s08_t01_feature_engineering

April 3, 2022

1 IT Academy - Data Science Itinerary

1.1 S08 T01: Feature Engineering

---

[1]:
```
#importing libraries
import pandas as pd
import kaggle
import os
from sklearn import preprocessing
import missingno as msno
import matplotlib.pyplot as plt
from sklearn.preprocessing import OneHotEncoder,\
OrdinalEncoder, StandardScaler, RobustScaler,\
MinMaxScaler,MaxAbsScaler,Normalizer,QuantileTransformer,\
PowerTransformer
from sklearn.decomposition import PCA
import numpy as np
import seaborn as sns
import warnings
import plotly.express as px
warnings.filterwarnings('ignore')
```

____ #### Exercise 1

Grab a sports-themed dataset that you like and normalize categorical attributes in dummy. Standardize numeric attributes with StandardScaler.

____

for this exercice we are going to use this dataset

The columns are:

- ID - Unique number for each athlete
- Name - Athlete's name
- Sex - M or F
- Age - Integer
- Height - In centimeters
- Weight - In kilograms
- Team - Team name
- NOC - National Olympic Committee 3-letter code
- Games - Year and season
- Year - Integer
- Season - Summer or Winter
- City - Host city
- Sport - Sport
- Event - Event
- Medal - Gold, Silver, Bronze, or NA

let's use the following code to download the dataset from kagggle:

[2]:
```
PATH = "./data"

if not os.path.exists(PATH):
    os.makedirs(PATH)
if not os.listdir(PATH):
    !kaggle datasets download -d "heesoo37/
↪120-years-of-olympic-history-athletes-and-results" --unzip -p $PATH
```

[3]:
```
files = [os.path.join(PATH, f) for f in os.listdir(PATH)]
for f in files:
    print(f)
```

```
./data/noc_regions.csv
./data/athlete_events.csv
```

[4]:
```
df = pd.read_csv(files[1])
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 271116 entries, 0 to 271115
Data columns (total 15 columns):
 #   Column  Non-Null Count   Dtype
---  ------  --------------   -----
 0   ID      271116 non-null  int64
 1   Name    271116 non-null  object
 2   Sex     271116 non-null  object
 3   Age     261642 non-null  float64
 4   Height  210945 non-null  float64
```

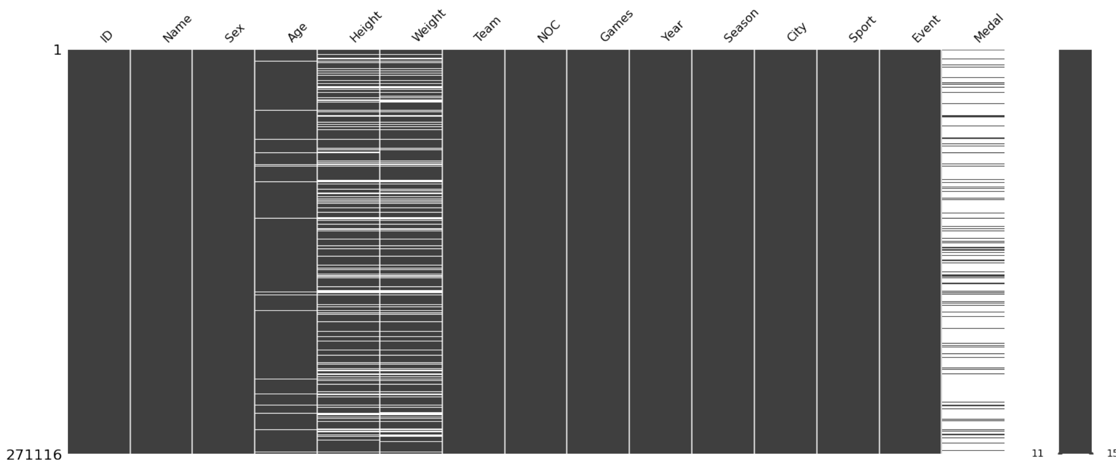
```
 5   Weight  208241 non-null  float64
 6   Team    271116 non-null  object
 7   NOC     271116 non-null  object
 8   Games   271116 non-null  object
 9   Year    271116 non-null  int64
 10  Season  271116 non-null  object
 11  City    271116 non-null  object
 12  Sport   271116 non-null  object
 13  Event   271116 non-null  object
 14  Medal   39783 non-null   object
dtypes: float64(3), int64(2), object(10)
memory usage: 31.0+ MB
```

[5]:
```
df.head()
```

[5]:
```
   ID                      Name Sex   Age  Height  Weight            Team  \
0   1                 A Dijiang   M  24.0   180.0    80.0           China
1   2                  A Lamusi   M  23.0   170.0    60.0           China
2   3       Gunnar Nielsen Aaby   M  24.0     NaN     NaN         Denmark
3   4      Edgar Lindenau Aabye   M  34.0     NaN     NaN  Denmark/Sweden
4   5  Christine Jacoba Aaftink   F  21.0   185.0    82.0     Netherlands

   NOC        Games  Year  Season       City          Sport  \
0  CHN  1992 Summer  1992  Summer  Barcelona     Basketball
1  CHN  2012 Summer  2012  Summer     London           Judo
2  DEN  1920 Summer  1920  Summer  Antwerpen       Football
3  DEN  1900 Summer  1900  Summer      Paris     Tug-Of-War
4  NED  1988 Winter  1988  Winter    Calgary  Speed Skating

                              Event Medal
0       Basketball Men's Basketball   NaN
1      Judo Men's Extra-Lightweight   NaN
2           Football Men's Football   NaN
3       Tug-Of-War Men's Tug-Of-War  Gold
4  Speed Skating Women's 500 metres   NaN
```

once we have the dataset exported in a pandas dataframe, let's make a plot to graphically see our data

[6]:
```
msno.matrix(df)
plt.show()
```



In the plot above we observe there are columns (with blank spaces) like “Age”, “Height”, “Weight” and “Medal”. with null or missing values.

For the “Medal” column we are going to substitute the NaN for the string “No_medal”. This will help us later when we have to transform the categorical variables

Since our data set is not small, for the first three columns we are going to choose to remove empty values. It is true that we could fill the empty values with zeros or the means. But, for this exercise we have decided to eliminate the rows with empty values

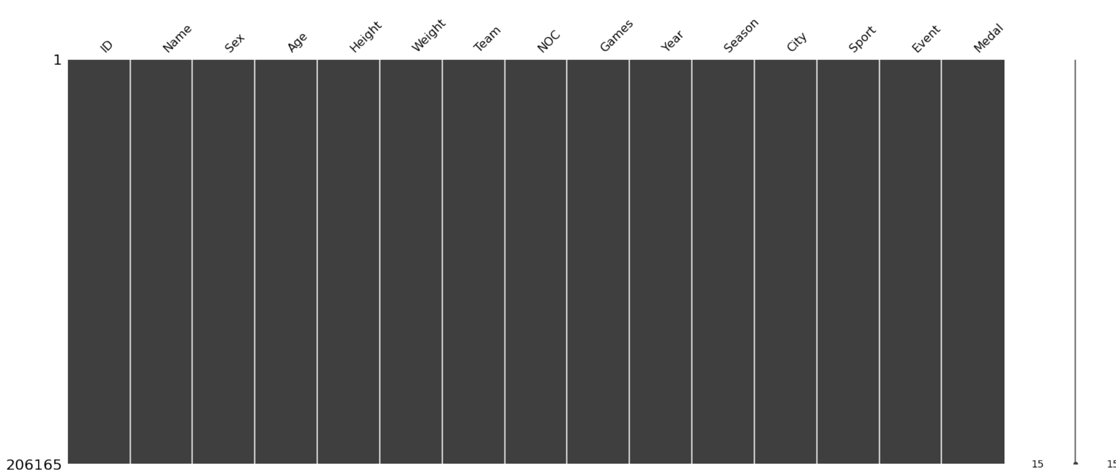
[7]:
```
#check number of NaN values in every column.
print(df.isnull().sum(axis=0))
```

```
ID            0
Name          0
Sex           0
Age        9474
Height    60171
Weight    62875
Team          0
NOC           0
Games         0
Year          0
Season        0
City          0
Sport         0
Event         0
```

```
Medal     231333
dtype: int64
```

[8]:
```
# remplace NaN by no_medal
df['Medal'] = df['Medal'].fillna("no_medal")

#drop row with Nan Values
df.dropna(subset=['Age', 'Height', 'Weight'],inplace=True)

#checking again the data
print(df.isnull().sum(axis=0))

#plotinh the data
msno.matrix(df)
plt.show()
```

```
ID        0
Name      0
Sex       0
Age       0
Height    0
Weight    0
Team      0
NOC       0
Games     0
Year      0
Season    0
City      0
Sport     0
Event     0
Medal     0
dtype: int64
```



once we have cleaned the dataset of Nan values, let's see if we have repeated values to remove them

[9]:
```
#check for duplicates
print("duplicated values: ", sum(df.duplicated()))

df[df.duplicated()].head()
```

```
duplicated values:  13
```

[9]:
```
          ID                 Name Sex   Age  Height  Weight           Team  \
87975  44600         Gavin Hadden   M  44.0   188.0    77.0  United States
87976  44600         Gavin Hadden   M  44.0   188.0    77.0  United States
87977  44600         Gavin Hadden   M  44.0   188.0    77.0  United States
92832  47034  Louis Hechenbleikner   M  38.0   178.0    67.0  United States
92833  47034  Louis Hechenbleikner   M  38.0   178.0    67.0  United States

       NOC        Games  Year  Season         City              Sport  \
87975  USA  1932 Summer  1932  Summer  Los Angeles  Art Competitions
87976  USA  1932 Summer  1932  Summer  Los Angeles  Art Competitions
87977  USA  1932 Summer  1932  Summer  Los Angeles  Art Competitions
92832  USA  1932 Summer  1932  Summer  Los Angeles  Art Competitions
92833  USA  1932 Summer  1932  Summer  Los Angeles  Art Competitions

                                                    Event     Medal
87975  Art Competitions Mixed Architecture, Unknown E…  no_medal
87976  Art Competitions Mixed Architecture, Unknown E…  no_medal
87977  Art Competitions Mixed Architecture, Unknown E…  no_medal
92832     Art Competitions Mixed Painting, Unknown Event  no_medal
92833     Art Competitions Mixed Painting, Unknown Event  no_medal
```

since there are duplicate values in our data we are going to remove it:

[10]:
```
#drop duplicates
df.drop_duplicates(inplace=True)
```

The next step: checking the type of the columns. Where we have a categorical variable -defined as "object"- we will convert it to type "category"

[11]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 206152 entries, 0 to 271115
Data columns (total 15 columns):
 #   Column  Non-Null Count   Dtype
---  ------  --------------   -----
 0   ID      206152 non-null  int64
 1   Name    206152 non-null  object
 2   Sex     206152 non-null  object
 3   Age     206152 non-null  float64
 4   Height  206152 non-null  float64
 5   Weight  206152 non-null  float64
 6   Team    206152 non-null  object
 7   NOC     206152 non-null  object
 8   Games   206152 non-null  object
 9   Year    206152 non-null  int64
 10  Season  206152 non-null  object
 11  City    206152 non-null  object
 12  Sport   206152 non-null  object
 13  Event   206152 non-null  object
 14  Medal   206152 non-null  object
dtypes: float64(3), int64(2), object(10)
memory usage: 25.2+ MB
```

[12]:
```
#filter object columns
print(df.select_dtypes(include=[object]).columns)

cat_var = df.select_dtypes(include=[object]).columns
```

```
Index(['Name', 'Sex', 'Team', 'NOC', 'Games', 'Season', 'City', 'Sport',
       'Event', 'Medal'],
      dtype='object')
```

[13]:
```
# we are going to exclude "Name" column from the type transformation

df[cat_var[1:]] = df[cat_var[1:]].astype("category")

print(df.info())
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 206152 entries, 0 to 271115
Data columns (total 15 columns):
 #   Column  Non-Null Count   Dtype
---  ------  --------------   -----
 0   ID      206152 non-null  int64
 1   Name    206152 non-null  object
 2   Sex     206152 non-null  category
 3   Age     206152 non-null  float64
 4   Height  206152 non-null  float64
 5   Weight  206152 non-null  float64
```

```
 6   Team    206152 non-null  category
 7   NOC     206152 non-null  category
 8   Games   206152 non-null  category
 9   Year    206152 non-null  int64
 10  Season  206152 non-null  category
 11  City    206152 non-null  category
 12  Sport   206152 non-null  category
 13  Event   206152 non-null  category
 14  Medal   206152 non-null  category
dtypes: category(9), float64(3), int64(2), object(1)
memory usage: 13.4+ MB
None
```

Once we have a clean the dataframe,the next step is normalize the categorical attributes in dummies variables. We are going to do it in two diferents ways:

- with with scikit-learn
- with the pandas library

with scikit-learn:

[14]:
```
#filter category (exclude name) columns

X = df[["Sex","NOC","Games","Season","City","Sport","Event"]]
```

At this point, we understand that the variable "Medal", in addition to being categorical, is also ordinal. In other words, "Medal"omprises a finite set of discrete values with a ranked ordering between values. For this reason, instead of "One hot encoding" we will use an "ordinal encoding"

[15]:
```
#Unorderer (nominal) data

coder = OneHotEncoder(handle_unknown="ignore",drop='if_binary') #drop binary to␣
 ↪avoid multicollinarity
result = coder.fit_transform(X)
X_categorical = pd.DataFrame.sparse.from_spmatrix(result)
X_categorical.columns = coder.get_feature_names_out()
X_categorical
```

[15]:
```
        Sex_M  NOC_AFG  NOC_AHO  NOC_ALB  NOC_ALG  NOC_AND  NOC_ANG  NOC_ANT  \
0         1.0      0.0      0.0      0.0      0.0      0.0      0.0      0.0
1         1.0      0.0      0.0      0.0      0.0      0.0      0.0      0.0
2         0.0      0.0      0.0      0.0      0.0      0.0      0.0      0.0
3         0.0      0.0      0.0      0.0      0.0      0.0      0.0      0.0
4         0.0      0.0      0.0      0.0      0.0      0.0      0.0      0.0
…         …        …        …        …        …        …        …        …
206147    1.0      0.0      0.0      0.0      0.0      0.0      0.0      0.0
206148    1.0      0.0      0.0      0.0      0.0      0.0      0.0      0.0
206149    1.0      0.0      0.0      0.0      0.0      0.0      0.0      0.0
206150    1.0      0.0      0.0      0.0      0.0      0.0      0.0      0.0
206151    1.0      0.0      0.0      0.0      0.0      0.0      0.0      0.0

        NOC_ANZ  NOC_ARG  …  \
0           0.0      0.0  …
1           0.0      0.0  …
2           0.0      0.0  …
3           0.0      0.0  …
4           0.0      0.0  …
…           …        …    …
206147      0.0      0.0  …
206148      0.0      0.0  …
206149      0.0      0.0  …
206150      0.0      0.0  …
206151      0.0      0.0  …

        Event_Wrestling Men's Super-Heavyweight, Greco-Roman  \
0                                                     0.0
1                                                     0.0
2                                                     0.0
3                                                     0.0
4                                                     0.0
…                                                   …
206147                                                0.0
206148                                                0.0
206149                                                0.0
206150                                                0.0
206151                                                0.0

        Event_Wrestling Men's Unlimited Class, Greco-Roman  \
0                                                     0.0
1                                                     0.0
2                                                     0.0
3                                                     0.0
4                                                     0.0
…                                                   …
206147                                                0.0
```

```
206148                                              0.0
206149                                              0.0
206150                                              0.0
206151                                              0.0

        Event_Wrestling Men's Welterweight, Freestyle  \
0                                                0.0
1                                                0.0
2                                                0.0
3                                                0.0
4                                                0.0
…                                              …
206147                                           0.0
206148                                           0.0
206149                                           0.0
206150                                           0.0
206151                                           0.0

        Event_Wrestling Men's Welterweight, Greco-Roman  \
0                                                  0.0
1                                                  0.0
2                                                  0.0
3                                                  0.0
4                                                  0.0
…                                                …
206147                                             0.0
206148                                             0.0
206149                                             0.0
206150                                             0.0
206151                                             0.0

        Event_Wrestling Women's Featherweight, Freestyle  \
0                                                   0.0
1                                                   0.0
2                                                   0.0
3                                                   0.0
4                                                   0.0
…                                                 …
206147                                              0.0
206148                                              0.0
206149                                              0.0
206150                                              0.0
206151                                              0.0

        Event_Wrestling Women's Flyweight, Freestyle  \
0                                               0.0
1                                               0.0
```

```
2                                                0.0
3                                                0.0
4                                                0.0
…                                              …
206147                                           0.0
206148                                           0.0
206149                                           0.0
206150                                           0.0
206151                                           0.0

        Event_Wrestling Women's Heavyweight, Freestyle  \
0                                                  0.0
1                                                  0.0
2                                                  0.0
3                                                  0.0
4                                                  0.0
…                                                …
206147                                             0.0
206148                                             0.0
206149                                             0.0
206150                                             0.0
206151                                             0.0

        Event_Wrestling Women's Light-Heavyweight, Freestyle  \
0                                                     0.0
1                                                     0.0
2                                                     0.0
3                                                     0.0
4                                                     0.0
…                                                   …
206147                                                0.0
206148                                                0.0
206149                                                0.0
206150                                                0.0
206151                                                0.0

        Event_Wrestling Women's Lightweight, Freestyle  \
0                                                  0.0
1                                                  0.0
2                                                  0.0
3                                                  0.0
4                                                  0.0
…                                                …
206147                                             0.0
206148                                             0.0
206149                                             0.0
206150                                             0.0
```

```
206151                                             0.0

        Event_Wrestling Women's Middleweight, Freestyle
0                                                   0.0
1                                                   0.0
2                                                   0.0
3                                                   0.0
4                                                   0.0
…                                                 …
206147                                              0.0
206148                                              0.0
206149                                              0.0
206150                                              0.0
206151                                              0.0

[206152 rows x 967 columns]
```

[16]:
```
# create a dataframe with the ordinal data
medal_column = df[["Medal"]].copy()

# Create Ordinal encoder

encoder = OrdinalEncoder(categories=[["Gold","Silver","Bronze","no_medal"]])

encoder = encoder.fit_transform(medal_column[["Medal"]])

# Assign back encoded values to new dataframe medal_encoder

medal_encoder =pd.DataFrame(encoder).rename(columns={0:"medal_encoder"})

medal_encoder.head()
```

[16]:

	medal_encoder
0	3.0
1	3.0
2	3.0
3	3.0
4	3.0

now we can group all the categorical variables

[17]:
```
df_encode_categorical = pd.concat([medal_encoder, X_categorical], axis=1)
```

[18]:
```
print(df_encode_categorical.shape)
df_encode_categorical.head()
```

```
(206152, 968)
```

[18]:

```
   medal_encoder  Sex_M  NOC_AFG  NOC_AHO  NOC_ALB  NOC_ALG  NOC_AND  NOC_ANG  \
0            3.0    1.0      0.0      0.0      0.0      0.0      0.0      0.0
1            3.0    1.0      0.0      0.0      0.0      0.0      0.0      0.0
2            3.0    0.0      0.0      0.0      0.0      0.0      0.0      0.0
3            3.0    0.0      0.0      0.0      0.0      0.0      0.0      0.0
4            3.0    0.0      0.0      0.0      0.0      0.0      0.0      0.0

   NOC_ANT  NOC_ANZ  …  \
0      0.0      0.0  …
1      0.0      0.0  …
2      0.0      0.0  …
3      0.0      0.0  …
4      0.0      0.0  …

   Event_Wrestling Men's Super-Heavyweight, Greco-Roman  \
0                                                0.0
1                                                0.0
2                                                0.0
3                                                0.0
4                                                0.0

   Event_Wrestling Men's Unlimited Class, Greco-Roman  \
0                                                0.0
1                                                0.0
2                                                0.0
3                                                0.0
4                                                0.0

   Event_Wrestling Men's Welterweight, Freestyle  \
0                                            0.0
1                                            0.0
2                                            0.0
3                                            0.0
4                                            0.0

   Event_Wrestling Men's Welterweight, Greco-Roman  \
0                                              0.0
1                                              0.0
2                                              0.0
3                                              0.0
4                                              0.0

   Event_Wrestling Women's Featherweight, Freestyle  \
0                                               0.0
1                                               0.0
```

```
2                                                  0.0
3                                                  0.0
4                                                  0.0

   Event_Wrestling Women's Flyweight, Freestyle  \
0                                              0.0
1                                              0.0
2                                              0.0
3                                              0.0
4                                              0.0

   Event_Wrestling Women's Heavyweight, Freestyle  \
0                                                0.0
1                                                0.0
2                                                0.0
3                                                0.0
4                                                0.0

   Event_Wrestling Women's Light-Heavyweight, Freestyle  \
0                                                   0.0
1                                                   0.0
2                                                   0.0
3                                                   0.0
4                                                   0.0

   Event_Wrestling Women's Lightweight, Freestyle  \
0                                                0.0
1                                                0.0
2                                                0.0
3                                                0.0
4                                                0.0

   Event_Wrestling Women's Middleweight, Freestyle
0                                                 0.0
1                                                 0.0
2                                                 0.0
3                                                 0.0
4                                                 0.0

[5 rows x 968 columns]
```

With pandas library:

let's try to do the same but using pandas.

Note that we have two binomial variables: "Sex" and "Season". Which we are going to separate and then work on them separately

[19]:
```
X[["Sex","Season"]]
```

[19]:
```
       Sex  Season
0        M  Summer
1        M  Summer
4        F  Winter
5        F  Winter
6        F  Winter
…       ..     …
271111   M  Winter
271112   M  Winter
271113   M  Winter
271114   M  Winter
271115   M  Winter

[206152 rows x 2 columns]
```

[20]:
```
#the variable "Sex" and "Season" are excluded because they are binomial

X_cat_pandas = pd.get_dummies(X[X.columns.difference(["Sex","Season"])],␣
↪drop_first=False)

#dummie binomial
binomials= pd.get_dummies(X[["Sex","Season"]], drop_first=True)

#concat
X_cat_pandas = pd.concat([binomials, X_cat_pandas], axis=1)
```

[21]:
```
print(X_cat_pandas.shape)
X_cat_pandas.head()
```

```
(206152, 967)
```

[21]:
```
   Sex_M  Season_Winter  City_Albertville  City_Amsterdam  City_Antwerpen  \
0      1              0                 0               0               0
1      1              0                 0               0               0
4      0              1                 0               0               0
5      0              1                 0               0               0
6      0              1                 1               0               0

   City_Athina  City_Atlanta  City_Barcelona  City_Beijing  City_Berlin  …  \
0            0             0               1             0            0  …
1            0             0               0             0            0  …
4            0             0               0             0            0  …
5            0             0               0             0            0  …
```

```
6          0          0          0          0          0  …

   Sport_Table Tennis  Sport_Taekwondo  Sport_Tennis  Sport_Trampolining  \
0                   0                0             0                   0
1                   0                0             0                   0
4                   0                0             0                   0
5                   0                0             0                   0
6                   0                0             0                   0

   Sport_Triathlon  Sport_Tug-Of-War  Sport_Volleyball  Sport_Water Polo  \
0                0                 0                 0                 0
1                0                 0                 0                 0
4                0                 0                 0                 0
5                0                 0                 0                 0
6                0                 0                 0                 0

   Sport_Weightlifting  Sport_Wrestling
0                    0                0
1                    0                0
4                    0                0
5                    0                0
6                    0                0

[5 rows x 967 columns]
```

once we have done the non ordinals variables, let's work with our ordinal variable:

[22]:
```python
#define a fuction to covert categories to numbers
def category_to_numeric(x):
    if x == "Gold":
        return 0
    if x == "Silver":
        return 1
    if x == "Bronze":
        return 2
    if x == "no_medal":
        return 3
```

[23]:
```python
#apply the fuction
medal_encoder_pandas = df["Medal"].apply(category_to_numeric)
medal_encoder_pandas.to_frame(name="medal_encoder")
```

[23]:
```
   medal_encoder
0              3
1              3
```

```
4                    3
5                    3
6                    3
…               …
271111               3
271112               3
271113               3
271114               3
271115               3

[206152 rows x 1 columns]
```

Standardize numeric attributes:

First, let's organize the numeric data:

[24]:
```
#filter numeric attributes
X_num = df[df.select_dtypes(include=["float64"]).columns]

X_num.head()
```

[24]:

	Age	Height	Weight
0	24.0	180.0	80.0
1	23.0	170.0	60.0
4	21.0	185.0	82.0
5	21.0	185.0	82.0
6	25.0	185.0	82.0

We are going to work with this 3 variables ("Age", "Height", "Weight")

let's see some general statistics information of our variables:

[25]:
```
X_num.describe()
```

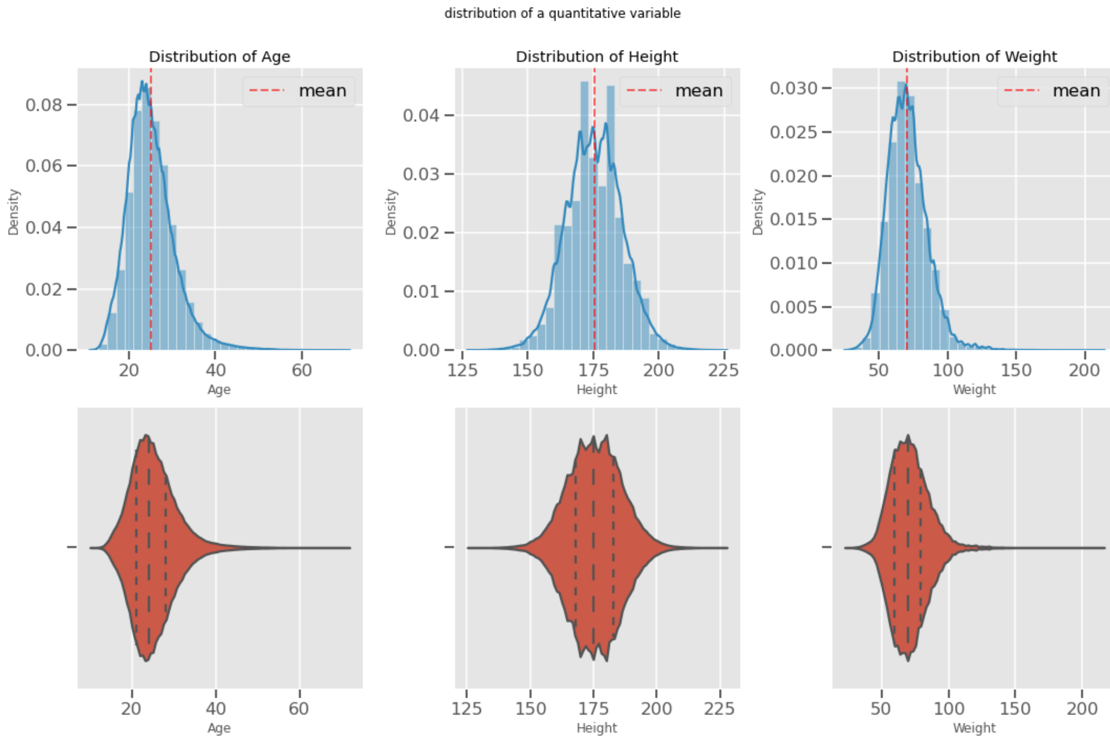
[25]:

	Age	Height	Weight
count	206152.000000	206152.000000	206152.000000
mean	25.054736	175.372056	70.688332
std	5.481679	10.545816	14.340633
min	11.000000	127.000000	25.000000
25%	21.000000	168.000000	60.000000
50%	24.000000	175.000000	70.000000

```
75%        28.000000    183.000000    79.000000
max        71.000000    226.000000   214.000000
```

Before Standardize the data, it is recommended to visualize the distribution of the data. So let's make some plots:

[26]:
```
sns.set_context("talk")
plt.style.use('ggplot')


fig, axes = plt.subplots(2,3,figsize=(15,10),alpha=0.5)

fig.subplots_adjust(top=.90)

fig.suptitle("distribution of a quantitative variable",fontsize=12)

z=0
for x in X_num:
    sns.histplot(X_num[x], bins=30, kde=True, ax= axes[0,z],stat="density").
 ↪set_title("Distribution of "+x)
    axes[0,z].axvline(x=X_num[x].mean(), linewidth=2, color='r', label="mean",␣
 ↪alpha=0.6,ls='--')
    axes[0,z].legend()
    axes[0,z].spines['top'].set_visible(False)
    axes[0,z].spines['right'].set_visible(False)
    z = z+1
v=0
for x in X_num:
    sns.violinplot(x=X_num[x],ax=axes[1,v],inner="quartile")
    v =  v+1


plt.tight_layout()
fig.subplots_adjust(top=.90)

plt.show()
```



From what we see in the plot above, we can confirm: the data not only has a fairly standard deviation but also has a many outliers. Note: the data is strongly affected by outliers

Standardize numeric attributes with StandardScaler:

As the scikit-learn documentation says, StandardScaler utility class is a quick and easy way to perform Standardization. so let's standardize our data using **StandardScaler()** .

[27]:
```
## X_num our variables to Standarize

#list for cols to scale
columns = X_num.columns
# initialize the scaler
scaler = StandardScaler()
##scale selected data
scaler = scaler.fit_transform(X_num)
X_num_standar = pd.DataFrame(scaler,columns=columns)
```

```python
display(X_num_standar.head())
display(X_num_standar.describe().round(2))
```

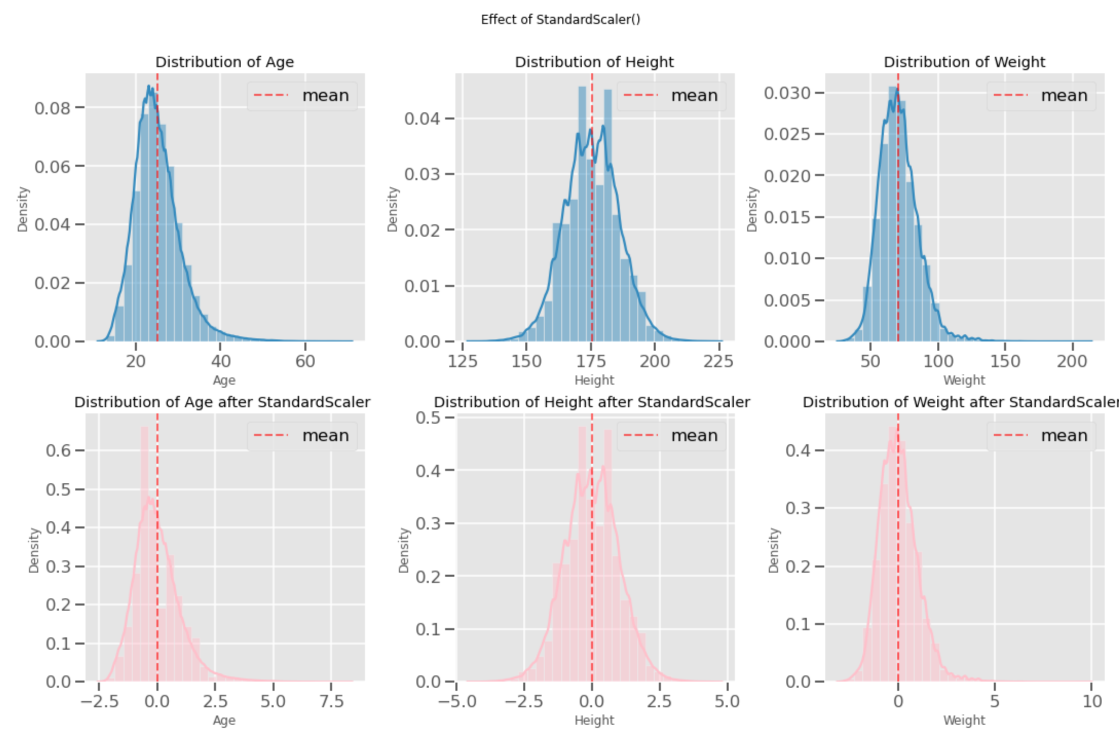
```
        Age    Height    Weight
0 -0.192412  0.438843  0.649322
1 -0.374838 -0.509403 -0.745320
2 -0.739691  0.912966  0.788786
3 -0.739691  0.912966  0.788786
4 -0.009985  0.912966  0.788786
```

	Age	Height	Weight
count	206152.00	206152.00	206152.00
mean	0.00	0.00	-0.00
std	1.00	1.00	1.00
min	-2.56	-4.59	-3.19
25%	-0.74	-0.70	-0.75
50%	-0.19	-0.04	-0.05
75%	0.54	0.72	0.58
max	8.38	4.80	9.99

see how after StandardScaler() the the standard deviation of our data is equal to 1, and the mean is equal to 0

[28]:
```python
fig, axes = plt.subplots(2,3,figsize=(15,10),alpha=0.5)

fig.subplots_adjust(top=.90)

fig.suptitle("Effect of StandardScaler()",fontsize=12)

z=0
for x in X_num:
    sns.histplot(X_num[x], bins=30, kde=True, ax= axes[0,z],stat="density").
 ↪set_title("Distribution of "+x)
    axes[0,z].axvline(x=X_num[x].mean(), linewidth=2, color='r', label="mean",␣
 ↪alpha=0.6,ls='--')
    axes[0,z].legend()
    axes[0,z].spines['top'].set_visible(False)
    axes[0,z].spines['right'].set_visible(False)

    z +=1

v=0
for x in X_num_standar:
```

```
    sns.histplot(X_num_standar[x], bins=30, kde=True, color="pink", ax=␣
↪axes[1,v],stat="density").set_title("Distribution of "+x +" after␣
↪StandardScaler ")
    axes[1,v].axvline(x=X_num_standar[x].mean(), linewidth=2, color='r',␣
↪label="mean",alpha=0.6,ls='--')
    axes[1,v].legend()
    axes[1,v].spines['top'].set_visible(False)
    axes[1,v].spines['right'].set_visible(False)

    v +=1

plt.tight_layout()
fig.subplots_adjust(top=.90)

plt.show()
```



The scikit-learn library offers other functions to standardize the data, we are going to try some of these and then plot to see the differences betweet the fuctions.

MinMaxScaler():

[29]:
```
#list for cols to scale
columns = X_num.columns
# initialize the scaler
scaler = MinMaxScaler()
##scale selected data
scaler = scaler.fit_transform(X_num)
X_MinMaxScaler = pd.DataFrame(scaler,columns=columns)

display(X_MinMaxScaler.head())
display(X_MinMaxScaler.describe().round(2))
```

	Age	Height	Weight
0	0.216667	0.535354	0.291005
1	0.200000	0.434343	0.185185
2	0.166667	0.585859	0.301587
3	0.166667	0.585859	0.301587
4	0.233333	0.585859	0.301587

	Age	Height	Weight
count	206152.00	206152.00	206152.00
mean	0.23	0.49	0.24
std	0.09	0.11	0.08
min	0.00	0.00	0.00
25%	0.17	0.41	0.19
50%	0.22	0.48	0.24
75%	0.28	0.57	0.29
max	1.00	1.00	1.00

see how after using **X__MinMaxScaler()** the minimum values of our data are equal to 0 and the maximum equal to 1

MaxAbsScaler():

[30]:
```
## X_num our variables to Standarize

#list for cols to scale
columns = X_num.columns
# initialize the scaler
scaler = MaxAbsScaler()
##scale selected data
scaler = scaler.fit_transform(X_num)
```

```
X_MaxAbsScaler = pd.DataFrame(scaler,columns=columns)

display(X_MaxAbsScaler.head())
display(X_MaxAbsScaler.describe().round(2))
```

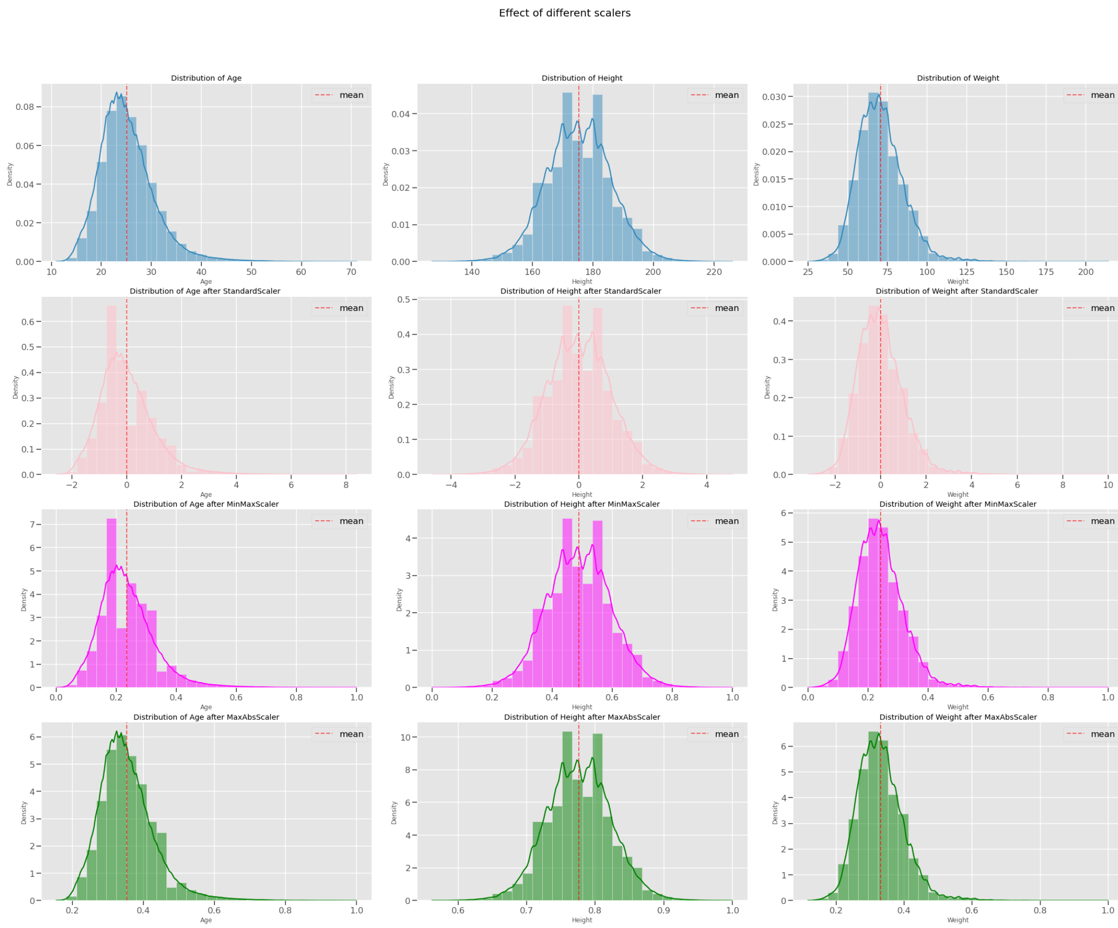
```
        Age    Height    Weight
0  0.338028  0.796460  0.373832
1  0.323944  0.752212  0.280374
2  0.295775  0.818584  0.383178
3  0.295775  0.818584  0.383178
4  0.352113  0.818584  0.383178

             Age     Height     Weight
count  206152.00  206152.00  206152.00
mean        0.35       0.78       0.33
std         0.08       0.05       0.07
min         0.15       0.56       0.12
25%         0.30       0.74       0.28
50%         0.34       0.77       0.33
75%         0.39       0.81       0.37
max         1.00       1.00       1.00
```

see how after using **X_MaxAbsScaler()** the maximum values of our data are equal to 1 and the minimum values are close to 0

let's see and compare in the next plot the effect of different scalers on our data

[31]:
```
fig, axes = plt.subplots(4,3,figsize=(30,25),alpha=0.5)

fig.subplots_adjust(top=0.90)

fig.suptitle("Effect of different scalers",fontsize=20)

z=0
for x in X_num:
    sns.histplot(X_num[x], bins=30, kde=True, ax= axes[0,z],stat="density").
 ↪set_title("Distribution of "+x)
    axes[0,z].axvline(x=X_num[x].mean(), linewidth=2, color='r', label="mean",␣
 ↪alpha=0.6,ls='--')
    axes[0,z].legend()
    axes[0,z].spines['top'].set_visible(False)
    axes[0,z].spines['right'].set_visible(False)

    z +=1
```

```
v=0
for x in X_num_standar:
    sns.histplot(X_num_standar[x], bins=30, kde=True, color="pink", ax=␣
 ↪axes[1,v],stat="density").set_title("Distribution of "+x +" after␣
 ↪StandardScaler ")
    axes[1,v].axvline(x=X_num_standar[x].mean(), linewidth=2, color='r',␣
 ↪label="mean",alpha=0.6,ls='--')
    axes[1,v].legend()
    axes[1,v].spines['top'].set_visible(False)
    axes[1,v].spines['right'].set_visible(False)

    v +=1

i=0

for x in X_MinMaxScaler:
    sns.histplot(X_MinMaxScaler[x], bins=30, kde=True, color="magenta", ax=␣
 ↪axes[2,i],stat="density").set_title("Distribution of "+x+ " after␣
 ↪MinMaxScaler")
    axes[2,i].axvline(x=X_MinMaxScaler[x].mean(), linewidth=2, color='r',␣
 ↪label="mean", alpha=0.6,ls='--')
    axes[2,i].legend()
    axes[2,i].spines['top'].set_visible(False)
    axes[2,i].spines['right'].set_visible(False)

    i +=1

j=0
for x in X_MaxAbsScaler:
    sns.histplot(X_MaxAbsScaler[x], bins=30,color="green", kde=True, ax=␣
 ↪axes[3,j],stat="density").set_title("Distribution of "+x+ " after␣
 ↪MaxAbsScaler")
    axes[3,j].axvline(x=X_MaxAbsScaler[x].mean(), linewidth=2, color='r',␣
 ↪label="mean", alpha=0.6,ls='--')
    axes[3,j].legend()
    axes[3,j].spines['top'].set_visible(False)
    axes[3,j].spines['right'].set_visible(False)

    j +=1


plt.tight_layout()
fig.subplots_adjust(top=0.90)

plt.show()
```



_____ #### Exercise 2

Continue with the sports theme data set you like and perform principal component analysis.

_____

First, let's try to make a PCA for the whole dataset (970 variables):

[32]:
```
X_to_PCA = pd.concat([X_categorical,X_num_standar], axis=1)

print(X_to_PCA.shape)
```

```
(206152, 970)
```

[33]:
```
pca = PCA(n_components=100, random_state=42)
pca.fit_transform(X_to_PCA)
```

[33]:
```
array([[ 7.65653625e-01, -4.17833739e-01, -5.96640507e-02, …,
         -6.56857142e-03,  1.24826290e-02, -2.92839728e-02],
       [-8.63776408e-01, -1.17718060e-01, -2.12993598e-01, …,
          2.25703083e-02,  3.38179380e-03, -4.55348899e-02],
       [ 8.01013754e-01, -1.01440846e+00,  7.31985119e-01, …,
         -1.37490346e-02,  2.14245523e-03,  6.51068111e-03],
       …,
       [-3.49622404e-01,  4.51768159e-01,  6.01189369e-01, …,
          3.08886395e-02, -4.91758284e-02,  1.47234616e-02],
       [ 2.09908429e+00,  3.99211844e-01,  1.09713235e+00, …,
          3.51808776e-02, -6.72694460e-03,  3.70347801e-05],
       [ 2.29598297e+00,  1.09894240e+00,  1.08764513e+00, …,
          3.47245962e-02, -1.49016849e-02,  2.93152111e-04]])
```

Let's see how much variance PCA is able to explain as we increase the number of components:

[34]:
```
exp_var_cumul = np.cumsum(pca.explained_variance_ratio_)
total_var = pca.explained_variance_ratio_.sum() * 100

px.area(
    x=range(1, exp_var_cumul.shape[0] + 1),
    y=exp_var_cumul,
    labels={"x": "Num Components", "y": "Explained Variance"},
    title=f'Total Explained Variance: {total_var:.2f}%',
)
```

let's do the same, but now with fewer variables (5 variables) : "Age","Height", "Weight", "Sex_M","Season_Winter"

PCA (n_components=3):

[35]:
```
X_to_PCA3 = pd.concat([X_num_standar,X_categorical[["Sex_M","Season_Winter"]]],␣
 ↪axis=1)

pca3 = PCA(n_components=3, random_state=20)

pca3.fit_transform(X_to_PCA3)
```

[35]:
```
array([[ 0.73955203, -0.39060213,  0.1445424 ],
       [-0.87501424, -0.14139292, -0.13526616],
       [ 0.80064559, -1.01892152,  0.0489156 ],
```

```
…,
[-0.35491368,  0.44123601, -0.42675906],
[ 2.09271692,  0.39462841,  0.70176638],
[ 2.2883418 ,  1.09564772,  0.65240317]])
```

[36]:
```
print(X_to_PCA3.shape)
```

```
(206152, 5)
```

Now let's see how much variance this PCA (n_components=3) is able to explain:

[37]:
```
components = pca3.fit_transform(X_to_PCA3)

exp_var_cumul = np.cumsum(pca3.explained_variance_ratio_)

total_var = pca3.explained_variance_ratio_.sum() * 100

px.area(
    x=range(1, exp_var_cumul.shape[0] + 1),
    y=exp_var_cumul,
    title=f'Total Explained Variance: {total_var:.2f}%',
    labels={"x": "Num Components", "y": "Explained Variance"}
)
```

Let's Visualize the PCA:

[38]:
```
total_var = pca3.explained_variance_ratio_.sum() * 100

fig = px.scatter_3d(
    components, x=0, y=1, z=2, color=df['Medal'],
    title=f'Total Explained Variance: {total_var:.2f}%',
    labels={'0': 'PC 1', '1': 'PC 2', '2': 'PC 3','color':'Medal'}
)
fig.show()
```

finally, let's try a PCA with n_components=2

PCA (n_components=2):

[39]:
```python
X_to_PCA2 = pd.concat([X_num_standar,X_categorical[["Sex_M","Season_Winter"]]],␣
↪axis=1)
```

[40]:
```python
pca2 = PCA(n_components=2, random_state=20)

pca2.fit_transform(X_to_PCA2)
```

[40]:
```
array([[ 0.73955203, -0.39060213],
       [-0.87501424, -0.14139292],
       [ 0.80064559, -1.01892152],
       …,
       [-0.35491368,  0.44123601],
       [ 2.09271692,  0.39462841],
       [ 2.2883418 ,  1.09564772]])
```

[41]:
```python
components = pca2.fit_transform(X_to_PCA2)

features = X_to_PCA2.columns

loadings = pca2.components_.T * np.sqrt(pca2.explained_variance_)

total_var = pca2.explained_variance_ratio_.sum() * 100


fig = px.scatter(components, x=0, y=1, color=df["Medal"],title=f'Total␣
↪Explained Variance: {total_var:.2f}%',
                labels={'0': 'PC 1', '1': 'PC 2',"color":"Medal"})

for i, feature in enumerate(features):
    fig.add_shape(
        type='line',
        x0=0, y0=0,
        x1=loadings[i, 0],
        y1=loadings[i, 1]
    )
    fig.add_annotation(
        x=loadings[i, 0],
        y=loadings[i, 1],
        ax=0, ay=0,
        xanchor="center",
        yanchor="bottom",
        text=feature,
    )
fig.show()
```

____ #### Exercise 3

Continue with the sports theme data set you like and normalize the data taking into account the outliers.

Following what the ScikitLearn documentation says: *"if your data contains many outliers, scaling using the mean and variance of the data is likely to not work very well. In these cases, you can use* ***RobustScaler()****"*, let's first use **RobustScaler()**.

Then we will use other alternatives which the same library offers and, finally, we will see and compare the effect of the different methods used

RobustScaler()

*This Scaler removes the median and scales the data according to the quantile range (defaults to IQR: Interquartile Range). for this case, let's use a quantile range (25, 75) in orden to leaving out the extreme outliers values

[42]:
```
#list for cols to normalize
columns = X_num.columns
# initialize the scaler
Rscaler = RobustScaler(quantile_range=(25, 75))
##scale selected data
scaler = Rscaler.fit_transform(X_num)
X_Rscaler = pd.DataFrame(scaler,columns=columns)

display(X_MinMaxScaler.head())
display(X_MinMaxScaler.describe())
```

	Age	Height	Weight
0	0.216667	0.535354	0.291005
1	0.200000	0.434343	0.185185
2	0.166667	0.585859	0.301587
3	0.166667	0.585859	0.301587
4	0.233333	0.585859	0.301587

	Age	Height	Weight
count	206152.000000	206152.000000	206152.000000
mean	0.234246	0.488607	0.241737
std	0.091361	0.106523	0.075876
min	0.000000	0.000000	0.000000
25%	0.166667	0.414141	0.185185
50%	0.216667	0.484848	0.238095

```
75%          0.283333       0.565657       0.285714
max          1.000000       1.000000       1.000000
```

Another option is to use Normalizer():

Normalizer()

[43]:
```
#list for cols to normalize
columns = X_num.columns
# initialize the scaler
Norscaler = Normalizer()
##scale selected data
scaler = Norscaler.fit_transform(X_num.T)
X_Norscaler = pd.DataFrame(scaler.T,columns=columns)

display(X_Norscaler.head())
display(X_Norscaler.describe())
```

```
        Age    Height    Weight
0  0.002061  0.002256  0.002443
1  0.001975  0.002131  0.001832
2  0.001803  0.002319  0.002504
3  0.001803  0.002319  0.002504
4  0.002147  0.002319  0.002504
```

```
                 Age          Height          Weight
count  206152.000000  206152.000000  206152.000000
mean        0.002152       0.002198       0.002158
std         0.000471       0.000132       0.000438
min         0.000945       0.001592       0.000763
25%         0.001803       0.002106       0.001832
50%         0.002061       0.002194       0.002137
75%         0.002404       0.002294       0.002412
max         0.006097       0.002833       0.006535
```

QuantileTransformer()

for this normalizer we can choose between output_distribution="normal" or output_distribution="uniform". We are going to use normal

[44]:
```
#list for cols to normalize
columns = X_num.columns
# initialize the scaler
Qscaler = QuantileTransformer(output_distribution="normal")
##scale selected data
scaler = Qscaler.fit_transform(X_num)
X_Qscaler = pd.DataFrame(scaler,columns=columns)

display(X_Qscaler.head())
display(X_Qscaler.describe())
```

```
        Age    Height    Weight
0 -0.076604  0.425228  0.739737
1 -0.295296 -0.493553 -0.729877
2 -0.763030  0.926176  0.868016
3 -0.763030  0.926176  0.868016
4  0.128317  0.926176  0.868016
```

```
                 Age          Height          Weight
count  206152.000000  206152.000000  206152.000000
mean        0.001260       0.000718      -0.000517
std         0.995561       0.998210       0.997209
min        -5.199338      -5.199338      -5.199338
25%        -0.763030      -0.694311      -0.729877
50%        -0.076604      -0.031369       0.038901
75%         0.687939       0.724973       0.665852
max         5.199338       5.199338       5.199338
```

Finally, let's try **PowerTransformer()**, it's a normalizer which supports Box-Cox transformation and Yeo-Johnson transformation.

Let's try both transformations:

PowerTransformer (method="box-cox")

[45]:
```
#list for cols to normalize
columns = X_num.columns
# initialize the scaler
ptscaler =  PowerTransformer(method="box-cox")
##scale selected data
scaler = ptscaler.fit_transform(X_num)
X_pt_box_scaler = pd.DataFrame(scaler,columns=columns)
```

```
display(X_pt_box_scaler.head())
display(X_pt_box_scaler.describe().round(2))
```

```
        Age    Height    Weight
0 -0.065530  0.440531  0.718031
1 -0.272477 -0.507759 -0.721453
2 -0.724544  0.913256  0.842052
3 -0.724544  0.913256  0.842052
4  0.130297  0.913256  0.842052
```

```
             Age     Height     Weight
count  206152.00  206152.00  206152.00
mean       -0.00      -0.00      -0.00
std         1.00       1.00       1.00
min        -4.35      -4.63      -5.04
25%        -0.72      -0.70      -0.72
50%        -0.07      -0.03       0.05
75%         0.66       0.72       0.65
max         4.35       4.76       5.72
```

PowerTransformer(method="yeo-johnson):

[46]:
```
#list for cols to normalize
columns = X_num.columns
# initialize the scaler
pt_J_scaler = PowerTransformer(method="yeo-johnson")
##scale selected data
scaler = pt_J_scaler .fit_transform(X_num)
X_pt_yeo_scaler = pd.DataFrame(scaler,columns=columns)

display(X_pt_yeo_scaler.head())
display(X_pt_yeo_scaler.describe().round(2))
```

```
        Age    Height    Weight
0 -0.065559  0.440534  0.718273
1 -0.272768 -0.507757 -0.721621
2 -0.725304  0.913257  0.842294
3 -0.725304  0.913257  0.842294
4  0.130517  0.913257  0.842294
```

```
             Age     Height     Weight
count  206152.00  206152.00  206152.00
mean        0.00       0.00      -0.00
std         1.00       1.00       1.00
min        -4.33      -4.63      -5.03
25%        -0.73      -0.70      -0.72
```

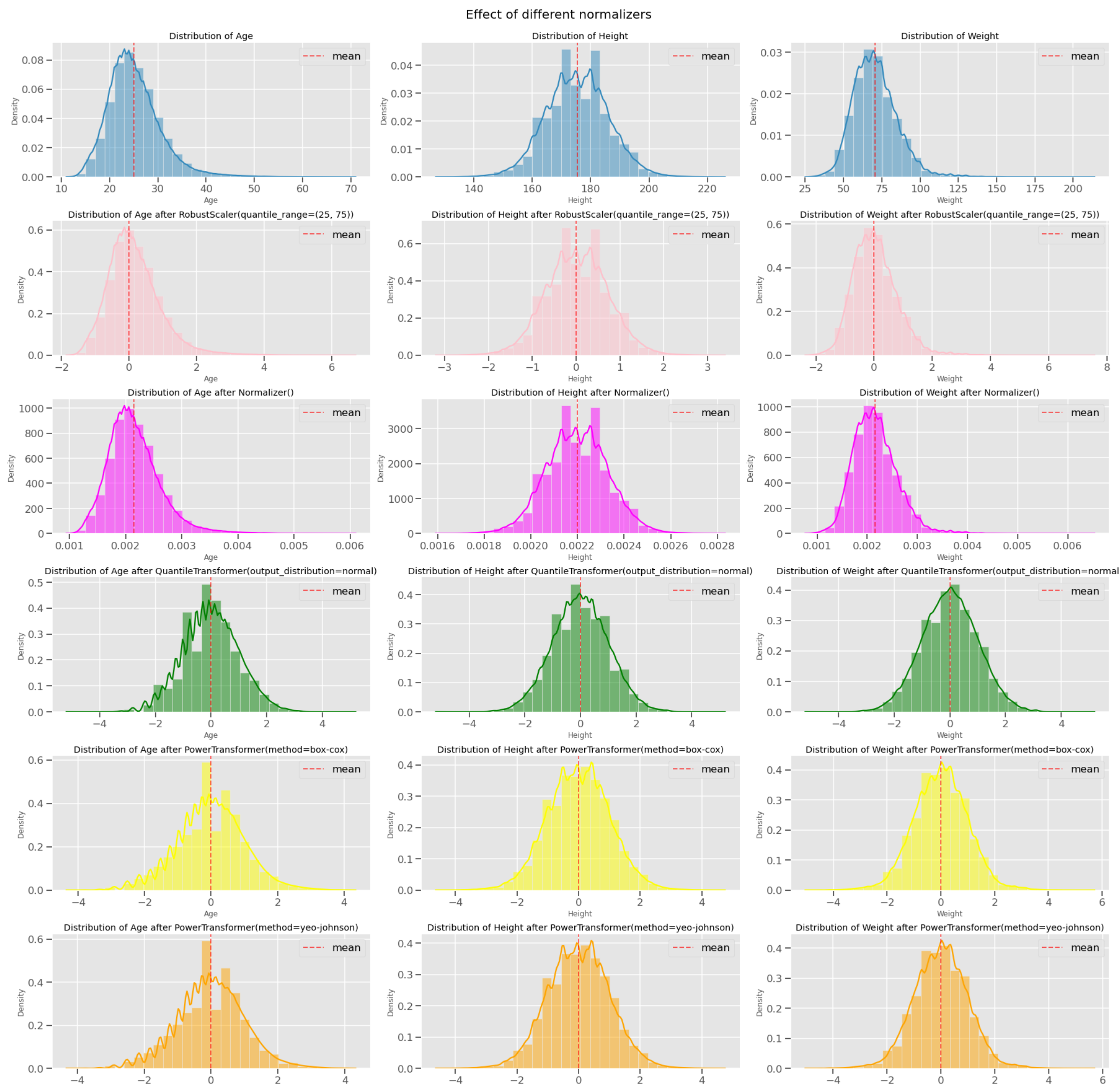
```
50%        -0.07      -0.03       0.05
75%         0.66       0.72       0.66
max         4.32       4.76       5.70
```

Compare the effect of different scalers on data with outliers:

[47]:
```python
fig, axes = plt.subplots(6,3,figsize=(25,25),alpha=0.5)

fig.subplots_adjust(top=0.95)

fig.suptitle("Effect of different normalizers",fontsize=20)

z=0
for x in X_num:
    sns.histplot(X_num[x], bins=30, kde=True, ax= axes[0,z],stat="density").
 ↪set_title("Distribution of "+x)
    axes[0,z].axvline(x=X_num[x].mean(), linewidth=2, color='r', label="mean",␣
 ↪alpha=0.6,ls='--')
    axes[0,z].legend()
    axes[0,z].spines['top'].set_visible(False)
    axes[0,z].spines['right'].set_visible(False)

    z +=1

v=0
for x in X_Rscaler:
    sns.histplot(X_Rscaler[x], bins=30, kde=True, color="pink", ax=␣
 ↪axes[1,v],stat="density").set_title("Distribution of "+x +" after␣
 ↪RobustScaler(quantile_range=(25, 75))")
    axes[1,v].axvline(x=X_num_standar[x].mean(), linewidth=2, color='r',␣
 ↪label="mean",alpha=0.6,ls='--')
    axes[1,v].legend()
    axes[1,v].spines['top'].set_visible(False)
    axes[1,v].spines['right'].set_visible(False)

    v +=1

i=0

for x in X_Norscaler:
    sns.histplot(X_Norscaler[x], bins=30, kde=True, color="magenta", ax=␣
 ↪axes[2,i],stat="density").set_title("Distribution of "+x+ " after␣
 ↪Normalizer()")
```

```
    axes[2,i].axvline(x=X_Norscaler[x].mean(), linewidth=2, color='r',␣
↪label="mean", alpha=0.6,ls='--')
    axes[2,i].legend()
    axes[2,i].spines['top'].set_visible(False)
    axes[2,i].spines['right'].set_visible(False)

    i +=1

j=0
for x in X_Qscaler:
    sns.histplot(X_Qscaler[x], bins=30,color="green", kde=True, ax=␣
↪axes[3,j],stat="density").set_title("Distribution of "+x+ " after␣
↪QuantileTransformer(output_distribution=normal)")
    axes[3,j].axvline(x=X_Qscaler[x].mean(), linewidth=2, color='r',␣
↪label="mean", alpha=0.6,ls='--')
    axes[3,j].legend()
    axes[3,j].spines['top'].set_visible(False)
    axes[3,j].spines['right'].set_visible(False)

    j +=1

k=0
for x in X_pt_box_scaler:
    sns.histplot(X_pt_box_scaler[x], bins=30,color="yellow", kde=True, ax=␣
↪axes[4,k],stat="density").set_title("Distribution of "+x+ " after␣
↪PowerTransformer(method=box-cox)")

    axes[4,k].axvline(x=X_pt_box_scaler[x].mean(), linewidth=2, color='r',␣
↪label="mean", alpha=0.6,ls='--')
    axes[4,k].legend()
    axes[4,k].spines['top'].set_visible(False)
    axes[4,k].spines['right'].set_visible(False)

    k +=1


l=0
for x in X_pt_yeo_scaler:
    sns.histplot(X_pt_yeo_scaler[x], bins=30,color="orange", kde=True, ax=␣
↪axes[5,l],stat="density").set_title("Distribution of "+x+ " after␣
↪PowerTransformer(method=yeo-johnson)")

    axes[5,l].axvline(x=X_pt_yeo_scaler[x].mean(), linewidth=2, color='r',␣
↪label="mean", alpha=0.6,ls='--')
    axes[5,l].legend()
    axes[5,l].spines['top'].set_visible(False)
```

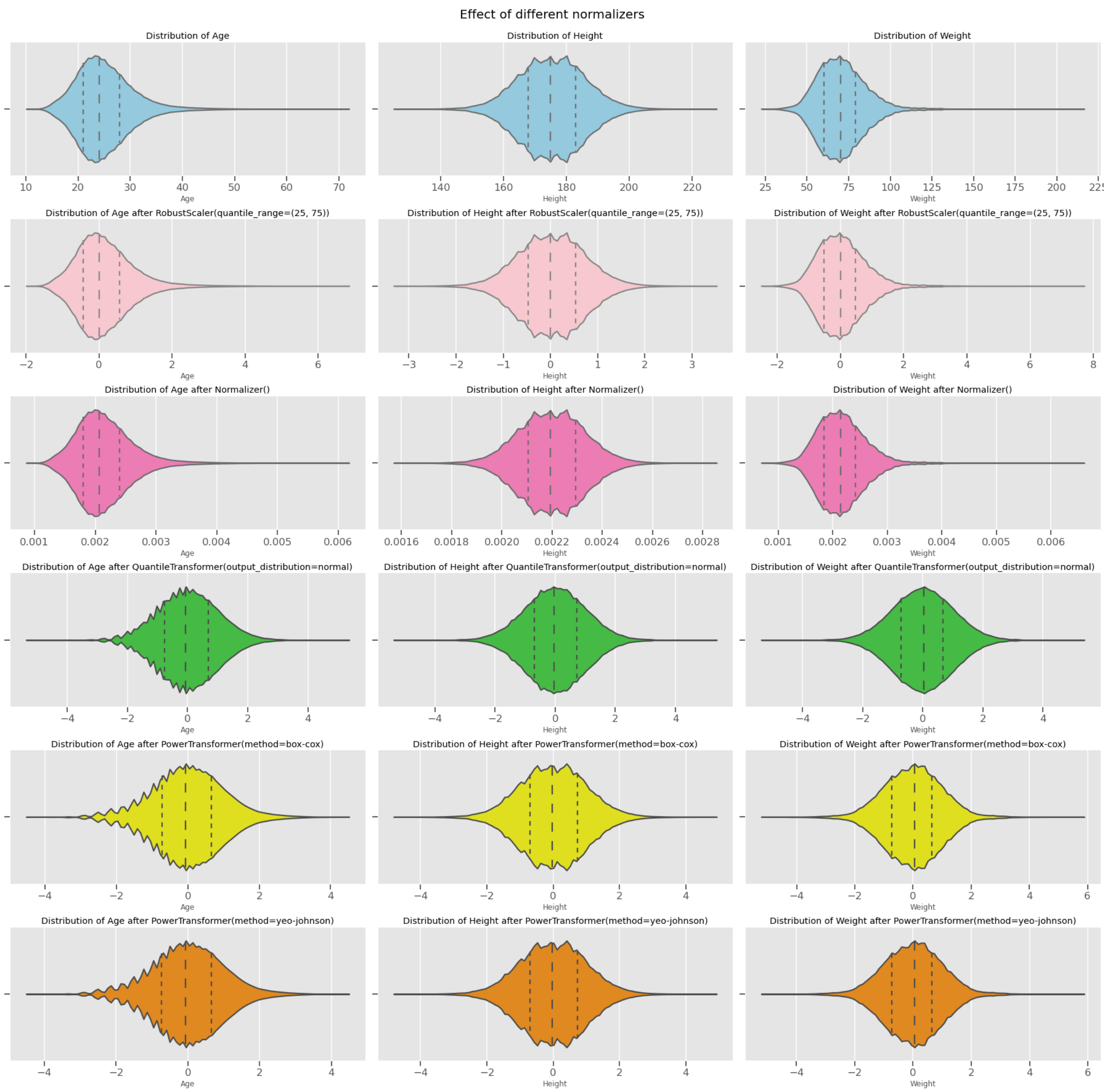
```
    axes[5,l].spines['right'].set_visible(False)

    l+=1


plt.tight_layout()
fig.subplots_adjust(top=0.95)

plt.show()
```



[48]:
```
plt.style.use('ggplot')

fig, axes = plt.subplots(6, 3, figsize=(25, 25), alpha=0.5)
```

```
fig.subplots_adjust(top=0.95)

fig.suptitle("Effect of different normalizers", fontsize=20)

z = 0
for x in X_num:
    sns.violinplot(X_num[x], inner="quartile", color="skyblue", ax=axes[0, z],
                   stat="density").set_title("Distribution of "+x)

    axes[0, z].spines['top'].set_visible(False)
    axes[0, z].spines['right'].set_visible(False)

    z += 1

v = 0
for x in X_Rscaler:
    sns.violinplot(X_Rscaler[x], inner="quartile",  color="pink", ax=axes[1,␣
 ↪v], stat="density").set_title(
        "Distribution of "+x + " after RobustScaler(quantile_range=(25, 75))")

    axes[1, v].spines['top'].set_visible(False)
    axes[1, v].spines['right'].set_visible(False)

    v += 1

i = 0

for x in X_Norscaler:
    sns.violinplot(X_Norscaler[x], inner="quartile",  color="hotpink",␣
 ↪ax=axes[2, i],
                   stat="density").set_title("Distribution of "+x + " after␣
 ↪Normalizer()")

    axes[2, i].spines['top'].set_visible(False)
    axes[2, i].spines['right'].set_visible(False)

    i += 1

j = 0
for x in X_Qscaler:
    sns.violinplot(X_Qscaler[x], inner="quartile", color="limegreen", ␣
 ↪ax=axes[3, j], stat="density").set_title(
        "Distribution of "+x + " after␣
 ↪QuantileTransformer(output_distribution=normal)")

    axes[3, j].spines['top'].set_visible(False)
```

```
    axes[3, j].spines['right'].set_visible(False)

    j += 1

k = 0
for x in X_pt_box_scaler:
    sns.violinplot(X_pt_box_scaler[x], inner="quartile", color="yellow", ␣
 ↪ax=axes[4, k], stat="density").set_title(
        "Distribution of "+x + " after PowerTransformer(method=box-cox)")

    axes[4, k].spines['top'].set_visible(False)
    axes[4, k].spines['right'].set_visible(False)

    k += 1


l = 0
for x in X_pt_yeo_scaler:
    sns.violinplot(X_pt_yeo_scaler[x], inner="quartile", color="darkorange", ␣
 ↪ax=axes[5, l], stat="density").set_title(
        "Distribution of "+x + " after PowerTransformer(method=yeo-johnson)")

    axes[5, l].spines['top'].set_visible(False)
    axes[5, l].spines['right'].set_visible(False)

    l += 1


plt.tight_layout()
fig.subplots_adjust(top=0.95)

plt.show()
```



____ #### Conclusions

- Before performing any transformation of the data (standardization, normalization, etc.) it is necessary to clean the data of null values (or Nan) -decide what is going to be done with it- and eliminate the repeated values

- Categorical variables can be treated with scikit-learn or with pandas to obtain dummy variables.
  - It must be taken into account that there are ordinal and non-ordinal categorical variables.
  - It should be observed if there are binomial categorical variables in the dataset.
- the standardization process can be perform with different methods, we have seen some of these.

+ Before perform a PCA the data have to be previously processed (clean and standarize)

```
+ the process is affected by the number of variables: the more variables the more will be the r
```

- to performance a normalize process (when there are outliers) there are different methods. We have seen some of these, and choosing one will depend on what we are going to use the data for.

References:

- Ordinal and One-Hot Encodings for Categorical Data
- Beware of the Dummy variable trap in pandas
- Scale, Standardize, or Normalize with Scikit-Learn
- Compare the effect of different scalers on data with outliers¶