S08T01 Feature Engineering

In [323…
```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
pd.set_option('display.max_columns', 500)
import warnings
warnings.filterwarnings('ignore')

from IPython.display import display_html
pd.options.mode.chained_assignment = None  # default='warn'
pd.options.display.float_format = '{:,.2f}'.format
```

I wil continue with the dataset of the Olympic Games that I already used in previous exercises. This dataset contains information about athletes competing in the Olympic Games, both Winter and Summer Season from Athens 1896 to Rio 2016.

This dataset is taken from: https://www.kaggle.com/heesoo37/120-years-of-olympic-history-athletes-and-results

For this exercise I will select the group of athletes that compete in the Sport 'Athletics' and exclude all the rest. We will create a new dataframe for them:

In [324…
```python
# open dataset
raw_data = pd.read_csv('athlete_events.csv')
```

In [325…
```python
raw_data.head()
```

Out[325…

	ID	Name	Sex	Age	Height	Weight	Team	NOC	Games	Year	Season	City	Sport	Event	Medal
0	1	A Dijiang	M	24.00	180.00	80.00	China	CHN	1992 Summer	1992	Summer	Barcelona	Basketball	Basketball Men's Basketball	NaN
1	2	A Lamusi	M	23.00	170.00	60.00	China	CHN	2012 Summer	2012	Summer	London	Judo	Judo Men's Extra-Lightweight	NaN
2	3	Gunnar Nielsen Aaby	M	24.00	NaN	NaN	Denmark	DEN	1920 Summer	1920	Summer	Antwerpen	Football	Football Men's Football	NaN
3	4	Edgar Lindenau Aabye	M	34.00	NaN	NaN	Denmark/Sweden	DEN	1900 Summer	1900	Summer	Paris	Tug-Of-War	Tug-Of-War Men's Tug-Of-War	Gold
4	5	Christine Jacoba Aaftink	F	21.00	185.00	82.00	Netherlands	NED	1988 Winter	1988	Winter	Calgary	Speed Skating	Speed Skating Women's 500 metres	NaN

In [326…
```python
# create new dataframe for Athletics athletes
athletics_data = raw_data[raw_data['Sport'] == 'Athletics']
```

In [327…
```python
# take a look at the data
athletics_data.sample(3)
```

Out[327…

	ID	Name	Sex	Age	Height	Weight	Team	NOC	Games	Year	Season	City	Sport	Event	Medal
31860	16367	Michael Anthony "Mike" Bull	M	25.00	184.00	80.00	Great Britain	GBR	1972 Summer	1972	Summer	Munich	Athletics	Athletics Men's Pole Vault	NaN
262033	131102	Hansjrg Wirz	M	29.00	177.00	70.00	Switzerland	SUI	1972 Summer	1972	Summer	Munich	Athletics	Athletics Men's 400 metres Hurdles	NaN
250304	125365	Mara del Monte Vasc Pes Gallardo	F	32.00	157.00	45.00	Spain	ESP	2008 Summer	2008	Summer	Beijing	Athletics	Athletics Women's 20 kilometres Walk	NaN

In [328…
```python
print('This dataset has', athletics_data.shape[0], 'rows and', athletics_data.shape[1], 'columns. ')
```

```
This dataset has 38624 rows and 15 columns.
```

Let´s look at the columns:

- ID : Unique identification number for each athlete
- Name: Name of athlete
- Sex: M (male) of F (female)
- Age: in years
- Height : in cm.
- Weight: in kg
- Team: country they represent
- NOC : National Olympic Comitee of the Team, 3 letters
- Games : Year and Season
- Year: when games took place
- Season: Winter or Summer Games
- City: city that hosts the Games
- Sport : category of competition
- Event : Subcategory of conmpetition
- Medal: Nan (no medal), Gold, Silver and Bronze

We´ll drop some columns that we are not going to need for this exercise:

- NOC : same info in column Team
- Games and city : we´ll only keep 'Year' column.
- Season : all rows are Summer
- Sport: all rows are Athletics

In [329…
```python
athletics_data.drop(['NOC', 'Games', 'Season', 'City', 'Sport'], axis= 1, inplace = True)
```

In [330…
```python
athletics_data.sample(3)
```

Out[330…

	ID	Name	Sex	Age	Height	Weight	Team	Year	Event	Medal
57817	29604	Vtor Hugo Silva Mouro dos Santos	M	20.00	185.00	74.00	Brazil	2016	Athletics Men's 100 metres	NaN
188073	94477	Hector "Hec" Phillips	M	30.00	175.00	NaN	Canada	1920	Athletics Men's 800 metres	NaN
200643	100770	Carolina Rieuwpassa	F	27.00	159.00	48.00	Indonesia	1976	Athletics Women's 200 metres	NaN

Even though we have selected 'Athletics' as type of Sport, we have a lot of subcategories for these athletes, let´s take a look:

In [331…
```python
athletics_data.Event.unique()
```

Out[331…
```
array(["Athletics Women's 100 metres",
       "Athletics Women's 4 x 100 metres Relay",
       "Athletics Men's Shot Put", "Athletics Men's Pole Vault",
       "Athletics Men's High Jump", "Athletics Men's 1,500 metres",
       "Athletics Men's 4 x 100 metres Relay",
       "Athletics Men's Long Jump", "Athletics Women's Javelin Throw",
       "Athletics Men's 5,000 metres", "Athletics Women's Shot Put",
       "Athletics Men's 110 metres Hurdles", "Athletics Women's Marathon",
       "Athletics Men's 100 metres", "Athletics Men's 400 metres Hurdles",
       "Athletics Men's 400 metres", "Athletics Men's Hammer Throw",
       "Athletics Men's 800 metres", "Athletics Men's Marathon",
       "Athletics Men's 4 x 400 metres Relay",
       "Athletics Men's 10,000 metres", "Athletics Women's 3,000 metres",
       "Athletics Men's 200 metres", "Athletics Men's Javelin Throw",
       "Athletics Men's 3,000 metres Steeplechase",
       "Athletics Women's 200 metres", "Athletics Men's Triple Jump",
       "Athletics Women's Long Jump", "Athletics Women's 5,000 metres",
       "Athletics Men's Discus Throw", "Athletics Women's 10,000 metres",
       "Athletics Men's Decathlon", "Athletics Women's Discus Throw",
       "Athletics Women's 1,500 metres", "Athletics Women's Pole Vault",
       "Athletics Women's 4 x 400 metres Relay",
       "Athletics Women's 800 metres", "Athletics Women's 400 metres",
       "Athletics Men's Javelin Throw, Both Hands",
       "Athletics Women's 400 metres Hurdles",
       "Athletics Women's Pentathlon", "Athletics Women's High Jump",
       "Athletics Men's Standing High Jump",
       "Athletics Men's Standing Long Jump",
       "Athletics Men's 20 kilometres Walk",
       "Athletics Men's 50 kilometres Walk",
       "Athletics Women's 100 metres Hurdles",
       "Athletics Men's Discus Throw, Greek Style",
       "Athletics Women's Heptathlon", "Athletics Men's Stone Throw",
       "Athletics Men's Javelin Throw, Freestyle",
       "Athletics Men's Pentathlon (Ancient)",
       "Athletics Women's 3,000 metres Steeplechase",
       "Athletics Men's Shot Put, Both Hands",
       "Athletics Men's 10 kilometres Walk",
       "Athletics Women's 20 kilometres Walk",
       "Athletics Women's 80 metres Hurdles",
       "Athletics Women's Triple Jump",
       "Athletics Women's 10 kilometres Walk",
       "Athletics Men's Cross-Country, Individual",
       "Athletics Men's Cross-Country, Team",
       "Athletics Men's 3,000 metres, Team", "Athletics Men's Pentathlon",
       "Athletics Men's 3,000 metres Walk", "Athletics Men's 5 mile",
       "Athletics Women's Hammer Throw",
       "Athletics Men's 3,200 metres Steeplechase",
       "Athletics Men's Standing Triple Jump",
       "Athletics Men's 4,000 metres Steeplechase",
       "Athletics Men's 5,000 metres, Team",
       "Athletics Men's 1,600 metres Medley Relay",
       "Athletics Men's 60 metres", "Athletics Men's 1,500 metres Walk",
       "Athletics Men's 3 mile, Team",
       "Athletics Men's 3,500 metres Walk",
       "Athletics Men's 10 mile Walk",
       "Athletics Men's Discus Throw, Both Hands",
       "Athletics Men's 200 metres Hurdles",
       "Athletics Men's 56-pound Weight Throw",
       "Athletics Men's 2,500 metres Steeplechase",
       "Athletics Men's All-Around Championship",
       "Athletics Men's 2,590 metres Steeplechase",
       "Athletics Men's 4 mile, Team"], dtype=object)
```

This seems like a wide range of different categories and possibly means the attributes in the dataset will be very diverse too.

In the dataset there are repeated athletes but we won't delete them because the reason they are repeated is not by mistake, they may play in different Events of Sports at the same Games or even in different Games so we´ll keep them.

In [332…
```python
duplicate_rows = len(athletics_data[athletics_data.duplicated(['ID'])])
print('We have a total number of', duplicate_rows, 'athletes that appear more than once in the dataset.')
```

```
We have a total number of 16553 athletes that appear more than once in the dataset.
```

Let´s check info about our dataframe, data types and missing values:

In [333…
```python
athletics_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 38624 entries, 26 to 271103
Data columns (total 10 columns):
 #   Column  Non-Null Count  Dtype  
---  ------  --------------  -----  
 0   ID      38624 non-null  int64  
 1   Name    38624 non-null  object 
 2   Sex     38624 non-null  object 
 3   Age     37507 non-null  float64
 4   Height  32907 non-null  float64
 5   Weight  32601 non-null  float64
 6   Team    38624 non-null  object 
 7   Year    38624 non-null  int64  
 8   Event   38624 non-null  object 
 9   Medal   3969 non-null   object 
dtypes: float64(3), int64(2), object(5)
memory usage: 3.2+ MB
```

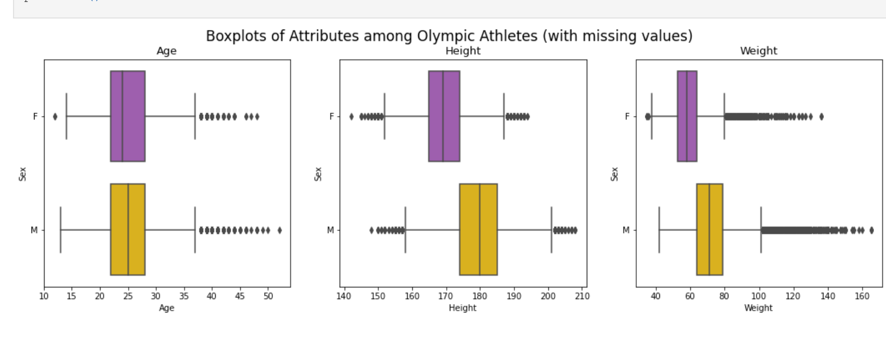
In [334…
```python
athletics_data.isnull().sum()
```

Out[334…
```
ID            0
Name          0
Sex           0
Age        1117
Height     5717
Weight     6023
Team          0
Year          0
Event         0
Medal     34655
dtype: int64
```

We have 34655 missing values in column Medal, in this case, Nan means no medal, later we will change the format of this column.

In this exercise, we´ll fill columns 'Age', 'Height' and 'Weight' that have null values. To do that, first we´ll draw boxplots to look at the information about the variability of the data. Since our dataset contains male and female athletes and their distributions may be different, we´ll do boxplots for each group:

In [335…
```python
fig, axes = plt.subplots(1,3, figsize= (18,5))
fig.suptitle('Boxplots of Attributes among Olympic Athletes (with missing values)', fontsize = 17)

sns.boxplot(ax= axes[0], data = athletics_data, x = 'Age', y = 'Sex', palette= ['#a652bb','#f8c300'])
axes[0].set_title('Age', fontsize = 13)

sns.boxplot(ax= axes[1], data = athletics_data, x = 'Height', y= 'Sex', palette= ['#a652bb','#f8c300'])
axes[1].set_title('Height', fontsize = 13)

sns.boxplot(ax= axes[2], data = athletics_data, x = 'Weight', y= 'Sex', palette= ['#a652bb','#f8c300'])
axes[2].set_title('Weight', fontsize = 13)

plt.show()
```



- We see we have some outliers in the data, specially in Age and Weight atributes.
- Age boxplot is similar for both male and female athletes, but is quite different in the case of Height and Weight: Male athletes have higher central values and more variability than female athletes. In these two cases we´ll split the dataframe before doing the imputation of missing values.

Let´s look at the descriptive statistics of our features before doing the imputation:

In [336…
```python
athletics_data[['Age', 'Height', 'Weight']].describe().round(2)
```

Out[336…

	Age	Height	Weight
count	37,507.00	32,907.00	32,601.00
mean	25.16	176.26	69.25
std	4.40	9.31	15.39
min	12.00	142.00	35.00
25%	22.00	170.00	59.00
50%	25.00	176.00	67.00
75%	28.00	183.00	76.00
max	52.00	208.00	165.00

In this summary we have a lot of info about our attributes: we can see mean, standard deviation, median(50% column), quantiles (25% and 75%) and even outliers (min and max values are very far from the mean).

To fill the missing values, we will use KNN imputer from sklearn because it maintains the value and variability of the dataset and it's more precise and efficient than using the average values.

In [337…
```python
# use Knn to impute
from sklearn.impute import KNNImputer
```

In [338…
```python
# make a copy of the dataset
athletics = athletics_data.copy()
# impute missing values for column 'Age'
knn_imputer = KNNImputer(n_neighbors= 7)

athletics[['Age']] = knn_imputer.fit_transform(athletics[['Age']] )
```

Now we´ll define separate dataframes for male and female athletes to impute 'Height' and 'Weight'.

In [339…
```python
athletics_male = athletics[athletics['Sex'] == 'M']
athletics_female = athletics[athletics['Sex'] == 'F']

print(athletics_male.shape)
print(athletics_female.shape)
```

```
(26958, 10)
(11666, 10)
```

We have 26958 observations for male athletes and 11666 for female athletes.

In [340…
```python
# imputing missing values:
athletics_male[['Height', 'Weight']] = knn_imputer.fit_transform(athletics_male[['Height', 'Weight']] )
# do the same fo female atghletes:
athletics_female[['Height', 'Weight']] = knn_imputer.fit_transform(athletics_female[['Height', 'Weight']])
```

In [341…
```python
# now let´s concatenate the two dataframes again:
athletics_df = pd.concat([athletics_male, athletics_female], ignore_index= True)
```

Now we don't any missing values in our numerical attributes. Let´s check descriptive statistics before and after imputation:

In [342…
```python
# create a funtion to display the two summaries side by side:

def side_by_side_df(df1, df2, title1, title2):

    df1_styler = df1.style.set_table_attributes("style='display:inline'").set_caption(title1)
    df2_styler = df2.style.set_table_attributes("style='display:inline'").set_caption(title2)
    space = "\xa0" * 10
    display_html(df1_styler._repr_html_()+ space  + df2_styler._repr_html_(), raw=True)
```

In [343…
```python
describe_before = athletics_data[['Age', 'Height', 'Weight']].describe().round(2)
describe_after = athletics_df[['Age', 'Height', 'Weight']].describe().round(2)

title1 = 'Summary statistics before imputation '
title2 = 'Summary statistics after imputation'
side_by_side_df(describe_before, describe_after ,title1, title2)
```

Summary statistics before imputation

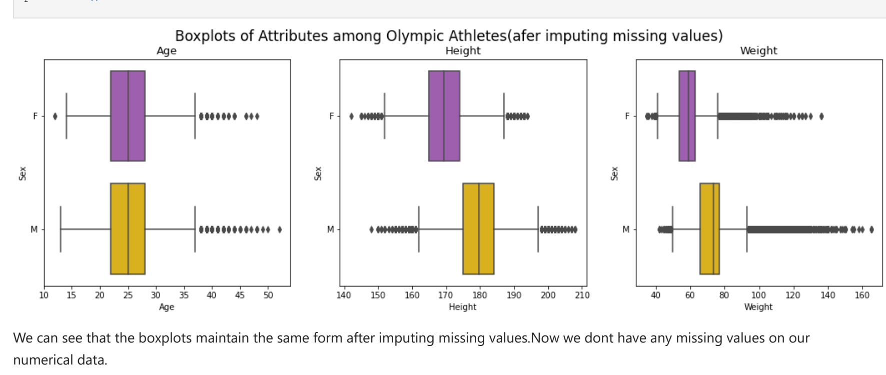
	Age	Height	Weight
count	37507.000000	32907.000000	32601.000000
mean	25.160000	176.260000	69.250000
std	4.400000	9.310000	15.390000
min	12.000000	142.000000	35.000000
25%	22.000000	170.000000	59.000000
50%	25.000000	176.000000	67.000000
75%	28.000000	183.000000	76.000000
max	52.000000	208.000000	165.000000

Summary statistics after imputation

	Age	Height	Weight
count	38624.000000	38624.000000	38624.000000
mean	25.160000	176.600000	69.710000
std	4.330000	8.730000	14.310000
min	12.000000	142.000000	35.000000
25%	22.000000	170.000000	60.000000
50%	25.000000	178.000000	69.000000
75%	28.000000	182.000000	74.000000
max	52.000000	208.000000	165.000000

Looking at the summary we see that KNN imputer has filled missing values without changing the central statistics of the data, let´s look again at the boxplots afer imputation to see if data variability has stayed the same too:

In [344…
```python
# same boxplot as before but after imputing missing values:

fig, axes = plt.subplots(1,3, figsize= (18,5))
fig.suptitle('Boxplots of Attributes among Olympic Athletes(afer imputing missing values)', fontsize = 17)

sns.boxplot(ax= axes[0], data = athletics_df, x = 'Age', y = 'Sex', palette= ['#a652bb','#f8c300'], order= ['F'
axes[0].set_title('Age', fontsize = 13)

sns.boxplot(ax= axes[1], data = athletics_df, x = 'Height', y= 'Sex', palette= ['#a652bb','#f8c300'], order= ['
axes[1].set_title('Height', fontsize = 13)

sns.boxplot(ax= axes[2], data = athletics_df, x = 'Weight', y = 'Sex', palette= ['#a652bb','#f8c300'], order= [
axes[2].set_title('Weight', fontsize = 13)

plt.show()
```



We can see that the boxplots maintain the same form after imputing missing values.Now we dont have any missing values on our numerical data.

- Exercici 1

Agafa un conjunt de dades de tema esportiu que t'agradi i normalitza els atributs categòrics en dummy. Estandaritza els atributs numèrics amb StandardScaler.

In [345…
```python
athletics_df.head(3)
```

Out[345…

	ID	Name	Sex	Age	Height	Weight	Team	Year	Event	Medal
0	18	Timo Antero Aaltonen	M	31.00	189.00	130.00	Finland	2000	Athletics Men's Shot Put	NaN
1	31	Evald rma (rman-)	M	24.00	174.00	70.00	Estonia	1936	Athletics Men's Pole Vault	NaN
2	32	Olav Augunson Aarnes	M	23.00	179.77	73.84	Norway	1912	Athletics Men's High Jump	NaN

- **Standardize numerical attributes with StandardScaler**

Standardization is a scaling technique where the values are centered around the mean with a unit standard deviation. This means that the mean of the attribute becomes zero and the resultant distribution has a unit standard deviation. This is useful because sometimes ML algorithms perform better with scaled attributes.

Besides, it makes it easier if we want to compare attributes that have different scale or want to interpret its values: just looking at them we know that positive values will be above the mean and negative values will be lower than the mean.

We´ll use this tranformation with our numerical columns for 'Age', 'Height' and 'Weight'.

In [346…
```python
# create a copy of the dataframe
athletics_scaled = athletics_df.copy()
```

In [347…
```python
from sklearn.preprocessing import StandardScaler
# choose the columns
col_names =['Age', 'Weight', 'Height']
features = athletics_scaled[col_names]
# apply scaler
scaler = StandardScaler().fit(features.values)
features = scaler.transform(features.values)
# assign new values
athletics_scaled[col_names] = features
athletics_scaled.head(3)
```

Out[347…

	ID	Name	Sex	Age	Height	Weight	Team	Year	Event	Medal
0	18	Timo Antero Aaltonen	M	1.35	1.42	4.21	Finland	2000	Athletics Men's Shot Put	NaN
1	31	Evald rma (rman-)	M	-0.27	-0.30	0.02	Estonia	1936	Athletics Men's Pole Vault	NaN
2	32	Olav Augunson Aarnes	M	-0.50	0.36	0.29	Norway	1912	Athletics Men's High Jump	NaN

Let´s check now for the mean and the standard deviation of the features:

In [348…
```python
athletics_scaled[['Age', 'Height', 'Weight']].mean().round(2)
```

Out[348…
```
Age      -0.00
Height    0.00
Weight    0.00
dtype: float64
```

In [349…
```python
athletics_scaled[['Age', 'Height', 'Weight']].std().round(2)
```

Out[349…
```
Age      1.00
Height   1.00
Weight   1.00
dtype: float64
```

Now the mean for this attributes is zero and the standard deviation is one. Standardization makes easier to interpret the data, any value above 0 is higher than the mean and lower than zero is smaller than the mean. If we use describe():

In [350…
```python
athletics_scaled[['Age', 'Height', 'Weight']].describe()
```

Out[350…

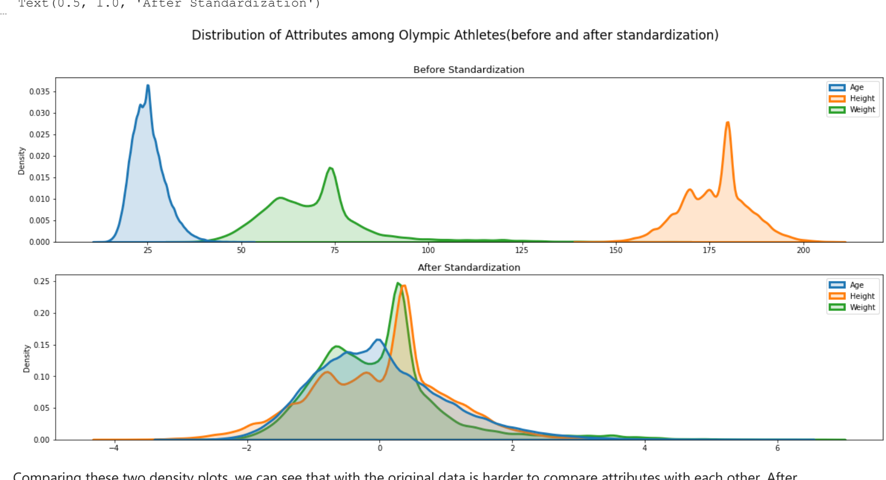
	Age	Height	Weight
count	38,624.00	38,624.00	38,624.00
mean	-0.00	0.00	0.00
std	1.00	1.00	1.00
min	-3.04	-3.96	-2.43
25%	-0.73	-0.76	-0.68
50%	-0.04	0.16	-0.05
75%	0.65	0.62	0.30
max	6.19	3.60	6.66

- For Age and Weight, the 50 % of values are below the mean.
- We can see that in Age and Weight the max values are much more far from the mean than in the case of Height.

In [351…
```python
# create a dataframe with only the three scaled columns:
standard_df = athletics_scaled[['Age', 'Height', 'Weight']]
```

In [352…
```python
# let´s plot this together:

fig, axes = plt.subplots(2,1, figsize= (20,9))
fig.suptitle('Distribution of Attributes among Olympic Athletes(before and after standardization)', fontsize =

sns.kdeplot(ax=axes[0], data= athletics_df[['Age', 'Height', 'Weight']] , linewidth = 3, fill= True, alpha = 0.
axes[0].set_title('Before Standardization', fontsize = 13)

sns.kdeplot(ax=axes[1], data= standard_df, linewidth = 3, fill= True, alpha = 0.2)
axes[1].set_title('After Standardization', fontsize = 13)
```

Out[352…
```
Text(0.5, 1.0, 'After Standardization')
```



Comparing these two density plots, we can see that with the original data is harder to compare attributes with each other. After standardization we have all attributes in the same ranges of data so it may be easier to reach conclusions about the distributions. We observe now that almost all observations fall in the range of 0 +/- 2, Age distribution seems closer to a normal distribution and in the cases of Weight and Height distributions it is more obvious that we have multimodal distributions that are similar among each other.

- **Categorical attributes**

We´ll map 'Sex' and 'Medal' column manually and later we will use get_dummies for the other categorical columns we are interested: Team and Event.

In [353…
```python
# use map for sex column:
athletics_scaled['Sex'] = athletics_scaled['Sex'].map({'M': 0, 'F': 1})
```

In [354…
```python
# use map for medal column: Nan: 0, Bronze: 1, Silver: 2, Gold: 3
athletics_scaled['Medal'] = athletics_scaled['Medal'].map({np.nan: 0, 'Bronze': 1, 'Silver': 2, 'Gold': 3})
```

In [355…
```python
# use get_dummies for Team and Event:
athletics_scaled = pd.get_dummies(data = athletics_scaled, columns= ['Event', 'Team'])
```

In [356…
```python
athletics_scaled.sample(3)
```

Out[356…

	ID	Name	Sex	Age	Height	Weight	Year	Medal	Event_Athletics Men's 1,500 metres	Event_Athletics Men's 1,500 metres Walk	Event_Athletics Men's 1,600 metres Medley Relay	Event_Athletics Men's 10 kilometres Walk	Event_At Men's 1
12573	61877	Paul Kipsilgich Koech	0	0.42	-0.76	-0.68	1996	0	0	0	0	0	0
24133	119949	Richard Thompson	0	-0.50	1.31	0.72	2008	2	0	0	0	0	0
6815	32033	Fouad El-Kaam	0	0.65	1.31	0.02	2016	0	1	0	0	0	0

Since get_dummies() in Pandas assigns a new column for each category of our categorical features(assigning 1 or 0 to each row if it belongs to that category), now our dataframe has a lot of new colums, let´s look at the shape:

In [357…
```python
print('The dataframe after using get_dummies() has', athletics_scaled.shape[0], ' rows and ',
      athletics_scaled.shape[1], 'columns.')
```

```
The dataframe after using get_dummies() has 38624  rows and  322 columns.
```

- Exercici 2

Continua amb el conjunt de dades de tema esportiu que t'agradi i aplica l'anàlisi de components principals.

PCA technique allows us to reduce the number of features preserving most of the variance among them, this way we can reduce the number of dimensions in our dataset. It' recommended to use for numerical and continuous data. In our case we only have three numerical columns for athletes attributes, which are Age, Weight and Height.

In this case we will also add 'Medal' column, which we have mapped the following way: .

- 0: no medal
- 1: Bronze medal
- 2: Silver medal
- 3: Gold medal

This way we will aplly the PCA analysis on these 4 variables we have selected (Age, Height, Weight and Medal):

In [358…
```python
# import library
from sklearn.decomposition import PCA

pca_features = ['Age', 'Height', 'Weight', 'Medal']
# separating the features  into an array
x = athletics_scaled.loc[:, pca_features].values

# apply PCA to our selected features
pca = PCA()
pca_components = pca.fit_transform(x)

# turn the arrays into a dataframe
pca_df = pd.DataFrame(data= pca_components, columns = ['PCA1', 'PCA2', 'PCA3', 'PCA4'])

# preview of our new dataframe
pca_df.head(3)
```

Out[358…

	PCA1	PCA2	PCA3	PCA4
0	4.06	1.16	-0.30	1.92
1	-0.22	-0.24	-0.20	0.24
2	0.42	-0.53	-0.23	-0.03

First, let´s look at how much variability is explained by each principal component:

In [359…
```python
pca.explained_variance_ratio_
```

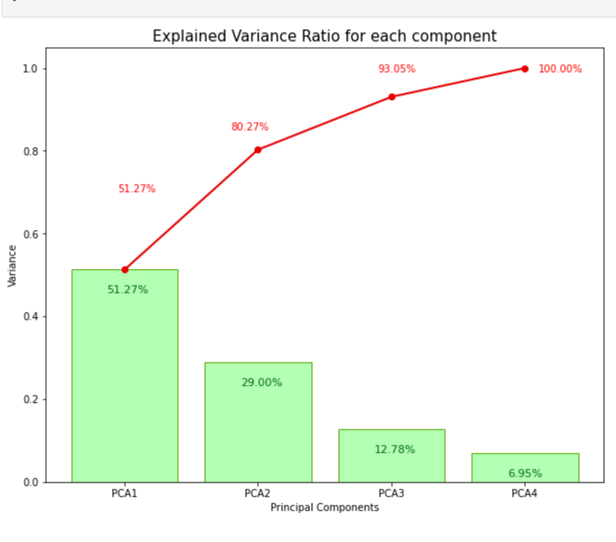
Out[359…
```
array([0.51267699, 0.29001159, 0.12779794, 0.06951347])
```

We got an array with 4 numbers, sklearn orders each component from more to less variability explained, so this means:

- PCA1: Component 1 explains 51,27 % of the variability of the data.
- PCA2: Component 2 contributes with 29,00 % of the variability of the dataset.
- PCA3: Component 3 provides 12,78 % of the variability.
- PCA4: Component 4 explains 6,95 % of the variability.

We can plot this to have a better view, we will plot the individual contribution to Variance of each component and also the cumulative Variance:

In [360…
```python
fig = plt.figure(figsize= (10,8))
PC_values = np.arange(pca.n_components_) + 1
plt.plot(PC_values, pca.explained_variance_ratio_.cumsum(), 'o-', linewidth = 2, color = '#e60000')
plt.text(0.95, 0.70, "%.2f%%" % (pca.explained_variance_ratio_.cumsum()[0]*100), color= 'red')
plt.text(1.80, 0.85, "%.2f%%" % (pca.explained_variance_ratio_.cumsum()[1]*100), color= 'red')
plt.text(2.90, 0.99, "%.2f%%" % (pca.explained_variance_ratio_.cumsum()[2]*100), color= 'red')
plt.text(4.10, 0.99, "%.2f%%" % (pca.explained_variance_ratio_.cumsum()[3]*100), color= 'red')

x1 = np.array([1,2,3, 4])
y = pca.explained_variance_ratio_
pl = plt.bar(x1, y, edgecolor = '#5da909', color= '#b3ffb3')
for bar in pl:
    plt.annotate("%.2f%%" % (bar.get_height()*100), xy= (bar.get_x()+ 0.27, bar.get_height() -0.057 ),
                 color = '#006622', fontsize= 11)

plt.title('Explained Variance Ratio for each component', fontsize= 15)
plt.xlabel('Principal Components')
labels = ['PCA1', 'PCA2', 'PCA3', 'PCA4']
plt.xticks(x1, labels)
plt.ylabel('Variance')
plt.show()
```



- In red we can see the cumulative variance of the Principal Components from 1 component to 4 components (4 components provide with 100 % of the Variance).
- In green is shown the individual contribution of each PCA to the Total Variance.

If we wanted to **reduce the dimensionality of data but at the same time we wanted to keep at least 90% of the explained Variance, looking at this graph we can observe that we could do that choosing the first 3 components**, which provide an Explained Variance of 93,05 %.

We can also reach the same conclusion if we use the PCA method:

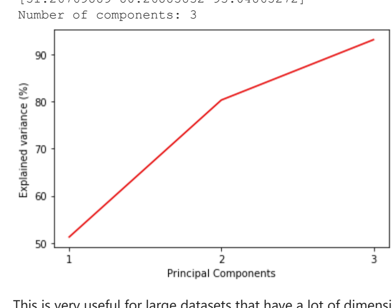
In [361…
```python
# we put inside PCA function the Explained Variance we want to achieve, in this case 90% :
pca = PCA(0.90).fit(x)
pca.n_components_
```

Out[361…
```
3
```

The result is 3, meaning that we need 3 Principal Components to achieve at least 90% of explained Variance. We can also plot this:

In [380…
```python
# make a plot of the components to achieve 90% of explained variance
pca = PCA(n_components = 0.90)
pca.fit(x)

print("Cumulative Variances (Percentage):")
print(np.cumsum(pca.explained_variance_ratio_ * 100))

components = len(pca.explained_variance_ratio_)
print(f'Number of components: {components}')

# Make the plot
plt.plot(range(1, components + 1), np.cumsum(pca.explained_variance_ratio_ * 100), color = '#e60000')
plt.xlabel("Number of components")
plt.xlabel('Principal Components')
plt.xticks([1,2,3])
plt.ylabel('Variance')
plt.ylabel("Explained variance (%)")
plt.show()
```

```
Cumulative Variances (Percentage):
[51.26769889 80.26885832 93.04865272]
Number of components: 3
```



This is very useful for large datasets that have a lot of dimensions, **in our example we would take our first 3 components to fit the model,** this way we could reduce dimensionality of data but at the same time we'd still keep 90% of the Variance that the original variables provide.

- Exercici 3

Continua amb el conjunt de dades de tema esportiu que t'agradi i normalitza les dades tenint en compte els outliers.

Machine learning algorithms tend to perform better or converge faster when the different features (variables) are on a smaller scale. Therefore it is common practice to normalize or standardize the data before training machine learning models on it.

Before we did standardization on numerical variables, now we are goig to do normalization. Normalizations means to scale the values to a minimum of 1 and a maximum of 0.

We will do this using sklearn MinMaxScaler, we´ll do this first with outliers and then we will do the same without outliers and see the differences. Lastly, we will use RobustScaler that is a technique that scales the features but is more robust to outliers, because it scales data not according to the median but using 1st quartile and 3rd quartile.

- #### Normalization including outliers

In [363…
```python
from sklearn.preprocessing import MinMaxScaler

# use the athletics_df that has the original values
athletics_df.sample(3)
```

Out[363…

	ID	Name	Sex	Age	Height	Weight	Team	Year	Event	Medal
16809	82857	Istvn (Stefan) Mudin	M	24.00	179.77	73.84	Hungary	1906	Athletics Men's Pentathlon (Ancient)	Silver
15150	75027	Bruno Marie-Rose	M	23.00	193.00	83.00	France	1988	Athletics Men's 200 metres	NaN
20420	101004	Jeffrey "Jeff" Riseley	M	29.00	192.00	74.00	Australia	2016	Athletics Men's 800 metres	NaN

In [364…
```python
# Select the numerical columns we want to normalize:
X = athletics_df[['Age', 'Height', 'Weight']]

mms = MinMaxScaler()
# apply transformation on selected columns:
Xnorm = mms.fit_transform(X)
```

In [365…
```python
# look for the min and max ranges for the normalized matrix:
print(Xnorm.min(axis=0))
print(Xnorm.max(axis=0))
```

```
[0. 0. 0.]
[1. 1. 1.]
```

In [366…
```python
print(Xnorm.std(axis=0))
```

```
[0.10835344 0.13323521 0.11008015]
```

Since all three columns are in the same scale now, we can compare the values between them. Since standard deviation is similar in the three columns, we can say that the data is similarly dispersed for the three variables.

We will convert this again to a dataframe:

In [367…
```python
norm_df = pd.DataFrame(Xnorm, columns = ['Age', 'Height', 'Weight'])
```

- #### Normalization without outliers

Now we are going to do the same but first we will remove outliers, we´ll use the IQR approach (IQR for Inter Quartile Range).

IQR = Quartile3 - Quartile1

To define an outlier we must set an Upper and Lower bound, using IQR we´ll define this bounds and the remove all the values that don't fit this:

- upper bound = Quartile3 + 1.5 x IQR
- lower bound = Quartile1 - 1.15 x IQR

In [368...

```
# we´ll define the columns
cols = ['Age', 'Height', 'Weight']

# define quantile 1
Q1 = athletics_df[cols].quantile(0.25) # 25% of the observations in this quantile
Q3 = athletics_df[cols].quantile(0.75) # 75% of the observations in this quantile
# define inter quartile range
IQR = Q3 - Q1
# new dataframe with onbservations from outliers removed(removes whole row if finds any outlier)
athletics_df_no_outliers = athletics_df[~((athletics_df[cols] < (Q1 - 1.5 * IQR))
                                         |(athletics_df[cols] > (Q3 + 1.5 * IQR))).any(axis=1)]
```

In [369...

```
print('We have removed ',len(athletics_df) - len(athletics_df_no_outliers), ' observations from our dataset.')
print('After removing outliers we have ', len(athletics_df_no_outliers), ' rows in the dataframe.')
```

```
We have removed  2385  observations from our dataset.
After removing outliers we have  36239  rows in the dataframe.
```

Before applying normalization let´s check describe method:

In [370...

```
athletics_df_no_outliers[['Age', 'Height', 'Weight']].describe().round(2)
```

Out[370...

	Age	Height	Weight
count	36,239.00	36,239.00	36,239.00
mean	24.91	175.99	67.54
std	4.04	8.16	10.29
min	13.00	152.00	39.00
25%	22.00	170.00	60.00
50%	25.00	178.00	68.00
75%	27.00	180.00	73.84
max	37.00	200.00	95.00

If we compare this summary statistics with the ones before removing outliers we can say that most outliers were 'Greater' values, since removing them the means and specially the max values have dicreased (minimum values too but not so much).

If we look at standard deviation before removing outliers:

In [371...

```
athletics_df[['Age', 'Height', 'Weight']].std().round(2)
```

Out[371...

```
Age       4.33
Height    8.73
Weight   14.31
dtype: float64
```

Looking at Standard deviations we notice that they also have dicreased a little, specially in the case of 'Weight' attribute, which was the one with most outliers (we saw that when we draw the boxplots). This effect is caused because standard deviation is a measure of dispersion of the data, and removing outliers makes data less dispersed.

Now let´s normalize the data:

In [372...

```
X_without_outliers = athletics_df_no_outliers[['Age', 'Height', 'Weight']]
mms = MinMaxScaler()

Xnorm_without_outliers = mms.fit_transform(X_without_outliers)
```

In [373...

```
Xnorm_without_outliers[:3]
```

Out[373...

```
array([[0.45833333, 0.45833333, 0.55357143],
       [0.41666667, 0.57850631, 0.6221273 ],
       [0.70833333, 0.72916667, 0.66071429]])
```

In [374...

```
# look for the min and max ranges for the normalized matrix:
print(Xnorm_without_outliers.min(axis=0))
print(Xnorm_without_outliers.max(axis=0))
```

```
[0. 0. 0.]
[1. 1. 1.]
```

As before, we see that now all of our values are in the scale between 0 and 1. Now we will convert the array to a dataframe:

In [375...

```
norm_df_no_outliers = pd.DataFrame(Xnorm_without_outliers,columns = ['Age', 'Height', 'Weight'] )
```

- **Scale using RobustScaler**

Another approach to scale variables in the presence of outliers is 'robust data scaling', that ignores the outliers from the calculation of the mean and standard deviation, then uses the calculated values to scale the variable. The result then is not skewed by the outliers.

In [376...

```
from sklearn import preprocessing

robust_scaler = preprocessing.RobustScaler()
robust_df = robust_scaler.fit_transform(athletics_df[['Age', 'Height', 'Weight']])
print('Number of observations in new dataframe is',  len(robust_df), ', same as the original df:',len(athletics
```

```
Number of observations in new dataframe is 38624 , same as the original df: 38624
```

We see that RobustScaler has not removed any values, the way it works is that it preserves outliers but tries to not let them influence the scaling of the non outliers values.

In [377...

```
robust_df = pd.DataFrame(robust_df, columns =['Age', 'Height', 'Weight'])
robust_df.describe().round(2)
```

Out[377...

	Age	Height	Weight
count	38,624.00	38,624.00	38,624.00
mean	0.03	-0.12	0.05
std	0.72	0.73	1.02
min	-2.17	-3.00	-2.43
25%	-0.50	-0.67	-0.64
50%	0.00	0.00	0.00
75%	0.50	0.33	0.36
max	4.50	2.50	6.86

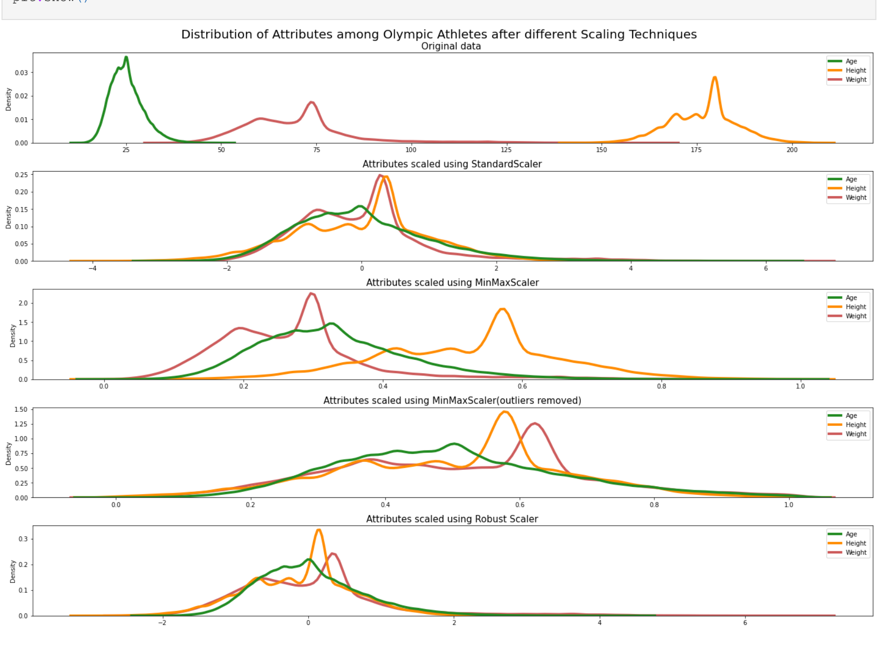
We observe that RobustScaler works similar to standarization, the mean is centered around 0 (in the parameters, with_centering: boolean: It is True by default), but in the case of Standard Deviation, it´s not equal to 1 in all cases (only in the case of weight). It may be because making the scale more robust to outliers may have lowered std.

Let´s plot this together and see if we coud reach any conclusions. We have scaled the data in four different ways:

- standard_df : we have used StandardScaler
- norm_df : we have used MinMaxScaler
- norm_df_no_outliers : firts we have removed outliers and then used MinMaxScaler
- robust_df : we have used RobustScaler

In [378...

```
# ploting the distributions in each case
fig, axes = plt.subplots(5,1, figsize= (20,14))
fig.suptitle('Distribution of Attributes among Olympic Athletes after different Scaling Techniques', fontsize =

colors = ['#228b22', '#ff8c00', '#cd5c5c']

sns.kdeplot(ax = axes[0], data = athletics_df[['Age', 'Height', 'Weight']], linewidth = 4, palette = colors)
axes[0].set_title('Original data ', fontsize = 15)

sns.kdeplot(ax = axes[1], data = standard_df, linewidth = 4, palette = colors)
axes[1].set_title('Attributes scaled using StandardScaler', fontsize = 15)

sns.kdeplot(ax = axes[2], data = norm_df, linewidth = 4,  palette = colors)
axes[2].set_title('Attributes scaled using MinMaxScaler', fontsize = 15)

sns.kdeplot(ax = axes[3], data = norm_df_no_outliers, linewidth = 4,  palette = colors)
axes[3].set_title('Attributes scaled using MinMaxScaler(outliers removed)', fontsize = 15)

sns.kdeplot(ax = axes[4], data = robust_df,  linewidth = 4,  palette = colors)
axes[4].set_title('Attributes scaled using Robust Scaler', fontsize = 15)

plt.tight_layout()
plt.show()
```

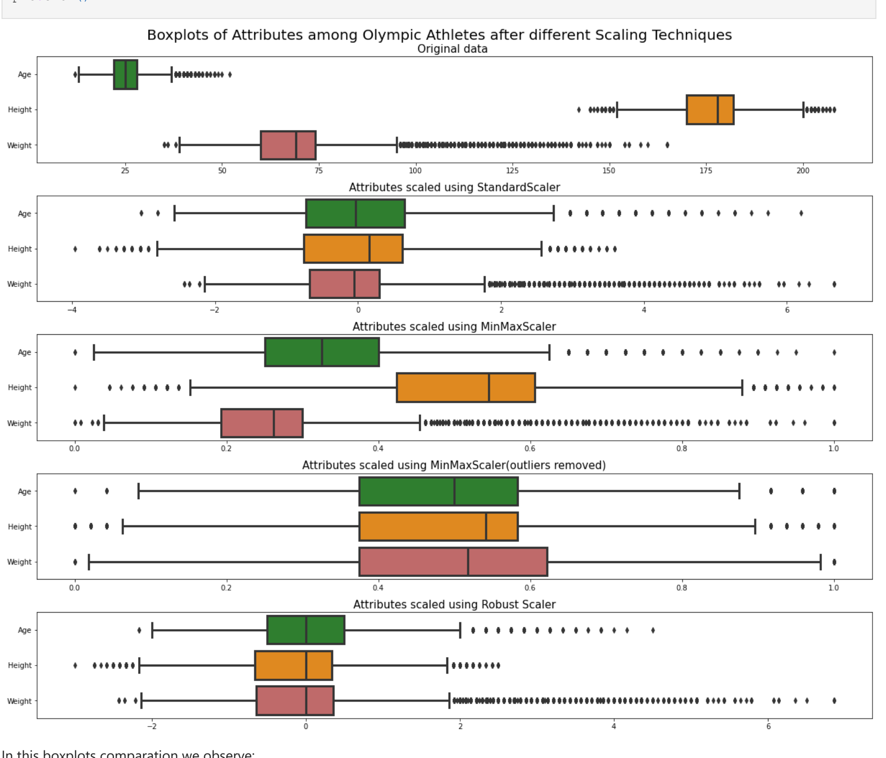


- In the original data, the distributions are far from each other because they are measured in different scales.
- Using StandarScaler Height and Weight look more similar to each other and Age resembles more like a normal distribution.
- MinMaxAScaler inluding outliers makes the three distributions look different from each other, Height has the data more dispersed and Weight is the attribute that has more outliers and is clearly right-skewed.
- MinMaxScaler with ouliers removed: now Height and Weight look more similar, the removing of the outliers has made the Weight distribution shift to the right. Age distribution is the one that looks closer to normal and is the most centered one.
- RobustScaler makes the data much more concentrated around the center than the other techniques, the three atributes look closer to each other but the outliers are still there.

Now let´s do the boxplots to see if we see anything else:

In [379...

```
fig, axes = plt.subplots(5,1, figsize= (17,14))
fig.suptitle('Boxplots of Attributes among Olympic Athletes after different Scaling Techniques', fontsize = 20)

colors = ['#228b22', '#ff8c00', '#cd5c5c']

sns.boxplot(ax = axes[0], data = athletics_df[['Age', 'Height', 'Weight']], linewidth = 3,
           palette = colors, orient = 'h')
axes[0].set_title('Original data ', fontsize = 15)

sns.boxplot(ax = axes[1], data = standard_df, linewidth = 3, orient = 'h' , palette = colors)
axes[1].set_title('Attributes scaled using StandardScaler', fontsize = 15)

sns.boxplot(ax = axes[2], data = norm_df, linewidth = 3, orient = 'h', palette = colors)
axes[2].set_title('Attributes scaled using MinMaxScaler', fontsize = 15)

sns.boxplot(ax = axes[3], data = norm_df_no_outliers, linewidth = 3, orient= 'h', palette = colors)
axes[3].set_title('Attributes scaled using MinMaxScaler(outliers removed)', fontsize = 15)

sns.boxplot(ax = axes[4], data = robust_df,  linewidth = 3, orient='h', palette = colors)
axes[4].set_title('Attributes scaled using Robust Scaler', fontsize = 15)

plt.tight_layout()
plt.show()
```



In this boxplots comparation we observe:

- Using StandarScaler: all boxplots are centered around the same area (median is close to zero). Weight attribute has so many outliers that makes the boxplot more tight compared to the other two. In the three cases we see that outliers mostly come from higher values.
- Using MinMaxScaler : Even though all three attributes are in the same scale, they are centered around differente values, we already saw that looking at the distributions, maybe this is the effect the outliers have on each type of data.
- MixMaxScaler after removing outliers: In this case we get a more symmetric distrtibution of the three attributes, and we observe that Weight boxplot is much more wider in this case than in the previous ones, being the attribute that has more outliers it is the one that changes the most after removing them.
- Use of RobustScaler: here we see that the three attributes are quite similar to each other, their boxplot's size is almost equal between them an their medians are centered around zero. We continue to see the outliers so no information is lost in this transformation.

Conclusions After using all this techniques of scaling and treatment of outliers, we can reach some conclusions. Removing outliers may not be such a good idea in all cases because outliers may provide with some information about the data or tendencies that we would be missing if we delete them. In our dataset there are a lot of outliers but they do not seem to be measurement errors. Even though we are studying athletes of the same sport (Athletics), there are subcategories in this sport in which some athletes may have different attributes (for instance, athletes that compete in Hammer Throw clearly will not have the same values as the ones running the Marathon). The different types of events we have leads to different attributes in the athletes, that can be a possible explanation of the reason we have so many outliers.

RobustScaler seems like a good technique, it´s a quick way to scale the different attributes to the same values but at the same time we keep the information that outliers provide us.

In [ ]: