WHAT MAKES A GOOD CLIMBER?

Statistical Principles of Data Science – Group Project
Group 4

Authors
Christina Kohlbacher, k11824719
David Obermann, k11717395
Fabio Pernegger, k11714227
Richard Wolfmayr, k11714228

Submission
June 2022

Abstract

We analyzed the climber dataset in order to find out which attributes make a good climber. We used a visual analytics approach to get a general understanding of the data and to show correlations of attributes. Also we used regression and classification with the mean grade of the climbers as a target and analyzed which features were important for the regression/classification. We found that the years a person climbed is the most important factor for being a good climber, and that e.g. weight only has minimal influence. This means that the focus of the sports minister should be on motivating young people to climb and on providing ample opportunities to climb. This is important for getting many good climbers and for Austria to become a well known climbing nation.

Table of Contents

Figures

Introduction

Climate change is melting our Austrian glaciers, leading not only to an increase of catastrophic floods but also to a decline in skiers and therefore a decline in Olympic medals. Also, climbing is soon to be made an official Olympic sport. This is why the Austrian minister for sports tasked us with finding attributes that make a good climber. The goal of the project is to provide a solid foundation on which future work can build to start producing world-class climbers "made in Austria" and maybe turn climbing into our new national sport in the long run.

Earning many Olympic medals may seem like a petty goal, but in fact, it is absolutely crucial for Austrian tourism to find alternatives to skiing and create publicity that Austria is a country where you can do other sports like, for example, climbing. Just like everyone knows that you can go to Nepal for great hiking training, everyone should know that you can go to Austria for great climbing training.

The dataset in use provides a large amount of information that can be used to find common attributes of good climbers. The insights should be used in several ways. If, for example, it turns out that starting training at a very young age is crucial, then schools could get subsidies for climbing weeks, just like they get now for skiing weeks. This way, we propose a suggestion for the Austrian ministry of sports on how to proceed with the support of sport climbing.

Data Set

We use the "climb dataset" that is provided on kaggle under the following link: https://www.kaggle.com/datasets/jordizar/climb-dataset. The dataset is a large collection of climbers and route information. This collection is obtained from the largest online climbing logbook in the world 8a.nu: https://www.8a.nu/. The author David Cohen built a Python-based web scraper that collects all users, ascend and route information and stores it in an SQLite database.

The original database built by Cohen is available on Kaggle as well under the following link: https://www.kaggle.com/datasets/dcohen21/8anu-climbing-logbook. Further

Cohen also provides the scrapper that retrieved said data on GitHub: https://github.com/dcohen21/8a.nu-Scraper.

Jordi Zaragoza later on compiled the big collection of information in the database into a cleaned and ready-to-use set of CSV files. These CSV files contain less information than the original database logged by Cohen. However, to fulfill the needs of our task the given data should be more than enough.

Below, we introduce the heads of the tables the dataset holds. A full dictionary to the tables can be found in the appendix. Most interesting to us is definitely the climbers table. This table contains information about the climbers generated from logs of route ascends logged by the climbers themselves.

The grade conversion table allows to convert the french grading system that involves numbers from 1 to 9 (for now) and subcategories using letters from a to c allow more fine grained evaluation. Those grades can then be tuned with +/- to hint for hard and soft and those can again be disputed within the community and are then separated by a /. An example would be 7a+/b or 7a/+. These are sorted and enumerated to create a continuous value e.g. the mean grade.

The route table contains a list of routes logged into the database. We could use this information to link the amount of hard routes to the number of good climbers for example.

	user_id	country	sex	height	weight	age	years_cl	date_first	date_last	grades_count	grades_first	grades_last	grades_max	grades_mean	year_first	year_last
0	1	SWE	0	177	73	41.0	21	1999-02-06 23:00:00	2001-07-31 22:00:00	84	36	55	62	46.750000	1999	2001
1	3	SWE	0	180	78	44.0	22	1999-03-31 22:00:00	2000-07-19 22:00:00	12	53	51	59	52.833333	1999	2000
2	4	SWE	1	165	58	33.0	16	2004-06-30 22:00:00	2009-05-26 22:00:00	119	53	49	64	53.890756	2004	2009
3	10	SWE	0	167	63	52.0	25	2000-01-14 23:00:00	2017-06-01 22:00:00	298	53	49	63	49.406040	2000	2017
4	16	NOR	0	177	68	44.0	21	1998-02-27 23:00:00	2010-05-13 22:00:00	5	53	49	53	51.400000	1998	2010

Figure 1: Climbers Dataset Head

	Unnamed: 0	grade_id	grade_fra
0	0	0	-
1	1	1	-
2	2	2	-
3	3	3	1
4	4	4	1a
5	5	5	1b
6	6	6	1c
7	7	7	1+
8	8	8	2
9	9	9	2a

Figure 2: Grade Dataset Head

	Unnamed: 0	name_id	country	crag	sector	name	tall_recommend_sum	grade_mean	cluster	rating_tot
0	0	0	and	montserrat	prohibitivo	diagonal de la x	-1	49.250000	3	-0.045211
1	1	1	and	montserrat	prohibitivo	mehir	-1	49.000000	2	0.116464
2	2	2	and	montserrat	prohibitivo	pas de la discordia	0	49.000000	2	0.178722
3	3	3	and	tartareu	bombo suis	tenedor libre	0	44.333333	3	0.158449
4	4	4	arg	bandurrias	rincon	tendinitis	1	48.500000	0	0.075797

Figure 3: Routes Dataset Head

Methods

The Methods that we employed are regression since we want to create a statistical framework that helps us identify traits and features of a good climber. Classification was also useful to find out which attributes are important. For the most part, we employ visual analysis techniques to gain a general understanding of the data.

We used a train-test split of 70 percent training and 30 percent test data and scaled the features since it is a best practice. Also, it makes sense to scale since we have very different attributes with highly differing values. We implemented every model twice, once for male and once for female. Stratifying it to have 50/50 males and females in the dataset would not be feasible, since we only have two percent of females. We included all columns that are not directly influencing the mean_grade. The count_grades and years_cl were borderline, and we expected them to be highly influential, but we still kept them to find out for sure. The following figure shows a code snippet for creating the training and test data.

```
x_column_names_c = ['countryenc', 'height', 'weight', 'age', 'years_cl', 'grades_count', 'year_first', 'year_last']

X_m_c = df_climber_m[x_column_names_c]
y_m_c = df_climber_m.grades_mean_discrete
X_train_m_c, X_test_m_c, y_train_m_c, y_test_m_c = train_test_split(X_m_c, y_m_c, test_size=0.3, random_state=random_state)

X_f_c = df_climber[x_column_names_c]
y_f_c = df_climber.grades_mean_discrete
X_train_f_c, X_test_f_c, y_train_f_c, y_test_f_c = train_test_split(X_f_c, y_f_c, test_size=0.3, random_state=random_state)


scaler = preprocessing.StandardScaler().fit(X_train_m_c)
X_train_m_scaled_c = scaler.transform(X_train_m_c)
X_test_m_scaled_c = scaler.transform(X_test_m_c)

scaler = preprocessing.StandardScaler().fit(X_train_f_c)
X_train_f_scaled_c = scaler.transform(X_train_f_c)
X_test_f_scaled_c = scaler.transform(X_test_f_c)
```

Figure 4: Code Snippet - Training and Test Data Creation

Regression

We chose to use linear regression as one of our methods, because it is best fitted for continuous data as the target. As our target variable we use the mean grade, which is the mean difficulty of all climbs logged. With its coefficients we were able to find the most and least beneficial features to achieve a high mean grade.

We fit three regression models: two for sex-splitted data respectively and one on the whole dataset including the sex column to gain insight into how important that feature is. In the Results chapter we then compare these three models and interpret the results of those in the discussion. We think further elaboration of the code is not necessary since we simply fit a simple linear regression model.

Classification

For the classification it would not make sense to use every single possible grade as a class. This would be 85 classes from 0 to 85. This is why we decided to discretise it into three distinct classes. We split the grade_mean into beginner=0, enthusiast=1, pro=2. Using a simple split that divided into three groups, each covering ⅓ of the points range would not have led to useful results since in the lowest grades almost no one climbs. We used our "expert knowledge" to find the following borders of the three classes in the bullet list below.

- 1 to 6b → beginner...45=6c
- 6c to 8 → enthusiast...61=8a
- 8a and upwards → pro...>=61

Decision Tree

We decided to use a decision tree because it is easy to interpret and visualize. As a reminder, we do not want to build a classifier only for being able to predict good climbers, but mainly to find out which attributes are important for classification. Decision trees show at which attribute the splits are made and can be read by humans. The following two figures are two code snippets which show the creation, fitting, and plotting of the decision trees, as well as the calculation and plotting of the accuracy scores and feature importances.

```
# male tree
tree_m = tree.DecisionTreeClassifier(criterion="entropy", random_state=random_state)
tree_m = tree_m.fit(X_train_m_scaled_c,y_train_m_c)
plt.figure(figsize=(15,25))
tree.plot_tree(tree_m, max_depth=3, feature_names=x_column_names_c, fontsize=8, class_names=["beginner", "enthusiast", "pro"])

# female tree
tree_f = tree.DecisionTreeClassifier(criterion="entropy", random_state=random_state)
tree_f = tree_f.fit(X_train_f_scaled_c,y_train_f_c)
plt.figure(figsize=(15,25))
tree.plot_tree(tree_f, max_depth=3, feature_names=x_column_names_c, fontsize=8, class_names=["beginner", "enthusiast", "pro"])
```

Figure 5: Code Snippet - Decision Tree Creation, Fitting, Plotting

```
# check accuracy
y_pred_m_c = tree_m.predict(X_test_m_scaled_c)
accuracy_m_c = accuracy_score(y_test_m_c.values, y_pred_m_c)

y_pred_f_c = tree_f.predict(X_test_f_scaled_c)
accuracy_f_c = accuracy_score(y_test_f_c, y_pred_f_c)

print(f"Accuracy for male tree: {accuracy_m_c}")
print(f"Accuracy for female tree: {accuracy_f_c}")

# feature importance
feature_importances_c = tree_m.feature_importances_
plt.figure(figsize=(10,5))
plt.bar([i for i in range(0, len(feature_importances_c))], feature_importances_c)
plt.xticks([i for i in range(0, len(x_column_names_c))], x_column_names_c)
plt.title(f'feature importance for Male Decision Tree')
plt.xlabel('features')
plt.ylabel('importance score')

feature_importances_c = tree_f.feature_importances_
plt.figure(figsize=(10,5))
plt.bar([i for i in range(0, len(feature_importances_c))], feature_importances_c)
plt.xticks([i for i in range(0, len(x_column_names_c))], x_column_names_c)
plt.title(f'feature importance for Female Decision Tree')
plt.xlabel('features')
plt.ylabel('importance score')
```

Figure 6: Code Snippet - Decision Tree - Calculation and Plotting of Accuracy and Feature Importances

Random Forest

We used a random forest in addition to the simple decision tree since random forests generally just perform better. This was the case for our project as well, the accuracy was significantly higher for the random forest compared to the decision tree. We do not include much code for the random forests since they are fairly similar to the decision trees. The figure below shows a code snippet containing the creation of the random forests, as well as the calculation and plotting of the accuracy scores.

```
randforest_m = RandomForestClassifier(random_state=random_state)
randforest_m = randforest_m.fit(X_train_m_scaled_c, y_train_m_c)

randforest_f = RandomForestClassifier(random_state=random_state)
randforest_f = randforest_f.fit(X_train_f_scaled_c, y_train_f_c)

y_pred_m_c = randforest_m.predict(X_test_m_scaled_c)
accuracy_m_c = accuracy_score(y_test_m_c.values, y_pred_m_c)

y_pred_f_c = randforest_f.predict(X_test_f_scaled_c)
accuracy_f_c = accuracy_score(y_test_f_c, y_pred_f_c)

print(f"Accuracy for male forest: {accuracy_m_c}")
print(f"Accuracy for female forest: {accuracy_f_c}")
```

Figure 7: Code Snippet - Random Forest Creation, Accuracy Calculation and -Plotting

Results

Exploratory Analysis

For an appropriate result, we decided to split the data and execute the exploratory analysis as well as the data modeling for both data sets individually. We decided this because the features for males and females are differently distributed and the original data set is highly skewed in terms of sex (see figure).

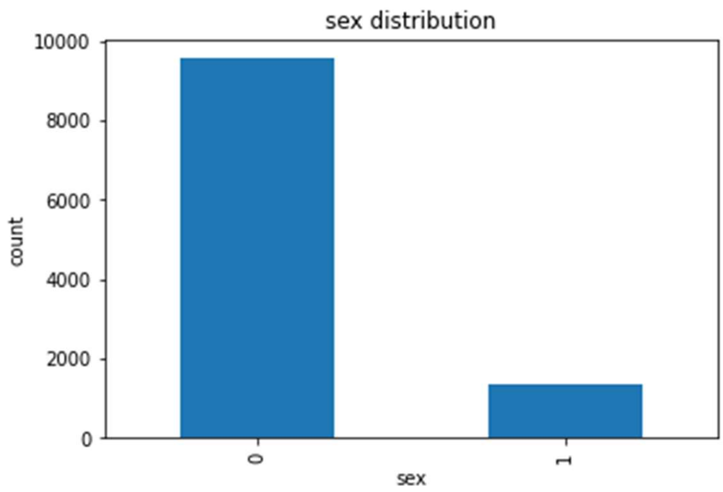


Figure 8: Dataset Sex Distribution

The next features we looked into are the country distribution, as well as the average grades mean per country - in other words, which countries have the best climbers (on average). The three countries with the most documented climbers in the data set (for male as well as female) are Spain, USA, and Poland. The least number of documented climbers are from Finland, Czech

Republic and Denmark. The original plot can be found in the notebook in the appendices.

On average, the best male, as well as female, cimbers (regarding the grades mean) are from Czech Republic. While Slovenia and Austria have the second- and third-best female climbers, France and Great Britain have the top two and three male climbers. A detailed overview regarding this analysis is shown in the plot below.

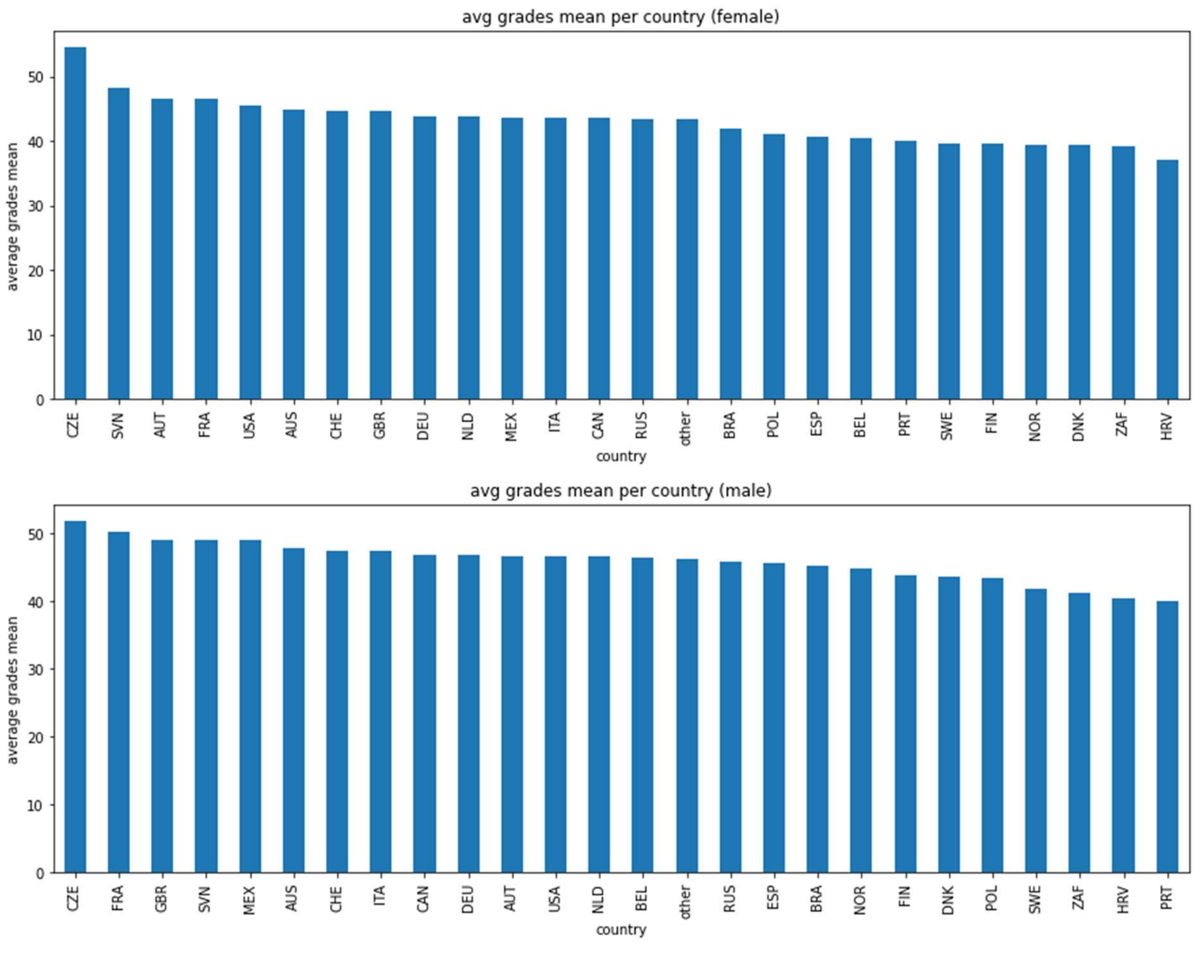


Figure 9: Average Grade Mean per Country, Split by Male and Female

Subsequently, we looked into the overall pro climber count by country. The most female pros in the data set are from Austria (three in total). Belgium and Czech Republic have two female pro climbers. For the male pro climbers, France gets the first spot with around 50 climbers, while Italy and “other” are on places two and three.

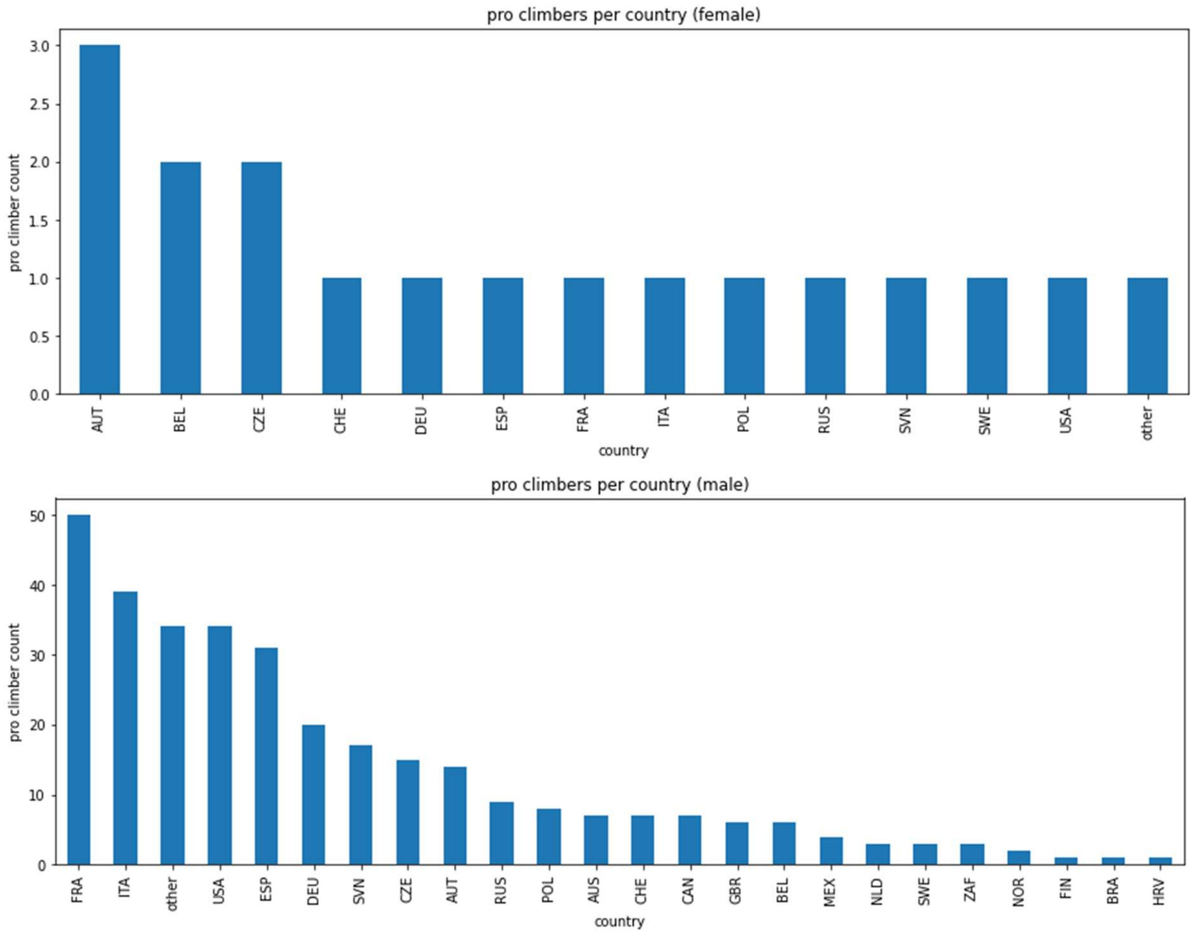


Figure 10: Pro Climbers per Country, Split by Male and Female

Furthermore, we created boxplots (female and male) for the features height, weight, age, years_cl (years climbed), grades_count, grades_first, grades_last, grades_max, grades_mean, year_first, and year_last. Each of the plots can be found in the notebook in the appendices. While the interquartile range of height is between approximately 175cm and 185cm in males, it is between around 160cm and 170cm for female climbers. While the middle 50 percent of male climbers weigh roughly 65-75kg, the female mid-50 percent is roughly 10kg lower with approximately 47-58kg. The distribution of the grade features is overall highly similar for females and males with the exception of grades_max where the interquartile range for males is roughly 48-62 while it is around 45-55 for females.

Finally, we created a correlation plot of the features we used for model creation later on. For instance, the results of the correlation plot show that weight and height are highly (positive) correlated and both are highly (negative) correlated with sex. Another high, positive correlation was identified between years climbed and age. Furthermore, the year of the last ascension positively correlates with grade count.

	sex	height	weight	age	years_cl	grades_count	grades_mean	year_first	year_last
sex	1.00	-0.52	-0.54	-0.03	-0.07	-0.03	-0.12	0.01	-0.00
height	-0.52	1.00	0.75	0.10	0.03	0.01	-0.02	-0.00	-0.02
weight	-0.54	0.75	1.00	0.23	0.06	-0.00	-0.12	-0.01	-0.03
age	-0.03	0.10	0.23	1.00	0.53	0.10	-0.07	-0.07	-0.25
years_cl	-0.07	0.03	0.06	0.53	1.00	0.11	0.37	-0.11	-0.36
grades_count	-0.03	0.01	-0.00	0.10	0.11	1.00	0.14	-0.02	0.38
grades_mean	-0.12	-0.02	-0.12	-0.07	0.37	0.14	1.00	-0.04	0.10
year_first	0.01	-0.00	-0.01	-0.07	-0.11	-0.02	-0.04	1.00	0.09
year_last	-0.00	-0.02	-0.03	-0.25	-0.36	0.38	0.10	0.09	1.00

Figure 11: Correlation Matrix of Climber Features

Linear Regression Model Results

We provide a table that compares the Linear Regression models below. The top insights we can gain from this are the coefficients learned by the model. Mainly interesting are the highest positive and negative values. In this case, the years climbed, the age, the height and, in contrast to this, the weight. year_last could be important although, as we will discuss later, this has some threats to its validity given the data. The model was evaluated using the mean squared error. We tried the most basic models to understand the results the best. A further look into the results is made in the discussion.

Full Data:		**Males:**		**Females:**	
countryenc:	-0.4122	countryenc:	-0.4955	countryenc:	-0.1750
sex :	-1.334				
height :	0.5881	height :	0.4034	height :	0.7641
weight:	-1.8561	weight:	-1.4163	weight:	-1.570
age :	-2.2582	age :	-2.4639	age :	-1.858
years_cl:	4.4833	years_cl:	4.6403	years_cl:	5.0662
grades_count:	-0.0999	grades_count:	-0.1200	grades_count:	0.1665
year_first:	-0.5528	year_first:	-0.5136	year_first:	-0.3051
year_last:	2.1599	year_last:	2.0128	year_last:	2.9760
MSE: 41.01		MSE: 41.09		MSE: 38.15	

Figure 12: Mean-Squared Error and Coefficients of the Regression Models

Classification Model Results

The decision tree for males and females look fairly similar. We plotted only until depth of three, even though the actual trees are much deeper. On the following pages, the two decision trees are shown. The first split is in years climbedfor male and female. years_cl is also used in later splits again. Age, weight, year_last and grades_count are also shown in the first few splits; the other attributes are used further down in the tree.

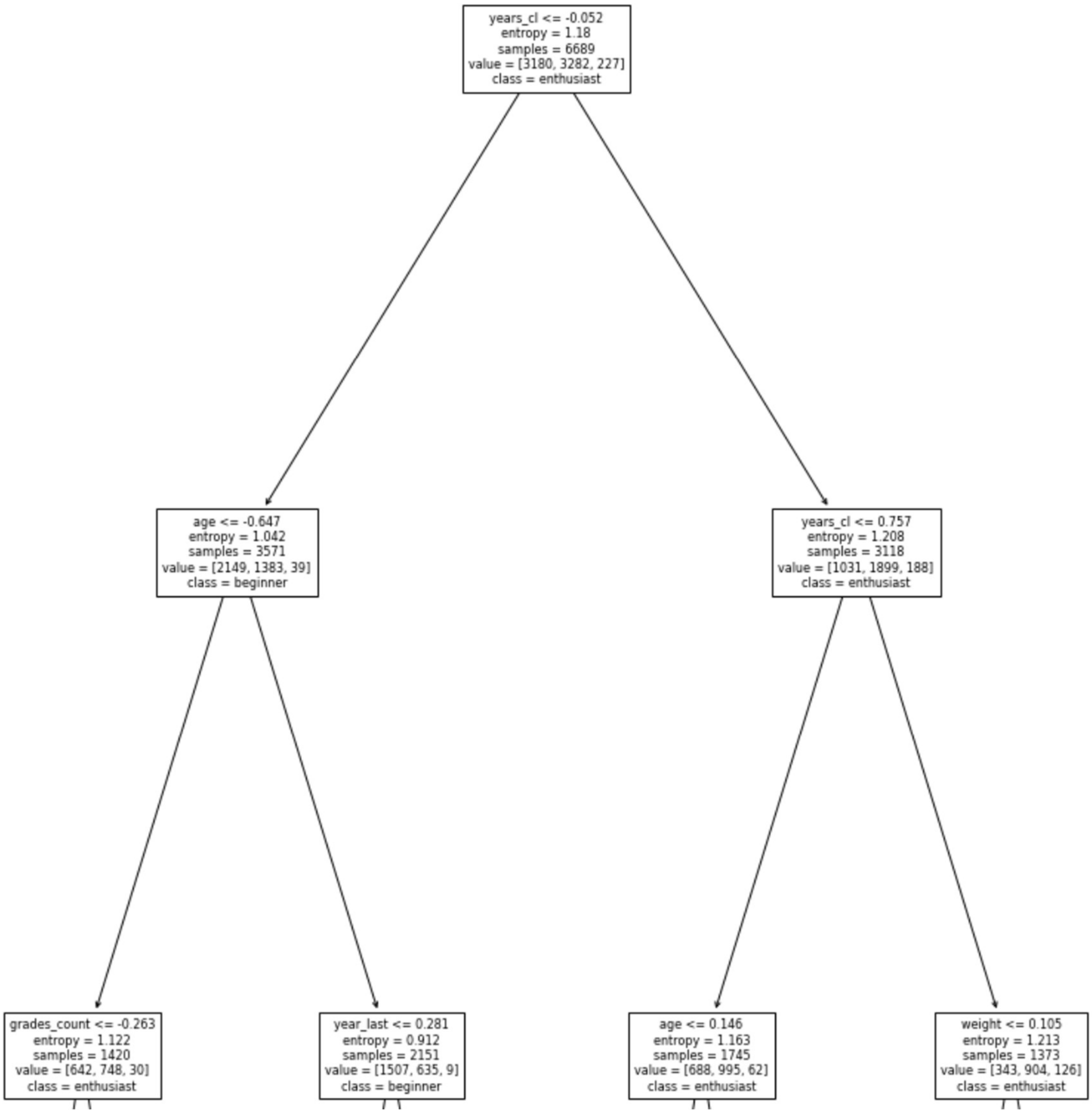


Figure 13: Decision Tree Male Dataset

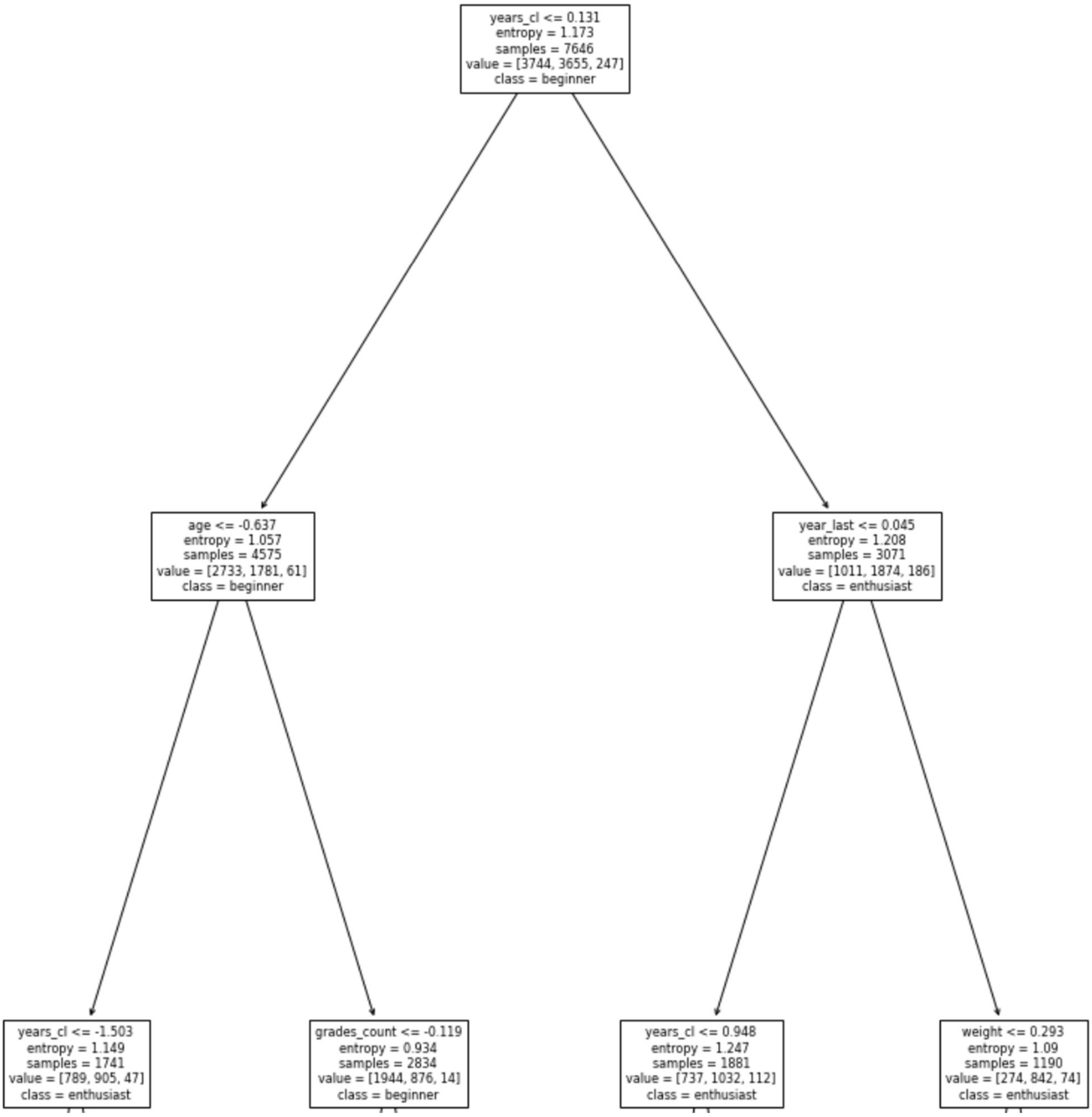


Figure 14: Decision Tree Female Dataset

For a more concrete description of which attribute is important for the decision tree, we included a visualization of the importance score below. The three most important features for both male and female climbers are grades_count, years_cl , and age. The least important feature is weight, also for both sexes.

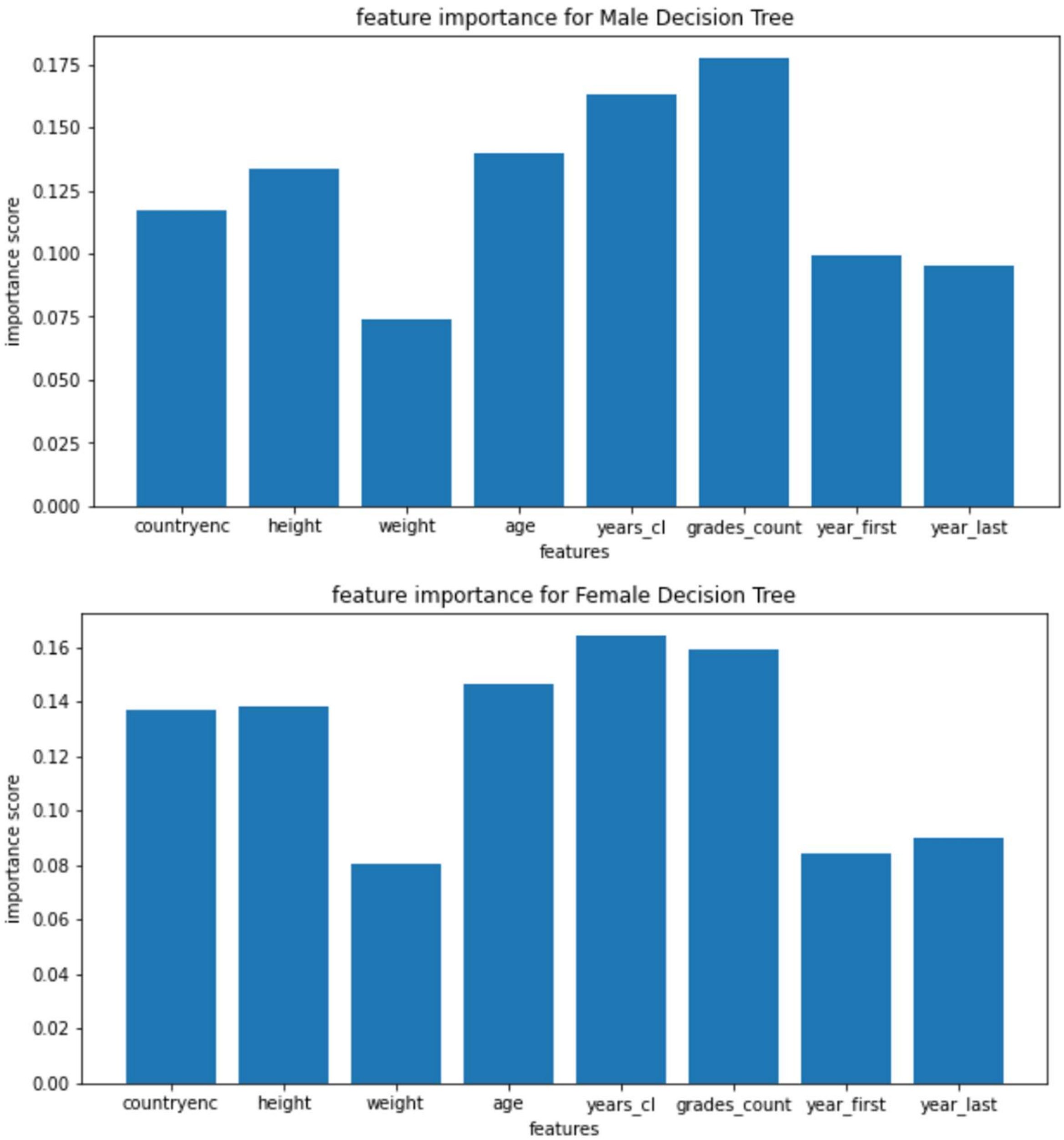


Figure 15: Decision Tree Feature Importance; Split by Male and Female

For the random forest, the visualized importance scores can be found in the illustration below. Here, grades_count, years_cl, and age are again the three most important features for both male and female climbers, and weight is the least important one once more.

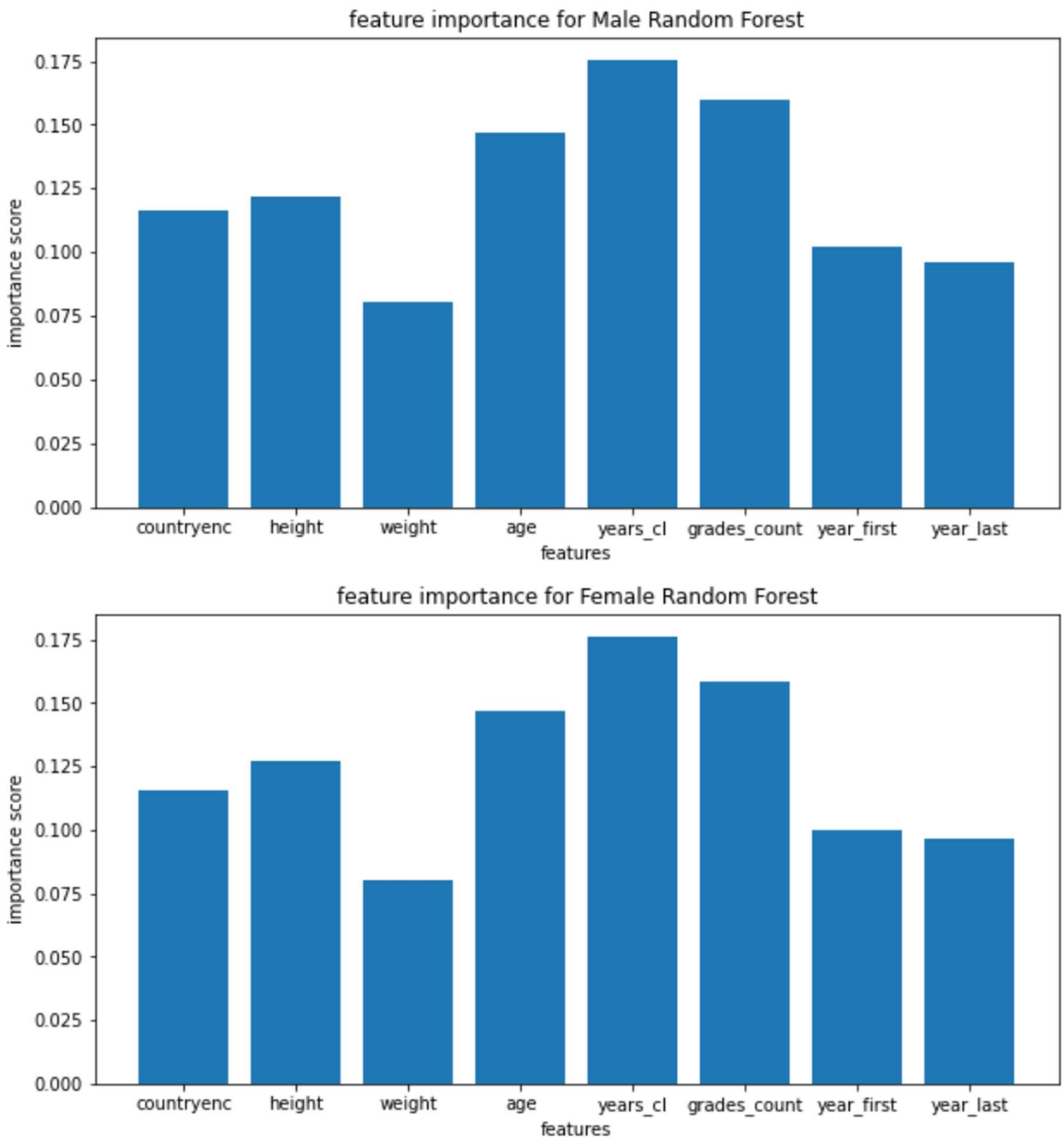


Figure 16: Random Forest Feature Importance; Split by Male and Female

The accuracy scores for each of the classifiers are shown in the bullet list below.

- Decision Tree

    - Accuracy for male tree: 0.595536959553696
    - Accuracy for female tree: 0.5951799877974374
- Random Forest
    - Accuracy for male forest: 0.6722454672245467
    - Accuracy for female forest: 0.6851738865161684

Discussion

Exploratory Analysis

When looking at the results for female climbers, it has to be taken into account that the data set of females is very small in contrast to the male data set. Therefore, the female climber analysis results are likely less accurate than those of male climbers.

It was interesting to see that, while the average grade_means both sexes are fairly similar for all the countries, there are clear differences in the pro climb_counts, both for female and male climbers. Moreover, the differences in height and weight distributions for males and females was as expected, the females being on average smaller and lighter than their male counterparts.

As presented in the Results Chapter, the grade features are, for the most part, very similar for both male and female climbers. The only visible difference is a higher interquartile range for max_grade for male climbers by approximately 10 grading points. However, it is interesting that the highest max_grade are the same for both sexes once more.

Regression

What we can see from the coefficients of the trained models is that generally the years climbed influence the mean grade the most. Firstly, this is obviously to expect, secondly we can further derive from that: possibly and logically the earlier a climber starts with their climbing career the better. This is also backed by the high negative importance of age. The importance of sex is negligible since this is not within the scope of our observation, although it is interesting to see how large the gap between the performance really is. Apparently, the country does not influence the regression much, although we suspect this could be further analyzed.

The high importance of year_last seems to be that the sport develops over the years and the general performance level rises with the development. An elevated negative importance of weight followed by a positively hinting importance of height suggest that high performing climbers are taller and lighter, which is obvious in the nature of the sport. A negative importance in the year first values shows that climbers that started later also perform higher, this again is probably linkable to the development of the sport.

Classification

As we expected, the first split for the decision tree is by years_cl. If you have climbed for a long time, apparently you will probably be a good climber. The visualization of the importance score again shows that years_cl is the most important factor. The grades_count, which are just as important, are actually very dependent on years_cl, since if you climb for a longer time, you will have more opportunity to accumulate grades. Interestingly and surprisingly, weight is the least influential attribute for the decision tree and random forests, in stark contrast to the importance of weight for the regression classifier. It should be noted that the climbers are relatively light in comparison to average persons.

Country of the climber also is fairly important for the classifiers, which is why we took a closer look at where the best climbers come from. Refer to the Exploratory Analysis to see what we found out. Another interesting observation is that higher age of course correlates negatively, so climbers should be young, but also have a lot of experience.

Threats to validity

The online logging service was introduced in 1999. Logs before that thus only exist rarely. Further this means that the popularity of the logging service was increasing over the years. During this time, the sport developed rapidly, harder routes got established over the years and rock climbing reached its bloom late into the 2000s and 2010s. This also increases the population that gets into climbing. So over the years the data gets more reliable. But also in our case it is important to keep in mind that the result regarding first ascends over the years and last logged ascends are to be interpreted with care given the circumstances of the sport development and the increased usage of the logging service. We also think that the logging service generally is more used by younger people, this might also influence the validity of our drawn conclusions.

Conclusion

We want to give the sports minister some advice on how to increase the amount of good climbers in Austria:

- Talk to Belgium, France, USA, and Italy since they have the most pro climbers apart from Austria
- Talk to Czech Republic, Great Britain, France, and Slovenia since they have on average the best climbers
- Start promoting at a young age since years climbed is very important and climbers are best when they are young

Appendix

Climber Data Dictionary

Columns:

- **user_id**: unique key for each row (Integer)
- **country**: the 3 letter short form of the origin country of the climber (String)
- **sex**: The sex of the climber (Integer)
- **height**: the height of the climber (Integer)
- **weight**: the weight of the climber (Integer)
- **age**: age of the climber (Float)
- **years_cl**: how many year the climber is practicing this sport (Integer)

- **date_first**: the date of the first ascension (Date)
- **date_last**: the date of the last ascension (Date)
- **grades_count**: the number of routes done by the climber (Integer)
- **grades_first**: the difficulty of the first route described by a number from 0 to 84 (Integer)
- **grades_last**: the difficulty of the last route described by a number from 0 to 84 (Integer)
- **grades_max**: the difficulty of the hardest route described by a number from 0 to 84 (Integer)
- **grades_mean**: the mean difficulty of all routes climbed described by a number from 0 to 84 (Float)
- **year_first**: year of the first ascension (Integer)
- **year_last**: year of the last ascension (Integer)

Grades Conversion Data Dictionary

Columns:

- **grade_id**: shows difficulty based on a scale from 0 to 84 (Integer)
- **grade_fra**: shows difficulty based on the french grading which is the most used system by rock-climbers (String)

Routes Data Dictionary

Columns:

- **name_id**: unique key for each row (Integer)
- **country**: the 3 letter short form of the origin country of the climber (String)
- **crag**: Name of the cliff/location (String)
- **sector**: Area in the crag (String)
- **name**: Name of the route (String)
- **tall_recommend_sum**: Value that shows if the route is easier for tall people. Negative Value: Easier for short people. Positive Value: Easier for tall people. (Integer)

- **grade_mean**: Value for the difficulty. High value = difficult. Various people graded the route ⇒ mean(Integer)
- **cluster**: The author of the dataset has clustered the routes as follows:
    - 0 - Soft routes
    - 1 - Routes for some reason preferred by women
    - 2 - Famouse routes
    - 3 - Very hard routes
    - 4 - Very repeated routes
    - 5 - Chipped routes, with soft rate
    - 6 - Traditional, not chipped routes
    - 7 - Easy to On-sight routes, not very repeated
    - 8 - Very famous routes but not so repeated and not so traditional
- **rating_tot**: The author did this calculation based on 3 features (comment sentiment, rating, recommendations) and took the first component of the PCA:

Code - Jupyter Notebook (including exploratory analysis plots)

(See following page)

Sheet

Statistical Principles of Data Science - Group Project

What makes a good climber?

Hand-In Date: xx.xx.xxxx

Christina Kohlbacher, k11824719
David Obermann, k11717395
Fabio Pernegger, k11714227
Richard Wolfmayr, k11714228

Imports

```
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.preprocessing import LabelEncoder
from sklearn import preprocessing
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score
from sklearn.model_selection import train_test_split
from sklearn import tree
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
```

Load Data Set

```
df_climber = pd.read_csv('climber_df.csv')
df_climber_orig = df_climber.copy()
df_climber.head()
random_state = 1337
```

```
df_routes = pd.read_csv('routes_rated.csv')
df_routes_orig = df_routes.copy()
df_routes.head()
```

	Unnamed: 0	name_id	country	crag	sector	name	tall_recommend_sum	grade_mean	cluster	rating_tot
0	0	0	and	montserrat	prohibitivo	diagonal de la x	-1	49.250000	3	-0.045211
1	1	1	and	montserrat	prohibitivo	mehir	-1	49.000000	2	0.116464
2	2	2	and	montserrat	prohibitivo	pas de la discordia	0	49.000000	2	0.178722
3	3	3	and	tartareu	bombo suis	tenedor libre	0	44.333333	3	0.158449
4	4	4	arg	bandurrias	rincon	tendinitis	1	48.500000	0	0.075797

```
df_grades = pd.read_csv('grades_conversion_table.csv')
df_grades_orig = df_grades.copy()
df_grades.head()
```

	Unnamed: 0	grade_id	grade_fra
0	0	0	-
1	1	1	-
2	2	2	-
3	3	3	1
4	4	4	1a

Data Understanding - Exploratory Analysis

First look into the climbers dataframe - print info

As you can see below, there are no missing values in the data set.

```
df_climber.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10927 entries, 0 to 10926
Data columns (total 16 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   user_id       10927 non-null  int64
 1   country       10927 non-null  object
 2   sex           10927 non-null  int64
 3   height        10927 non-null  int64
 4   weight        10927 non-null  int64
 5   age           10927 non-null  float64
 6   years_cl      10927 non-null  int64
 7   date_first    10927 non-null  object
 8   date_last     10927 non-null  object
 9   grades_count  10927 non-null  int64
 10  grades_first  10927 non-null  int64
 11  grades_last   10927 non-null  int64
 12  grades_max    10927 non-null  int64
 13  grades_mean   10927 non-null  float64
 14  year_first    10927 non-null  int64
 15  year_last     10927 non-null  int64
dtypes: float64(2), int64(11), object(3)
memory usage: 1.3+ MB
```

Next, we want to get a description with basic statistical measures of the features.

```
df_climber[['height', 'weight', 'age', 'years_cl', 'grades_count', 'grades_first',
            'grades_last', 'grades_max', 'grades_mean', 'year_first', 'year_last']].describe()
```

	height	weight	age	years_cl	grades_count	grades_first	grades_last	grades_max	grades_mean	year_first	year_last
count	10927.000000	10927.000000	10927.000000	10927.000000	10927.000000	10927.000000	10927.000000	10927.000000	10927.000000	10927.000000	10927.000000
mean	176.152009	67.608676	33.333852	12.672188	79.794546	45.648851	46.983802	53.764437	45.505055	2008.621946	2012.311613
std	8.508669	9.677316	7.590989	6.108451	141.411297	9.478173	9.418087	9.679533	7.891356	28.833298	4.161484
min	137.000000	40.000000	12.000000	1.000000	1.000000	28.000000	28.000000	29.000000	28.500000	0.000000	1991.000000
25%	171.000000	63.000000	28.000000	8.000000	8.000000	38.000000	40.000000	46.000000	39.400000	2006.000000	2009.000000
50%	177.000000	68.000000	33.000000	12.000000	28.000000	46.000000	48.000000	55.000000	45.151899	2009.000000	2013.000000
75%	182.000000	73.000000	38.000000	17.000000	90.000000	53.000000	53.000000	62.000000	51.210084	2012.000000	2016.000000
max	202.000000	93.000000	69.000000	29.000000	2445.000000	75.000000	77.000000	77.000000	75.272727	2017.000000	2017.000000

The mode of the nominal features is shown below.

```
df_climber[['country', 'sex']].mode(axis=0)
```

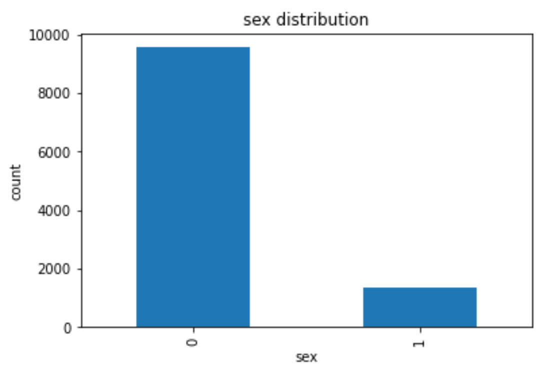
	country	sex
0	ESP	0

Let's look at the specific features and their distributions explicitly.

```
def plot_description(title, xlabel, ylabel):
    plt.title(title)
    plt.xlabel(xlabel)
    plt.ylabel(ylabel)
    #plt.show()
```

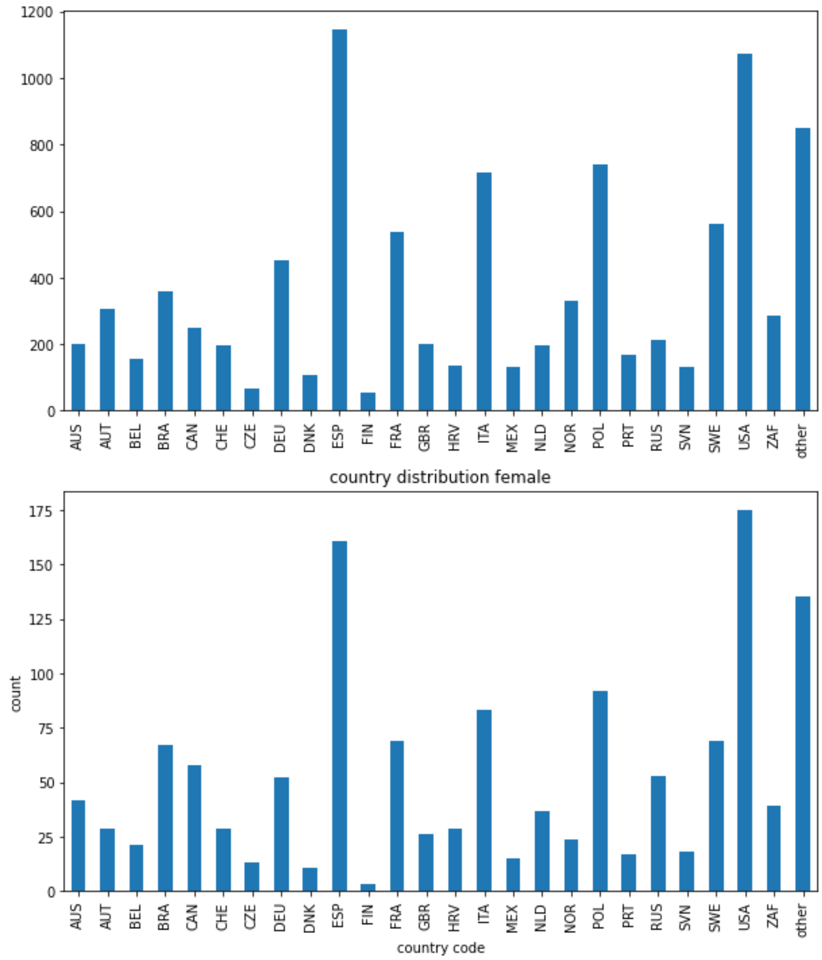
```
def plot_my_boxplot(col, unit):
    fig, axs = plt.subplots(1,2,figsize=(10,5))
    df_climber[df_climber.sex == 0][col].plot(kind='box' ,ax=axs[0])
    axs[0].set_title(f'{col} distribution male'), axs[0].set_xlabel(' '), axs[0].set_ylabel(f'{col} {unit}')
    df_climber[df_climber.sex == 1][col].plot(kind='box' , ax=axs[1])
    axs[1].set_title(f'{col} distribution female'), axs[0].set_xlabel(' ') , axs[1].set_ylabel(f'{col} {unit}')
```

```
df_climber.sex.value_counts().plot(kind='bar')
plot_description('sex distribution', 'sex', 'count')
```

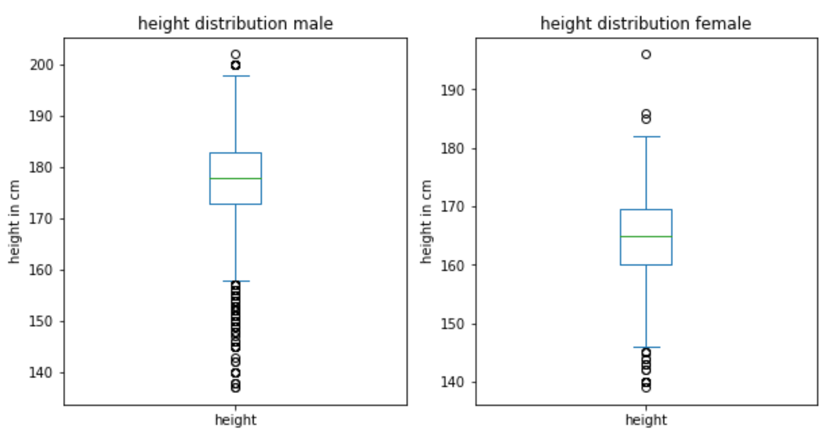


As shown in the plot above, the data is highly skewed in terms of sex distribution. We decided to split the data into two dataframes and create the models for both of the groups because different features might be important for each of them, and each feature is differently distributed. We also perform the exploratory analysis for both groups.

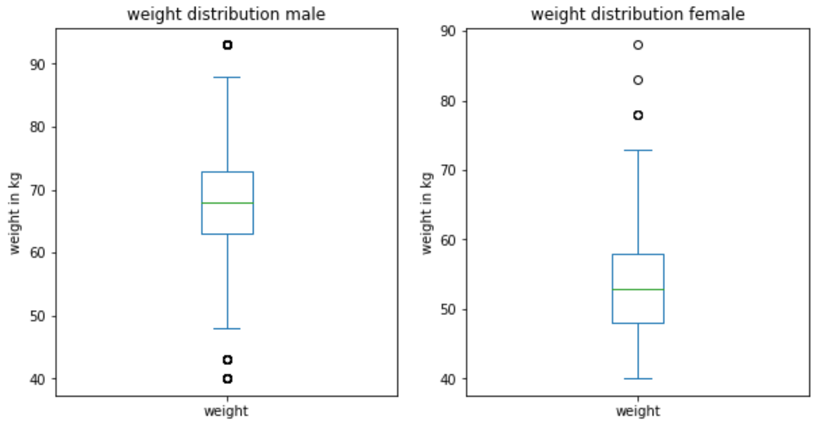
```
fig, axs = plt.subplots(2,1,figsize=(10,12))
df_climber[df_climber.sex == 0].country.value_counts().sort_index().plot(kind='bar', ax= axs[0])
plot_description('country distribution male', 'country code', 'count')
df_climber[df_climber.sex == 1].country.value_counts().sort_index().plot(kind='bar', ax= axs[1])
plot_description('country distribution female', 'country code', 'count')
```



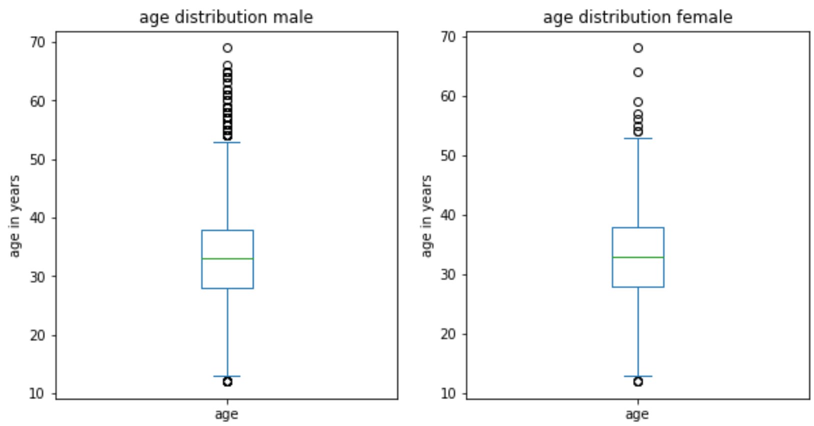

```
plot_my_boxplot('height', 'in cm')
```



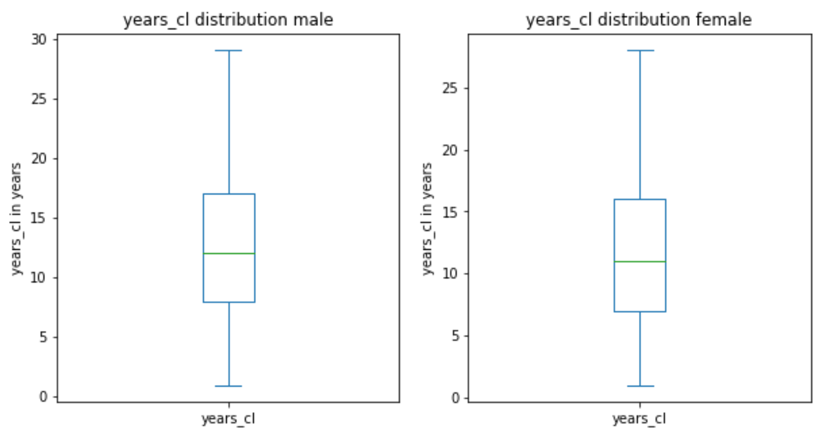

```
plot_my_boxplot('weight', 'in kg')
```



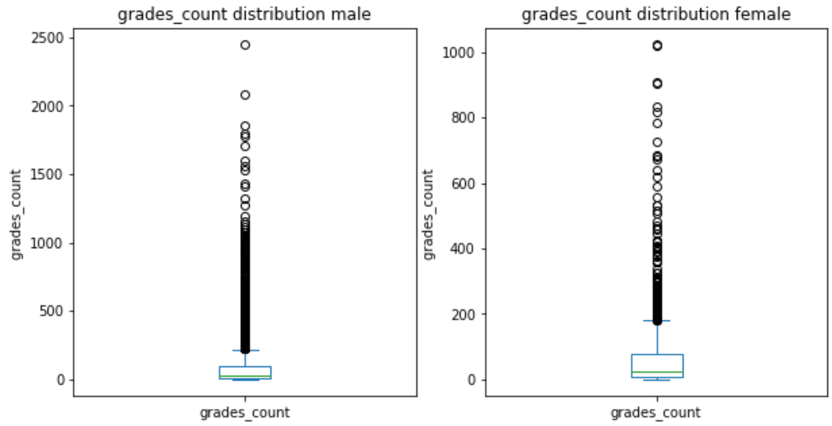

```
plot_my_boxplot('age', 'in years')
```



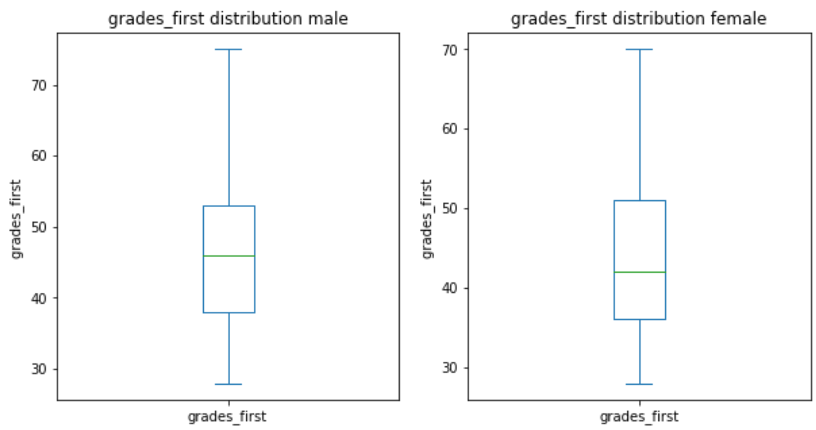

```
plot_my_boxplot('years_cl', 'in years')
```



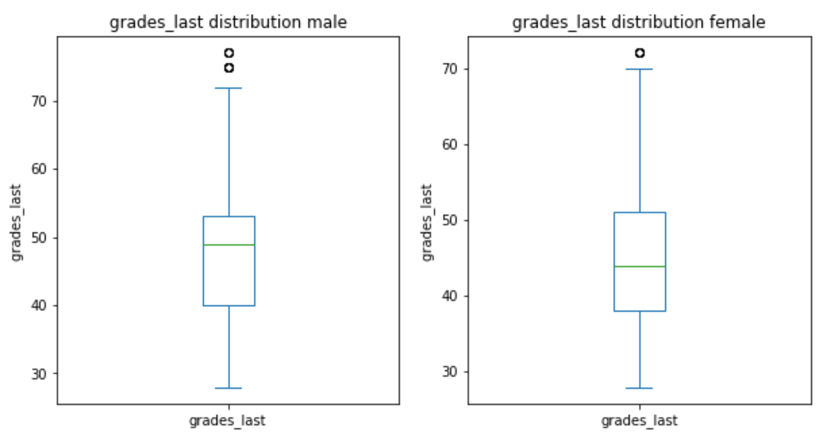

```
plot_my_boxplot('grades_count', '')
```



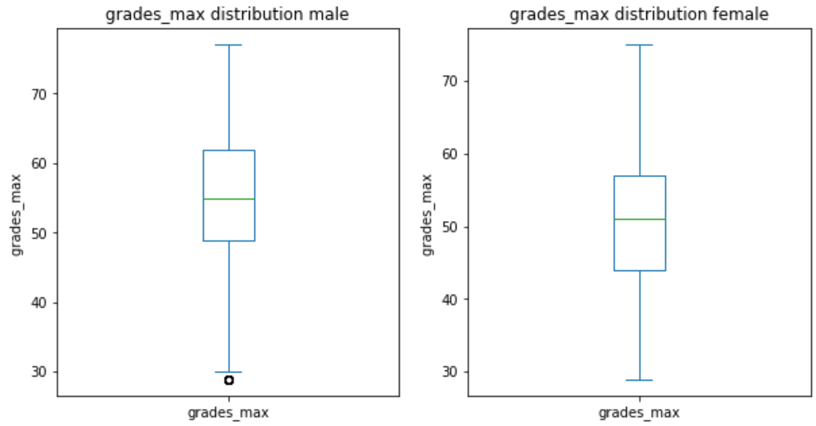

```
plot_my_boxplot('grades_first', '')
```



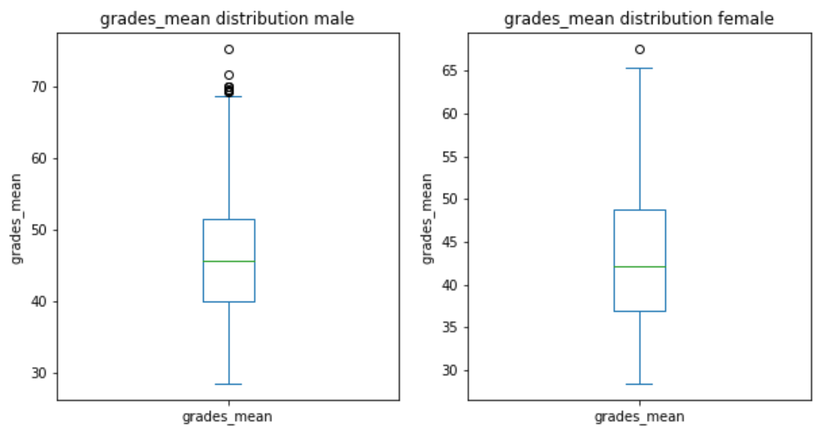

```
plot_my_boxplot('grades_last', '')
```



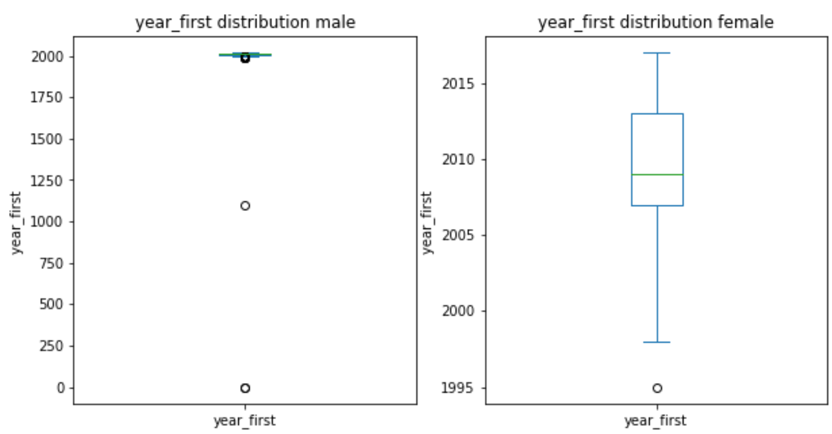

```
plot_my_boxplot('grades_max', '')
```




```
plot_my_boxplot('grades_mean', '')
```

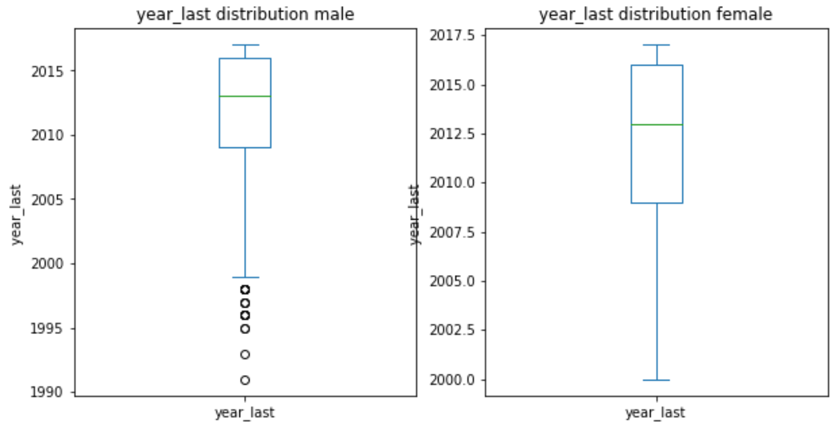



```
plot_my_boxplot('year_first', '')
```

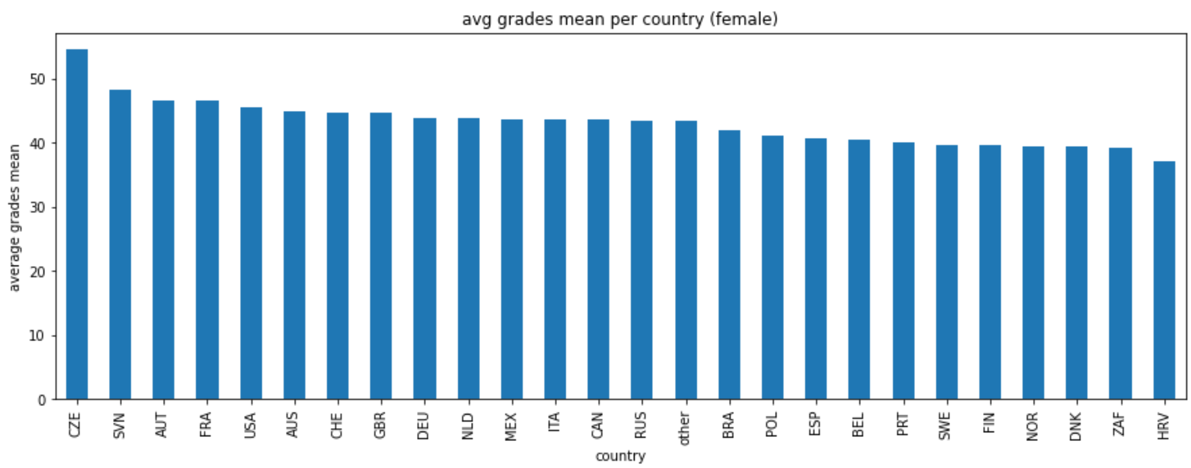


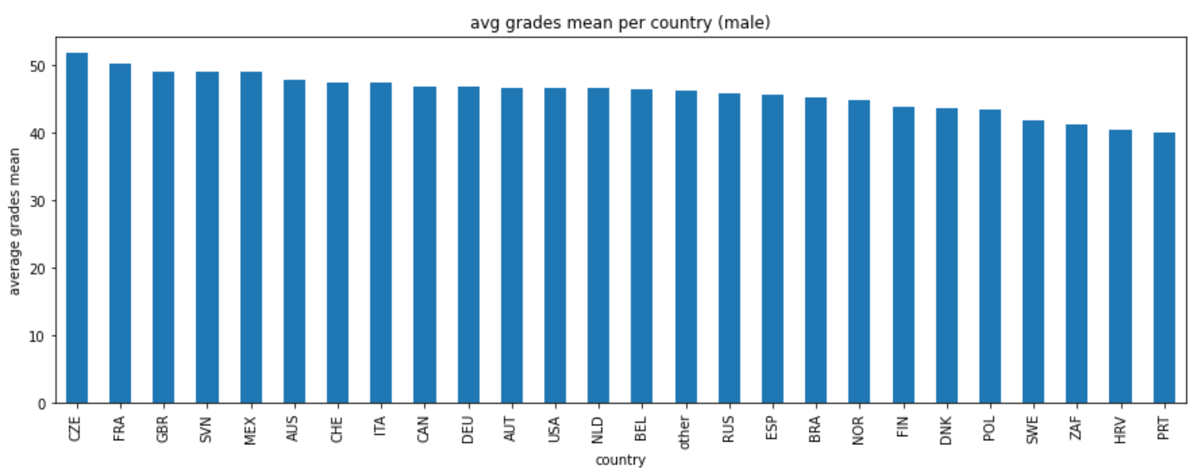
the climber rows with year_first below 1950 should be omitted from the data set since it is not realistic to have years 0 or 1100.

```
plot_my_boxplot('year_last', '')
```




```
df_climber[['country','grades_mean']][df_climber.sex == 1].groupby('country').mean().sort_values('grades_mean', ascending=False).plot(
df_climber[['country','grades_mean']][df_climber.sex == 0].groupby('country').mean().sort_values('grades_mean', ascending=False).plot(
```






```
(<AxesSubplot:title={'center':'avg grades mean per country (male)'}, xlabel='country', ylabel='average grades mean'>,
 None)
```

```
corr_plt = df_climber.drop(columns=['user_id', 'grades_max', 'grades_first', 'grades_last']).corr()
corr_plt.style.background_gradient(cmap='coolwarm').format(precision=2)
```

	sex	height	weight	age	years_cl	grades_count	grades_mean	year_first	year_last
sex	1.00	-0.52	-0.54	-0.03	-0.07	-0.03	-0.12	0.01	-0.00
height	-0.52	1.00	0.75	0.10	0.03	0.01	-0.02	-0.00	-0.02
weight	-0.54	0.75	1.00	0.23	0.06	-0.00	-0.12	-0.01	-0.03
age	-0.03	0.10	0.23	1.00	0.53	0.10	-0.07	-0.07	-0.25
years_cl	-0.07	0.03	0.06	0.53	1.00	0.11	0.37	-0.11	-0.36
grades_count	-0.03	0.01	-0.00	0.10	0.11	1.00	0.14	-0.02	0.38
grades_mean	-0.12	-0.02	-0.12	-0.07	0.37	0.14	1.00	-0.04	0.10
year_first	0.01	-0.00	-0.01	-0.07	-0.11	-0.02	-0.04	1.00	0.09
year_last	-0.00	-0.02	-0.03	-0.25	-0.36	0.38	0.10	0.09	1.00

Preprocessing

Drop the rows with first year < 1950

```
df_climber.shape
```

```
(10927, 16)
```

```
df_climber = df_climber[df_climber.year_first >1950]
```

```
df_climber.shape
```

```
(10924, 16)
```

3 rows were dropped

```
df_climber.describe()
```

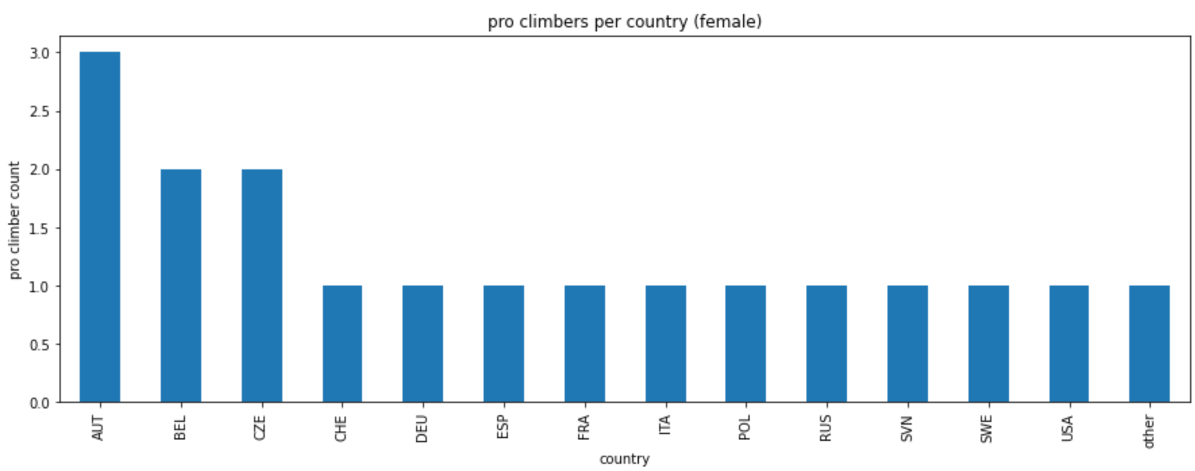
	user_id	sex	height	weight	age	years_cl	grades_count	grades_first	grades_last	grades_max	grades_mean	year_first	yea
count	10924.000000	10924.000000	10924.000000	10924.000000	10924.000000	10924.000000	10924.000000	10924.000000	10924.000000	10924.000000	10924.000000	10924.000000	109
mean	29414.960912	0.125137	176.152051	67.608111	33.331655	12.669810	79.796778	45.647382	46.981234	53.762450	45.503111	2009.073325	201
std	18022.383870	0.330890	8.509278	9.677087	7.590556	6.106819	141.423144	9.477138	9.417402	9.679826	7.891007	4.029715	4.1
min	1.000000	0.000000	137.000000	40.000000	12.000000	1.000000	1.000000	28.000000	28.000000	29.000000	28.500000	1991.000000	199
25%	14656.500000	0.000000	171.000000	63.000000	28.000000	8.000000	8.000000	38.000000	40.000000	46.000000	39.400000	2006.000000	200
50%	27323.500000	0.000000	177.000000	68.000000	33.000000	12.000000	28.000000	46.000000	48.000000	55.000000	45.147214	2009.000000	201
75%	43241.500000	0.000000	182.000000	73.000000	38.000000	17.000000	90.000000	53.000000	53.000000	62.000000	51.207983	2012.000000	201
max	67020.000000	1.000000	202.000000	93.000000	69.000000	29.000000	2445.000000	75.000000	77.000000	77.000000	75.272727	2017.000000	201

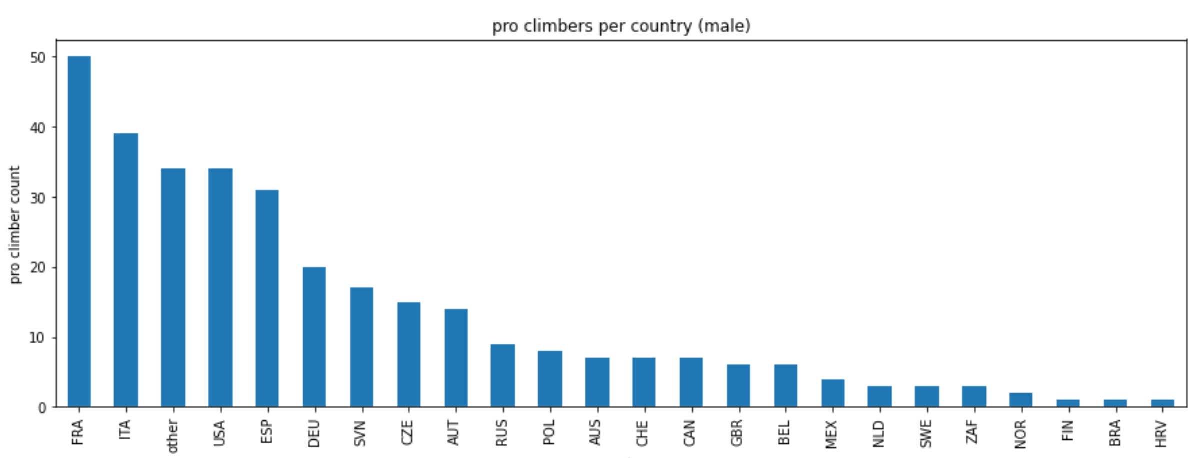
For the classification it would not make sense to use every single possible grade as a class. This would be 85 classes from 0 to 85. This is why we decided to discretise to three distinct classes. We simply split it into beginner=0, intermediate=1, expert=2. We used our "expert knowledge" to find the following borders of these three classes:
Until exclusive 6c -> beginner...45=6c
6c to exclusive 8a -> enthusiast...61=8a
Upwards of 8a -> pro...>62

```
# df_grades
beginner_upperbound = 45
intermediate_upperbound = 61
df_climber["grades_mean_discrete"] = 0
df_climber.loc[df_climber["grades_mean"]<beginner_upperbound, ["grades_mean_discrete"]] = 0
df_climber.loc[(df_climber["grades_mean"]>=beginner_upperbound) & (df_climber["grades_mean"]<intermediate_upperbound), ["grades_mean_d
df_climber.loc[(df_climber["grades_mean"]>=intermediate_upperbound), ["grades_mean_discrete"]] = 2
#df_climber.describe()
df_climber.head()
```

	user_id	country	sex	height	weight	age	years_cl	date_first	date_last	grades_count	grades_first	grades_last	grades_max	grades_mean	year_first	year_last	grades_mean_discrete
0	1	SWE	0	177	73	41.0	21	1999-02-06 23:00:00	2001-07-31 22:00:00	84	36	55	62	46.750000	1999	2001	1
1	3	SWE	0	180	78	44.0	22	1999-03-31 22:00:00	2000-07-19 22:00:00	12	53	51	59	52.833333	1999	2000	1
2	4	SWE	1	165	58	33.0	16	2004-06-30 22:00:00	2009-05-26 22:00:00	119	53	49	64	53.890756	2004	2009	1
3	10	SWE	0	167	63	52.0	25	2000-01-14 23:00:00	2017-06-01 22:00:00	298	53	49	63	49.406040	2000	2017	1
4	16	NOR	0	177	68	44.0	21	1998-02-27 23:00:00	2010-05-13 22:00:00	5	53	49	53	51.400000	1998	2010	1

```
goodies = df_climber[df_climber.grades_mean_discrete == 2 ]
df_climber[df_climber.grades_mean_discrete == 2 ]
goodies[['country','grades_mean_discrete']][goodies.sex == 1].groupby('country').count().sort_values('grades_mean_discrete', ascending
goodies[['country','grades_mean_discrete']][goodies.sex == 0].groupby('country').count().sort_values('grades_mean_discrete', ascending
```






```
(<AxesSubplot:title={'center':'pro climbers per country (male)'}, xlabel='country', ylabel='pro climber count'>,
 None)
```

```
le = LabelEncoder()
le.fit(df_climber['country'])
df_climber['countryenc'] = le.transform(df_climber['country'])
```

```
df_climber_f = df_climber[df_climber.sex == 1]
df_climber_m = df_climber[df_climber.sex == 0]

df_climber_f.info(), df_climber_m.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1367 entries, 2 to 10915
Data columns (total 18 columns):
 #   Column                Non-Null Count  Dtype  
---  ------                --------------  -----  
 0   user_id               1367 non-null   int64  
 1   country               1367 non-null   object 
 2   sex                   1367 non-null   int64  
 3   height                1367 non-null   int64  
 4   weight                1367 non-null   int64  
 5   age                   1367 non-null   float64
 6   years_cl              1367 non-null   int64  
 7   date_first            1367 non-null   object 
 8   date_last             1367 non-null   object 
 9   grades_count          1367 non-null   int64  
 10  grades_first          1367 non-null   int64  
 11  grades_last           1367 non-null   int64  
 12  grades_max            1367 non-null   int64  
 13  grades_mean           1367 non-null   float64
 14  year_first            1367 non-null   int64  
 15  year_last             1367 non-null   int64  
 16  grades_mean_discrete  1367 non-null   int64  
 17  countryenc            1367 non-null   int64  
dtypes: float64(2), int64(13), object(3)
memory usage: 202.9+ KB
<class 'pandas.core.frame.DataFrame'>
Int64Index: 9557 entries, 0 to 10926
Data columns (total 18 columns):
 #   Column                Non-Null Count  Dtype  
---  ------                --------------  -----  
 0   user_id               9557 non-null   int64  
 1   country               9557 non-null   object 
 2   sex                   9557 non-null   int64  
 3   height                9557 non-null   int64  
 4   weight                9557 non-null   int64  
 5   age                   9557 non-null   float64
 6   years_cl              9557 non-null   int64  
 7   date_first            9557 non-null   object 
 8   date_last             9557 non-null   object 
 9   grades_count          9557 non-null   int64  
 10  grades_first          9557 non-null   int64  
 11  grades_last           9557 non-null   int64  
 12  grades_max            9557 non-null   int64  
 13  grades_mean           9557 non-null   float64
 14  year_first            9557 non-null   int64  
 15  year_last             9557 non-null   int64  
 16  grades_mean_discrete  9557 non-null   int64  
 17  countryenc            9557 non-null   int64  
dtypes: float64(2), int64(13), object(3)
memory usage: 1.4+ MB


(None, None)
```

Data Modeling

Splitting for Regression Tasks:

```
x_column_names = ['countryenc', 'sex', 'height', 'weight', 'age', 'years_cl', 'grades_count', 'year_first', 'year_last']
X = df_climber[x_column_names]
y = df_climber.grades_mean
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=1337)

x_column_names = ['countryenc', 'height', 'weight', 'age', 'years_cl', 'grades_count', 'year_first', 'year_last']
X_m = df_climber_m[x_column_names]
y_m = df_climber_m.grades_mean
X_train_m, X_test_m, y_train_m, y_test_m = train_test_split(X_m, y_m, test_size=0.3, random_state=1337)

X_f = df_climber_f[x_column_names]
y_f = df_climber_f.grades_mean
X_train_f, X_test_f, y_train_f, y_test_f = train_test_split(X_f, y_f, test_size=0.3, random_state=1337)

scaler = preprocessing.StandardScaler().fit(X_train)
X_train_scaled = scaler.transform(X_train)
X_test_scaled = scaler.transform(X_test)

scaler = preprocessing.StandardScaler().fit(X_train_m)
X_train_m_scaled = scaler.transform(X_train_m)
X_test_m_scaled = scaler.transform(X_test_m)

scaler = preprocessing.StandardScaler().fit(X_train_f)
X_train_f_scaled = scaler.transform(X_train_f)
X_test_f_scaled = scaler.transform(X_test_f)
```

Regression

```
linreg = LinearRegression()
linreg.fit(X_train_scaled, y_train)

preds_linreg = linreg.predict(X_test_scaled)
print("Coefficients: \n")
for n, c in zip(linreg.coef_,['countryenc', 'sex\t', 'height\t', 'weight\t', 'age\t', 'years_cl', 'grades_count', 'year_first', 'year_
    print(c + ':\t' + str(n))
print("Mean squared error: %.2f" % mean_squared_error(y_test, preds_linreg))
```

```
Coefficients: 

countryenc:     -0.41222904816770223
sex     :       -1.334703783256014
height  :       0.5881433243934915
weight  :       -1.8561157719893229
age     :       -2.258266625676187
years_cl:       4.4833931642609866
grades_count:   -0.0999660765729169
year_first:     -0.5528963128252122
year_last:      2.1599360335226203
Mean squared error: 41.01
```

```
linreg_m = LinearRegression()
linreg_m.fit(X_train_m_scaled, y_train_m)

preds_linreg_m = linreg_m.predict(X_test_m_scaled)
print("Coefficients: \n")
for n, c in zip(linreg_m.coef_,['countryenc', 'height\t', 'weight\t', 'age\t', 'years_cl', 'grades_count', 'year_first', 'year_last'])
    print(c + ':\t' + str(n))
print("Mean squared error: %.2f" % mean_squared_error(y_test_m, preds_linreg_m))
```

```
Coefficients: 

countryenc:     -0.49550519720654346
height  :       0.4034575661201658
weight  :       -1.4163267439159546
age     :       -2.4639917777643245
years_cl:       4.64036468639629
grades_count:   -0.12000934776398942
year_first:     -0.5136493868778782
year_last:      2.012828389731777
```

```
Mean squared error: 41.09
```

```
linreg_f = LinearRegression()
linreg_f.fit(X_train_f_scaled, y_train_f)

preds_linreg_f = linreg_f.predict(X_test_f_scaled)
print("Coefficients: \n")
for n, c in zip(linreg_f.coef_,['countryenc', 'height\t', 'weight\t', 'age\t', 'years_cl', 'grades_count', 'year_first', 'year_last'])
    print(c + ':\t' + str(n))
print("Mean squared error: %.2f" % mean_squared_error(y_test_f, preds_linreg_f))
```

```
Coefficients:

countryenc:     -0.17508797926311492
height  :       0.7641456784649397
weight  :       -1.5702487419092612
age     :       -1.8581920852298326
years_cl:       5.066266017955897
grades_count:   0.16657976539620492
year_first:     -0.3051140977746958
year_last:      2.9760350856012208
Mean squared error: 38.15
```

Interpreting the results:

Seeing that for both, males and females, have the highest coefficient for years climbed we can draw the obvious conlusion that climbing for more years improves the performance. This is quite obvious, but what more can we see that helps us understand the data? We can see that the second most important score seems to be for females year last. This indicates that female climbers got better in recent years.

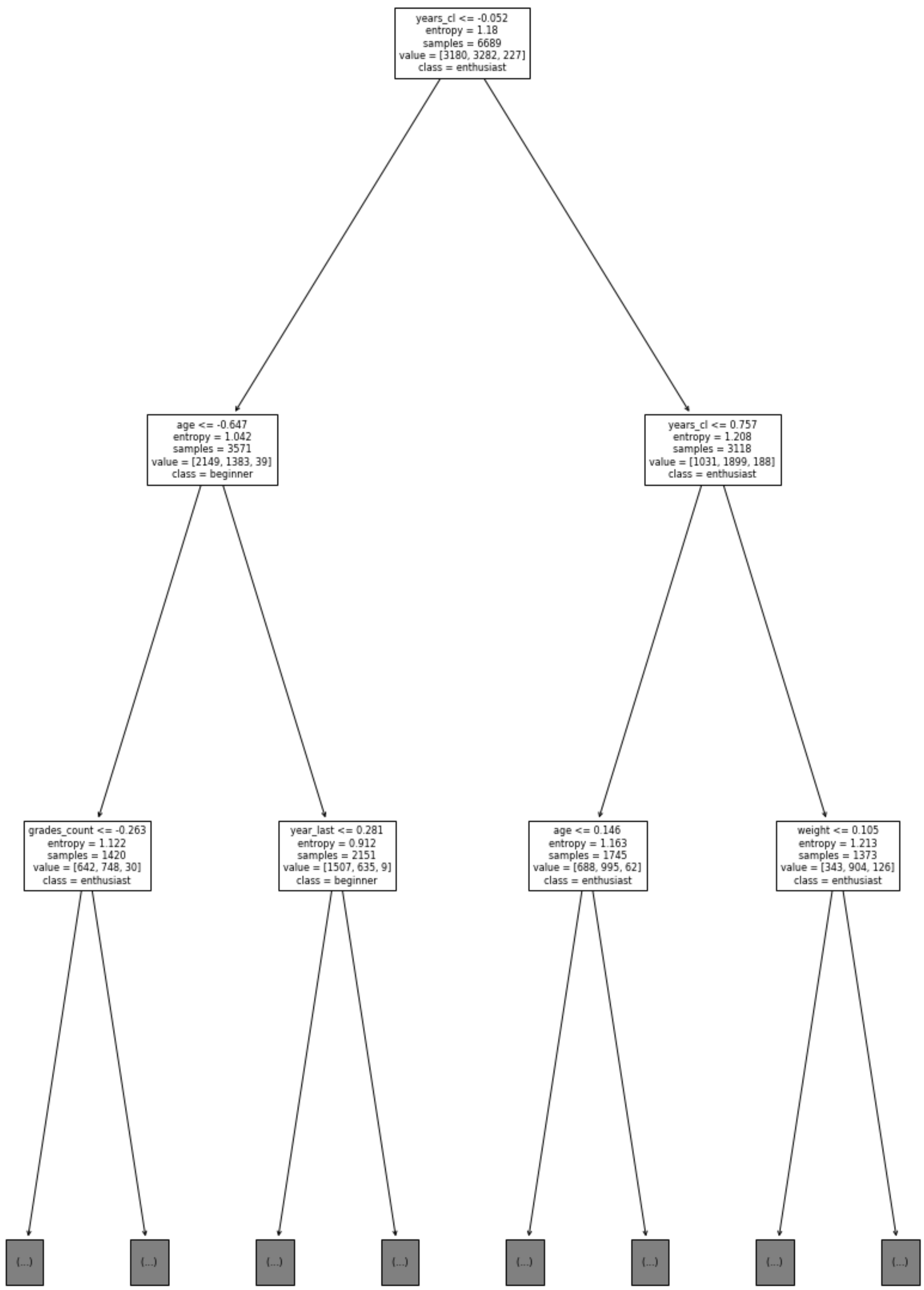
Tree

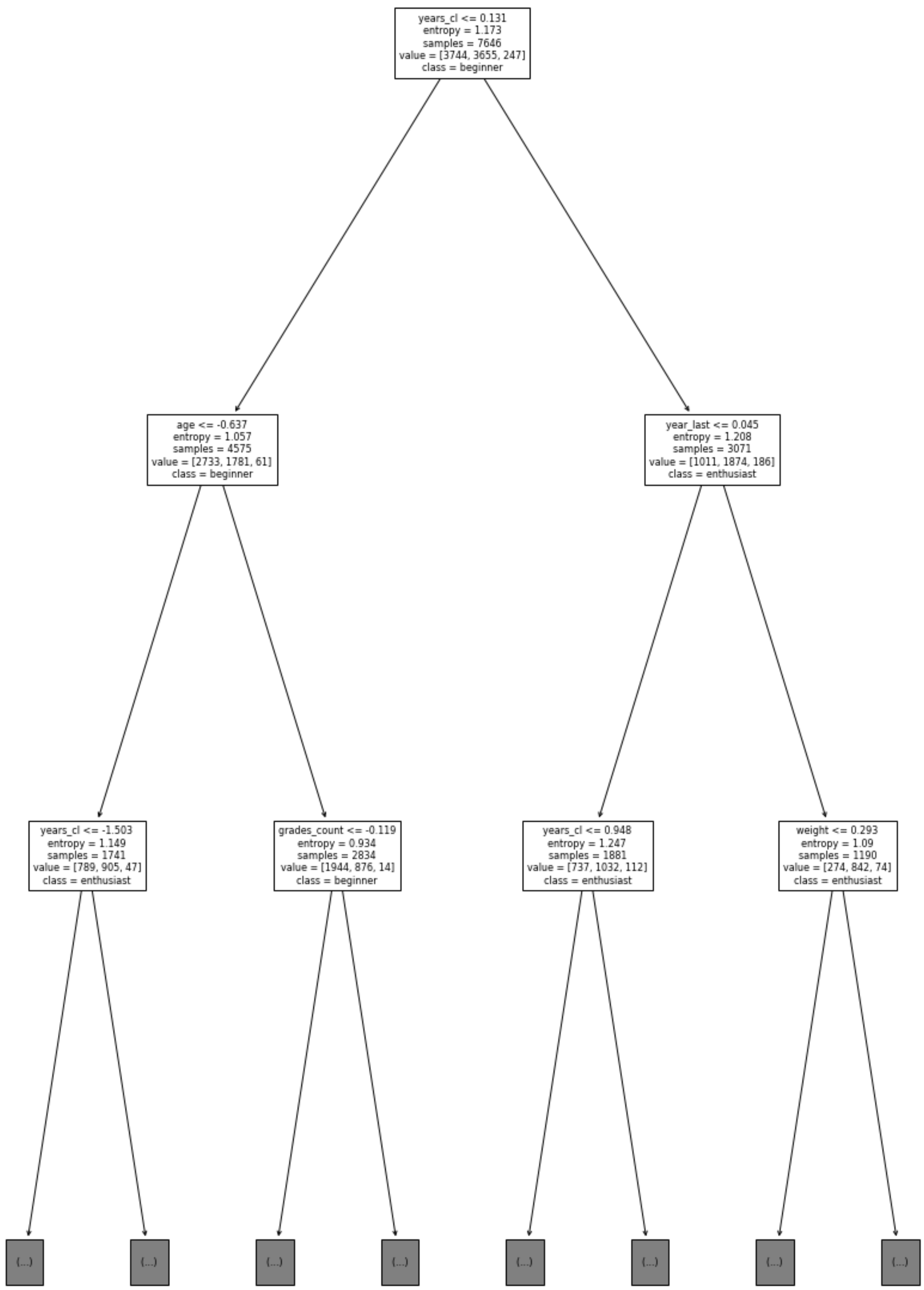
```
x_column_names_c = ['countryenc', 'height', 'weight', 'age', 'years_cl', 'grades_count', 'year_first', 'year_last']

X_m_c = df_climber_m[x_column_names_c]
y_m_c = df_climber_m.grades_mean_discrete
X_train_m_c, X_test_m_c, y_train_m_c, y_test_m_c = train_test_split(X_m_c, y_m_c, test_size=0.3, random_state=random_state)

X_f_c = df_climber[x_column_names_c]
y_f_c = df_climber.grades_mean_discrete
X_train_f_c, X_test_f_c, y_train_f_c, y_test_f_c = train_test_split(X_f_c, y_f_c, test_size=0.3, random_state=random_state)


scaler = preprocessing.StandardScaler().fit(X_train_m_c)
X_train_m_scaled_c = scaler.transform(X_train_m_c)
X_test_m_scaled_c = scaler.transform(X_test_m_c)

scaler = preprocessing.StandardScaler().fit(X_train_f_c)
X_train_f_scaled_c = scaler.transform(X_train_f_c)
X_test_f_scaled_c = scaler.transform(X_test_f_c)
```

```
# Fit a decision tree and plot the tree
print("To understand the tree: left is always True, right is always false... for e.g.  age<=49.5 all the samples that ARE smaller go l
# male tree
tree_m = tree.DecisionTreeClassifier(criterion="entropy", random_state=random_state)
tree_m = tree_m.fit(X_train_m_scaled_c,y_train_m_c)
plt.figure(figsize=(15,25))
tree.plot_tree(tree_m, max_depth=2, feature_names=x_column_names_c, fontsize=8, class_names=["beginner", "enthusiast", "pro"])

# female tree
tree_f = tree.DecisionTreeClassifier(criterion="entropy", random_state=random_state)
tree_f = tree_f.fit(X_train_f_scaled_c,y_train_f_c)
plt.figure(figsize=(15,25))
tree.plot_tree(tree_f, max_depth=2, feature_names=x_column_names_c, fontsize=8, class_names=["beginner", "enthusiast", "pro"])
```

```
To understand the tree: left is always True, right is always false... for e.g.  age<=49.5 all the samples that ARE smaller go left
```

```
[Text(418.5, 1189.125, 'years_cl <= 0.131\nentropy = 1.173\nsamples = 7646\nvalue = [3744, 3655, 247]\nclass = beginner'),
 Text(209.25, 849.375, 'age <= -0.637\nentropy = 1.057\nsamples = 4575\nvalue = [2733, 1781, 61]\nclass = beginner'),
 Text(104.625, 509.625, 'years_cl <= -1.503\nentropy = 1.149\nsamples = 1741\nvalue = [789, 905, 47]\nclass = enthusiast'),
 Text(52.3125, 169.875, '\n  (...)  \n'),
 Text(156.9375, 169.875, '\n  (...)  \n'),
 Text(313.875, 509.625, 'grades_count <= -0.119\nentropy = 0.934\nsamples = 2834\nvalue = [1944, 876, 14]\nclass = beginner'),
 Text(261.5625, 169.875, '\n  (...)  \n'),
 Text(366.1875, 169.875, '\n  (...)  \n'),
 Text(627.75, 849.375, 'year_last <= 0.045\nentropy = 1.208\nsamples = 3071\nvalue = [1011, 1874, 186]\nclass = enthusiast'),
 Text(523.125, 509.625, 'years_cl <= 0.948\nentropy = 1.247\nsamples = 1881\nvalue = [737, 1032, 112]\nclass = enthusiast'),
 Text(470.8125, 169.875, '\n  (...)  \n'),
 Text(575.4375, 169.875, '\n  (...)  \n'),
 Text(732.375, 509.625, 'weight <= 0.293\nentropy = 1.09\nsamples = 1190\nvalue = [274, 842, 74]\nclass = enthusiast'),
 Text(680.0625, 169.875, '\n  (...)  \n'),
 Text(784.6875, 169.875, '\n  (...)  \n')]
```

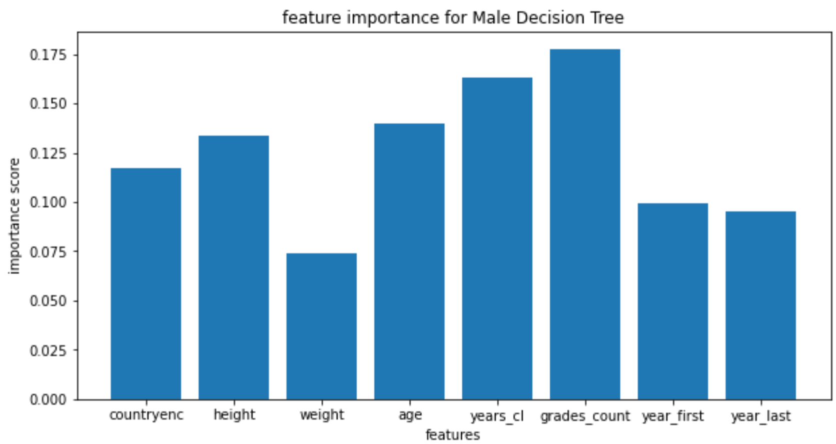

years_cl <= -0.052
entropy = 1.18
samples = 6689
value = [3180, 3282, 227]
class = enthusiast
age <= -0.647
entropy = 1.042
samples = 3571
value = [2149, 1383, 39]
class = beginner
years_cl <= 0.757
entropy = 1.208
samples = 3118
value = [1031, 1899, 188]
class = enthusiast
grades_count <= -0.263
entropy = 1.122
samples = 1420
value = [642, 748, 30]
class = enthusiast
year_last <= 0.281
entropy = 0.912
samples = 2151
value = [1507, 635, 9]
class = beginner
age <= 0.146
entropy = 1.163
samples = 1745
value = [688, 995, 62]
class = enthusiast
weight <= 0.105
entropy = 1.213
samples = 1373
value = [343, 904, 126]
class = enthusiast
(...)
(...)
(...)
(...)
(...)
(...)
(...)
(...)

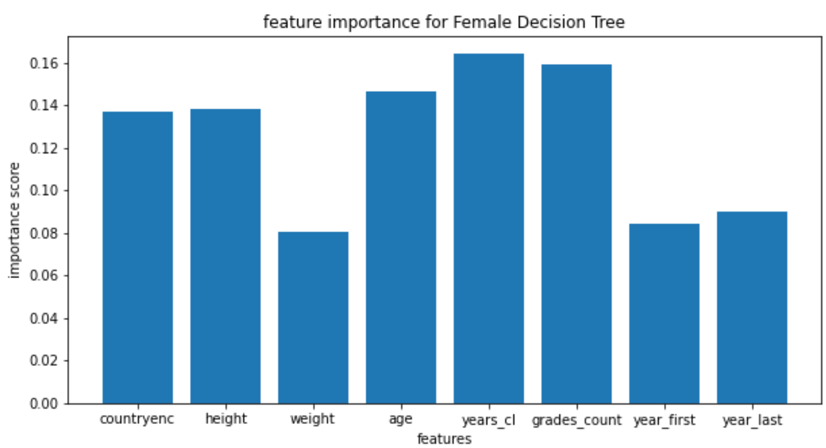

years_cl <= 0.131
entropy = 1.173
samples = 7646
value = [3744, 3655, 247]
class = beginner
age <= -0.637
entropy = 1.057
samples = 4575
value = [2733, 1781, 61]
class = beginner
year_last <= 0.045
entropy = 1.208
samples = 3071
value = [1011, 1874, 186]
class = enthusiast
years_cl <= -1.503
entropy = 1.149
samples = 1741
value = [789, 905, 47]
class = enthusiast
grades_count <= -0.119
entropy = 0.934
samples = 2834
value = [1944, 876, 14]
class = beginner
years_cl <= 0.948
entropy = 1.247
samples = 1881
value = [737, 1032, 112]
class = enthusiast
weight <= 0.293
entropy = 1.09
samples = 1190
value = [274, 842, 74]
class = enthusiast
(...)
(...)
(...)
(...)
(...)
(...)
(...)
(...)

```
# check accuracy
y_pred_m_c = tree_m.predict(X_test_m_scaled_c)
accuracy_m_c = accuracy_score(y_test_m_c.values, y_pred_m_c)

y_pred_f_c = tree_f.predict(X_test_f_scaled_c)
accuracy_f_c = accuracy_score(y_test_f_c, y_pred_f_c)

print(f"Accuracy for male tree: {accuracy_m_c}")
print(f"Accuracy for female tree: {accuracy_f_c}")

# feature importance
feature_importances_c = tree_m.feature_importances_
plt.figure(figsize=(10,5))
plt.bar([i for i in range(0, len(feature_importances_c))], feature_importances_c)
plt.xticks([i for i in range(0, len(x_column_names_c))], x_column_names_c)
plt.title(f'feature importance for Male Decision Tree')
plt.xlabel('features')
plt.ylabel('importance score')

feature_importances_c = tree_f.feature_importances_
plt.figure(figsize=(10,5))
plt.bar([i for i in range(0, len(feature_importances_c))], feature_importances_c)
plt.xticks([i for i in range(0, len(x_column_names_c))], x_column_names_c)
plt.title(f'feature importance for Female Decision Tree')
plt.xlabel('features')
plt.ylabel('importance score')
```

```
Accuracy for male tree: 0.595536959553696
Accuracy for female tree: 0.5951799877974374
```

```
Text(0, 0.5, 'importance score')
```





Forest

```
randforest_m = RandomForestClassifier(random_state=random_state)
randforest_m = randforest_m.fit(X_train_m_scaled_c, y_train_m_c)

randforest_f = RandomForestClassifier(random_state=random_state)
randforest_f = randforest_f.fit(X_train_f_scaled_c, y_train_f_c)

y_pred_m_c = randforest_m.predict(X_test_m_scaled_c)
accuracy_m_c = accuracy_score(y_test_m_c.values, y_pred_m_c)

y_pred_f_c = randforest_f.predict(X_test_f_scaled_c)
accuracy_f_c = accuracy_score(y_test_f_c, y_pred_f_c)

print(f"Accuracy for male forest: {accuracy_m_c}")
print(f"Accuracy for female forest: {accuracy_f_c}")

feature_importances_c = randforest_m.feature_importances_
plt.figure(figsize=(10,5))
plt.bar([i for i in range(0, len(feature_importances_c))], feature_importances_c)
plt.xticks([i for i in range(0, len(x_column_names_c))], x_column_names_c)
plt.title(f'feature importance for Male Random Forest')
plt.xlabel('features')
plt.ylabel('importance score')

feature_importances_c = randforest_f.feature_importances_
plt.figure(figsize=(10,5))
plt.bar([i for i in range(0, len(feature_importances_c))], feature_importances_c)
plt.xticks([i for i in range(0, len(x_column_names_c))], x_column_names_c)
plt.title(f'feature importance for Female Random Forest')
plt.xlabel('features')
plt.ylabel('importance score')
```

```
Accuracy for male forest: 0.6722454672245467
Accuracy for female forest: 0.6851738865161684
```

```
Text(0, 0.5, 'importance score')
```