

Exploration of Threshold Heart Rate Data on Performance

Will Dauenhauer

University of Montana

MSBA 2023

Table of Contents:

I. Executive Summary

II. Background

III. Overview

IV. Research Questions

V. Data

VI. Code Explanations

VII. Analysis

VIII. Limitations

IX. Recommendations

X. Conclusion

I. Executive Summary

This paper explores the relationship between desired heart rate during training and performance outcomes, specifically distance covered, speed, and time taken. The study utilizes personal data collected from Strava's API to examine this relationship. The findings suggest that there is a correlation between heart rate and performance outcomes, although the degree of correlation varies. The results suggest that desired heart rate during training can have an impact on performance outcomes, and thus should be considered when designing training programs. However, further research is needed to fully understand the relationship between heart rate and performance outcomes, and to identify the most effective training programs for different individuals. Overall, this study highlights the importance of monitoring heart rate during training and its potential impact on performance outcomes.

II. Background

I am an athlete here at the University of Montana and have been running Cross Country and Track, and with that I have been recording my runs and other aerobic activities on Strava since 2020. Since this point, every activity that I record on my Garmin Forerunner 35 watch will upload metrics to Strava to showcase your activity to the public. Strava is a social fitness network that provides a range of features for runners, including GPS tracking, route mapping, and activity analysis. The platform allows runners to track their runs, rides, and other workouts, and provides insights into their performance, including distance covered, pace, elevation, and calories burned. Strava provides runners with a comprehensive platform to track, analyze, and improve their performance, however it does not give all the answers to the true capabilities for my analysis which brings us to the Strava API integration. Strava's API (Application Programming Interface) provides runners with a way to access their data and perform custom analysis beyond what is available through Strava's website or mobile app. The API allows developers to create custom applications and tools that can access Strava data, including workout data, segment data, and athlete data. This allows for more detailed analysis of workouts, performance metrics, and trends over time.

III. Overview

What is already known and what are we trying to figure out. Figure 1 below shows the different heart rate zones that Strava has given me over the years based on my performance. Training at threshold heart rate can lead to improvements in running performance, but the extent of these improvements can vary depending on several factors, including an individual's fitness level, training history, and genetics. Threshold heart rate training involves working out at or near the maximum sustainable heart rate for a prolonged period of time. This type of training can help improve aerobic capacity, increase lactate threshold, and improve overall endurance. It is commonly used by runners to improve their speed, distance, and race performance. Research has shown that training at threshold heart rate can lead to improvements in running performance, including increased VO2 max, improved lactate threshold, and increased running economy. However, the magnitude of these improvements can vary depending on individual factors and the specific training program.

Table

Description automatically generated

*Figure 1: Chart of Heart Rate Zones*

In this analysis, I wanted to see to what effect this threshold has on my overall performance personally and see how well it can predict my performance. The why in this approach is due to the fact that over the years, my training has been all over the place due to different coaching styles and philosophies on heart rate, training for different distance races, different workout types, switching between cross country and track, and overall consistency in training.

IV. Research Question

Given the varied nature of the training over the years, I wanted to explore the impact of using heart rate as a threshold for training intensity on my performance. Specifically, I am interested in how consistently maintaining or hitting a certain heart rate threshold during training affects my overall performance in terms of distance, pace, and time. Additionally, I am curious about how well this heart rate threshold can predict my performance in future races. With these questions in mind.

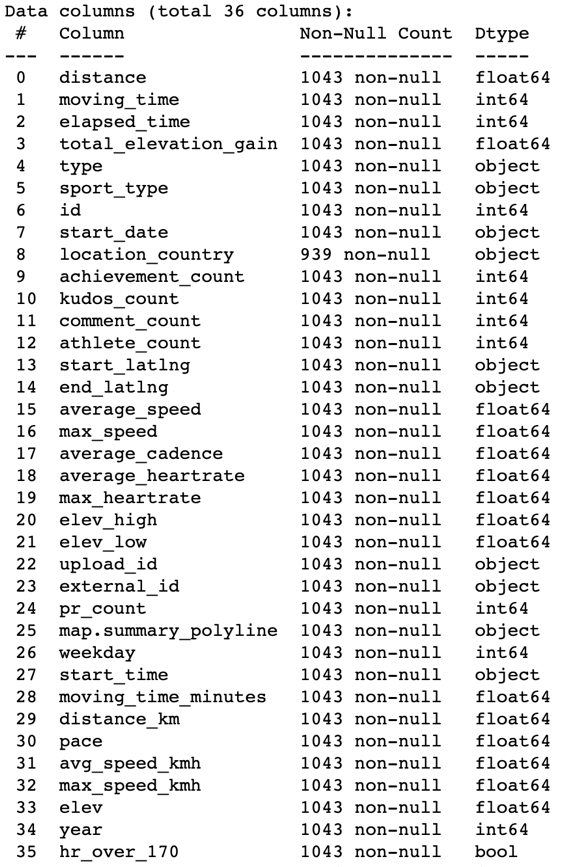
RQ 1: How does reaching a threshold heart rate during training affect performance outcomes?

RQ 2: How well does threshold heart rate serve as a predictor for performance outcomes?

Given this question, one of the response variables that I will introduce into my data set (besides the max\_heartrate) will be a variable called hr\_over\_170 which will account for 10 bpm’s of error given sweat can interfere with the heart rate recording and become a Boolean variable for if the activity had a heart rate over 170 bpm. Regarding the choice of using max\_heartrate as our other response allows us to show that I was able to hit a heart rate in that threshold. The limitations to this which I will discuss later will be on how given measures of max\_heartrate, this is not the best variable as we lack the fluctuation of heart rate throughout the activity.

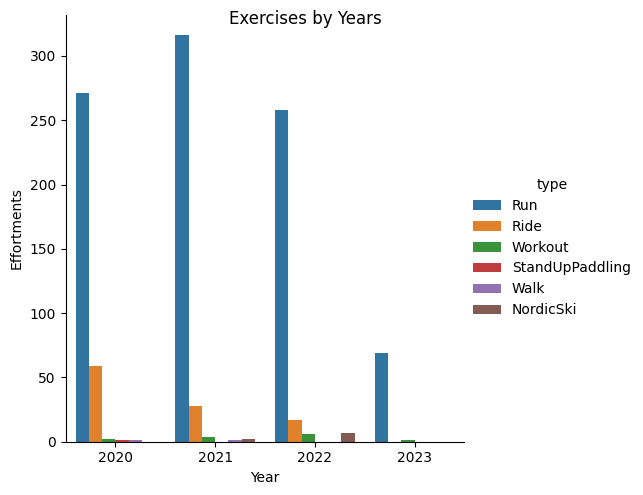
V. Data

The type of data that was pulled from Strava including the new variable created called ‘hr\_over\_170’ is shown below in figure 2. The data includes 1043 observations, and the variables recorded include distance, moving time, elapsed time, total elevation gain, achievement count, kudos count, comment count, athlete count, average speed, and other variables such as elevation low, pr count, weekday, moving time in minutes, distance in kilometers, pace, average speed in kilometers per hour, maximum speed in kilometers per hour, elevation, and year.



*Figure 2: Data Description*

From this we cleaned the data (seen in code appendix) and took out variables that were useless to our analysis which included 'id', 'achievement\_count', 'kudos\_count', 'comment\_count', 'pr\_count',and 'athlete\_count'. To give a reference for the size of the dataset, figure 3 shows the spread of the activities over the years.

****

*Figure 3: Exercises by Years*

The next three figures show basis for the reasoning of my analysis, with Exercise Time vs Max Heart Rate, Distance (meters) vs Max Heartrate, and Average Max Heartrate throughout the week. The figure 4 shows the relationship between exercise time and maximum heart rate. This plot allows us to visualize whether there is a correlation between the duration of the exercise and the maximum heart rate that is achieved during that exercise. It can help us identify any trends or patterns that may emerge from the data.

*Chart, scatter chart

Description automatically generated*

*Figure 4: Exercise Time vs Max Heartrate*

Figure 5 plots the distance covered during each exercise session against the maximum heart rate achieved during that session. This plot can help us identify whether there is a relationship between the distance covered and the maximum heart rate achieved. It can also help us see if there are any outliers in the data, such as unusually high or low maximum heart rates.

Chart, scatter chart

Description automatically generated

*Figure 5: Distance vs Max Heartrate*

Figure 6 shows the average maximum heart rate throughout the week. This can help us see if there are any trends or patterns that emerge over time. It can also help us identify if there are any days of the week where the maximum heart rate is higher or lower than average.

**Chart, bar chart

Description automatically generated**

Figure 6: Average Max Heart Rate during a Week

Overall, the figures provided give the reader a visual representation of the data and help them understand the basis for the analysis.

The correlation matrix (figure 7) is a table that shows the correlation coefficient between variables. In this case, the focus is on variables related to heart rate. A correlation coefficient is a statistical measure that indicates the strength and direction of the relationship between two variables. It ranges from -1 to +1, where -1 indicates a perfect negative correlation, 0 indicates no correlation, and +1 indicates a perfect positive correlation.

A picture containing chart

Description automatically generated*Figure 7: Correlation Matrix of Cleaned and Useful Variables*

The correlation matrix shows the correlation coefficient for each pair of variables in a table format. In this case, the matrix is focused on variables surrounding heart rate. The lighter squares in the matrix indicate stronger correlations between these variables. The correlation between two variables can be positive or negative. A positive correlation means that when one variable increases, the other variable also increases. A negative correlation means that when one variable increases, the other variable decreases.

By focusing on the lighter squares related to variables surrounding heart rate, we can see which variables are strongly correlated with heart rate. This information is useful for understanding how different variables affect heart rate and for developing predictive models that include heart rate as a predictor variable.

VI. Code Explanations

Recursive Feature Elimination (RFE) is a method that is used to train a selected model and eliminate the least important features recursively. However, to perform RFE, it is necessary to have the models already selected for this run. In this particular case, Linear Regression and Stochastic Gradient Descent (SGD) models are chosen, each with a different response variable.

It is important to note that RFE can be computationally expensive for high-dimensional datasets.

To implement RFE, a function is defined and used for linear regression and SGD. The function trains the selected model and eliminates features based on their importance recursively. For each iteration, the model is trained on the remaining features until only the most important ones are left. In this case, seven features are selected using the chi-squared (chi2) statistical test.

In summary, RFE is a powerful feature selection method that can help to improve the performance of machine learning models by eliminating irrelevant or redundant features.Top of Form

Bottom of Form

**Input 1**

**def** get\_best\_rfe\_features(X,y, model):

rfe **=** RFE(model, step**=**0.05)**.**fit(X, y)

selected\_features **=** [i **for** i, j **in** zip(X**.**columns, rfe**.**support\_) **if** j]

**return** selected\_features

y **=** tweak\_runs['max\_heartrate']

X **=** tweak\_runs**.**drop('max\_heartrate',1)

encoded\_y **=** preprocessing**.**LabelEncoder()**.**fit\_transform(y)

model **=** LinearRegression()

linear\_feats **=** get\_best\_rfe\_features(X, encoded\_y, model)

**Output 1**

['moving\_time', 'average\_speed', 'max\_speed', 'average\_heartrate', 'moving\_time\_minutes', 'distance\_km', 'avg\_speed\_kmh', 'max\_speed\_kmh', 'hr\_over\_170']

This code defines a function called **get\_best\_rfe\_features** that takes in three parameters:

* **X**: A pandas DataFrame containing the input features for the model.
* **y**: A pandas Series containing the response variable to be predicted by the model.
* **model**: A scikit-learn estimator object representing the model to be used for feature selection.

Within the function, the Recursive Feature Elimination (RFE) method is used to recursively eliminate features that are not deemed important by the model. Specifically, the **RFE** function is called with **model** as the estimator object and **step=0.05**, which means that 5% of the features will be eliminated in each iteration. The **fit** method is then called on **rfe** with **X** and **y** as inputs to train the model and select the best features. The **selected\_features** list is then created by zipping the column names in **X** with the Boolean values in **rfe.support\_** (which indicate whether or not each feature was selected), and filtering out the features that were not selected.

After defining the function, the code sets **y** and **X** to the response variable and input features from a pandas DataFrame called **tweak\_runs**, respectively. The **LabelEncoder** from scikit-learn is then used to encode **y** into integer labels, which are necessary for using **LinearRegression**. Finally, **get\_best\_rfe\_features** is called with **X**, **encoded\_y**, and a **LinearRegression** object to select the best features for a linear regression model. The selected features are stored in a list called **linear\_feats**.

In regression models, the objective is to predict a target variable y using a set of independent variables x. The model learns the values of the coefficients in the equation of a line by adjusting several lines in the feature map and selecting the line that yields the smallest error between the predicted and actual values of y.

In this specific application of linear regression, only the features selected in the previous step and stored in the variable **linear\_feats** will be used. To evaluate the performance of the algorithm, the dataset will be split into 80% for training and 20% for testing. This enables us to measure how well the model generalizes to unseen data.

After the data split, the training dataset is passed to the fit function of the Linear Regression model, which adjusts the coefficients of the line that best fits the training data. The model is then used to make predictions on the test dataset, and the performance of the model is evaluated by comparing the predicted values with the actual values.

**Input 2**

y = runs['max\_heartrate']

X = runs[linear\_feats]

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2)

model = LinearRegression()

model.fit(X\_train,y\_train)

y\_pred = model.predict(X\_test)

Code chunk 2 is an implementation of linear regression on a dataset using the scikit-learn library in Python.

* The first line of code selects the target variable 'max\_heartrate' from the dataset 'runs' and assigns it to the variable 'y'.
* The second line selects the feature variables from the dataset 'runs' based on the features selected in the previous step (stored in the variable 'linear\_feats') and assigns them to the variable 'X'.
* The third line splits the dataset into training and testing sets using the 'train\_test\_split' function from scikit-learn. The training set will contain 80% of the data and the testing set will contain 20% of the data. The feature and target variables are split into separate training and testing sets.
* The fourth line creates an instance of the LinearRegression model.
* The fifth line fits the model on the training data using the 'fit' method of the LinearRegression object. This step trains the model by finding the best fit line that predicts the target variable based on the selected feature variables.
* The sixth line uses the trained model to predict the target variable 'max\_heartrate' for the testing data set by calling the 'predict' method of the LinearRegression object and passing it the feature variables 'X\_test'. The predicted values are assigned to the variable 'y\_pred'.

**Input 3**

y = tweak\_runs['hr\_over\_170']

X = tweak\_runs.drop('hr\_over\_170',1)

model = SGDClassifier(loss="hinge", penalty="l2", max\_iter=5)

class\_feats = get\_best\_rfe\_features(X, y, model)

**Output 3**

['distance', 'moving\_time', 'elapsed\_time', 'total\_elevation\_gain', 'average\_heartrate', 'max\_heartrate', 'elev\_high', 'elev\_low', 'year']

Code chunk 3 is performing feature selection using Recursive Feature Elimination (RFE) for a binary classification problem.

The target variable is defined as y which corresponds to the column 'hr\_over\_170' in the tweak\_runs dataframe. The feature matrix is defined as X which is obtained by dropping the target variable column using drop('hr\_over\_170',1) method.

Then, a SGDClassifier model is defined with specific parameters. loss="hinge" is the loss function used to measure the difference between predicted and actual class labels, penalty="l2" is the type of regularization used to prevent overfitting, and max\_iter=5 is the maximum number of passes over the training data.

The function get\_best\_rfe\_features is called to perform RFE feature selection on the input data X and target variable y using the SGDClassifier model. The selected features are stored in class\_feats variable. The output is the list of class\_feats variable.

**Input 4**

y = runs['hr\_over\_170']

X = runs[class\_feats]

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2)

model = SGDClassifier()

model.fit(X\_train,y\_train)

y\_pred = model.predict(X\_test)

Code chunk 4 performs classification using Stochastic Gradient Descent (SGD) on a dataset consisting of features in X and corresponding labels in y.

The code first defines y as the label column, which is the target variable we want to predict, and X as the features matrix, which includes all the features that were previously selected by Recursive Feature Elimination (RFE) using the class\_feats variable.

Then, the code uses the train\_test\_split function from scikit-learn to split the data into training and testing sets with a test size of 0.2, meaning that 20% of the data will be used for testing while 80% will be used for training.

Next, the code initializes an SGDClassifier object with default settings and fits the training data to the model using the fit function. Finally, the code predicts the labels of the test data using the trained model and stores the predictions in the y\_pred variable.

VII. Analysis

Regression

The Mean Squared Error (MSE) can be calculated using the predicted and actual values, and it provides a measure of how much the model is deviating from the true values. A high MSE value indicates that the model is making a large number of errors in its predictions, which can lead to unreliable or inaccurate results. In addition to the MSE, a scatter plot can also be used to visually compare the predicted values with the actual values. In this case, the calculated MSE was 49.419, indicating that the model may have significant errors in its predictions.

Figure 8 is evaluating the performance of a regression model by calculating the mean squared error (MSE) between the predicted values and the true values of the test set. The value of the MSE is 49.419, which indicates the average squared difference between the predicted values and the true values is 49.419.

Chart, scatter chart

Description automatically generated

Figure 7: Scatterplot of Predicted vs True Values

The scatter plot of the predicted values versus the true values, with the predicted values on the x-axis and the true values on the y-axis. The points on the scatter plot are colored crimson. The y-axis and x-axis are set to a logarithmic scale.

The line to the scatter plot to represent the ideal case where the predicted values are equal to the true values. The line is drawn from the maximum predicted value to the minimum predicted value and is colored blue.

From here we can train our model using the code snippet below:

model.predict(

pd.DataFrame(data={

'moving\_time': 114,

'average\_speed': 6,

'max\_speed': 6.67,

'average\_heartrate': 200,

'moving\_time\_minutes': 2,

'distance\_km': .8,

'avg\_speed\_kmh': 23,

'max\_speed\_kmh': 24,

'hr\_over\_170': True

},

index=[0]

)

)

This code is using a trained model to make a prediction on a new data point. The input features are given as a dictionary with the following keys and values:

* **moving\_time**: an integer representing the duration of the activity in seconds.
* **average\_speed**: a float representing the average speed of the activity in meters per second.
* **max\_speed**: a float representing the maximum speed reached during the activity in meters per second.
* **average\_heartrate**: an integer representing the average heart rate during the activity.
* **moving\_time\_minutes**: an integer representing the duration of the activity in minutes.
* **distance\_km**: a float representing the distance of the activity in kilometers.
* **avg\_speed\_kmh**: a float representing the average speed of the activity in kilometers per hour.
* **max\_speed\_kmh**: a float representing the maximum speed reached during the activity in kilometers per hour.
* **hr\_over\_170:** a boolean indicating if the heart rate during the activity was over 170 beats per minute.

The values for these features are passed to a Pandas DataFrame, which is then passed to the **predict** method of the **model** object. The **predict** method will return the predicted class label for the new data point based on the learned patterns in the model.

Classification

In contrast to regressions, classification systems use different evaluation metrics such as precision, which calculates the ratio of correct predictions to total predictions. In this case, the precision score was 61.2%

The Confusion Matrix which can be seen in figure 8 shows the number of true positives, true negatives, false positives, and false negatives, which are the four possible outcomes of a binary classification problem. It can be used to determine the precision, recall, and F1-score of the model.

**Chart, treemap chart

Description automatically generated**

Figure 8: Confusion Matrix

The accuracy score, which is a measure of how many of the predictions made by the model were correct. In this case, the accuracy score is 0.612, indicating that the model was able to correctly predict the target variable for about 61.2% of the test data.

The code below is similar to above to predicting a target variable based on a set of input features using a machine learning model. Specifically, the code is using the **predict** method of a trained model to predict the target variable for a new data point.

model.predict(

pd.DataFrame(data={

'distance': 800,

'moving\_time': 120,

'elapsed\_time' : 120,

'total\_elevation\_gain': 0,

'average\_heartrate': 200,

'max\_heartrate': 220,

'elev\_high': 900,

'elev\_low': 900,

'year' : 2023

},

index=[0]

)

)

The input features are provided as a dictionary of values wrapped inside a Pandas DataFrame with a single row and the column names matching the feature names used in training the model.

The features include:

* **distance**: the distance of the activity in meters
* **moving\_time**: the duration of the activity in seconds while in motion
* **elapsed\_time**: the total duration of the activity in seconds
* **total\_elevation\_gain**: the cumulative elevation gain over the activity in meters
* **average\_heartrate**: the average heart rate over the activity
* **max\_heartrate**: the maximum heart rate over the activity
* **elev\_high**: the highest elevation point of the activity in meters
* **elev\_low**: the lowest elevation point of the activity in meters
* **year**: the year when the activity was performed.

Once the new data point is constructed, the **predict** method of the trained model is called on the input features to generate a predicted output value for the target variable.

The fact that the models were able to make correct predictions based on the meet data, even with a moderate accuracy score and MSE values, suggests that the models have some predictive power when it comes to athletic performance and max heart rate. While the accuracy score and MSE values indicate that the models are not perfectly accurate in their predictions, they are still able to provide some valuable insights into the relationship between these variables.

It's important to note that while the models were able to make correct predictions based on the meet data, this does not necessarily mean that they will always be accurate in their predictions. There are many factors that can impact athletic performance, and it's possible that the models may not be able to account for all of these factors. Additionally, the models were trained on a specific dataset and may not perform as well on new, unseen data.

Overall, while the models may not be perfect, they are still a useful tool for understanding the relationship between max heart rate and athletic performance. By taking into account the limitations of the models and using them in conjunction with other sources of information, such as expert knowledge and personal experience, athletes and coaches can make more informed decisions about training and competition strategies.

VIII. Limitations

While classification and regression models can be useful tools for predicting performance from max heart rate, they both have limitations.

One limitation of classification models is that they rely on a binary outcome and may not accurately capture the complexity and nuances of athletic performance. In addition, classification models are sensitive to the choice of threshold used to define the binary outcome, and different thresholds may result in different classification outcomes.

On the other hand, regression models can capture a more continuous range of outcomes, but they may oversimplify the relationship between max heart rate and athletic performance. Regression models assume a linear relationship between predictor and outcome variables, which may not hold in reality. Additionally, regression models may be sensitive to outliers and may not capture non-linear relationships between variables.

In the case of the models used in this analysis, a classification model achieved an accuracy score of 61.2%, indicating that the model correctly predicted the outcome for 61.2% of the cases. While this may seem like a relatively high score, it is important to note that the model is only accurate 61.2% of the time and is incorrect almost 40% of the time. This could be due to the limitations of the binary outcome and the choice of threshold used to define the outcome.

Similarly, the mean squared error of the regression model was 49.419, indicating that there was a relatively high amount of error between the predicted and actual values. This could be due to the limitations of assuming a linear relationship between max heart rate and athletic performance, as well as the potential for outliers and non-linear relationships to be present in the data.

Overall, while classification and regression models can provide valuable insights into the relationship between max heart rate and athletic performance, it is important to consider their limitations and the potential for other factors to impact athletic performance.

IX. Recommendations

The recommendation to create a new variable by binning the average max heart rates in the activity can be a useful way to provide the model with additional information for better prediction of performance. The idea behind binning is to categorize a continuous variable, such as max\_heartrate, into discrete intervals or bins. In this case, we could divide the range of max\_heartrate values into 10 equal sections, and create a new variable that assigns each observation to one of these sections.

By creating this new variable, we are effectively converting a continuous variable into a categorical variable, which can be easier to work with in some cases. In addition, this can provide the model with more information to use in its predictions, as it can now take into account the specific range of max\_heartrate values that an observation falls into.

However, it is important to note that binning also has some limitations. For example, it can lead to loss of information, as we are reducing the precision of the variable by categorizing it into discrete intervals. Additionally, the choice of the number and size of the bins can affect the results of the analysis. If the bins are too small, we may end up with too few observations in each category, which can reduce the power of the analysis. On the other hand, if the bins are too large, we may lose some of the variability in the data and may not capture important patterns.

Therefore, when creating a new variable by binning the max\_heartrate variable, it is important to carefully consider the number and size of the bins, as well as to evaluate the impact of the binning on the model's performance. This can be done through techniques such as cross-validation, which can help us to assess the robustness of the model and ensure that it is not overfitting to the data.

X. Conclusion

Overall, we have highlighted the limitations of classification and regression models in predicting athletic performance based solely on max heart rate and stresses the importance of considering other factors that may impact performance. The recommendation to create a new variable by binning the average max heart rates in the activity is seen as a useful approach to provide the model with more information and improve performance prediction. By dividing the range of max\_heartrate values into 10 equal sections, the model can gain a more detailed understanding of how different ranges of max heart rates impact athletic performance. Overall, this approach can help to provide a more accurate and nuanced picture of the relationship between max heart rate and athletic performance.

Sources:

Silva, L. de B. (2022, December 30). *Data Analysis with Strava*. Medium. Retrieved March 26, 2023, from https://levelup.gitconnected.com/data-analysis-with-strava-7251327698e3

*The strava API*. Strava Developers. (n.d.). Retrieved March 26, 2023, from https://developers.strava.com/

*What you need to know about threshold training*. Runner's World. (2022, December 31). Retrieved March 26, 2023, from https://www.runnersworld.com/advanced/a20807282/threshold-training/

*Will dauenhauer – track and field results & statistics*. TFRRS. (n.d.). Retrieved March 26, 2023, from https://www.tfrrs.org/athletes/6876302/Montana/Will\_Dauenhauer.html

Code Appendix:

**Creating Token**

client\_id **=** 103568

client\_secret **=** 'b34f2e9c0061c8c3fec5cdb1fd6276669d92290e'

strava\_code **=** '3d2789128c08418e9ee5aa4b0bcc6cc418391674'

**import** os

**import** json

**import** requests

response **=** requests**.**post(

url**=**'https://www.strava.com/oauth/token',

data**=**{

'client\_id': client\_id,

'client\_secret': client\_secret,

'code': strava\_code,

'grant\_type': 'authorization\_code'

}

)

strava\_tokens **=** response**.**json()

**with** open('strava\_tokens.json', 'w', encoding**=**'utf-8') **as** outfile:

json**.**dump(strava\_tokens, outfile)

print(strava\_tokens)

**Getting Activities**

**import** requests

**import** pandas **as** pd

**import** os

**import** json

**import** glob

**import** time

**import** requests

**import** pandas **as** pd

**from** dotenv **import** load\_dotenv

load\_dotenv()

**def** main():

url **=** "https://www.strava.com/api/v3/activities"

access\_token **=** get\_credentials()

page **=** 1

print('Getting data from Strava')

**while** **True**:

response **=** get\_data(url, access\_token, 200, page)

**if** 'message' **in** response**.**columns:

**raise** Exception('Authorization Error')

**if** response**.**empty:

**break**

save\_csv(response, f'data/strava\_activities\_page\_{page}.csv')

page **+=** 1

merge\_files('data/', 'result/strava\_all\_activities.csv')

print('Done Successfully')

**def** get\_credentials():

**with** open('strava\_tokens.json', encoding**=**'utf-8') **as** json\_file:

strava\_tokens **=** json**.**load(json\_file)

**if** 'expires\_at' **not** **in** strava\_tokens**.**keys() **or** strava\_tokens['expires\_at'] **<** time**.**time():

strava\_tokens **=** refresh\_credentials(strava\_tokens)

**return** strava\_tokens['access\_token']

**def** refresh\_credentials(strava\_tokens):

response **=** requests**.**post(

url**=**'https://www.strava.com/oauth/token',

data**=**{

'client\_id': int(os**.**environ**.**get('client\_id')),

'client\_secret': os**.**environ**.**get('client\_secret'),

'grant\_type': 'refresh\_token',

'refresh\_token': strava\_tokens['refresh\_token']

}

)

strava\_tokens **=** response**.**json()

**with** open('strava\_tokens.json', 'w', encoding**=**'utf-8') **as** outfile:

json**.**dump(strava\_tokens, outfile)

**with** open('strava\_tokens.json', encoding**=**'utf-8') **as** check:

data **=** json**.**load(check)

**return** data

**def** get\_data(url, access\_token, numb\_item\_page, page):

print(f'Getting data from page {page}')

response **=** requests**.**get(

f'{url}?access\_token={access\_token}&per\_page={numb\_item\_page}&page={page}'

)

response **=** response**.**json()

dataframe **=** pd**.**json\_normalize(response)

**return** dataframe

**def** save\_csv(dataframe, filename):

print(f'Saving {filename}')

dataframe**.**to\_csv(filename)

**def** merge\_files(path, filename):

print('Merging files')

csv\_files **=** [pd**.**read\_csv(\_file)

**for** \_file **in** glob**.**glob(os**.**path**.**join(path, "\*.csv"))]

final\_df **=** csv\_files**.**pop(len(csv\_files)**-**1)

final\_df **=** final\_df**.**append(csv\_files)

save\_csv(final\_df, filename)

**if** \_\_name\_\_ **==** '\_\_main\_\_':

main()

**Strava API Analysis:**

**import** subprocess

**import** calendar

**from** geopy.geocoders **import** Nominatim

**import** pandas **as** pd

**import** numpy **as** np

**import** seaborn **as** sns

**import** matplotlib.pyplot **as** plt

**import** matplotlib.dates **as** mdates

*# machine learning*

**from** sklearn **import** preprocessing

**from** sklearn **import** metrics

**from** sklearn.feature\_selection **import** chi2

**from** sklearn.feature\_selection **import** SelectKBest

**from** sklearn.feature\_selection **import** RFE

**from** sklearn.linear\_model **import** LinearRegression

**from** sklearn.linear\_model **import** SGDClassifier

**from** sklearn.metrics **import** accuracy\_score

**from** sklearn.metrics **import** confusion\_matrix, ConfusionMatrixDisplay

**from** sklearn.model\_selection **import** train\_test\_split

**from** sklearn.pipeline **import** make\_pipeline

**from** sklearn.preprocessing **import** StandardScaler

df **=** pd**.**read\_csv('result/strava\_all\_activities.csv')

print('Dataframe Shape:', df**.**shape)

df**.**head()

Dataframe Shape: (1043, 60)

Out[222]:

5 rows × 60 columns

In [223]:

df2 **=** pd**.**read\_csv('MeetData2.csv')

df2**.**head()

Out[223]:

In [224]:

null\_df **=** [[col, df[col]**.**isnull()**.**sum()] **for** col **in** df**.**columns]

print('Null Data:', df**.**isnull()**.**sum()**.**sum())

list(filter(**lambda** x: x[1]**>**0, null\_df))

Null Data: 7888

Out[224]:

[['workout\_type', 974],

['location\_city', 1043],

['location\_state', 1043],

['location\_country', 104],

['gear\_id', 186],

['average\_cadence', 116],

['average\_heartrate', 114],

['max\_heartrate', 114],

['elev\_high', 57],

['elev\_low', 57],

['upload\_id', 8],

['upload\_id\_str', 8],

['external\_id', 8],

['suffer\_score', 114],

['map.summary\_polyline', 66],

['average\_watts', 948],

['kilojoules', 948],

['device\_watts', 939],

['average\_temp', 1041]]

In [225]:

selected\_columns **=** ['distance', 'moving\_time', 'elapsed\_time',

'total\_elevation\_gain', 'type','sport\_type', 'id', 'start\_date',

'start\_date\_local','location\_country', 'achievement\_count', 'kudos\_count',

'comment\_count','athlete\_count', 'start\_latlng',

'end\_latlng', 'average\_speed', 'max\_speed', 'average\_cadence',

'average\_heartrate', 'max\_heartrate', 'elev\_high','elev\_low',

'upload\_id', 'external\_id', 'pr\_count', 'map.summary\_polyline']

df **=** df[selected\_columns]

In [226]:

df['start\_date\_local'] **=** pd**.**to\_datetime(df['start\_date\_local'], errors**=**'coerce')

df **=** df**.**sort\_values(by**=**'start\_date\_local')

df['weekday'] **=** df['start\_date\_local']**.**map(**lambda** x: x**.**weekday)

df['start\_time'] **=** df['start\_date\_local']**.**dt**.**time

df['start\_time'] **=** df['start\_time']**.**astype(str)

df['start\_date'] **=** df['start\_date\_local']**.**dt**.**date

df **=** df**.**drop('start\_date\_local', 1)

df**.**head()

/Users/willd/anaconda3/lib/python3.7/site-packages/ipykernel\_launcher.py:9: FutureWarning: In a future version of pandas all arguments of DataFrame.drop except for the argument 'labels' will be keyword-only

if \_\_name\_\_ == "\_\_main\_\_":

Out[226]:

5 rows × 28 columns

In [227]:

df **=** df**.**drop(df[(df**.**distance **<** 1) **&** (df**.**type **==** 'Run')]**.**index)

df **=** df**.**drop(df[df**.**average\_speed **>** 30]**.**index)

df **=** df**.**reset\_index(drop**=True**)

df['elev\_high'] **=** df['elev\_high']**.**fillna(value**=**0)

df['elev\_low'] **=** df['elev\_low']**.**fillna(value**=**0)

df['upload\_id'] **=** df['upload\_id']**.**fillna(value**=**'unknown')

df['external\_id'] **=** df['external\_id']**.**fillna(value**=**'unknown')

df['map.summary\_polyline'] **=** df['map.summary\_polyline']**.**fillna(value**=**'unknown')

df['average\_cadence'] **=** df['average\_cadence']**.**fillna(value**=**df['average\_cadence']**.**mean())

df['average\_heartrate'] **=** df['average\_heartrate']**.**fillna(value**=**df['average\_heartrate']**.**mean())

df['max\_heartrate'] **=** df['max\_heartrate']**.**fillna(value**=**df['max\_heartrate']**.**mean())

df['moving\_time\_minutes'] **=** round(df['moving\_time']**/**60, 2)

df['distance\_km'] **=** round(df['distance'] **/** 1000, 2)

df['pace'] **=** df['moving\_time\_minutes'] **/** df['distance\_km']

df['avg\_speed\_kmh'] **=** round(60**/**df['pace'], 2)

df['max\_speed\_kmh'] **=** round(df['max\_speed']**\***3.6, 2)

df['elev'] **=** df['elev\_high'] **-** df['elev\_low']

df['year']**=** df['start\_date']**.**map(**lambda** x: x**.**year)

In [97]:

df**.**head()

Out[97]:

5 rows × 35 columns

In [98]:

**def** get\_city\_state\_from\_value(value):

value **=** value**.**replace('[','')**.**replace(']','')**.**split(',')

**if** value **!=** ['']:

location **=** geolocator**.**reverse(', '**.**join(value))

result **=** f'{location[0]**.**split(",")[1]}, {location[0]**.**split(",")[1]}'

**else**:

result **=** 'unknown'

**return** result

In [217]:

geolocator **=** Nominatim(user\_agent**=**"strava\_exploration\_data")

df['location'] **=** df['start\_latlng']**.**map(get\_city\_state\_from\_value)

In [228]:

df**.**head()

Out[228]:

In [229]:

df['hr\_over\_170'] **=** np**.**where(df['max\_heartrate']**>=**170, **True**, **False**)

df**.**head()

Out[229]:

5 rows × 36 columns

In [230]:

fig **=** sns**.**catplot(x**=**'year', hue**=**'type', data**=**df, kind**=**'count')

fig**.**fig**.**suptitle('Exercices by Years')

fig**.**set\_xlabels('Year')

fig**.**set\_ylabels('Effortments')

fig

Out[230]:

<seaborn.axisgrid.FacetGrid at 0x7fdb6a65cb10>

In [233]:

runs **=** df**.**loc[df['type'] **==** 'Run']

sns**.**regplot(x**=**'moving\_time\_minutes', y **=** 'max\_heartrate', data**=**runs)**.**set\_title("Exercice Time vs Max Heartrate")

Out[233]:

Text(0.5, 1.0, 'Exercice Time vs Max Heartrate')

In [234]:

sns**.**regplot(x**=**'distance', y **=** 'max\_heartrate', data**=**runs)**.**set\_title("Distance vs Max Heartrate")

Out[234]:

Text(0.5, 1.0, 'Distance vs Max Heartrate')

In [235]:

runs**.**groupby('weekday')**.**mean()['max\_heartrate']**.**plot**.**bar()

Out[235]:

<AxesSubplot:xlabel='weekday'>

In [236]:

corr **=** runs**.**corr()

plt**.**figure(figsize **=** (12,8))

sns**.**heatmap(corr, fmt**=**".2f");

plt**.**title('Correlation between dataset variables')

plt**.**show()

In [237]:

runs **=** runs**.**sample(frac**=**1)**.**reset\_index(drop**=True**)

categorical\_cols **=** [col **for** col **in** runs**.**columns **if** runs[col]**.**dtypes **==** 'O']

useless\_vars **=** ['id', 'achievement\_count', 'kudos\_count', 'comment\_count', 'pr\_count','athlete\_count']

tweak\_runs **=** runs**.**drop(categorical\_cols**+**useless\_vars, axis**=**1)

tweak\_runs

Out[237]:

914 rows × 20 columns

In [238]:

max\_hr **=** df['max\_heartrate']**.**nlargest(10)

print(max\_hr)

570 218.0

629 211.0

861 210.0

157 209.0

669 208.0

647 205.0

510 204.0

229 203.0

591 203.0

701 203.0

Name: max\_heartrate, dtype: float64

In [279]:

y **=** tweak\_runs['max\_heartrate']

X **=** tweak\_runs**.**drop('max\_heartrate',1)

best\_features **=** SelectKBest(chi2, k**=**7)**.**fit\_transform(X, y**.**astype(int))

best\_features

/Users/willd/anaconda3/lib/python3.7/site-packages/ipykernel\_launcher.py:2: FutureWarning: In a future version of pandas all arguments of DataFrame.drop except for the argument 'labels' will be keyword-only

Out[279]:

array([[6496.2, 1694, 2108, ..., 1014.4, 977.9, 36.5],

[11308.4, 2301, 3036, ..., 960.9, 956.9, 4.0],

[23872.3, 7756, 8027, ..., 1359.2, 976.0, 383.20000000000005],

...,

[11680.5, 2908, 3111, ..., 983.4, 968.5, 14.899999999999977],

[9684.9, 2573, 2609, ..., 989.7, 970.3, 19.40000000000009],

[6530.0, 1865, 2002, ..., 981.9, 976.5, 5.399999999999977]],

dtype=object)

In [280]:

**def** get\_best\_rfe\_features(X,y, model):

rfe **=** RFE(model, step**=**0.05)**.**fit(X, y)

selected\_features **=** [i **for** i, j **in** zip(X**.**columns, rfe**.**support\_) **if** j]

**return** selected\_features

In [281]:

y **=** tweak\_runs['max\_heartrate']

X **=** tweak\_runs**.**drop('max\_heartrate',1)

encoded\_y **=** preprocessing**.**LabelEncoder()**.**fit\_transform(y)

model **=** LinearRegression()

linear\_feats **=** get\_best\_rfe\_features(X, encoded\_y, model)

print(linear\_feats)

['moving\_time', 'average\_speed', 'max\_speed', 'average\_heartrate', 'moving\_time\_minutes', 'distance\_km', 'avg\_speed\_kmh', 'max\_speed\_kmh', 'hr\_over\_170']

/Users/willd/anaconda3/lib/python3.7/site-packages/ipykernel\_launcher.py:2: FutureWarning: In a future version of pandas all arguments of DataFrame.drop except for the argument 'labels' will be keyword-only

In [282]:

y **=** runs['max\_heartrate']

X **=** runs[linear\_feats]

X\_train, X\_test, y\_train, y\_test **=** train\_test\_split(X, y, test\_size**=**0.2)

In [283]:

model **=** LinearRegression()

model**.**fit(X\_train,y\_train)

y\_pred **=** model**.**predict(X\_test)

In [284]:

print('MSE:', metrics**.**mean\_squared\_error(y\_test, y\_pred))

plt**.**figure(figsize**=**(10,10))

plt**.**scatter(y\_test, y\_pred, c**=**'crimson')

plt**.**yscale('log')

plt**.**xscale('log')

p1 **=** max(max(y\_pred), max(y\_test))

p2 **=** min(min(y\_pred), min(y\_test))

plt**.**plot([p1, p2], [p1, p2], 'b-')

plt**.**xlabel('True Values', fontsize**=**15)

plt**.**ylabel('Predictions', fontsize**=**15)

plt**.**axis('equal')

plt**.**show()

MSE: 49.41907498120188

In [295]:

model**.**predict(

pd**.**DataFrame(data**=**{

'moving\_time': 120,

'average\_speed': 6,

'max\_speed': 6.67,

'average\_heartrate': 200,

'moving\_time\_minutes': 2,

'distance\_km': .8,

'avg\_speed\_kmh': 23,

'max\_speed\_kmh': 24,

'hr\_over\_170': **True**

},

index**=**[0]

)

)

Out[295]:

array([210.63988302])

In [300]:

y **=** tweak\_runs['hr\_over\_170']

X **=** tweak\_runs**.**drop('hr\_over\_170',1)

model **=** SGDClassifier(loss**=**"hinge", penalty**=**"l2", max\_iter**=**5)

class\_feats **=** get\_best\_rfe\_features(X, y, model)

print(class\_feats)

['distance', 'moving\_time', 'elapsed\_time', 'total\_elevation\_gain', 'average\_heartrate', 'max\_heartrate', 'elev\_high', 'elev\_low', 'year']

/Users/willd/anaconda3/lib/python3.7/site-packages/ipykernel\_launcher.py:2: FutureWarning: In a future version of pandas all arguments of DataFrame.drop except for the argument 'labels' will be keyword-only

In [301]:

y **=** runs['hr\_over\_170']

X **=** runs[class\_feats]

X\_train, X\_test, y\_train, y\_test **=** train\_test\_split(X, y, test\_size**=**0.2)

In [302]:

model **=** SGDClassifier()

model**.**fit(X\_train,y\_train)

y\_pred **=** model**.**predict(X\_test)

In [303]:

print('Accuracy:', accuracy\_score(y\_test, y\_pred))

cm **=** confusion\_matrix(y\_test, y\_pred, labels**=**model**.**classes\_)

disp **=** ConfusionMatrixDisplay(confusion\_matrix**=**cm,display\_labels**=**model**.**classes\_)

disp**.**plot()

plt**.**show()

Accuracy: 0.6120218579234973

In [312]:

model**.**predict(

pd**.**DataFrame(data**=**{

'distance': 800,

'moving\_time': 120,

'elapsed\_time' : 120,

'total\_elevation\_gain': 0,

'average\_heartrate': 200,

'max\_heartrate': 220,

'elev\_high': 900,

'elev\_low': 900,

'year' : 2023

},

index**=**[0]

)

)

Out[312]:

array([ True])

In [ ]: