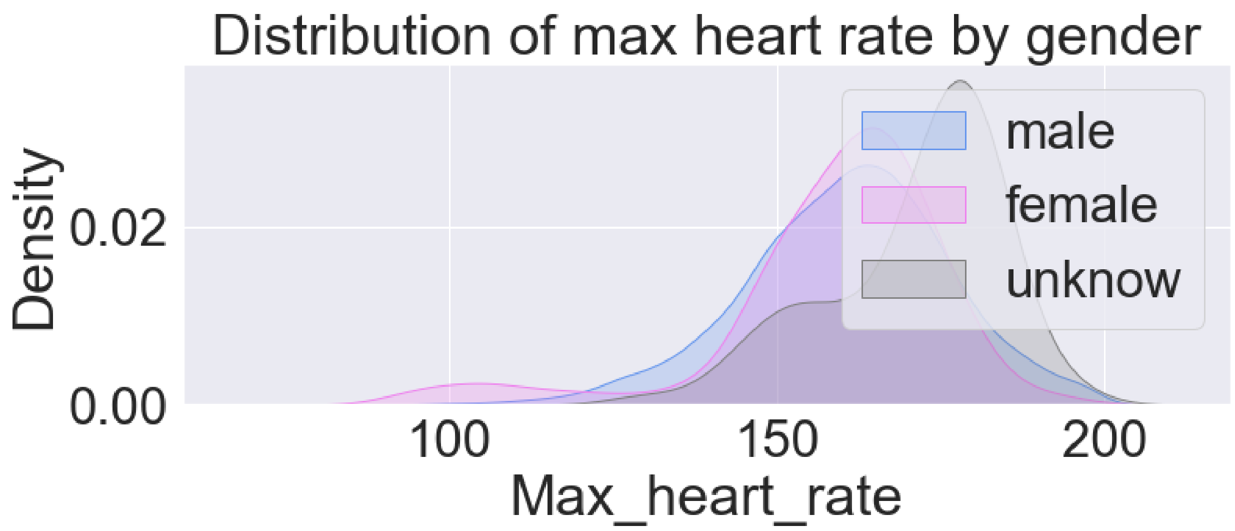
Out[12]:

	id	gender	Max_heart_rate
0	3930381	male	133.188170
1	3933514	male	147.056557
2	3940962	male	153.203170
3	4632763	male	144.093784
4	4651866	male	149.000000
...	...	...	...
9988	651598821	male	180.056087
9989	651793414	male	167.947758
9990	652776545	male	188.201302
9991	656149214	male	171.100294
9992	657584281	male	170.606347

9993 rows × 3 columns

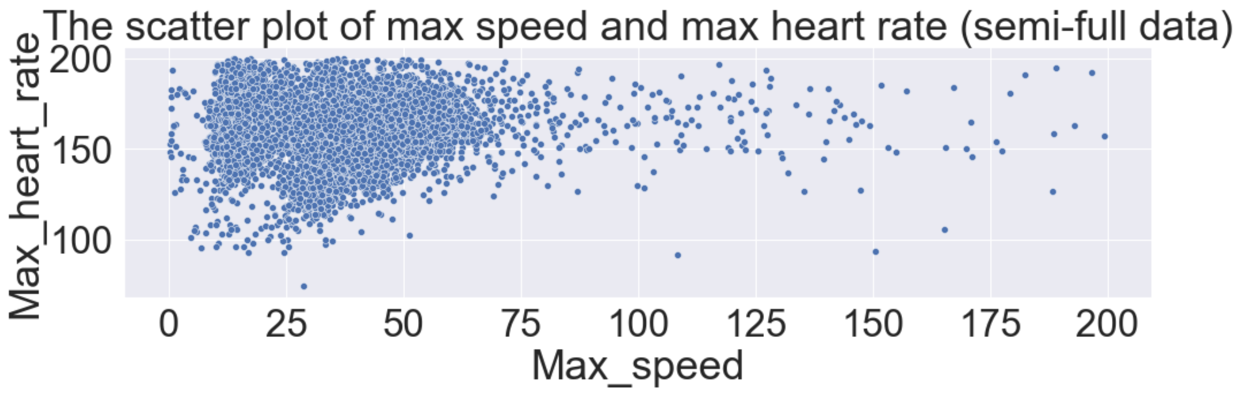
In [13]:
```
plt.figure(figsize=(12, 4))
maledata=max_heartrate_bygender[max_heartrate_bygender["gender"]=="male"]
femaledata=max_heartrate_bygender[max_heartrate_bygender["gender"]=="female"]
unknowgenderdata=max_heartrate_bygender[max_heartrate_bygender["gender"]=="unknown"]
sns.kdeplot(data=maledata,x="Max_heart_rate",color="cornflowerblue",fill=True,label="male")
sns.kdeplot(data=femaledata,x="Max_heart_rate",color="violet",fill=True,label="female")
sns.kdeplot(data=unknowgenderdata,x="Max_heart_rate",color="grey",fill=True,label="unknow")
plt.legend()
plt.title("Distribution of max heart rate by gender")
```

Out[13]:
```
Text(0.5, 1.0, 'Distribution of max heart rate by gender')
```



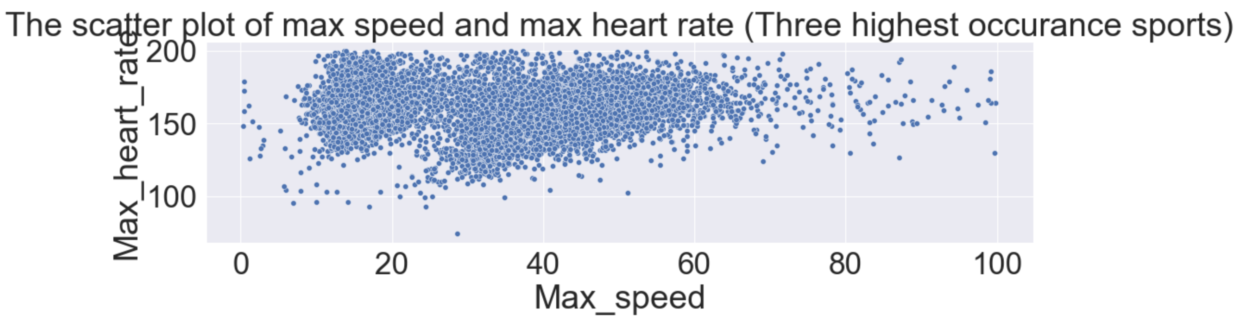
```
In [14]: data_exclude=data_enoughsample[(data_enoughsample["sport"] != "circuit training") &
                                        (data_enoughsample["sport"] != "core stability training")]

         MAXheartdataall=data_exclude.groupby("id")["tar_heart_rate"].max().to_frame("Max_heart_rate")
         MAXspeeddataall=data_exclude.groupby("id")["tar_derived_speed"].max().to_frame("Max_speed")
         plotdataall=MAXheartdataall.merge(MAXspeeddataall,on='id')
         plotdataall=plotdataall[plotdataall["Max_speed"]<=200]
         plt.figure(figsize=(16, 4))
         sns.scatterplot(data=plotdataall, x="Max_speed", y="Max_heart_rate")
         plt.title("The scatter plot of max speed and max heart rate (semi-full data)")
```

```
Out[14]: Text(0.5, 1.0, 'The scatter plot of max speed and max heart rate (semi-full data)')
```



```
In [15]: data_select=data_enoughsample[(data_enoughsample["sport"] == "run") |
                                       (data_enoughsample["sport"] == "bike") |
                                       (data_enoughsample["sport"] == "mountain bike")]

         MAXheartdata=data_select.groupby("id")["tar_heart_rate"].max().to_frame("Max_heart_rate")
         MAXspeeddata=data_select.groupby("id")["tar_derived_speed"].max().to_frame("Max_speed")
         plotdata=MAXheartdata.merge(MAXspeeddata,on='id')
         plotdata=plotdata[plotdata["Max_speed"]<=100]
         plt.figure(figsize=(16, 4))
         sns.scatterplot(data=plotdata, x="Max_speed", y="Max_heart_rate")
         plt.title("The scatter plot of max speed and max heart rate (Three highest occurance sports)")
```

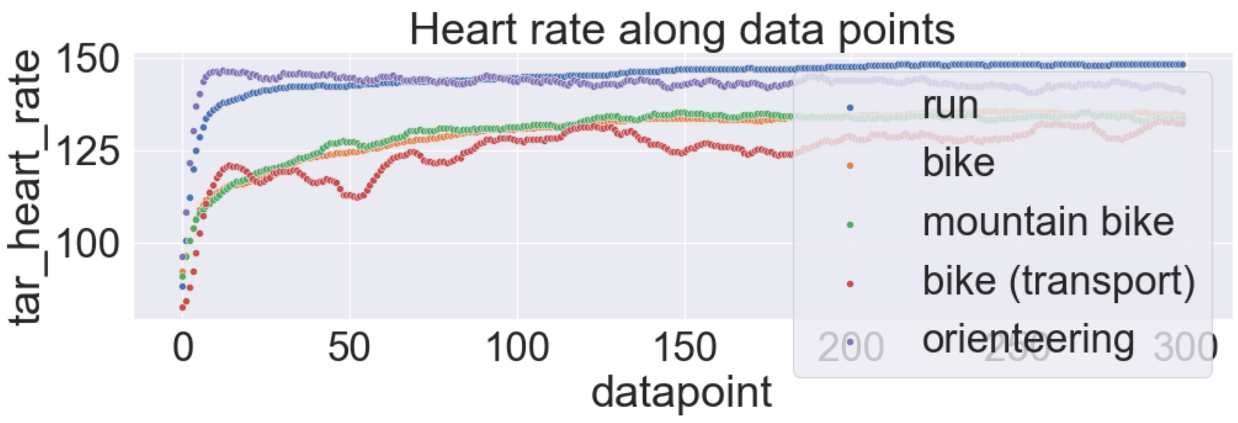
```
Out[15]: Text(0.5, 1.0, 'The scatter plot of max speed and max heart rate (Three highest occurance sports)')
```



```
In [16]: time_and_heartrate=data_enoughsample.groupby(["datapoint","sport"])["tar_heart_rate"].mean().to_frame().reset
```

```
plt.figure(figsize=(16, 4))
sns.scatterplot(data=time_and_heartrate[time_and_heartrate["sport"]=="run"],\
               x="datapoint",y="tar_heart_rate",label="run")
sns.scatterplot(data=time_and_heartrate[time_and_heartrate["sport"]=="bike"],\
               x="datapoint",y="tar_heart_rate",label="bike")
sns.scatterplot(data=time_and_heartrate[time_and_heartrate["sport"]=="mountain bike"],\
               x="datapoint",y="tar_heart_rate",label="mountain bike")
sns.scatterplot(data=time_and_heartrate[time_and_heartrate["sport"]=="bike (transport)"],\
               x="datapoint",y="tar_heart_rate",label="bike (transport)")
sns.scatterplot(data=time_and_heartrate[time_and_heartrate["sport"]=="orienteering"], \
               x="datapoint",y="tar_heart_rate",label="orienteering")

plt.legend()
plt.title("Heart rate along data points")
```

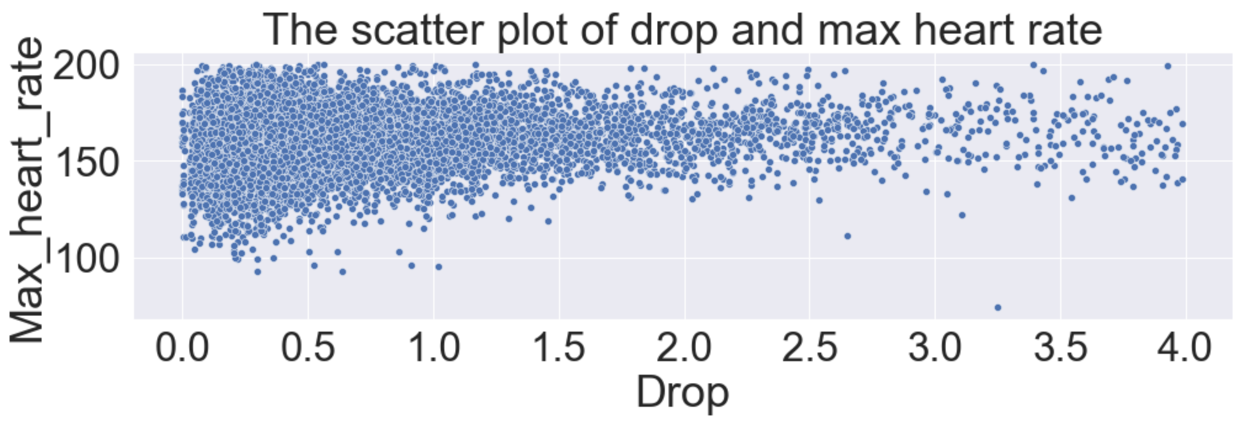
Out[16]:
```
Text(0.5, 1.0, 'Heart rate along data points')
```



In [17]:
```
def drop(a):
    return np.max(a)-np.min(a)

data_altitude=data_select.groupby("id")["altitude"].agg(drop).to_frame().reset_index()
data_altitude=data_altitude.merge((data_select.groupby("id")["tar_heart_rate"].max()),on="id")\
.rename(columns = {'tar_heart_rate':'Max_heart_rate','altitude':'Drop'})
```

In [18]:
```
data_altitude=data_altitude[data_altitude["Drop"]<=4]
plt.figure(figsize=(16, 4))
sns.scatterplot(data=data_altitude, x="Drop", y="Max_heart_rate")
plt.title("The scatter plot of drop and max heart rate")
```

Out[18]:
```
Text(0.5, 1.0, 'The scatter plot of drop and max heart rate')
```



In [ ]:

OLS_RF

December 9, 2022

In this part, we fit a linear regression model and two random forest models to the data to predict the heart rate (`tar_heart_rate`).

```
[2]: import numpy as np
     import pandas as pd
```

```
[3]: data = pd.read_csv("full_data.csv", encoding = 'unicode_escape')
```

```
[4]: data.head()
```

```
[4]:    Unnamed: 0   since_begin  time_elapsed   latitude gender  tar_heart_rate  \
     0           0  1.378479e+06     -0.122568  60.173349   male      100.000000
     1           1  1.378479e+06     -0.122122  60.173240   male      113.355469
     2           2  1.378479e+06     -0.121676  60.172980   male      120.214752
     3           3  1.378479e+06     -0.121230  60.172478   male      119.108221
     4           4  1.378479e+06     -0.120784  60.171861   male      120.569362

         timestamp         id  longitude   since_last  heart_rate sport  altitude  \
     0  1408898746  396826535  24.649770  2158.846078   -8.197369  bike -1.804467
     1  1408898754  396826535  24.650143  2158.846078   -5.369012  bike -1.818636
     2  1408898765  396826535  24.650911  2158.846078   -3.916386  bike -1.820717
     3  1408898778  396826535  24.650669  2158.846078   -4.150721  bike -1.847772
     4  1408898794  396826535  24.649145  2158.846078   -3.841288  bike -1.851729

        derived_speed  tar_derived_speed  distance    userId
     0      -7.082944       7.105427e-15 -4.372304  10921915
     1      -2.088780       1.255489e+01 -1.797320  10921915
     2      -0.351569       1.692208e+01 -0.055967  10921915
     3      -0.680039       1.609634e+01 -0.051062  10921915
     4      -0.279256       1.710387e+01  4.282176  10921915
```

```
[31]: data_agg = data.groupby('id').agg({
          'latitude': np.mean,
          'gender': 'first',
          'tar_heart_rate': 'max',
          'longitude': np.mean,
          'sport': 'first',
          'altitude': np.ptp,
    'derived_speed': np.mean,
    'distance': 'max'
})
data_agg.head()
```

```
[31]:          latitude gender  tar_heart_rate  longitude sport  altitude  \
      id
      3930381  43.858155   male      133.188170  10.550179  bike  0.322210
      3933514  43.863827   male      147.056557  10.603604  bike  0.850032
      3940962  43.809938   male      153.203170  10.507894  bike  1.038723
      4632763  43.828347   male      144.093784  10.473134  bike  0.456615
      4651866  43.843574   male      149.000000  10.635830  bike  0.672854

               derived_speed   distance
      id
      3930381       0.810823   4.111688
      3933514       0.094354   2.548592
      3940962       0.330715   4.183119
      4632763       0.238919  13.567686
      4651866       0.033350  19.045289
```

We then perform some feature engineering on the data. We use one hot encoding to encode `sport`:

```
[32]: sports = pd.get_dummies(data_agg.sport, prefix='sport')
      data_agg = data_agg.join(sports)
```

Then we use one hot encoding on `gender` as well. Note that since there are users with 'unknown' gender, we can take that as our baseline and only encode the 'male' and 'female'.

```
[33]: genders = pd.get_dummies(data_agg.gender, prefix='gender')
      genders = genders.drop(['gender_unknown'], axis=1)
      data_agg = data_agg.join(genders)
```

Then we can proceed to separate `X` and `y` and generate a training set and a test set by using `train_test_split`. Note that we have to drop the columns that could not be taken in the model as a feature.

```
[34]: data_agg.columns.values
```

```
[34]: array(['latitude', 'gender', 'tar_heart_rate', 'longitude', 'sport',
             'altitude', 'derived_speed', 'distance', 'sport_basketball',
             'sport_bike', 'sport_bike (transport)', 'sport_circuit training',
             'sport_core stability training', 'sport_cross-country skiing',
             'sport_hiking', 'sport_indoor cycling', 'sport_kayaking',
             'sport_mountain bike', 'sport_orienteering', 'sport_rowing',
             'sport_run', 'sport_skate', 'sport_soccer', 'sport_tennis',
             'sport_walk', 'sport_weight training', 'gender_female',
             'gender_male'], dtype=object)
```

[41]:
```python
from sklearn.model_selection import train_test_split

X = data_agg.drop(['tar_heart_rate', 'gender', 'sport'], axis=1)
y = data_agg['tar_heart_rate']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33,␣
 ↪random_state=42)
```

We use `statsmodels.api` package to fit the linear regression model.

[43]:
```python
import statsmodels.api as sm

X_train_linreg = sm.add_constant(X_train)

reg = sm.OLS(y_train, X_train_linreg).fit()

print(reg.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:         tar_heart_rate   R-squared:                       0.106
Model:                            OLS   Adj. R-squared:                  0.103
Method:                 Least Squares   F-statistic:                     33.07
Date:                Sun, 20 Nov 2022   Prob (F-statistic):          4.08e-143
Time:                        20:56:15   Log-Likelihood:                -27683.
No. Observations:                6700   AIC:                         5.542e+04
Df Residuals:                    6675   BIC:                         5.559e+04
Df Model:                          24
Covariance Type:            nonrobust
================================================================================
=================
                                    coef    std err          t      P>|t|
[0.025      0.975]
--------------------------------------------------------------------------------
-----------------
const                           158.6857      2.702     58.734      0.000
153.389     163.982
latitude                         -0.0202      0.011     -1.881      0.060
-0.041       0.001
longitude                         0.0121      0.005      2.475      0.013
0.003       0.022
altitude                          1.7587      0.175     10.061      0.000
1.416       2.101
derived_speed                     0.0374      0.019      2.009      0.045
0.001       0.074
distance                         -0.0002      0.001     -0.439      0.661
-0.001       0.001
sport_basketball                 -4.5751     14.394     -0.318      0.751
-32.793      23.642
```

```
sport_bike                        6.7406      1.789      3.768      0.000
3.233      10.248
sport_bike (transport)            5.1751      2.173      2.382      0.017
0.916       9.435
sport_circuit training           21.4377      5.355      4.003      0.000
10.940      31.936
sport_core stability training     8.2531      5.085      1.623      0.105
-1.715      18.221
sport_cross-country skiing       24.7394      4.664      5.305      0.000
15.597      33.882
sport_hiking                    -24.2736      6.096     -3.982      0.000
-36.223     -12.324
sport_indoor cycling             16.4152      4.343      3.779      0.000
7.901      24.930
sport_kayaking                   28.8133      5.683      5.070      0.000
17.672      39.954
sport_mountain bike              10.1841      1.893      5.380      0.000
6.473      13.895
sport_orienteering               19.4365      2.255      8.619      0.000
15.016      23.857
sport_rowing                      7.8534      5.129      1.531      0.126
-2.201      17.908
sport_run                        14.2131      1.790      7.942      0.000
10.705      17.721
sport_skate                      -8.2814     14.396     -0.575      0.565
-36.501      19.939
sport_soccer                     34.2873     10.261      3.342      0.001
14.173      54.401
sport_tennis                     19.7091     14.395      1.369      0.171
-8.509      47.928
sport_walk                      -34.9324      3.498     -9.986      0.000
-41.790     -28.075
sport_weight training            13.4904     10.292      1.311      0.190
-6.685      33.666
gender_female                    -7.9623      2.330     -3.417      0.001
-12.530      -3.395
gender_male                      -9.2769      2.168     -4.279      0.000
-13.527      -5.027
==============================================================================
Omnibus:                      164.000   Durbin-Watson:                   1.995
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              213.815
Skew:                          -0.300   Prob(JB):                     3.72e-47
Kurtosis:                       3.638   Cond. No.                     5.99e+15
==============================================================================

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly
specified.
```

```
[2] The smallest eigenvalue is 2.23e-23. This might indicate that there are
strong multicollinearity problems or that the design matrix is singular.
```

We can see that the linear regression model has a poor performance: the R^2 is only 0.106. Just in case, we can explore the collinearity using VIF, as defined below. Note that we do not calculate the VIF for the one-hot-encoded variables because the VIF will definitely be infinity as they must add to 1 by construction.

```
[9]: from statsmodels.stats.outliers_influence import variance_inflation_factor

     def VIF(df, columns):

         values = sm.add_constant(df[columns]).values  # the dataframe passed to VIF␣
      ↪must include the intercept term
         num_columns = len(columns)+1
         vif = [variance_inflation_factor(values, i) for i in range(num_columns)]

         return pd.Series(vif[1:], index=columns)
```

```
[44]: VIF(X_train_linreg, ['latitude', 'longitude', 'altitude', 'derived_speed',␣
       ↪'distance'])
```

```
[44]: latitude         1.267990
      longitude        1.305368
      altitude         1.047534
      derived_speed    1.002630
      distance         1.001181
      dtype: float64
```

We can see that all the VIF values for numerical variables are well below 5, so there is no severe collinearity. Therefore, we just need to remove the variables with p-values > 0.05.

```
[88]: var_large_p_values = ['latitude', 'distance', 'sport_basketball', 'sport_core␣
       ↪stability training', 'sport_rowing',
                            'sport_skate', 'sport_tennis', 'sport_weight training']

      X_train_linreg = X_train_linreg.drop(var_large_p_values, axis=1)
      X_test_linreg = sm.add_constant(X_test.drop(var_large_p_values, axis=1))

      reg = sm.OLS(y_train, X_train_linreg).fit()
      print(reg.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:         tar_heart_rate   R-squared:                       0.105
Model:                            OLS   Adj. R-squared:                  0.103
Method:                 Least Squares   F-statistic:                     46.31
Date:                Wed, 07 Dec 2022   Prob (F-statistic):          2.29e-147
Time:                        15:58:35   Log-Likelihood:                -27687.
```

```
No. Observations:                6700   AIC:                         5.541e+04
Df Residuals:                    6682   BIC:                         5.553e+04
Df Model:                          17
Covariance Type:            nonrobust
==================================================================================
==============
                                  coef    std err          t      P>|t|
[0.025      0.975]
----------------------------------------------------------------------------------
--------------
const                         165.3643      3.675     44.997      0.000
158.160     172.569
longitude                       0.0162      0.004      3.709      0.000
0.008       0.025
altitude                        1.7918      0.174     10.307      0.000
1.451       2.133
derived_speed                   0.0372      0.019      1.999      0.046
0.001       0.074
sport_bike                     -0.8827      3.186     -0.277      0.782
-7.128       5.362
sport_bike (transport)         -1.9260      3.429     -0.562      0.574
-8.649       4.797
sport_circuit training         13.4744      6.213      2.169      0.030
1.295      25.654
sport_cross-country skiing     16.7470      5.554      3.015      0.003
5.859      27.635
sport_hiking                  -31.9535      6.937     -4.606      0.000
-45.553     -18.354
sport_indoor cycling            8.6072      5.260      1.636      0.102
-1.704      18.919
sport_kayaking                 21.3078      6.534      3.261      0.001
8.499      34.116
sport_mountain bike             2.7174      3.254      0.835      0.404
-3.661       9.096
sport_orienteering             11.5261      3.500      3.293      0.001
4.666      18.387
sport_run                       6.5040      3.185      2.042      0.041
0.260      12.748
sport_soccer                   27.1165     11.145      2.433      0.015
5.270      48.963
sport_walk                    -42.7805      4.448     -9.617      0.000
-51.500     -34.061
gender_female                  -8.0189      2.309     -3.473      0.001
-12.545      -3.492
gender_male                    -9.2627      2.155     -4.299      0.000
-13.486      -5.039
==============================================================================
Omnibus:                      166.340   Durbin-Watson:                   1.994
```

```
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              216.583
Skew:                          -0.303   Prob(JB):                     9.32e-48
Kurtosis:                       3.639   Cond. No.                     3.32e+03
==============================================================================

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly
specified.
[2] The condition number is large, 3.32e+03. This might indicate that there are
strong multicollinearity or other numerical problems.
```

We can observe that there are still coefficients with p-value greater than 0.05, and they all seem to be related to bike (`bike`, `bike (transport)`, `indoor cycling`, `mountain bike`). As discussed in EDA, we only keep `bike`, which has the most workouts.

```
[89]: var_large_p_values2 = ['sport_bike (transport)', 'sport_indoor cycling',␣
       ↪'sport_mountain bike']

      X_train_linreg = X_train_linreg.drop(var_large_p_values2, axis=1)
      X_test_linreg = X_test_linreg.drop(var_large_p_values2, axis=1)

      reg = sm.OLS(y_train, X_train_linreg).fit()
      print(reg.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:         tar_heart_rate   R-squared:                       0.104
Model:                            OLS   Adj. R-squared:                  0.102
Method:                 Least Squares   F-statistic:                     55.22
Date:                Wed, 07 Dec 2022   Prob (F-statistic):          3.42e-147
Time:                        16:00:46   Log-Likelihood:                -27693.
No. Observations:                6700   AIC:                         5.542e+04
Df Residuals:                    6685   BIC:                         5.552e+04
Df Model:                          14
Covariance Type:            nonrobust
================================================================================
==============
                                 coef    std err          t      P>|t|
[0.025      0.975]
--------------------------------------------------------------------------------
--------------
const                        167.0133      2.226     75.045      0.000
162.651     171.376
longitude                      0.0144      0.004      3.318      0.001
0.006       0.023
altitude                       1.7844      0.174     10.265      0.000
1.444       2.125
derived_speed                  0.0372      0.019      2.000      0.046
```

```
0.001       0.074
sport_bike                       -2.6135      0.656     -3.985      0.000
-3.899      -1.328
sport_circuit training           11.7381      5.378      2.183      0.029
1.196       22.280
sport_cross-country skiing       15.0344      4.596      3.271      0.001
6.025       24.044
sport_hiking                    -33.7487      6.200     -5.444      0.000
-45.902     -21.595
sport_kayaking                   19.5240      5.746      3.398      0.001
8.261       30.787
sport_orienteering                9.7821      1.586      6.166      0.000
6.672       12.892
sport_run                         4.7575      0.662      7.183      0.000
3.459       6.056
sport_soccer                     25.2735     10.707      2.360      0.018
4.284       46.263
sport_walk                      -44.4671      3.252    -13.673      0.000
-50.843     -38.092
gender_female                    -7.9560      2.308     -3.447      0.001
-12.481      -3.431
gender_male                      -9.1267      2.148     -4.250      0.000
-13.336      -4.917
==============================================================================
Omnibus:                      168.153   Durbin-Watson:                   1.993
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              218.976
Skew:                          -0.305   Prob(JB):                     2.82e-48
Kurtosis:                       3.641   Cond. No.                     2.78e+03
==============================================================================

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly
specified.
[2] The condition number is large, 2.78e+03. This might indicate that there are
strong multicollinearity or other numerical problems.
```

We can see that all p-values are below 0.05. We can then calculate the Out-of-Sample R2 to see how the model perform on the test set

[16]:
```python
def OSR2(y_train, y_pred, y_test):
    SSE = np.sum((y_test - y_pred)**2)
    SST = np.sum((y_test - np.mean(y_train))**2)
    return 1 - SSE/SST
```

[90]:
```python
y_pred = reg.predict(X_test_linreg)
OSR2(y_train, y_pred, y_test)
```

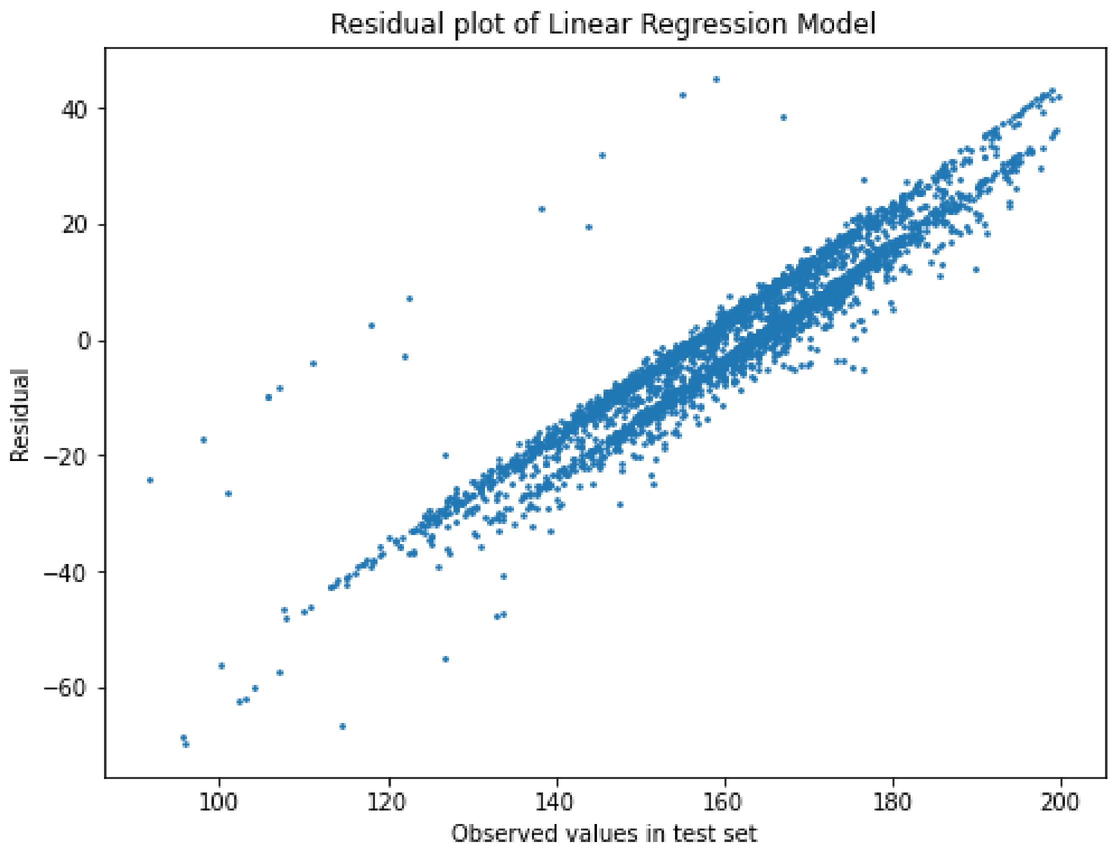
[90]:
```
0.08905431650590756
```

Although the linear regression model performs poorly on the test set as well, we can see that the OSR2 (0.089) is not too far away from the training R2 (0.104), which means we did not overfit the data. We are either underfitting the data, which means we need more features, or linear models are simply not suitable for the data.

We can visualize the residual against the actual observed values in the test set:

[63]:
```
import matplotlib.pyplot as plt

plt.figure(figsize=(8, 6))
plt.scatter(y_test, y_test - y_pred, s=3)
plt.title('Residual plot of Linear Regression Model')
plt.xlabel('Observed values in test set')
plt.ylabel('Residual')
```

[63]:
```
Text(0, 0.5, 'Residual')
```



We can see that in most part the residuals are greater than 0 for higher observed heart rates, and less than 0 for lower observed heart rates, which means the model is over-predicting low heart rates and under-predicting high heart rates. We may conclude that the relationship between heart rates and other features could not be well explained linearly.

In order to capture the non-linear relationship, we can fit a Random Forest model with a 5-fold cross validation to find the optimal `max_features` parameter of the random forest regressor.

```
[49]: len(X_train.columns.values)
```

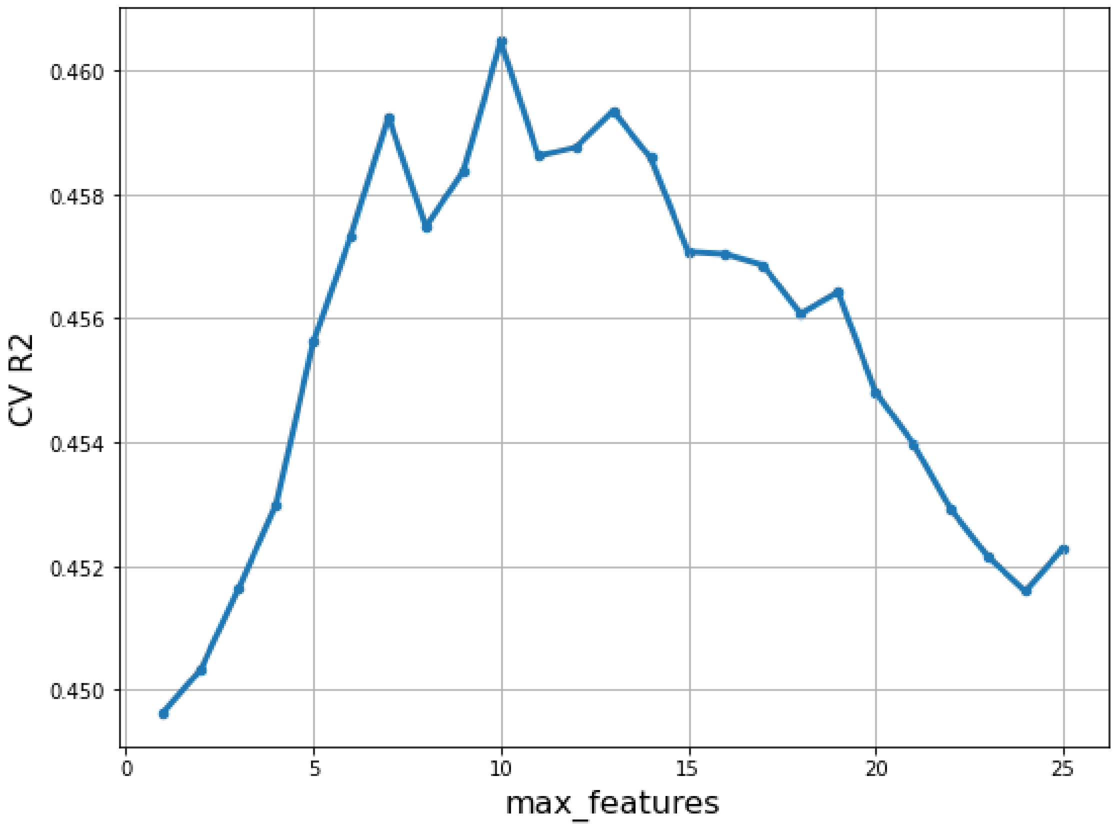
```
[49]: 25
```

```
[92]: from sklearn.ensemble import RandomForestRegressor
      from sklearn.model_selection import GridSearchCV

      grid_values = {'max_features': np.linspace(1, 25, 25, dtype = 'int32'),
                     'criterion': ['mse'],
                     #'min_samples_leaf': [5],
                     'n_estimators': [500],
                     'random_state': [88]}

      rf = RandomForestRegressor()
      rf_cv = GridSearchCV(rf, param_grid=grid_values, scoring='r2', cv=5, verbose=1)
      rf_cv.fit(X_train, y_train)
```

```
Fitting 5 folds for each of 25 candidates, totalling 125 fits

[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done 125 out of 125 | elapsed: 12.7min finished
```

```
[92]: GridSearchCV(cv=5, estimator=RandomForestRegressor(),
                   param_grid={'criterion': ['mse'],
                               'max_features': array([ 1,  2,  3,  4,  5,  6,  7,  8,
       9, 10, 11, 12, 13, 14, 15, 16, 17,
              18, 19, 20, 21, 22, 23, 24, 25], dtype=int32),
                               'n_estimators': [500], 'random_state': [88]},
                   scoring='r2', verbose=1)
```

```
[93]: max_features = rf_cv.cv_results_['param_max_features'].data
      ACC_scores = rf_cv.cv_results_['mean_test_score']

      plt.figure(figsize=(8, 6))
      plt.xlabel('max_features', fontsize=16)
      plt.ylabel('CV R2', fontsize=16)
      plt.scatter(max_features, ACC_scores, s=20)
      plt.plot(max_features, ACC_scores, linewidth=3)
      plt.grid(True, which='both')

      plt.tight_layout()
      plt.show()

      print('Best parameters', rf_cv.best_params_)
      print('Best training R-square', round(rf_cv.best_score_, 4))
```



```
Best parameters {'criterion': 'mse', 'max_features': 10, 'n_estimators': 500,
'random_state': 88}
Best training R-square 0.4605
```

The `max_features` that produces the highest training R^2 is 10, with training R^2 of 0.4605. Compared to the training R^2 of 0.104 in the linear regression model, the random forest regressor performs much better. Yet 0.4605 is still not a very decent R^2.

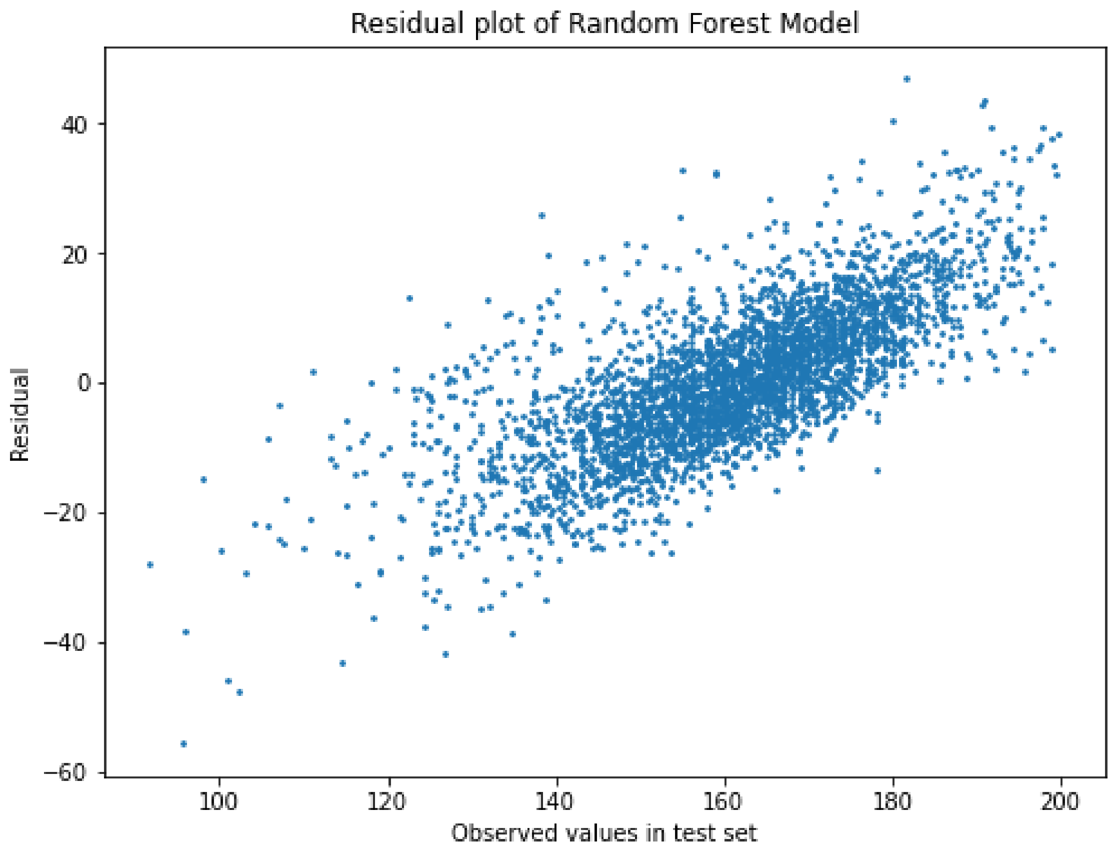
[94]:
```
y_pred = rf_cv.predict(X_test)
print('OSR2:', OSR2(y_train, y_pred, y_test))
```

```
OSR2: 0.48200717800456794
```

The OSR2 is even higher that the training R^2, which means the random forest regressor does not have an overfitting problem as well. We can plot the residual on the test set.

[60]:
```
plt.figure(figsize=(8, 6))
plt.scatter(y_test, y_test - y_pred, s=3)
plt.title('Residual plot of Random Forest Model')
plt.xlabel('Observed values in test set')
plt.ylabel('Residual')
```

[60]:
```
Text(0, 0.5, 'Residual')
```



We can see that more points are closer to 0 compared to the residual plot of the linear regression model, meaning that the error is smaller and the model's performance on the test set is better. However, the range of the residual is still very large (from -60 to +40), and we can still observe a rough trend of over-predicting low heart rates and under-predicting high heart rates.

Since random forest model does not have coefficients for the variables, we check the importance scores for the variables to see their influence on the model.

```
[54]: pd.DataFrame({'Feature': X_train.columns,
                     'Importance Score': 100*rf_cv.best_estimator_.
       ↪feature_importances_}).round(2)
```

```
[54]:                       Feature  Importance Score
      0                    latitude             23.88
      1                   longitude             19.34
      2                    altitude             16.84
      3               derived_speed             17.25
      4                    distance             12.95
      5            sport_basketball              0.00
      6                  sport_bike              1.94
      7        sport_bike (transport)            0.70
```

```
8          sport_circuit training          0.01
9   sport_core stability training          0.02
10     sport_cross-country skiing          0.02
11                   sport_hiking          0.38
12           sport_indoor cycling          0.02
13                 sport_kayaking          0.05
14            sport_mountain bike          0.48
15             sport_orienteering          0.24
16                   sport_rowing          0.01
17                      sport_run          2.39
18                    sport_skate          0.01
19                   sport_soccer          0.02
20                   sport_tennis          0.00
21                     sport_walk          2.40
22          sport_weight training          0.00
23                  gender_female          0.51
24                    gender_male          0.53
```

To improve the performance, we can also build a random forest classifier. According to CDC (https://www.cdc.gov/physicalactivity/basics/measuring/heartrate.htm), the maximum heart rate can be calculated with the formula $MaximumHeartRate = 220 - Age$. For example, for an 30-year-old person, the maximum heart rate would be $220 - 30 = 190$. For vigorous-intensity physical activity, the target heart rate should be between 77% and 93% of the maximum heart rate. In other words, a heart rate above 93% of the maximum heart rate could signify an overwhelming intensity of the sport. We can use a random forest classifier to predict the sports that could raise such problems.

Since we do not have the age data for the users, we may assume an age of 30 to avoid unnecessary warnings. The 93% of the maximum heart rate would be $190 \times 93\% = 176.7$. We first create a new `y` which contains `0` if the heart rate is below 177 and `1` if the heart rate is equal to or above 177.

[66]:
```
y_train_new = (y_train >= 177).astype('int')
y_test_new = (y_test >= 177).astype('int')
```

[76]:
```
from sklearn.ensemble import RandomForestClassifier

grid_values = {'max_features': np.linspace(1, 25, 25, dtype = 'int32'),
               'min_samples_leaf': [5],
               'n_estimators': [500],
               'random_state': [88]}

rfc = RandomForestClassifier()
rfc_cv = GridSearchCV(rfc, param_grid=grid_values, scoring='accuracy', cv=5,␣
 ↪verbose=1)
rfc_cv.fit(X_train, y_train_new)
```

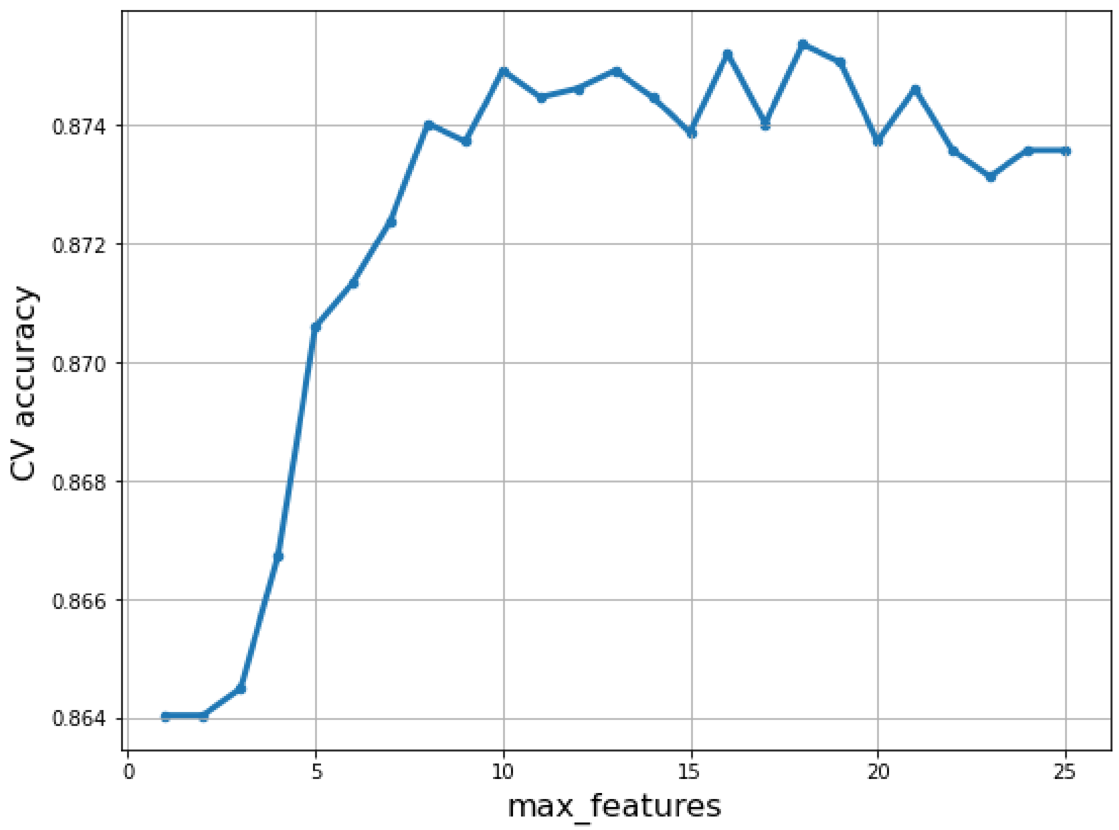
```
Fitting 5 folds for each of 25 candidates, totalling 125 fits
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
```

```
[Parallel(n_jobs=1)]: Done 125 out of 125 | elapsed: 12.0min finished
```

```
[76]: GridSearchCV(cv=5, estimator=RandomForestClassifier(),
                   param_grid={'max_features': array([ 1,  2,  3,  4,  5,  6,  7,  8,
      9, 10, 11, 12, 13, 14, 15, 16, 17,
             18, 19, 20, 21, 22, 23, 24, 25], dtype=int32),
                               'min_samples_leaf': [5], 'n_estimators': [500],
                               'random_state': [88]},
                   scoring='accuracy', verbose=1)
```

```
[80]: max_features = rfc_cv.cv_results_['param_max_features'].data
      acc_scores = rfc_cv.cv_results_['mean_test_score']

      plt.figure(figsize=(8, 6))
      plt.xlabel('max_features', fontsize=16)
      plt.ylabel('CV accuracy', fontsize=16)
      plt.scatter(max_features, acc_scores, s=20)
      plt.plot(max_features, acc_scores, linewidth=3)
      plt.grid(True, which='both')

      plt.tight_layout()
      plt.show()

      print('Best parameters', rf_cv.best_params_)
      print('Best training accuracy', round(rfc_cv.best_score_, 4))
```



```
Best parameters {'criterion': 'gini', 'max_features': 16, 'n_estimators': 500,
'random_state': 88}
Best training accuracy 0.8754
```

The `max_features` that produces the highest accuracy is 16, with accuracy of 0.8754.

We then calculate the confusion matrix on the test set to evaluate the model's performance.

[82]:
```
from sklearn.metrics import confusion_matrix

y_pred_new = rfc_cv.predict(X_test)
tn, fp, fn, tp = confusion_matrix(y_test_new, y_pred_new).ravel()
print(f'True Negative: {tn}\tFalse Positive: {fp}')
print(f'False Negative: {fn}\tTrue Positive: {tp}')
```

```
True Negative: 2798     False Positive: 45
False Negative: 357     True Positive: 100
```

Using the confusion matrix, we can calculate some key metrics:

[83]:
```
acc = (tp + tn) / (tn + fp + fn + tp)
precision = tp / (tp + fp)
recall = tp / (tp + fn)
```

```
far = fp / (tn + fp)
print(f'Accuracy: {round(acc, 4)}\tPrecision: {round(precision, 4)}')
print(f'Recall: {round(recall, 4)}\tFalse Alarm Rate: {round(far, 4)}')
```

```
Accuracy: 0.8782        Precision: 0.6897
Recall: 0.2188  False Alarm Rate: 0.0158
```

Calculate the same set of metrics for the baseline model, which in this case is a zero estimator which simply predict 0 for all y (i.e. no warning raised) since 0 is the mode of y.

[84]:
```
acc_zero = 1 - sum(y_test_new)/len(y_test_new)
recall_zero = 0
far_zero = 0
print('Zero estimater:')
print(f'Accuracy: {round(acc_zero, 4)}\tPrecision: N/A')
print(f'Recall: {round(recall_zero, 4)}\tFalse Alarm Rate: {round(far_zero,␣
↪4)}')
```

```
Zero estimater:
Accuracy: 0.8615        Precision: N/A
Recall: 0       False Alarm Rate: 0
```

We can see that the accuracy of the random forest classifier is in fact not much higher than that of the zero estimator. The random forest model's recall is also only 0.2188, which means a large portion of heart rates that should raise a warning is still not detected. However, since the random forest model improves the recall by 0.2188 and only under a sacrifice of 0.0158 false alarm rate, it is still useful.

[81]:
```
pd.DataFrame({'Feature': X_train.columns,
              'Importance Score': 100*rfc_cv.best_estimator_.
↪feature_importances_}).round(2)
```

[81]:
```
                        Feature  Importance Score
0                      latitude             28.51
1                     longitude             21.38
2                      altitude             13.29
3                 derived_speed             17.12
4                      distance             13.66
5              sport_basketball              0.00
6                    sport_bike              2.05
7         sport_bike (transport)             0.10
8         sport_circuit training             0.01
9   sport_core stability training            0.00
10     sport_cross-country skiing            0.06
11                 sport_hiking              0.00
12         sport_indoor cycling              0.04
13                sport_kayaking             0.08
14           sport_mountain bike             1.31
15           sport_orienteering              0.93
```

```
16               sport_rowing          0.00
17                  sport_run          1.00
18                sport_skate          0.00
19               sport_soccer          0.00
20               sport_tennis          0.00
21                 sport_walk          0.01
22      sport_weight training          0.00
23              gender_female          0.12
24                gender_male          0.33
```

The importance score table is similar to that of the random forest regressor.

```
[ ]:
```

```
In [ ]: import pandas as pd
        from sklearn.neural_network import MLPClassifier
        from sklearn.svm import SVC

        from sklearn.preprocessing import StandardScaler, MinMaxScaler
        from sklearn.preprocessing import LabelEncoder, OneHotEncoder
        from sklearn.feature_extraction import DictVectorizer
        from sklearn.decomposition import PCA #Principal Component Analysis
        from sklearn.manifold import TSNE #T-Distributed Stochastic Neighbor Embedding
        from sklearn.cluster import KMeans #K-Means Clustering

        from sklearn.pipeline import Pipeline
        from sklearn.metrics import accuracy_score
        from sklearn.model_selection import train_test_split
        from sklearn.model_selection import GridSearchCV, ParameterGrid

        from scipy.stats import uniform, randint
        from sklearn.metrics import auc, accuracy_score, confusion_matrix, mean_squared_error
        from sklearn.model_selection import cross_val_score, GridSearchCV, KFold, RandomizedSearchCV, train_test_split
        import numpy as np

        import collections
        from sklearn.cluster import KMeans
        from sklearn.metrics import silhouette_score
        import numpy as np
        from sklearn.preprocessing import normalize

        import matplotlib
        import matplotlib.pyplot as plt

        import plotly as py
        import plotly.graph_objs as go
        from plotly.offline import download_plotlyjs, init_notebook_mode, plot, iplot
```

Sample

This part only used in the first time processing, that is aimed to export seeveral new, useful data sets, including the `full_data` that consists of the first 10000 records in the npy file, `heart_data`, `altitude_data`, and `speed_data`, and those three data sets are Descriptive Statistics by each record id.

Don't run the following code once you have saved the 4 csv files on the computer.

```
In [ ]: data = np.load("/Users/kai/Downloads/processed_endomondoHR_proper_interpolate.npy", allow_pickle=True)[0]
```

```
In [ ]: df = pd.DataFrame()
```

```
In [ ]: for i in range(10000):
            df = pd.concat([df, pd.DataFrame(data[i])])
```

```
In [ ]: df.head()
```

Out[ ]:

	since_begin	time_elapsed	latitude	gender	tar_heart_rate	timestamp	id	longitude	since_last	heart_rate	sport	altitude
0	1.378479e+06	-0.122568	60.173349	male	100.000000	1408898746	396826535	24.649770	2158.846078	-8.197369	bike	-1.804467
1	1.378479e+06	-0.122122	60.173240	male	113.355469	1408898754	396826535	24.650143	2158.846078	-5.369012	bike	-1.818636
2	1.378479e+06	-0.121676	60.172980	male	120.214752	1408898765	396826535	24.650911	2158.846078	-3.916386	bike	-1.820717
3	1.378479e+06	-0.121230	60.172478	male	119.108221	1408898778	396826535	24.650669	2158.846078	-4.150721	bike	-1.847772
4	1.378479e+06	-0.120784	60.171861	male	120.569362	1408898794	396826535	24.649145	2158.846078	-3.841288	bike	-1.851729

```
In [ ]: user = df[['id','gender','sport','userId']].drop_duplicates()
```

Generate two new data sets for project use

```
In [ ]: user.to_csv('/Users/kai/Desktop/user_data.csv')
```

```
In [ ]: df.to_csv('/Users/kai/Desktop/full_data.csv')
```

Aggregate heart data

```
In [ ]: meta = df.groupby(['id']).agg({'tar_heart_rate':['min','max','mean','median','var','std']})
        meta
```

Out[ ]:

	tar_heart_rate					
	min	max	mean	median	var	std
id						
3930381	107.000000	133.188170	126.052616	127.690308	24.679526	4.967849
3933514	102.737612	147.056557	129.107529	129.824025	40.781166	6.386013
3940962	119.000000	153.203170	135.745412	135.111115	51.729728	7.192338
4632763	99.000000	144.093784	125.688475	126.154797	50.713629	7.121350
4651866	95.000000	149.000000	131.088789	131.271737	65.589959	8.098763
...	...	...	...	...	...	...
651598821	105.000000	180.056087	153.262615	155.055382	237.087531	15.397647
651793414	92.000000	167.947758	144.486555	143.746036	76.864410	8.767235
652776545	78.000000	188.201302	148.425898	148.420723	607.459492	24.646693
656149214	104.000000	171.100294	162.785552	165.000000	74.066952	8.606216
657584281	89.532329	170.606347	144.081080	148.035233	285.101349	16.884944

10000 rows × 6 columns

In [ ]:
```python
meta.columns = meta.columns.droplevel()
```

In [ ]:
```python
meta = meta.reset_index()
```

In [ ]:
```python
# meta.to_csv('/Users/kai/Desktop/heart_data.csv')
```

Aggregate altitude data

In [ ]:
```python
meta_al = data.groupby(['id']).agg({'altitude':['min','max','mean','median','var','std']})
meta_al
```

Out[ ]:

	altitude					
	min	max	mean	median	var	std
id						
3930381	-2.274128	-1.951918	-2.117251	-2.119646	0.006619	0.081360
3933514	-2.413336	-1.563304	-2.087596	-2.120396	0.028579	0.169054
3940962	-2.391011	-1.352288	-2.109420	-2.256160	0.089336	0.298892
4632763	-1.752364	-1.295748	-1.636443	-1.659474	0.009159	0.095702
4651866	-2.218926	-1.546072	-1.983237	-2.056175	0.033427	0.182830
...	...	...	...	...	...	...
651598821	-2.495599	-1.458820	-2.022786	-1.958346	0.113292	0.336589
651793414	0.920737	1.192892	1.059817	1.062778	0.006510	0.080686
652776545	7.901693	8.254716	8.017489	7.993179	0.007225	0.085002
656149214	8.998983	10.753798	9.793342	9.657995	0.362602	0.602164
657584281	-2.447059	1.716663	-0.614704	-1.075103	2.207662	1.485820

10000 rows × 6 columns

In [ ]:
```python
meta_al.columns = meta_al.columns.droplevel()
meta_al = meta_al.reset_index()
# meta_al.to_csv('/Users/kai/Desktop/altitude_data.csv')
```

Aggregate Speed

In [ ]:
```python
data = pd.read_csv("/Users/kai/Course/Data100/full_data.csv", index_col=0)
meta_speed = data.groupby(['id']).agg({'derived_speed':['min','max','mean','median','var','std']})
```

In [ ]:
```python
meta_speed.columns = meta_speed.columns.droplevel()
meta_speed.reset_index().to_csv('/Users/kai/Course/Data100/speed_data.csv')
```

Clustering Analysis

Data preparation

In this sector, I want to have clean data sets with import variables to portain their sport behavior and health status. The granularity of the final data set would be the user, specifically, the median of each exercise of each user. And the variables I need are max heart rate `heart_max`, differences of the altitude in a period of time `altitude_diff`, average speed in a period of time `speed_mean`, and `gender`.

In [ ]:
```
data = pd.read_csv("/Users/kai/Course/Data100/full_data.csv", index_col=0)
data = data[data['gender'] != 'unknown']
```

drop the data points with unvalid gender

Merge `data` to the existing three data sets of the statistics for each exercise.

In [ ]:
```
alt_data = pd.read_csv("/Users/kai/Course/Data100/altitude_data.csv", index_col=0)
heart_data = pd.read_csv("/Users/kai/Course/Data100/heart_data.csv", index_col=0)
speed_data = pd.read_csv("/Users/kai/Course/Data100/speed_data.csv", index_col=0)
```

In [ ]:
```
data.head()
```

Out[ ]:

	since_begin	time_elapsed	latitude	gender	tar_heart_rate	timestamp	id	longitude	since_last	heart_rate	sport	altitude
0	1.378479e+06	-0.122568	60.173349	male	100.000000	1408898746	396826535	24.649770	2158.846078	-8.197369	bike	-1.804467
1	1.378479e+06	-0.122122	60.173240	male	113.355469	1408898754	396826535	24.650143	2158.846078	-5.369012	bike	-1.818636
2	1.378479e+06	-0.121676	60.172980	male	120.214752	1408898765	396826535	24.650911	2158.846078	-3.916386	bike	-1.820717
3	1.378479e+06	-0.121230	60.172478	male	119.108221	1408898778	396826535	24.650669	2158.846078	-4.150721	bike	-1.847772
4	1.378479e+06	-0.120784	60.171861	male	120.569362	1408898794	396826535	24.649145	2158.846078	-3.841288	bike	-1.851729

In [ ]:
```
heart_data.head()
```

Out[ ]:

	id	min	max	mean	median	var	std
0	3930381	107.000000	133.188170	126.052616	127.690308	24.679526	4.967849
1	3933514	102.737612	147.056557	129.107529	129.824025	40.781166	6.386013
2	3940962	119.000000	153.203170	135.745412	135.111115	51.729728	7.192338
3	4632763	99.000000	144.093784	125.688475	126.154797	50.713629	7.121350
4	4651866	95.000000	149.000000	131.088789	131.271737	65.589959	8.098763

In [ ]:
```
alt_data['diff'] = alt_data['max'] - alt_data['min']
```

In [ ]:
```
data_selected= data[['gender','sport','altitude','derived_speed','id','userId']]
data_merged = (
    data_selected.merge(heart_data[['max','id','std']], on='id')
    .merge(alt_data[['diff','std','id']], on='id')
    .merge(speed_data[['mean','std','id']], on='id')
)
```

In [ ]:
```
data_merged = data_merged.rename({'diff':'alt_diff','max':'heart_max','mean':'speed_mean'}, axis=1)
```

In [ ]:
```
data_merged
```

Out[ ]:

	gender	sport	altitude	derived_speed	id	userId	heart_max	std_x	alt_diff	std_y	speed_mean	std
0	male	bike	-1.804467	-7.082944	396826535	10921915	169.177154	10.119547	0.767201	0.226943	2.889815	2.510836
1	male	bike	-1.818636	-2.088780	396826535	10921915	169.177154	10.119547	0.767201	0.226943	2.889815	2.510836
2	male	bike	-1.820717	-0.351569	396826535	10921915	169.177154	10.119547	0.767201	0.226943	2.889815	2.510836
3	male	bike	-1.847772	-0.680039	396826535	10921915	169.177154	10.119547	0.767201	0.226943	2.889815	2.510836
4	male	bike	-1.851729	-0.279256	396826535	10921915	169.177154	10.119547	0.767201	0.226943	2.889815	2.510836
...	...	...	...	...	...	...	...	...	...	...	...	...
2977195	male	bike	-0.064222	-0.209647	176731991	331586	160.945059	14.183650	0.571151	0.140918	-0.413759	2.906451
2977196	male	bike	-0.078347	-1.329734	176731991	331586	160.945059	14.183650	0.571151	0.140918	-0.413759	2.906451
2977197	male	bike	-0.105896	-0.006515	176731991	331586	160.945059	14.183650	0.571151	0.140918	-0.413759	2.906451
2977198	male	bike	-0.124999	-3.231240	176731991	331586	160.945059	14.183650	0.571151	0.140918	-0.413759	2.906451
2977199	male	bike	-0.165820	-0.534229	176731991	331586	160.945059	14.183650	0.571151	0.140918	-0.413759	2.906451

2977200 rows × 12 columns

Now, let's preprocess categorical variables `sport`, into dummy variables.

In [ ]:
```
categorical = ['sport']
df_dummies = pd.get_dummies(data_merged[categorical], columns=categorical)
data_merged = data_merged.drop(categorical, axis = 1)
data_merged = data_merged.replace({'male':1, 'female':0}).merge(df_dummies, left_index = True, right_index= True)
```

In [ ]:
```
data['sport'].value_counts()
```

```
Out[ ]: bike                       1416000
        run                        1196100
        mountain bike               213600
        bike (transport)             62700
        orienteering                 48900
        walk                         10800
        indoor cycling                8100
        cross-country skiing          4800
        core stability training       3300
        rowing                        3300
        hiking                        3000
        kayaking                      2400
        circuit training              2400
        soccer                         600
        tennis                         300
        basketball                     300
        skate                          300
        weight training                300
        Name: sport, dtype: int64
```

```
In [ ]: data_merged = data_merged.drop(['altitude','derived_speed'],axis=1).drop_duplicates()
```

Let's label the heart_max over 180 as risky heart rates.

```
In [ ]: data_merged['heart_risk'] = data_merged['heart_max']>180
        data_merged['heart_risk'] = data_merged['heart_risk'].astype('int')
```

```
In [ ]: data_merged
```

Out[ ]:

	gender	id	userId	heart_max	std_x	alt_diff	std_y	speed_mean	std	sport_basketball	...	sport_mountain bike
0	1	396826535	10921915	169.177154	10.119547	0.767201	0.226943	2.889815	2.510836	0	...	0
300	1	392337038	10921915	172.577113	11.186082	0.726726	0.154062	3.310221	2.830463	0	...	0
600	1	389643739	10921915	162.156270	10.289886	0.668623	0.159920	2.280061	2.919536	0	...	0
900	1	386729739	10921915	178.140847	12.028911	0.758043	0.164667	3.436977	3.005276	0	...	0
1200	1	372368431	10921915	157.212850	13.193648	0.435232	0.111841	2.081560	2.453061	0	...	0
...	...	...	...	...	...	...	...	...	...	...	...	...
2975700	1	179541176	331586	166.025730	17.988476	0.688984	0.188743	0.974516	4.348646	0	...	0
2976000	1	179540799	331586	162.320343	7.113997	0.209992	0.053109	-2.964418	0.498286	0	...	0
2976300	1	178495706	331586	172.024154	11.566740	0.170236	0.039346	-2.131020	0.931947	0	...	0
2976600	1	176731930	331586	186.336447	16.392756	0.884187	0.222878	-3.330440	19.649112	0	...	0
2976900	1	176731991	331586	160.945059	14.183650	0.571151	0.140918	-0.413759	2.906451	0	...	0

9924 rows × 28 columns

```
In [ ]: X = data_merged.drop(['id','userId','heart_max','heart_risk'],axis=1)
        y = data_merged['heart_risk']
        X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, random_state=42)
```

```
In [ ]: X_train[:]
```

Out[ ]:

	gender	std_x	alt_diff	std_y	speed_mean	std	sport_basketball	sport_bike	sport_bike (transport)	sport_circuit training	...	sport_kayaki
390600	1	6.631605	0.215063	0.046746	-2.692430	0.544458	0	0	0	0	...	
1844400	1	12.941896	1.758425	0.405411	-2.265545	0.931623	0	0	0	0	...	
2471100	1	11.558490	0.463067	0.083175	8.996960	2.885595	0	1	0	0	...	
2365800	1	9.411876	1.444414	0.528165	2.322927	3.378014	0	1	0	0	...	
2744100	1	18.481579	3.849187	1.126091	1.714599	6.072005	0	1	0	0	...	
...	...	...	...	...	...	...	...	...	...	...	...	
1720200	1	22.434826	0.594080	0.195675	-2.486322	7.430934	0	1	0	0	...	
1557300	1	17.726846	0.674364	0.173670	-4.023859	1.381357	0	0	0	0	...	
1617000	1	13.424453	0.551636	0.107820	3.804666	3.441075	0	1	0	0	...	
258000	1	15.614705	0.325232	0.068716	4.821659	1.893535	0	1	0	0	...	
2181000	1	11.542973	1.775486	0.511649	-1.334234	4.403683	0	1	0	0	...	

6649 rows × 24 columns

Built a RF model

```
In [ ]: from sklearn.ensemble import RandomForestClassifier
        # from sklearn.metrics import mean_squared_error as mse
        # from sklearn.metrics import r2_score as r2


        rfc_100 = RandomForestClassifier(n_estimators=300, random_state=90)
```

```
rfc_100.fit(X_train, y_train)
y_pred_100 = rfc_100.predict(X_test)

print('Model accuracy score with 300 decision-trees : {0:0.4f}'. format(accuracy_score(y_test, y_pred_100)))
```

```
Model accuracy score with 300 decision-trees : 0.9002
```

The data scattered unevenly among different sort of sports. So, I would only focus on the top three sports, `bike`, `run`, and `mountain bike`, otherwise there woould be too few data points to perform clustering analysis

function `sport_subset` and `agg_median` helps us to get data set by sport and results in the users as the granularity.

In [ ]:
```
def sport_subset(sport):
    return data_merged[data_merged[sport]==1][['id','userId','gender','speed_mean','heart_max','alt_diff']].drop_duplicate
```

In [ ]:
```
data_bike = sport_subset('sport_bike')
data_run = sport_subset('sport_run')
data_mountain_bike = sport_subset('sport_mountain bike')
```

In [ ]:
```
data_bike.head()
```

Out[ ]:

	id	userId	gender	speed_mean	heart_max	alt_diff
0	396826535	10921915	1	2.889815	169.177154	0.767201
300	392337038	10921915	1	3.310221	172.577113	0.726726
600	389643739	10921915	1	2.280061	162.156270	0.668623
900	386729739	10921915	1	3.436977	178.140847	0.758043
1200	372368431	10921915	1	2.081560	157.212850	0.435232

In [ ]:
```
def agg_median(data):
    data = data.groupby('userId')[['gender','speed_mean','heart_max','alt_diff']].median().reset_index()
    cat = ['speed_mean','heart_max','alt_diff']
    for i in cat:
        data[i]= (data[i]-data[i].mean())/data[i].std()
    return data
```

In [ ]:
```
agg_median(data_bike).head()
```

Out[ ]:

	userId	gender	speed_mean	heart_max	alt_diff
0	16786	1.0	0.775573	-0.062170	0.575141
1	22260	1.0	0.716639	1.152143	0.116120
2	56291	1.0	-0.649830	-1.222555	-0.476478
3	69228	1.0	1.091222	1.205620	0.071944
4	182042	0.0	-1.362690	0.195841	-0.390095

In [ ]:
```
user_bike = agg_median(data_bike)
```

In [ ]:
```
user_bike
```

Out[ ]:

	userId	gender	speed_mean	heart_max	alt_diff
0	16786	1.0	0.775573	-0.062170	0.575141
1	22260	1.0	0.716639	1.152143	0.116120
2	56291	1.0	-0.649830	-1.222555	-0.476478
3	69228	1.0	1.091222	1.205620	0.071944
4	182042	0.0	-1.362690	0.195841	-0.390095
...	...	...	...	...	...
64	13276532	1.0	1.933232	0.931127	0.584110
65	13279851	1.0	0.448589	-0.126503	-0.396488
66	13469928	1.0	1.377630	0.042564	-0.322247
67	13693003	1.0	0.684712	-0.259039	0.086967
68	14066832	1.0	0.045764	0.254184	-0.566734

69 rows × 5 columns

In [ ]:
```
user_run = agg_median(data_run)
```

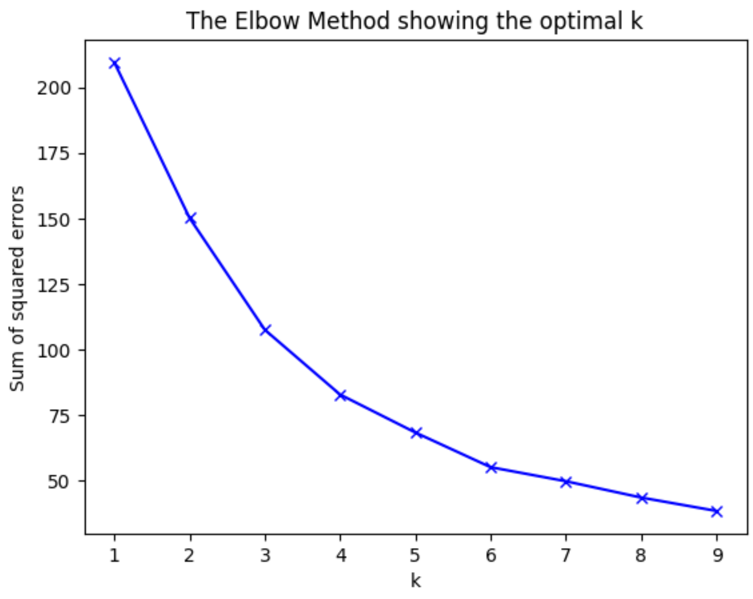
The `bike` data set have 69 users and each column records the median of the corresponding variable in all their exercises. For example, the column `speed_mean` refers to the median of the average speed of all a user's exercises, so it can be interpreted as median(avg(X)) as well. In addition, the `run` data set have 81 users and the `mountain bike` have 23 users.

Kmeans clustering

bike

So far only the bikers are analyzed, but the model is quite convenient to handle different sports. The concern is the description for the certain group that might with high risks of heart anomaly. Do they really would suffer some day? And what other characteristics they share in common that needs more scrutiny.

In [ ]:
```
X = user_bike[['gender','speed_mean','heart_max','alt_diff']]

# k means determine k
SSE = []
K = range(1,10)
for k in K:
    kmeans = KMeans(n_clusters=k, random_state=42)
    kmeans.fit(X)
    SSE.append(kmeans.inertia_)

# Plot the elbow
plt.plot(K, SSE, 'bx-')
plt.xlabel('k')
plt.ylabel('Sum of squared errors')
plt.title('The Elbow Method showing the optimal k')
plt.show()
```



Optimal k is 4, as it is the elbow point on the curve.

Also, we are not intended for a large k, because too many clusters cause a lot of trouble in analysis.

In [ ]:
```
kmeans = KMeans(n_clusters=4, random_state=42)
kmeans.fit(X)

user_bike['label'] = kmeans.labels_
```

In [ ]:
```
np.round(kmeans.cluster_centers_,1)
# 'speed_mean','heart_max','alt_diff'
```

Out[ ]:
```
array([[ 0.9, -1. ,  0.8, -0.3],
       [ 0.9,  0.7,  0.2, -0.1],
       [ 1. , -1.4,  0.1,  3.7],
       [ 0.9, -0.5, -1.4, -0.3]])
```

A simple tentative analysis

row 1: low speed, high max heart rate, moderate alt diff. This group indicates heart anomaly and entails careful analysis.

row 2: high speed, moderate max heart rate, moderate alt diff. Road biker

row 3: low speed, moderate max heart rate, high alt diff. Maybe advanced mountain off-road biker.

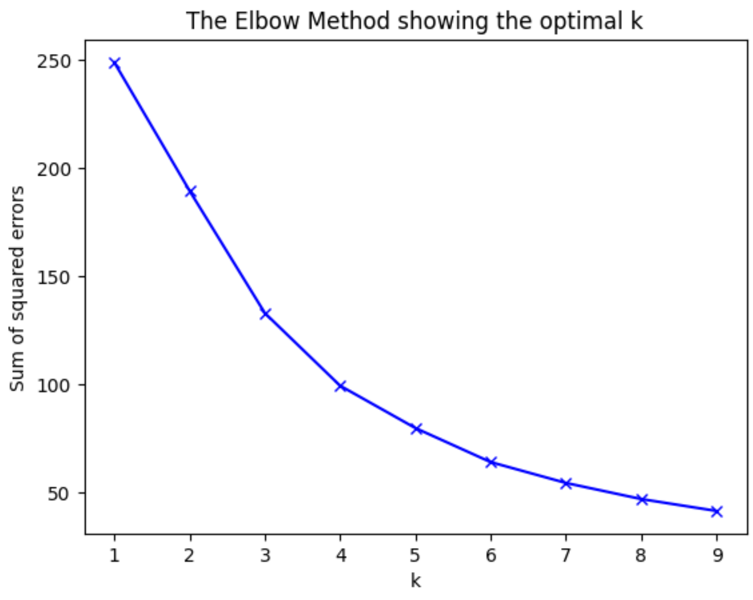
row 4: moderate speed, low max heart rate, moderate alt diff. Beginner cyclist

In [ ]:
```
user_bike['label'].value_counts()
```

Out[ ]:
```
1    36
3    15
0    15
2     3
Name: label, dtype: int64
```

run

In [ ]:
```
X = user_run[['gender','speed_mean','heart_max','alt_diff']]

# k means determine k
SSE = []
K = range(1,10)
for k in K:
    kmeans = KMeans(n_clusters=k, random_state=42)
    kmeans.fit(X)
    SSE.append(kmeans.inertia_)

# Plot the elbow
plt.plot(K, SSE, 'bx-')
plt.xlabel('k')
plt.ylabel('Sum of squared errors')
plt.title('The Elbow Method showing the optimal k')
plt.show()
```



In [ ]:
```
kmeans = KMeans(n_clusters=4, random_state=42)
kmeans.fit(X)

user_run['label'] = kmeans.labels_
```

In [ ]:
```
np.round(kmeans.cluster_centers_,1)
# 'speed_mean','heart_max','alt_diff'
```

Out[ ]:
```
array([[ 0.8, -1.1,  0.4,  0. ],
       [ 0.9,  0.7,  0.5, -0.1],
       [ 1. ,  0.9,  0.2,  7.5],
       [ 0.9, -0.2, -1.3, -0.1]])
```

In [ ]:
```
user_run['label'].value_counts()
```

Out[ ]:
```
1    38
3    21
0    21
2     1
Name: label, dtype: int64
```

In [ ]:
```
user_run[user_run['label']==0]
```

Out[ ]:

	userId	gender	speed_mean	heart_max	alt_diff	label
3	182042	0.0	-1.907443	-0.149179	-0.354930	0
10	407769	1.0	-1.073496	0.206922	-0.128483	0
12	732008	1.0	-0.882103	0.389957	1.637182	0
23	1543833	1.0	-0.162234	0.840292	0.662823	0
26	2104631	1.0	-0.498749	-0.047256	-0.743112	0
30	2486861	1.0	-2.067687	0.158777	-0.192096	0
33	2868369	1.0	-0.247047	0.538960	-0.252214	0
37	3545637	0.0	-2.268573	1.334910	-0.213536	0
39	3680369	1.0	-0.918755	0.393764	-0.421140	0
45	4433918	1.0	-1.137088	0.668590	0.483786	0
46	4654918	1.0	-0.567464	0.201472	1.063824	0
50	5273972	1.0	-0.731738	0.417115	-0.617365	0
52	5337796	0.0	-3.843496	0.710530	0.053444	0
54	5964610	1.0	-0.580315	0.277396	-0.720180	0
58	6479229	0.0	-1.847867	0.928408	-0.129009	0
59	6539051	1.0	-0.673965	-0.262371	-0.016042	0
63	7231044	1.0	-0.263465	-0.140157	0.232759	0
65	7898832	1.0	-1.307960	-0.283723	-0.532605	0
67	9275291	0.0	-0.306331	0.828447	0.430776	0
68	9985340	1.0	-0.286432	0.959934	0.554298	0
74	13279851	1.0	-0.941052	0.490438	-0.160867	0

PCA for visualization

In [ ]:
```
user_bike.loc[:,'gender':'label']
```

Out[ ]:

	gender	speed_mean	heart_max	alt_diff	label
0	1.0	0.775573	-0.062170	0.575141	1
1	1.0	0.716639	1.152143	0.116120	1
2	1.0	-0.649830	-1.222555	-0.476478	3
3	1.0	1.091222	1.205620	0.071944	1
4	0.0	-1.362690	0.195841	-0.390095	0
...	...	...	...	...	...
64	1.0	1.933232	0.931127	0.584110	1
65	1.0	0.448589	-0.126503	-0.396488	1
66	1.0	1.377630	0.042564	-0.322247	1
67	1.0	0.684712	-0.259039	0.086967	1
68	1.0	0.045764	0.254184	-0.566734	1

69 rows × 5 columns

In [ ]:
```
plotX = user_bike.loc[:,'gender':'label']

#Rename plotX's columns since it was briefly converted to an np.array above
plotX.columns = user_bike.loc[:,'gender':'label'].columns
```

In [ ]:
```
#PCA with one principal component
pca_1d = PCA(n_components=1)

#PCA with two principal components
pca_2d = PCA(n_components=2)

#PCA with three principal components
pca_3d = PCA(n_components=3)
```

In [ ]:
```
#This DataFrame holds that single principal component mentioned above
PCs_1d = pd.DataFrame(pca_1d.fit_transform(plotX.drop(["label"], axis=1)))

#This DataFrame contains the two principal components that will be used
#for the 2-D visualization mentioned above
PCs_2d = pd.DataFrame(pca_2d.fit_transform(plotX.drop(["label"], axis=1)))

#And this DataFrame contains three principal components that will aid us
#in visualizing our clusters in 3-D
PCs_3d = pd.DataFrame(pca_3d.fit_transform(plotX.drop(["label"], axis=1)))
```

In [ ]:
```
PCs_1d.columns = ["PC1_1d"]

#"PC1_2d" means: 'The first principal component of the components created for 2-D visualization, by PCA.'
#And "PC2_2d" means: 'The second principal component of the components created for 2-D visualization, by PCA.'
PCs_2d.columns = ["PC1_2d", "PC2_2d"]

PCs_3d.columns = ["PC1_3d", "PC2_3d", "PC3_3d"]
```

In [ ]:
```
plotX = pd.concat([plotX,PCs_1d,PCs_2d,PCs_3d], axis=1, join='inner')
```

In [ ]:
```
plotX
```

Out[ ]:

	gender	speed_mean	heart_max	alt_diff	label	PC1_1d	PC1_2d	PC2_2d	PC1_3d	PC2_3d	PC3_3d
0	1.0	0.775573	-0.062170	0.575141	1	-0.203693	-0.203693	0.313536	-0.203693	0.313536	-0.894882
1	1.0	0.716639	1.152143	0.116120	1	-0.854404	-0.854404	1.061504	-0.854404	1.061504	-0.054775
2	1.0	-0.649830	-1.222555	-0.476478	3	0.617689	0.617689	-1.310135	0.617689	-1.310135	0.185076
3	1.0	1.091222	1.205620	0.071944	1	-1.171910	-1.171910	1.101643	-1.171910	1.101643	-0.261339
4	0.0	-1.362690	0.195841	-0.390095	0	0.693381	0.693381	-0.162159	0.693381	-0.162159	1.264476
...	...	...	...	...	...	...	...	...	...	...	...
64	1.0	1.933232	0.931127	0.584110	1	-1.387314	-1.387314	1.204646	-1.387314	1.204646	-1.255810
65	1.0	0.448589	-0.126503	-0.396488	1	-0.516636	-0.516636	-0.297618	-0.516636	-0.297618	-0.131921
66	1.0	1.377630	0.042564	-0.322247	1	-1.208309	-1.208309	-0.065709	-1.208309	-0.065709	-0.735711
67	1.0	0.684712	-0.259039	0.086967	1	-0.357123	-0.357123	-0.125891	-0.357123	-0.125891	-0.633724
68	1.0	0.045764	0.254184	-0.566734	1	-0.456253	-0.456253	-0.098680	-0.456253	-0.098680	0.409309

69 rows × 11 columns

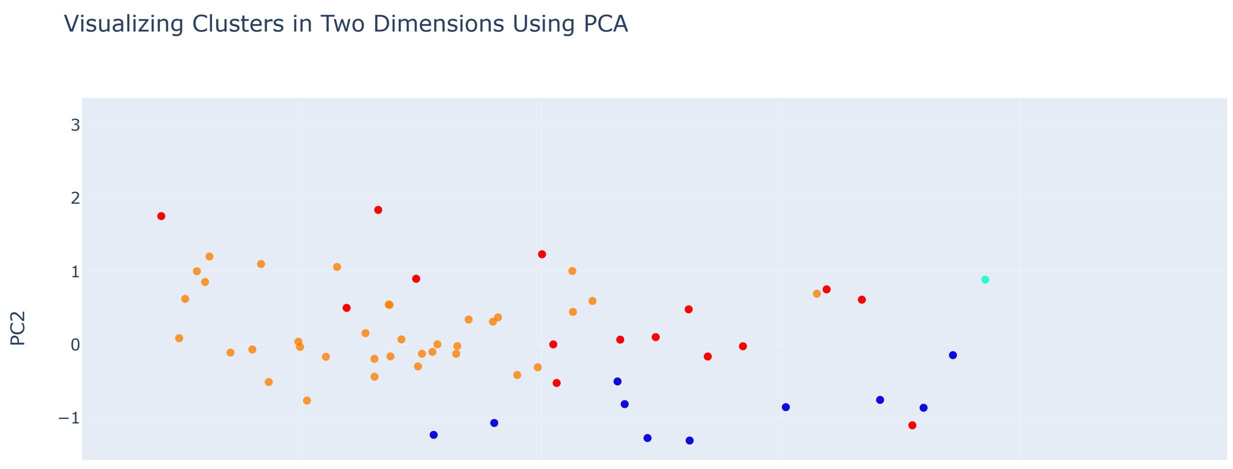
In [ ]:
```
#Note that all of the DataFrames below are sub-DataFrames of 'plotX'.
#This is because we intend to plot the values contained within each of these DataFrames.

cluster0 = plotX[plotX["label"] == 0]
cluster1 = plotX[plotX["label"] == 1]
cluster2 = plotX[plotX["label"] == 2]
cluster3 = plotX[plotX["label"] == 3]
```

visualization

In [ ]:
```
# https://www.kaggle.com/code/minc33/visualizing-high-dimensional-clusters/notebook#Method-#1:-Principal-Component-Analysi
```

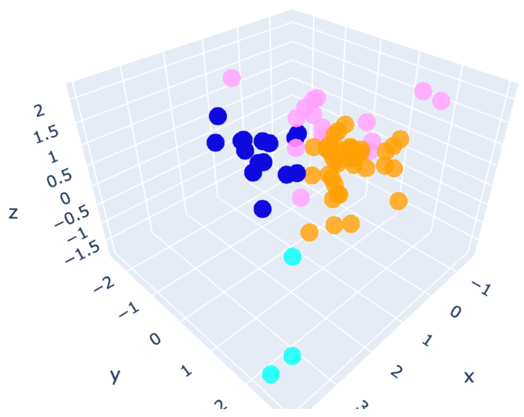
In [ ]:
```
init_notebook_mode(connected=True)
```

In [ ]:
```
trace1 = go.Scatter(
                    x = cluster0["PC1_2d"],
                    y = cluster0["PC2_2d"],
                    mode = "markers",
                    name = "Cluster 0",
                    marker = dict(color = 'rgba(255, 0, 0, 1)'),
                    text = None)

#trace2 is for 'Cluster 1'
trace2 = go.Scatter(
                    x = cluster1["PC1_2d"],
                    y = cluster1["PC2_2d"],
                    mode = "markers",
                    name = "Cluster 1",
                    marker = dict(color = 'rgba(255, 128, 2, 0.8)'),
                    text = None)

#trace3 is for 'Cluster 2'
trace3 = go.Scatter(
                    x = cluster2["PC1_2d"],
                    y = cluster2["PC2_2d"],
                    mode = "markers",
                    name = "Cluster 2",
                    marker = dict(color = 'rgba(0, 255, 200, 0.8)'),
                    text = None)

trace4 = go.Scatter(
                    x = cluster3["PC1_2d"],
                    y = cluster3["PC2_2d"],
                    mode = "markers",
                    name = "Cluster 3",
                    marker = dict(color = 'rgba(15, 10, 222, 1)'),
                    text = None)

data_all = [trace1, trace2, trace3, trace4]

title = "Visualizing Clusters in Two Dimensions Using PCA"

layout = dict(title = title,
              xaxis= dict(title= 'PC1',ticklen= 5,zeroline= False),
              yaxis= dict(title= 'PC2',ticklen= 5,zeroline= False)
             )

fig = dict(data = data_all, layout = layout)

iplot(fig)
```



In [ ]:
```
trace1 = go.Scatter3d(
                    x = cluster0["PC1_3d"],
                    y = cluster0["PC2_3d"],
                    z = cluster0["PC3_3d"],
                    mode = "markers",
                    name = "Cluster 0",
                    marker = dict(color = 'rgba(255, 128, 255, 0.8)'),
                    text = None)

#trace2 is for 'Cluster 1'
trace2 = go.Scatter3d(
                    x = cluster1["PC1_3d"],
                    y = cluster1["PC2_3d"],
                    z = cluster1["PC3_3d"],
                    mode = "markers",
                    name = "Cluster 1",
                    marker = dict(color = 'rgba(255, 128, 2, 0.8)'),
                    text = None)

#trace3 is for 'Cluster 2'
trace3 = go.Scatter3d(
                    x = cluster2["PC1_3d"],
                    y = cluster2["PC2_3d"],
                    z = cluster2["PC3_3d"],
                    mode = "markers",
                    name = "Cluster 2",
                    marker = dict(color = 'rgba(0, 255, 200, 0.8)'),
                    text = None)

trace4 = go.Scatter3d(
                    x = cluster3["PC1_3d"],
                    y = cluster3["PC2_3d"],
                    z = cluster3["PC3_3d"],
                    mode = "markers",
                    name = "Cluster 3",
                    marker = dict(color = 'rgba(15, 10, 222, 1)'),
                    text = None)

data_all = [trace1, trace2, trace3, trace4]

title = "Visualizing Clusters in Three Dimensions Using PCA"

layout = dict(title = title,
              xaxis= dict(title= 'PC1',ticklen= 5,zeroline= False),
              yaxis= dict(title= 'PC2',ticklen= 5,zeroline= False)
             )

fig = dict(data = data_all, layout = layout)

iplot(fig)
```

Visualizing Clusters in Three Dimensions Using PCA



A closer look at the risk group

In [ ]:
```
risk_biker = user_bike[user_bike["label"] == 0]
risk_biker.head()
```

Out[ ]:

	userId	gender	speed_mean	heart_max	alt_diff	label
4	182042	0.0	-1.362690	0.195841	-0.390095	0
10	338866	1.0	-1.378293	0.671521	-0.749274	0
18	1517642	1.0	0.214730	2.671329	-0.851151	0
20	1663599	1.0	-1.269819	0.511143	-0.459112	0
21	2020266	1.0	-0.827295	2.674227	-0.600488	0

In [ ]:
```
risk_runner = user_run[user_run['label']==0]
risk_runner.head()
```

Out[ ]:

	userId	gender	speed_mean	heart_max	alt_diff	label
3	182042	0.0	-1.907443	-0.149179	-0.354930	0
10	407769	1.0	-1.073496	0.206922	-0.128483	0
12	732008	1.0	-0.882103	0.389957	1.637182	0
23	1543833	1.0	-0.162234	0.840292	0.662823	0
26	2104631	1.0	-0.498749	-0.047256	-0.743112	0

In [ ]:
```
lst = list(risk_runner['userId']).extend(list(risk_biker['userId']))
```

In [ ]:
```
data_merged['risk'] = data_merged['userId'].isin(risk_biker['userId']) #| data_merged['userId'].isin(risk_runner['userId']
```

In [ ]:
```
data_merged['risk'].value_counts()
```

Out[ ]:
```
False    8951
True      973
Name: risk, dtype: int64
```

The gender of two groups is virtually the same.

Max Heart rate of the risk group is higher than this of the normal group, that is 165 vs 160

Simultaneouly, the alt diff shows the risk group tends to cycle on more moderate routes, and the speed_mean shows their cycling speed is fairly slow.

We also noticed that the heart rate std (that is std_x) for the risk group is slightly higher than the normal group, indicating the risk group experiences a drastic heart rate fluctuation.

In [ ]:
```
data_merged.groupby('risk').agg('mean')
```

Out[ ]:

	gender	id	userId	heart_max	std_x	alt_diff	std_y	speed_mean	std	sport_basketball	...	sport_mour
risk												
False	0.954083	3.689118e+08	4.946456e+06	159.96607	12.830497	0.919427	0.232597	-0.394173	3.754329	0.000112	...	0.079
True	0.917780	3.846787e+08	3.786466e+06	165.61522	13.777484	0.615809	0.156102	-2.265939	4.799635	0.000000	...	0.00

2 rows × 28 columns

In [ ]:

A hypo: It is also noteworthy that the risk group has fewer exercise records compared to the normal group, which might come from technical issues like lack of experience in scheduling and physical distribution, but can also come from the user's lack of exercises or unmatched exercises abilities.

In [ ]:

```python
data_merged.groupby(['userId','risk']).agg({'userId': 'count'}).groupby('risk').agg({'userId': 'mean'})
```

Out[ ]:

	userId
risk	
False	105.305882
True	64.866667

Conclusion and limitations

The clustering analysis shows how we can leverage an unsupervised machine learning model to detect the heart anomaly and identify the risk group. This result is useful when people are to design an alert system that provides the user with a heart health caveat on wearable devices. And once combined with geographical data like latitude deviation and route length, we are able to construct a route recommendation system that matches the level of physical ability and exercise habits of each user.

However, this analysis is not flawless. It suffers a lot from the restrictions of the data set. One concern is the data insufficiency that many common correlated demographical features (like age, and race) and personal information (like medical history, and exercise frequency) are inaccessible for this analysis; whereas they are likely to be essential in explaining the disparities. Another concern is a technical one, in that we are not sure what kind of matrices to assess the clustering model. Also, we would like to improve our analysis and insights to be more research-intensive if we received support for the clinical knowledge of cardiology.