LAB Assignment 6: EDA on Gait Dataset

Assigning Date : 08-Feb-2021

Due Date: 14-Feb-2021

Name:Rohit Byas

Roll No: 181210043

In [14]:

```
try:
    from google.colab import drive
    %tensorflow_version 2.x
    COLAB = True
    print("Assignment 6")
    print("Note: using Google CoLab")
except:
    print("Assignment 6")
    print("Note: not using Google CoLab")
    COLAB = False
```

```
Assignment 6
Note: not using Google CoLab
```

In [15]:

```
print("Rohit Byas", "\nRoll Number: 181210043")
```

```
Rohit Byas
Roll Number: 181210043
```

In [16]:

```
from datetime import date, datetime

today = date.today()
print("Current Date:", today)
```

```
Current Date: 2021-02-14
```

In [17]:

```
now = datetime.now()
print(now.strftime("Current Time: %H:%M:%S"))

Current Time: 20:49:01
```

Assignment Solutions

In [18]:

```
# Needed dependency to read xlsx files using pandas
!pip install openpyxl

Requirement already satisfied: openpyxl in c:\users\beeta\appdata\local\programs\python\python39\lib\
Requirement already satisfied: et-xmlfile in c:\users\beeta\appdata\local\programs\python\python39\li
Requirement already satisfied: jdcal in c:\users\beeta\appdata\local\programs\python\python39\lib\sit
```

In [19]:

```
!pip install scipy

Requirement already satisfied: scipy in c:\users\beeta\appdata\local\programs\python\python39\lib\sit
Requirement already satisfied: numpy>=1.16.5 in c:\users\beeta\appdata\local\programs\python\python39
```

In [20]:

```
# Importing required libraries:
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
from scipy import stats
import seaborn as sns
import math
from sklearn import preprocessing
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
```

4.0.1 Task 1: Getting to Know Your Data¶

4.0.2 Read Dataset from the link from LAB 1 [DM_Gait.xlsx]¶

In [21]:

```
# Downloading the xlsx file:
!gdown --id "1Kec4lVrgIi34Ty_3cvsbsWEojcr5_hMb"
```

```
Downloading...
From: https://drive.google.com/uc?id=1Kec4lVrgIi34Ty_3cvsbsWEojcr5_hMb
To: D:\Desktop Files\Beeta\College\6th Semester - Study Materials\Data Mining\Lab\Lab6\assignment\DM_

  0%|          | 0.00/39.8k [00:00<?, ?B/s]
100%|##########| 39.8k/39.8k [00:00<00:00, 2.29MB/s]
[Errno 13] Permission denied: 'DM_Gait.xlsx'
```

In [150]:

```
#Loading the excel file:
xlsx = pd.ExcelFile('./DM_Gait.xlsx')

#Declaring all_sheets list which will store all the sheets' dataframes
all_sheets = []

#Iterating through every sheet in the excel file and storing the data of each sheet in the all_sheets
for sheet_name in xlsx.sheet_names:
    all_sheets.append({"name": sheet_name,
                       "df": pd.read_excel(xlsx, sheet_name)})
```

```
c:\users\beeta\appdata\local\programs\python\python39\lib\site-packages\openpyxl\worksheet\_reader.py
  warn(msg)
```

In [151]:

```
#Deleting the last two null and duplicate rows:
try:
    all_sheets[0]["df"] = all_sheets[0]["df"].drop(index=[113, 114], axis=1)
except:
    print("Already deleted the rows!")
```

In [152]:

```
#Displaying the first sheet (data):
all_sheets[0]["df"]
```

Out[152]:

	S.No	Subject #	Gender	Age (Year)	Height (m)	Weight (Kg)	BMI	Leg Length (m)	Thigh Length (m)	Calf L
0	1.0	S21	M	30.0	1.740	61.0	20.147972	0.930	0.52	0.410
1	2.0	S22	M	27.0	1.855	65.9	19.151270	1.095	0.56	0.535
2	3.0	S23	F	27.0	1.570	62.1	25.193720	0.815	0.46	0.355
3	4.0	S24	F	22.0	1.580	59.5	23.834321	0.830	0.49	0.340
4	5.0	S25	M	20.0	1.740	80.0	26.423570	0.840	0.47	0.370
...	...	...	...	...	...	...	...	...	...	...
108	109.0	S129	M	30.0	1.800	85.0	26.234568	0.890	0.47	0.420
109	110.0	S130	M	25.0	1.670	62.0	22.230987	0.980	0.49	0.490
110	111.0	S131	M	28.0	1.700	68.0	23.529412	0.880	0.49	0.390
111	112.0	S132	M	55.0	1.670	62.3	22.338556	0.900	0.49	0.410
112	113.0	S133	M	42.0	1.680	60.2	21.329365	0.940	0.55	0.390

113 rows × 15 columns

In [153]:

```
#Deleting the last two null and duplicate rows:
try:
    all_sheets[1]["df"] = all_sheets[1]["df"].drop(index=[113,114], axis=1)
except:
    print("Already deleted the rows!")
```

In [154]:

```
#Displaying the second sheet (Spatio_Temporal):
all_sheets[1]["df"]
```

Out[154]:

	Unnamed: 0	Stance	Swing	Velocity	Cadence	Stride_len	Step_Length
0	S21	66.001777	33.998223	1.096667	119.460451	1.211936	0.605968
1	S22	71.024500	28.975500	0.953300	152.714500	1.179800	0.589900
2	S23	69.542400	30.457600	1.067500	152.810000	1.102700	0.551300
3	S24	63.074124	36.925876	1.170000	119.694881	1.337395	0.668697
4	S25	68.209900	31.790100	0.995000	112.888200	0.964900	0.482400
...	...	...	...	...	...	...	...
108	S129	75.959118	24.040882	1.325000	135.792569	1.345380	0.672690

	Unnamed: 0	Stance	Swing	Velocity	Cadence	Stride_len	Step_Length
109	S130	69.744575	30.255425	1.140000	138.123569	1.119287	0.559644
110	S131	67.857143	25.373134	1.120000	144.827586	1.158621	0.579310
111	S132	71.398990	28.601010	1.168000	152.549820	1.014113	0.507056
112	S133	68.125641	31.874359	1.212000	121.590259	1.468966	0.734483

113 rows × 7 columns

In [155]:

```
#Displaying the third sheet (Knee Angle):
all_sheets[2]["df"]
```

Out[155]:

	Unnamed: 0	Unnamed: 1	Unnamed: 2	Unnamed: 3	Unnamed: 4	Unnamed: 5	Unnamed: 6	Unnamed: 7	Unnamed: 8
0	S21	-0.138624	-0.105468	1.557842	4.082138	6.882636	9.716510	12.513191	15.092535
1	S22	2.920620	2.814374	4.541029	7.271030	10.262650	13.046121	15.359428	17.062091
2	S23	7.497443	8.860625	11.240413	13.829357	16.133285	18.078403	19.672057	20.809789
3	S24	6.076717	6.130036	7.605894	9.627953	11.463388	12.911771	14.191951	15.480457

4 rows × 63 columns

4.1 1 a). How many Attribute (or dimensions, features, variables) are there in this dataset in all three different sheet (data and Spatio-Temporal)¶

In [156]:

```
#Iterating through every sheet and printing the number of columns:
for sheet in range(len(all_sheets)):
    print("The number of attributes in", all_sheets[sheet]["name"], "is:", len(all_sheets[sheet]["df"
```

```
The number of attributes in data is: 15
The number of attributes in Spatio-Temporal is: 7
The number of attributes in Knee_angle is: 63
```

1 b). What are their Attributes Tyes [Nominal / Binary / Ordinal attributes/ Numeric] and Data types.¶

In [157]:

```
#Displaying all the data types for each sheet:
for sheet in range(len(all_sheets)):
    print("\nThe data types for", all_sheets[sheet]["name"], "are:\n", all_sheets[sheet]["df"].dtypes
```

```
    print("----------------------------------------")
```

```
The data types for data are:
 S.No                    float64
Subject #                object
Gender                   object
Age (Year)              float64
Height (m)              float64
Weight (Kg)             float64
BMI                     float64
Leg Length (m)          float64
Thigh Length (m)        float64
Calf Length             float64
Mallelous height -L     float64
Foot Length (cM)        float64
Shoe No                 float64
Waist width (inches)    float64
Bi-illiac width (m)     float64
dtype: object
----------------------------------------

The data types for Spatio-Temporal are:
 Unnamed: 0      object
Stance          float64
Swing           float64
Velocity        float64
Cadence         float64
Stride_len      float64
Step_Length     float64
dtype: object
----------------------------------------

The data types for Knee_angle are:
 Unnamed: 0      object
Unnamed: 1      float64
Unnamed: 2      float64
Unnamed: 3      float64
Unnamed: 4      float64
                 ...
Unnamed: 58     float64
Unnamed: 59     float64
Unnamed: 60     float64
Unnamed: 61     float64
Unnamed: 62     float64
```

```
Length: 63, dtype: object
----------------------------------------
```

1 c). How many data object (or samples , examples, instances, data points, objects, tuples) are there in this dataset¶

In [158]:

```
#Iterating through every sheet and printing the number of rows:
for sheet in range(len(all_sheets)):
    print("The number of tuples in", all_sheets[sheet]["name"], "is:", len(all_sheets[sheet]["df"]))
```

```
The number of tuples in data is: 113
The number of tuples in Spatio-Temporal is: 113
The number of tuples in Knee_angle is: 4
```

4.1.1 TASK 2: Handle Missing Data¶

2 a). Percentage of Missing DATA for each attributes¶

In [159]:

```
for sheet in range(len(all_sheets)):
    null_count = all_sheets[sheet]["df"].isnull().sum()
    not_null_count = all_sheets[sheet]["df"].count(axis=0)
    percentage_of_null_count = null_count/(not_null_count+null_count)*100
    print("\nThe percentage of missing values in", all_sheets[sheet]["name"], "sheet is:\n", percenta
    print("----------------------------------------------------------------")
```

```
The percentage of missing values in data sheet is:
 S.No                    0.000000
Subject #               0.000000
Gender                  0.000000
Age (Year)              0.000000
Height (m)              2.654867
Weight (Kg)             2.654867
BMI                     2.654867
Leg Length (m)          2.654867
Thigh Length (m)        2.654867
Calf Length             0.000000
Mallelous height -L     2.654867
Foot Length (cM)        2.654867
```

```
Shoe No                 2.654867
Waist width (inches)    2.654867
Bi-illiac width (m)     2.654867
dtype: float64
----------------------------------------------------------------

The percentage of missing values in Spatio-Temporal sheet is:
 Unnamed: 0     0.000000
Stance          2.654867
Swing           2.654867
Velocity        2.654867
Cadence         2.654867
Stride_len      2.654867
Step_Length     2.654867
dtype: float64
----------------------------------------------------------------

The percentage of missing values in Knee_angle sheet is:
 Unnamed: 0      0.0
Unnamed: 1       0.0
Unnamed: 2       0.0
Unnamed: 3       0.0
Unnamed: 4       0.0
                ...
Unnamed: 58     50.0
Unnamed: 59     50.0
Unnamed: 60     75.0
Unnamed: 61     75.0
Unnamed: 62     75.0
Length: 63, dtype: float64
----------------------------------------------------------------
```

2 b). Fill the Missing DATA¶

In [160]:

```
#Filling the missing data using linear interpolation:
for sheet in range(len(all_sheets)):
    all_sheets[sheet]["df"] = all_sheets[sheet]["df"].interpolate(method ='linear', limit_direction =
```

In [161]:

```
#Printing the count of null values in each attribute for each sheet:
for sheet in range(len(all_sheets)):
    print("\nThe number of null values in", all_sheets[sheet]["name"], "is:\n", all_sheets[sheet]["df
    print("-----------------------------------------------------")
```

```
The number of null values in data is:
 S.No                    0
Subject #               0
Gender                  0
Age (Year)              0
Height (m)              0
Weight (Kg)             0
BMI                     0
Leg Length (m)          0
Thigh Length (m)        0
Calf Length             0
Mallelous height -L     0
Foot Length (cM)        0
Shoe No                 0
Waist width (inches)    0
Bi-illiac width (m)     0
dtype: int64
-----------------------------------------------------

The number of null values in Spatio-Temporal is:
 Unnamed: 0     0
Stance          0
Swing           0
Velocity        0
Cadence         0
Stride_len      0
Step_Length     0
dtype: int64
-----------------------------------------------------

The number of null values in Knee_angle is:
 Unnamed: 0     0
Unnamed: 1      0
Unnamed: 2      0
Unnamed: 3      0
Unnamed: 4      0
               ..
Unnamed: 58     0
Unnamed: 59     0
```

```
Unnamed: 60    3
Unnamed: 61    3
Unnamed: 62    3
Length: 63, dtype: int64
--------------------------------------------------
```

4.1.2 TASK 3: Handling Noisy DATA / Outliers¶

Binning
– first sort data and partition into (equal-frequency) bins
– then one can smooth by bin means, smooth by bin median, smooth by bin boundaries, etc.

In [162]:

```
#Merging the first and second sheet for easier coding:
merged_df = pd.merge(all_sheets[0]["df"], all_sheets[1]["df"], right_on="Unnamed: 0", left_on="Subjec
merged_df = merged_df.drop(columns=['Unnamed: 0'])
```

In [163]:

```
merged_df
```

Out[163]:

	S.No	Subject #	Gender	Age (Year)	Height (m)	Weight (Kg)	BMI	Leg Length (m)	Thigh Length (m)	Calf L
0	1.0	S21	M	30.0	1.740	61.0	20.147972	0.930	0.52	0.410
1	2.0	S22	M	27.0	1.855	65.9	19.151270	1.095	0.56	0.535
2	3.0	S23	F	27.0	1.570	62.1	25.193720	0.815	0.46	0.355
3	4.0	S24	F	22.0	1.580	59.5	23.834321	0.830	0.49	0.340
4	5.0	S25	M	20.0	1.740	80.0	26.423570	0.840	0.47	0.370
...	...	...	...	...	...	...	...	...	...	...
108	109.0	S129	M	30.0	1.800	85.0	26.234568	0.890	0.47	0.420
109	110.0	S130	M	25.0	1.670	62.0	22.230987	0.980	0.49	0.490
110	111.0	S131	M	28.0	1.700	68.0	23.529412	0.880	0.49	0.390
111	112.0	S132	M	55.0	1.670	62.3	22.338556	0.900	0.49	0.410
112	113.0	S133	M	42.0	1.680	60.2	21.329365	0.940	0.55	0.390

113 rows × 21 columns

In [164]:

```
def equifreq(arr1, m):
    final_list = []
    a = len(arr1)
    n = int(a / m)
    count = 0
    for i in range(0, m+1):
        arr = []
        if (count+n) > len(arr1):
            arr = arr1[count::]
        else:
            arr = arr1[count:count+n]
        count=count+n
        mean = (np.array(arr)).mean()
        arr = [mean for elem in arr]
        final_list.append(np.array(arr))
    return np.hstack(np.array(final_list, dtype=object))

#Iterating through each column:
for col in merged_df.columns[4::]:
    merged_df = merged_df.sort_values([col], ascending=True)
    col_list = list(merged_df[col])
    smoothened_binned_list = equifreq(col_list, 12)
    merged_df.drop(columns=[col])
    merged_df[col]=smoothened_binned_list

merged_df = merged_df.sort_index()
```

In [165]:

```
merged_df
```

Out[165]:

	S.No	Subject #	Gender	Age (Year)	Height (m)	Weight (Kg)	BMI	Leg Length (m)	Thigh Length (m)	Calf L
0	1.0	S21	M	30.0	1.745556	61.811111	20.507301	0.935556	0.518889	0.412
1	2.0	S22	M	27.0	1.871000	66.622222	19.543862	1.019000	0.566000	0.506
2	3.0	S23	F	27.0	1.573889	61.811111	25.667457	0.816111	0.452778	0.355
3	4.0	S24	F	22.0	1.573889	59.977778	24.078185	0.816111	0.491667	0.201
4	5.0	S25	M	20.0	1.745556	78.755556	26.926892	0.846667	0.474444	0.369
...	...	...	...	...	...	...	...	...	...	...
108	109.0	S129	M	30.0	1.798889	83.888889	25.667457	0.897778	0.474444	0.421
109	110.0	S130	M	25.0	1.676667	61.811111	21.870881	0.962778	0.491667	0.462

	S.No	Subject #	Gender	Age (Year)	Height (m)	Weight (Kg)	BMI	Leg Length (m)	Thigh Length (m)	Calf L
110	111.0	S131	M	28.0	1.694444	66.622222	23.517249	0.880000	0.491667	0.388
111	112.0	S132	M	55.0	1.661667	61.811111	22.661131	0.905556	0.486667	0.412
112	113.0	S133	M	42.0	1.676667	59.977778	21.079829	0.935556	0.537778	0.388

113 rows × 21 columns

Drop a column or Row

In [166]:

```
#Dropping S.NO and Subject#:
columns_to_delete = ["S.No ", "Subject #"]
try:
    merged_df = merged_df.drop(columns=columns_to_delete)
except:
    print("S.No and Subject # columns don't exist anymore.")
```

In [167]:

```
#Displaying the dataframe after deleting:
merged_df
```

Out[167]:

	Gender	Age (Year)	Height (m)	Weight (Kg)	BMI	Leg Length (m)	Thigh Length (m)	Calf Length	Mallelous h
0	M	30.0	1.745556	61.811111	20.507301	0.935556	0.518889	0.412778	0.075000
1	M	27.0	1.871000	66.622222	19.543862	1.019000	0.566000	0.506000	0.080000
2	F	27.0	1.573889	61.811111	25.667457	0.816111	0.452778	0.355000	0.690000
3	F	22.0	1.573889	59.977778	24.078185	0.816111	0.491667	0.201111	0.072778
4	M	20.0	1.745556	78.755556	26.926892	0.846667	0.474444	0.369444	0.080000
...	...	...	...	...	...	...	...	...	...
108	M	30.0	1.798889	83.888889	25.667457	0.897778	0.474444	0.421111	0.062778
109	M	25.0	1.676667	61.811111	21.870881	0.962778	0.491667	0.462222	0.070000
110	M	28.0	1.694444	66.622222	23.517249	0.880000	0.491667	0.388889	0.070000
111	M	55.0	1.661667	61.811111	22.661131	0.905556	0.486667	0.412778	0.054444
112	M	42.0	1.676667	59.977778	21.079829	0.935556	0.537778	0.388889	0.062778

113 rows × 19 columns

4.1.3 TASK 4: Features encoding and decoding¶

In [168]:

```
# Identify the feature/s that needs encoding
# We need to map gender to a numerical 0 or 1; and map ages into a range
```

In [169]:

```
#Mapping gender:
genders = {"M": 0, "F": 1,
           1: 1, 0: 0}
merged_df['Gender'] = merged_df['Gender'].map(genders)
```

In [170]:

```
merged_df
```

Out[170]:

	Gender	Age (Year)	Height (m)	Weight (Kg)	BMI	Leg Length (m)	Thigh Length (m)	Calf Length	Mallelous h
0	0	30.0	1.745556	61.811111	20.507301	0.935556	0.518889	0.412778	0.075000
1	0	27.0	1.871000	66.622222	19.543862	1.019000	0.566000	0.506000	0.080000
2	1	27.0	1.573889	61.811111	25.667457	0.816111	0.452778	0.355000	0.690000
3	1	22.0	1.573889	59.977778	24.078185	0.816111	0.491667	0.201111	0.072778
4	0	20.0	1.745556	78.755556	26.926892	0.846667	0.474444	0.369444	0.080000
...	...	...	...	...	...	...	...	...	...
108	0	30.0	1.798889	83.888889	25.667457	0.897778	0.474444	0.421111	0.062778
109	0	25.0	1.676667	61.811111	21.870881	0.962778	0.491667	0.462222	0.070000
110	0	28.0	1.694444	66.622222	23.517249	0.880000	0.491667	0.388889	0.070000
111	0	55.0	1.661667	61.811111	22.661131	0.905556	0.486667	0.412778	0.054444
112	0	42.0	1.676667	59.977778	21.079829	0.935556	0.537778	0.388889	0.062778

113 rows × 19 columns

In [171]:

```
age_already_categorized = False
```

In [172]:

```
#Mapping ages to a range:

if not age_already_categorized:
    #Rounding the min and max to be divisible by 10:
    min = int(merged_df['Age (Year)'].min())
```

```
    min = min - (min%10)
    max = int(merged_df['Age (Year)'].max())
    max = max + (10-(max%10)) +1

    age_ranges = list(range(min, max, 10))
    merged_df['Age (Year)'] = merged_df['Age (Year)'].astype(int)

    for x in range(len(age_ranges)-1):
        merged_df.loc[(merged_df['Age (Year)']>age_ranges[x]) & (merged_df['Age (Year)']<=age_ranges[

    age_already_categorized = True

else:
    print("Already categorized the ages into their ranges")
```

In [173]:

```
#Displaying the dataframe:
merged_df
```

Out[173]:

	Gender	Age (Year)	Height (m)	Weight (Kg)	BMI	Leg Length (m)	Thigh Length (m)	Calf Length	Mallelous h
0	0	2	1.745556	61.811111	20.507301	0.935556	0.518889	0.412778	0.075000
1	0	2	1.871000	66.622222	19.543862	1.019000	0.566000	0.506000	0.080000
2	1	2	1.573889	61.811111	25.667457	0.816111	0.452778	0.355000	0.690000
3	1	2	1.573889	59.977778	24.078185	0.816111	0.491667	0.201111	0.072778
4	0	1	1.745556	78.755556	26.926892	0.846667	0.474444	0.369444	0.080000
...	...	...	...	...	...	...	...	...	...
108	0	2	1.798889	83.888889	25.667457	0.897778	0.474444	0.421111	0.062778
109	0	2	1.676667	61.811111	21.870881	0.962778	0.491667	0.462222	0.070000
110	0	2	1.694444	66.622222	23.517249	0.880000	0.491667	0.388889	0.070000
111	0	5	1.661667	61.811111	22.661131	0.905556	0.486667	0.412778	0.054444
112	0	4	1.676667	59.977778	21.079829	0.935556	0.537778	0.388889	0.062778

113 rows × 19 columns

4.1.4 TASK 5: Basic Statistical Descriptions of DataSet¶

4.1.5 Interpreting quantitative findings i.e. Descriptive Statistics:

Write your own function to find following Descriptive Statistics parameters for the attributes :

- count

- Min
- Max
- Mean
- Median
- Mode
- Midrange
- Variance and
- Standard deviation

Show the result of above Descriptive Statistics functions for the dataset.¶

In [174]:

```
all_properties = []
```

In [175]:

```
#Count
all_count = []

for col in merged_df.columns:
    count = {
        "attribute": col,
        "value": len(merged_df[col])
    }
    all_count.append(count)

all_properties.append({"property": "count", "values": all_count})

count_df = pd.DataFrame(all_count)
count_df
```

Out[175]:

	attribute	value
0	Gender	113
1	Age (Year)	113
2	Height (m)	113
3	Weight (Kg)	113
4	BMI	113
5	Leg Length (m)	113
6	Thigh Length (m)	113
7	Calf Length	113

	attribute	value
8	Mallelous height -L	113
9	Foot Length (cM)	113
10	Shoe No	113
11	Waist width (inches)	113
12	Bi-illiac width (m)	113
13	Stance	113
14	Swing	113
15	Velocity	113
16	Cadence	113
17	Stride_len	113
18	Step_Length	113

In [176]:

```
#Min
all_min = []

for col in merged_df.columns:
    min = {
        "attribute": col,
        "value": np.amin(np.array(merged_df[col]))
    }
    all_min.append(min)

all_properties.append({"property": "min", "values": all_min})

min_df = pd.DataFrame(all_min)
min_df
```

Out[176]:

	attribute	value
0	Gender	0.000000
1	Age (Year)	0.000000
2	Height (m)	1.444444
3	Weight (Kg)	41.244444
4	BMI	17.033706
5	Leg Length (m)	0.748333
6	Thigh Length (m)	0.399444
7	Calf Length	0.201111
8	Mallelous height -L	0.054444
9	Foot Length (cM)	0.211667

	attribute	value
10	Shoe No	4.222222
11	Waist width (inches)	27.444444
12	Bi-illiac width (m)	0.686667
13	Stance	59.972090
14	Swing	24.455553
15	Velocity	0.982107
16	Cadence	92.141404
17	Stride_len	0.978052
18	Step_Length	0.489026

In [177]:

```
#Max:
all_max = []

for col in merged_df.columns:
    max = {
        "attribute": col,
        "value": np.amax(np.array(merged_df[col]))
    }
    all_max.append(max)

all_properties.append({"property": "max", "values": all_max})

max_df = pd.DataFrame(all_max)
max_df
```

Out[177]:

	attribute	value
0	Gender	1.000000
1	Age (Year)	5.000000
2	Height (m)	1.871000
3	Weight (Kg)	103.820000
4	BMI	33.102478
5	Leg Length (m)	1.019000
6	Thigh Length (m)	0.566000
7	Calf Length	0.506000
8	Mallelous height -L	0.690000
9	Foot Length (cM)	0.291000
10	Shoe No	10.600000
11	Waist width (inches)	39.200000

	attribute	value
12	Bi-illiac width (m)	0.996000
13	Stance	76.611921
14	Swing	40.719783
15	Velocity	1.555133
16	Cadence	181.252880
17	Stride_len	1.606366
18	Step_Length	0.803183

In [178]:

```
#Mean:
all_mean = []

for col in merged_df.columns:
    mean = {
        "attribute": col,
        "value": np.mean(np.array(merged_df[col]))
    }
    all_mean.append(mean)

all_properties.append({"property": "mean", "values": all_mean})

mean_df = pd.DataFrame(all_mean)
mean_df
```

Out[178]:

	attribute	value
0	Gender	0.247788
1	Age (Year)	2.212389
2	Height (m)	1.681018
3	Weight (Kg)	66.938938
4	BMI	23.499537
5	Leg Length (m)	0.884602
6	Thigh Length (m)	0.480221
7	Calf Length	0.393894
8	Mallelous height -L	0.110796
9	Foot Length (cM)	0.251093
10	Shoe No	7.663717
11	Waist width (inches)	32.845133
12	Bi-illiac width (m)	0.833451
13	Stance	67.422111

	attribute	value
14	Swing	32.514351
15	Velocity	1.179267
16	Cadence	137.741276
17	Stride_len	1.235066
18	Step_Length	0.617533

In [179]:

```
#Median:
all_median = []

for col in merged_df.columns:
    median = {
        "attribute": col,
        "value": np.median(np.array(merged_df[col]))
    }
    all_median.append(median)

all_properties.append({"property": "median", "values": all_median})

median_df = pd.DataFrame(all_median)
median_df
```

Out[179]:

	attribute	value
0	Gender	0.000000
1	Age (Year)	2.000000
2	Height (m)	1.694444
3	Weight (Kg)	66.622222
4	BMI	23.517249
5	Leg Length (m)	0.889722
6	Thigh Length (m)	0.486667
7	Calf Length	0.401111
8	Mallelous height -L	0.070000
9	Foot Length (cM)	0.250000
10	Shoe No	8.000000
11	Waist width (inches)	32.000000
12	Bi-illiac width (m)	0.810000
13	Stance	67.640897
14	Swing	32.618600
15	Velocity	1.154481

	attribute	value
16	Cadence	139.062700
17	Stride_len	1.219254
18	Step_Length	0.609627

In [180]:

```
#Mode:
all_mode = []

for col in merged_df.columns:
    mode = {
        "attribute": col,
        "value": stats.mode(np.array(merged_df[col]))[0][0],
        "occurance": stats.mode(np.array(merged_df[col]))[1][0]
    }
    all_mode.append(mode)

all_properties.append({"property": "mode", "values": all_mode})

mode_df = pd.DataFrame(all_mode)
mode_df
```

Out[180]:

	attribute	value	occurance
0	Gender	0.000000	85
1	Age (Year)	2.000000	67
2	Height (m)	1.444444	9
3	Weight (Kg)	41.244444	9
4	BMI	17.033706	9
5	Leg Length (m)	0.748333	9
6	Thigh Length (m)	0.399444	9
7	Calf Length	0.201111	9
8	Mallelous height -L	0.070000	36
9	Foot Length (cM)	0.250000	18
10	Shoe No	8.000000	27
11	Waist width (inches)	32.000000	27
12	Bi-illiac width (m)	0.810000	27
13	Stance	59.972090	9
14	Swing	24.455553	9
15	Velocity	0.982107	9
16	Cadence	92.141404	9

	attribute	value	occurance
17	Stride_len	0.978052	9
18	Step_Length	0.489026	9

In [181]:

```
#Midrange:
all_midrange = []

for col in merged_df.columns:
    arr = np.array(merged_df[col])
    max = np.amax(arr)
    min = np.amin(arr)
    midrange = {
        "attribute": col,
        "value": (max-min)/2,
    }
    all_midrange.append(midrange)

all_properties.append({"property": "midrange", "values": all_midrange})

midrange_df = pd.DataFrame(all_midrange)
midrange_df
```

Out[181]:

	attribute	value
0	Gender	0.500000
1	Age (Year)	2.500000
2	Height (m)	0.213278
3	Weight (Kg)	31.287778
4	BMI	8.034386
5	Leg Length (m)	0.135333
6	Thigh Length (m)	0.083278
7	Calf Length	0.152444
8	Mallelous height -L	0.317778
9	Foot Length (cM)	0.039667
10	Shoe No	3.188889
11	Waist width (inches)	5.877778
12	Bi-illiac width (m)	0.154667
13	Stance	8.319916
14	Swing	8.132115
15	Velocity	0.286513

	attribute	value
16	Cadence	44.555738
17	Stride_len	0.314157
18	Step_Length	0.157079

In [182]:

```
#Variance:
all_variance = []

for col in merged_df.columns:
    variance = {
        "attribute": col,
        "value": np.var(np.array(merged_df[col]))
    }
    all_variance.append(variance)

all_properties.append({"property": "variance", "values": all_variance})

variance_df = pd.DataFrame(all_variance)
variance_df
```

Out[182]:

	attribute	value
0	Gender	0.186389
1	Age (Year)	0.822147
2	Height (m)	0.009766
3	Weight (Kg)	185.703349
4	BMI	13.990977
5	Leg Length (m)	0.003660
6	Thigh Length (m)	0.001667
7	Calf Length	0.004506
8	Mallelous height -L	0.017786
9	Foot Length (cM)	0.000343
10	Shoe No	2.578359
11	Waist width (inches)	8.576360
12	Bi-illiac width (m)	0.005885
13	Stance	15.495958
14	Swing	15.894669
15	Velocity	0.018720
16	Cadence	444.548736
17	Stride_len	0.024699

	attribute	value
18	Step_Length	0.006175

In [183]:

```
#Standard Deviation:
all_std = []

for col in merged_df.columns:
    std = {
        "attribute": col,
        "value": np.std(np.array(merged_df[col]))
    }
    all_std.append(std)

all_properties.append({"property": "standard_deviation", "values": all_std})

variance_df = pd.DataFrame(all_std)
variance_df
```

Out[183]:

	attribute	value
0	Gender	0.431728
1	Age (Year)	0.906723
2	Height (m)	0.098821
3	Weight (Kg)	13.627302
4	BMI	3.740451
5	Leg Length (m)	0.060500
6	Thigh Length (m)	0.040824
7	Calf Length	0.067127
8	Mallelous height -L	0.133366
9	Foot Length (cM)	0.018509
10	Shoe No	1.605727
11	Waist width (inches)	2.928542
12	Bi-illiac width (m)	0.076715
13	Stance	3.936491
14	Swing	3.986812
15	Velocity	0.136822
16	Cadence	21.084324
17	Stride_len	0.157159
18	Step_Length	0.078580

In [184]:

```
def map_to_df(elem):
    dict = { }
    dict['property'] = elem['property']
    for x in elem['values']:
        dict[x['attribute']]=x['value']
    return dict
my_describe = [map_to_df(elem) for elem in list(all_properties)]
custom_desc_df = pd.DataFrame(my_describe)
```

In [185]:

```
custom_desc_df
```

Out[185]:

	property	Gender	Age (Year)	Height (m)	Weight (Kg)	BMI	Leg Length (m)	Thigh Length (m)
0	count	113.000000	113.000000	113.000000	113.000000	113.000000	113.000000	113.000000
1	min	0.000000	0.000000	1.444444	41.244444	17.033706	0.748333	0.399444
2	max	1.000000	5.000000	1.871000	103.820000	33.102478	1.019000	0.566000
3	mean	0.247788	2.212389	1.681018	66.938938	23.499537	0.884602	0.480221
4	median	0.000000	2.000000	1.694444	66.622222	23.517249	0.889722	0.486667
5	mode	0.000000	2.000000	1.444444	41.244444	17.033706	0.748333	0.399444
6	midrange	0.500000	2.500000	0.213278	31.287778	8.034386	0.135333	0.083278
7	variance	0.186389	0.822147	0.009766	185.703349	13.990977	0.003660	0.001667
8	standard_deviation	0.431728	0.906723	0.098821	13.627302	3.740451	0.060500	0.040824

Compare your result with the standard python function eg: .describe()¶

In [186]:

```
#Extracting the required fields from the describe() method:
desc_df = merged_df.describe()
desc_df = desc_df.rename_axis('property')
desc_df = desc_df.sort_values(['property']).iloc[3:]
desc_df = desc_df.reset_index()
desc_df
```

Out[186]:

	property	Gender	Age (Year)	Height (m)	Weight (Kg)	BMI	Leg Length (m)	Thigh Length (m)	Calf Leng
0	count	113.000000	113.000000	113.000000	113.000000	113.000000	113.000000	113.000000	113.0000

	property	Gender	Age (Year)	Height (m)	Weight (Kg)	BMI	Leg Length (m)	Thigh Length (m)	Calf Leng
1	max	1.000000	5.000000	1.871000	103.820000	33.102478	1.019000	0.566000	0.506000
2	mean	0.247788	2.212389	1.681018	66.938938	23.499537	0.884602	0.480221	0.393894
3	min	0.000000	0.000000	1.444444	41.244444	17.033706	0.748333	0.399444	0.201111
4	std	0.433651	0.910762	0.099261	13.688003	3.757113	0.060770	0.041005	0.067426

In [187]:

```
#Extracting the required fields from the data that contains describe attributes made by our own metho
custom_desc_df = custom_desc_df.sort_values(['property']).loc[(custom_desc_df['property'] != 'median'
                                                               (custom_desc_df['property'] != 'mid
                                                               (custom_desc_df['property'] != 'mod
                                                               (custom_desc_df['property'] != 'var

try:
    custom_desc_df = custom_desc_df.drop(columns=['index'])
except:
    print("Index column doesn't exist. ")

custom_desc_df.iloc[4,0]="std"
custom_desc_df
```

Out[187]:

	property	Gender	Age (Year)	Height (m)	Weight (Kg)	BMI	Leg Length (m)	Thigh Length (m)	Calf Leng
0	count	113.000000	113.000000	113.000000	113.000000	113.000000	113.000000	113.000000	113.0000
1	max	1.000000	5.000000	1.871000	103.820000	33.102478	1.019000	0.566000	0.506000
2	mean	0.247788	2.212389	1.681018	66.938938	23.499537	0.884602	0.480221	0.393894
3	min	0.000000	0.000000	1.444444	41.244444	17.033706	0.748333	0.399444	0.201111
4	std	0.431728	0.906723	0.098821	13.627302	3.740451	0.060500	0.040824	0.067127

In [188]:

```
#Comparing the two dataframes:
desc_df==custom_desc_df
```

Out[188]:

	property	Gender	Age (Year)	Height (m)	Weight (Kg)	BMI	Leg Length (m)	Thigh Length (m)	Calf Length	Mallelou
0	True	True	True	True	True	True	True	True	True	True
1	True	True	True	True	True	True	True	True	True	True
2	True	True	True	True	True	True	True	True	True	True
3	True	True	True	True	True	True	True	True	True	True
4	True	False	False	False	False	False	False	False	False	False

Observation:¶

- It seems that there is a difference in the standard deviation and the way pandas calculate it in the describe() method is different from the method that i have used. ---------

---

4.1.6 TASK 6: Measuring the Dispersion of Data¶

Data dispersion:

- How the data spread out?
- analyzed with multiple granularities of precision : range, quartiles etc.
- Boxplot or quantile analysis on sorted intervals
- Useful for identifying outliers

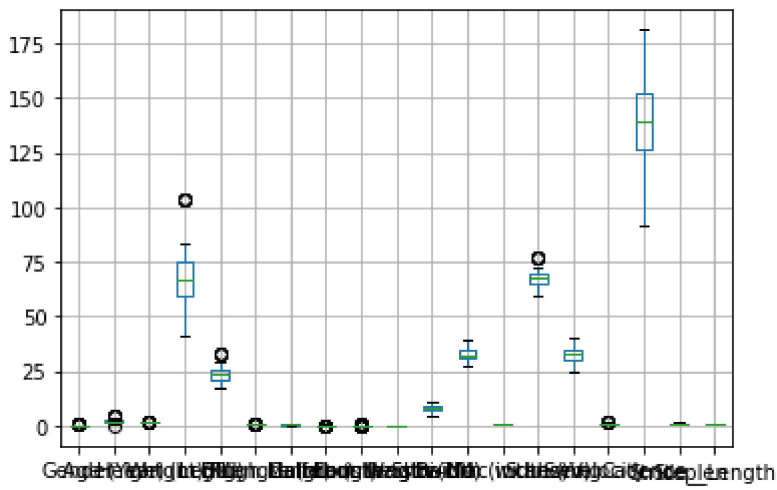
  Illustrate following Descriptive Statistics parameters for the attributes :
- Range
- Quantiles- show value of (Q1, Q2 ,Q3 and Q4)
- K- Percentile
- Inter-quartile range: IQR = Q3 – Q1

In [189]:

```
#Box plot:
merged_df.boxplot()
```

Out[189]:

```
<AxesSubplot:>
```



In [190]:

```
#Range:
min = merged_df.min()
max = merged_df.max()

df = pd.DataFrame(list(min))
df['Highest Range'] = list(max)
try:
    df = df.rename(columns = {0:'Lowest Range'})
except:
    print("Columns have already been renamed.")
```

In [191]:

```
df
```

Out[191]:

	Lowest Range	Highest Range
0	0.000000	1.000000
1	0.000000	5.000000
2	1.444444	1.871000
3	41.244444	103.820000
4	17.033706	33.102478
5	0.748333	1.019000
6	0.399444	0.566000
7	0.201111	0.506000
8	0.054444	0.690000
9	0.211667	0.291000
10	4.222222	10.600000
11	27.444444	39.200000
12	0.686667	0.996000
13	59.972090	76.611921
14	24.455553	40.719783
15	0.982107	1.555133
16	92.141404	181.252880
17	0.978052	1.606366
18	0.489026	0.803183

In [192]:

```
#Quantiles - Q1, Q2, Q3, Q4
merged_df.quantile([.25, .5, .75, 1])
```

Out[192]:

	Gender	Age (Year)	Height (m)	Weight (Kg)	BMI	Leg Length (m)	Thigh Length (m)	Calf Length	Mallelous h
0.25	0.0	2.0	1.644444	59.977778	21.079829	0.861389	0.452778	0.380000	0.070000
0.50	0.0	2.0	1.694444	66.622222	23.517249	0.889722	0.486667	0.401111	0.070000
0.75	0.0	3.0	1.745556	75.388889	25.667457	0.920000	0.508333	0.433333	0.079444
1.00	1.0	5.0	1.871000	103.820000	33.102478	1.019000	0.566000	0.506000	0.690000

In [193]:

```
#K-percentile:
k = int(input("Enter the value of K (out of 100 - percent): "))
merged_df.quantile([k*0.01])

Enter the value of K (out of 100 - percent): 47
```

Out[193]:

	Gender	Age (Year)	Height (m)	Weight (Kg)	BMI	Leg Length (m)	Thigh Length (m)	Calf Length	Mallelous h
0.47	0.0	2.0	1.676667	63.555556	22.661131	0.88	0.474444	0.393333	0.07

In [194]:

```
Q3_df = merged_df.quantile([.75]).reset_index()
Q1_df = merged_df.quantile([.25]).reset_index()

#Interquartile = Q3-Q1:
(Q3_df - Q1_df).drop(columns=['index'])
```

Out[194]:

	Gender	Age (Year)	Height (m)	Weight (Kg)	BMI	Leg Length (m)	Thigh Length (m)	Calf Length	Mallelous heigh
0	0.0	1.0	0.101111	15.411111	4.587628	0.058611	0.055556	0.053333	0.009444

4.1.7 TASK 7: Graphic Displays of Basic Statistical Descriptions¶

Basic Statistical Descriptions of Data by Visually inspect the data using the python packages - Bar charts - Pie charts - Line graphs - Histograms - Scatter plots - BOX plots

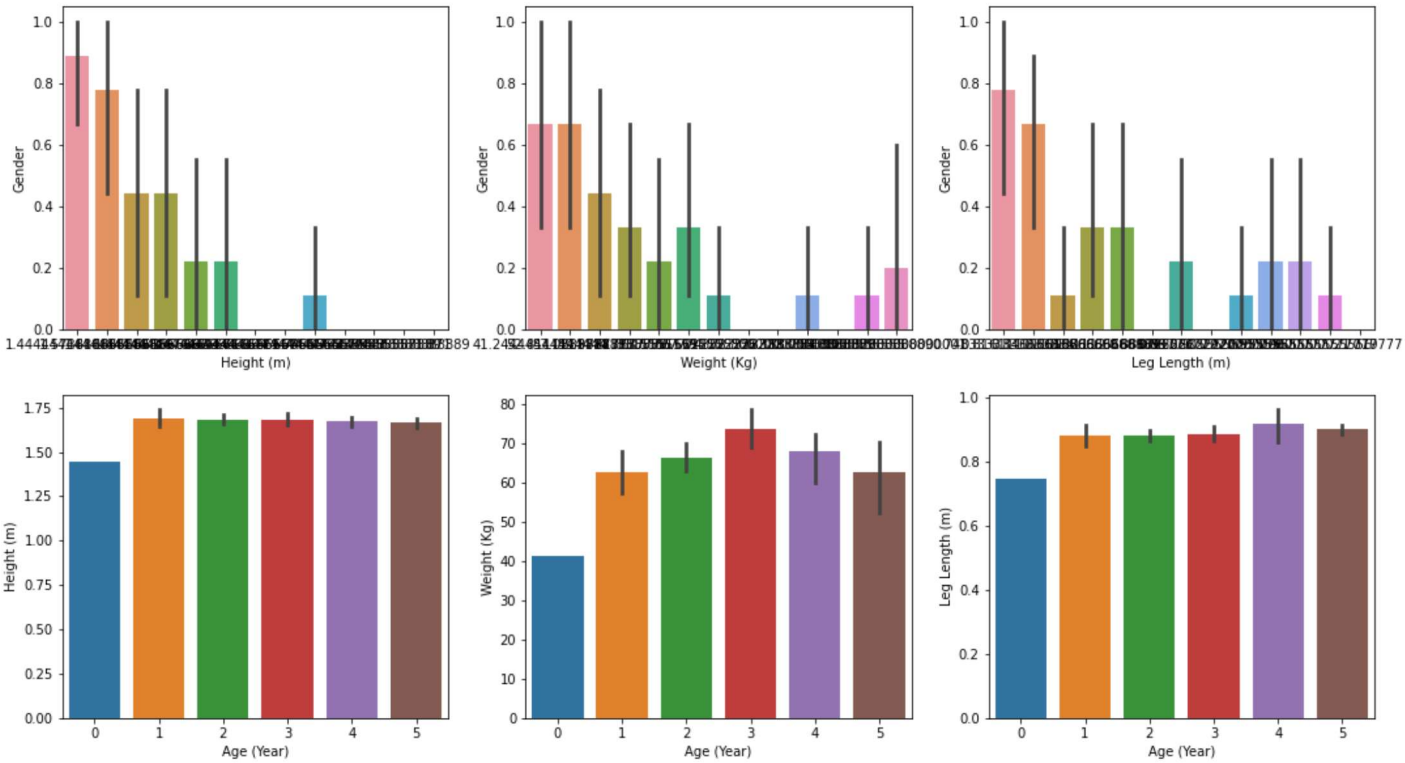
BAR Charts for Gender, Age, Height, Weight, Leg Length¶

In [195]:

```
cols = ['Gender', 'Age (Year)', 'Height (m)', 'Weight (Kg)', 'Leg Length (m)']

fig, axes = plt.subplots(2, 3, figsize=(18, 10))

#Drawing bar graphs for (Gender w/ every column) and (Age w/ every column)
for x in range(3):
    sns.barplot(ax=axes[0, x], data=merged_df, y=cols[0], x=cols[x+2])
    sns.barplot(ax=axes[1, x], data=merged_df, x=cols[1], y=cols[x+2])
```

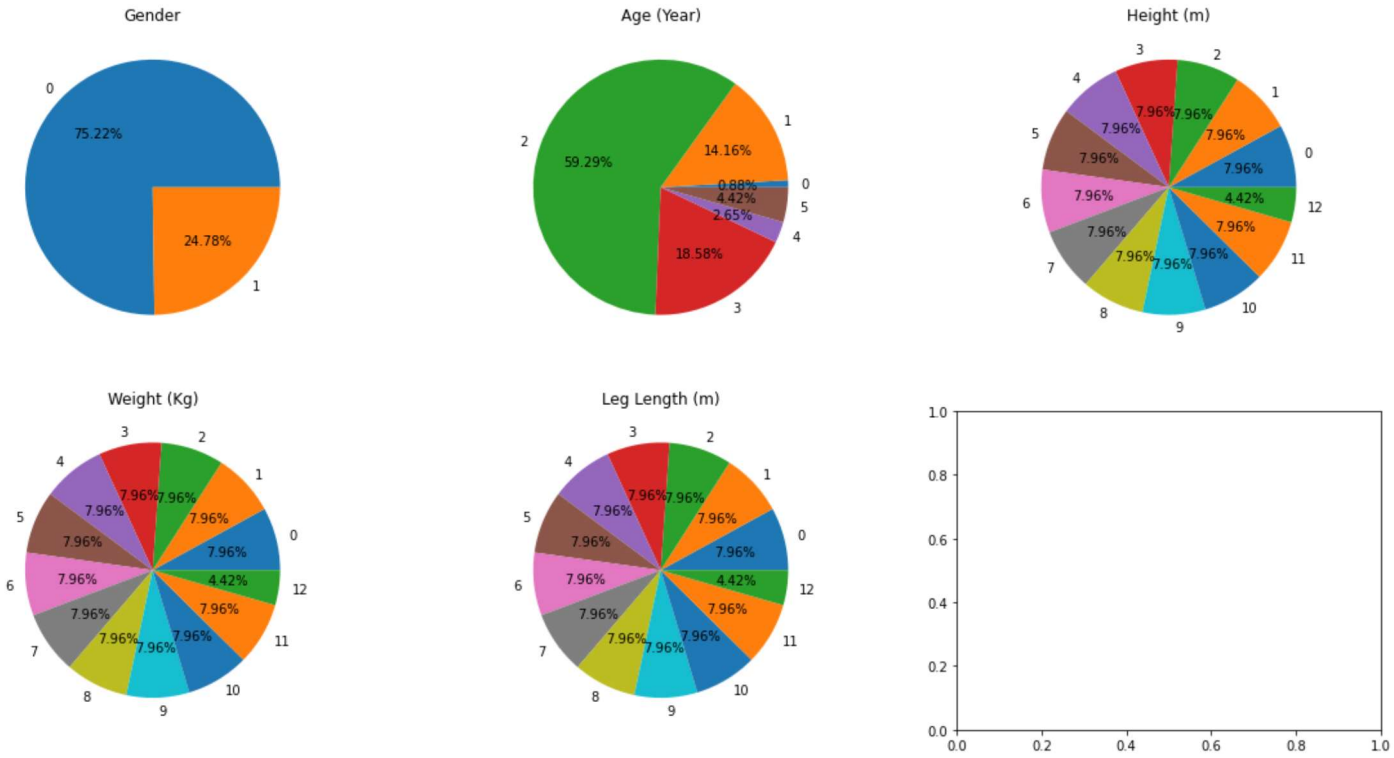


PIE Charts for Gender, Age, Height, Weight, Leg Length¶

In [196]:

```
cols = ['Gender', 'Age (Year)', 'Height (m)', 'Weight (Kg)', 'Leg Length (m)']

fig, axes = plt.subplots(2, 3, figsize=(20, 10))
for index in range(5):
    x = np.array(list(merged_df[cols[index]]))
    uniqw, inverse = np.unique(x, return_inverse=True)
    y = np.bincount(inverse)
    ii = np.nonzero(y)[0]
    x = list(y[ii])
    y = list(ii)
    axes[int(index/3), (index%3)].pie(x, labels =y, autopct='%1.2f%%')
    axes[int(index/3), (index%3)].title.set_text(cols[index])
```



LINE graph for thigh length, calf length, M-height , Foot Length, Shoe No¶

In [197]:

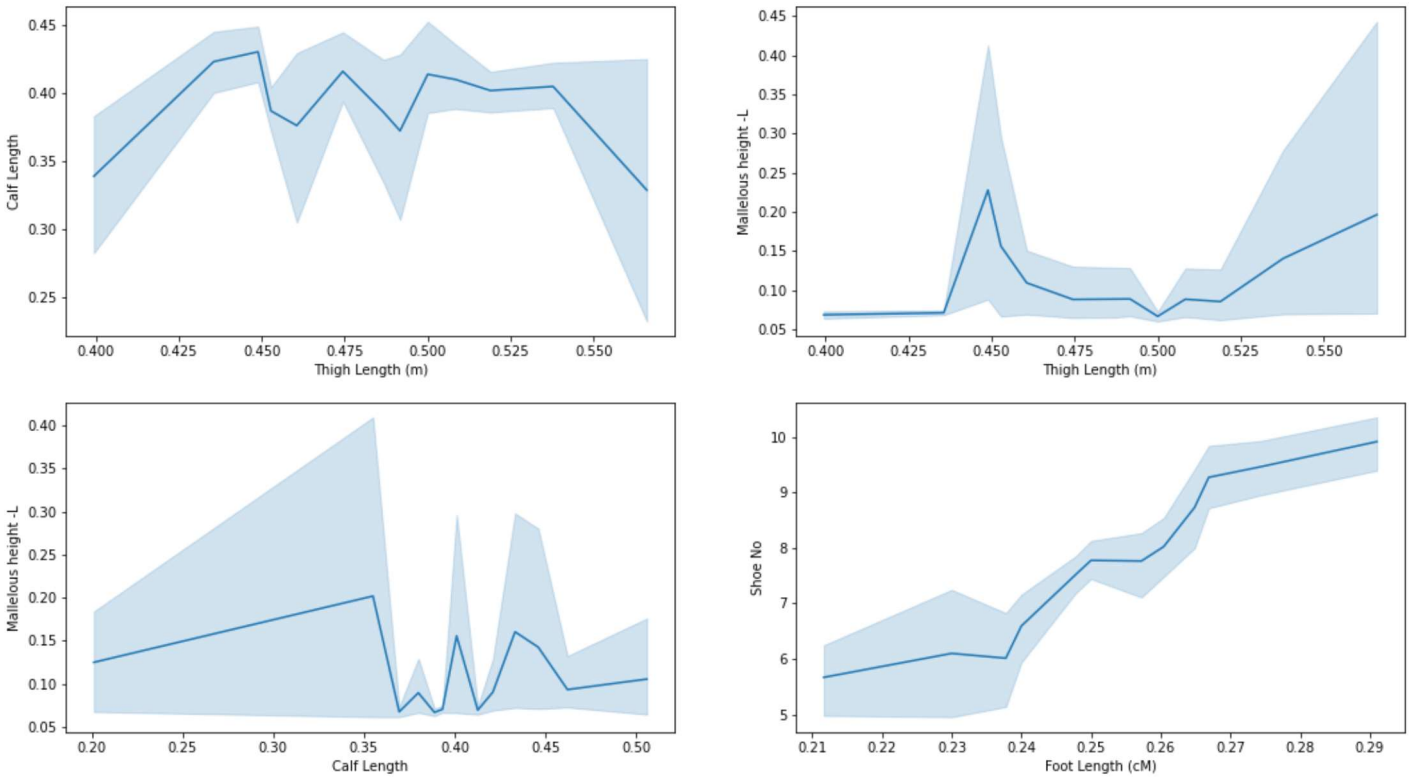
```
cols = ['Thigh Length (m)', 'Calf Length', 'Mallelous height -L', 'Foot Length (cM)', 'Shoe No']

fig, axes = plt.subplots(2, 2, figsize=(18, 10))

#Drawing bar graphs for (Gender w/ every column) and (Age w/ every column)
sns.lineplot(ax=axes[0, 0], x=cols[0], y=cols[1], data=merged_df)
sns.lineplot(ax=axes[0, 1], x=cols[0], y=cols[2], data=merged_df)
sns.lineplot(ax=axes[1, 0], x=cols[1], y=cols[2], data=merged_df)
sns.lineplot(ax=axes[1, 1], x=cols[3], y=cols[4], data=merged_df)
```

Out[197]:

```
<AxesSubplot:xlabel='Foot Length (cM)', ylabel='Shoe No'>
```



LINE graph for Knee_Angle for 4 subejcts in a single plot¶

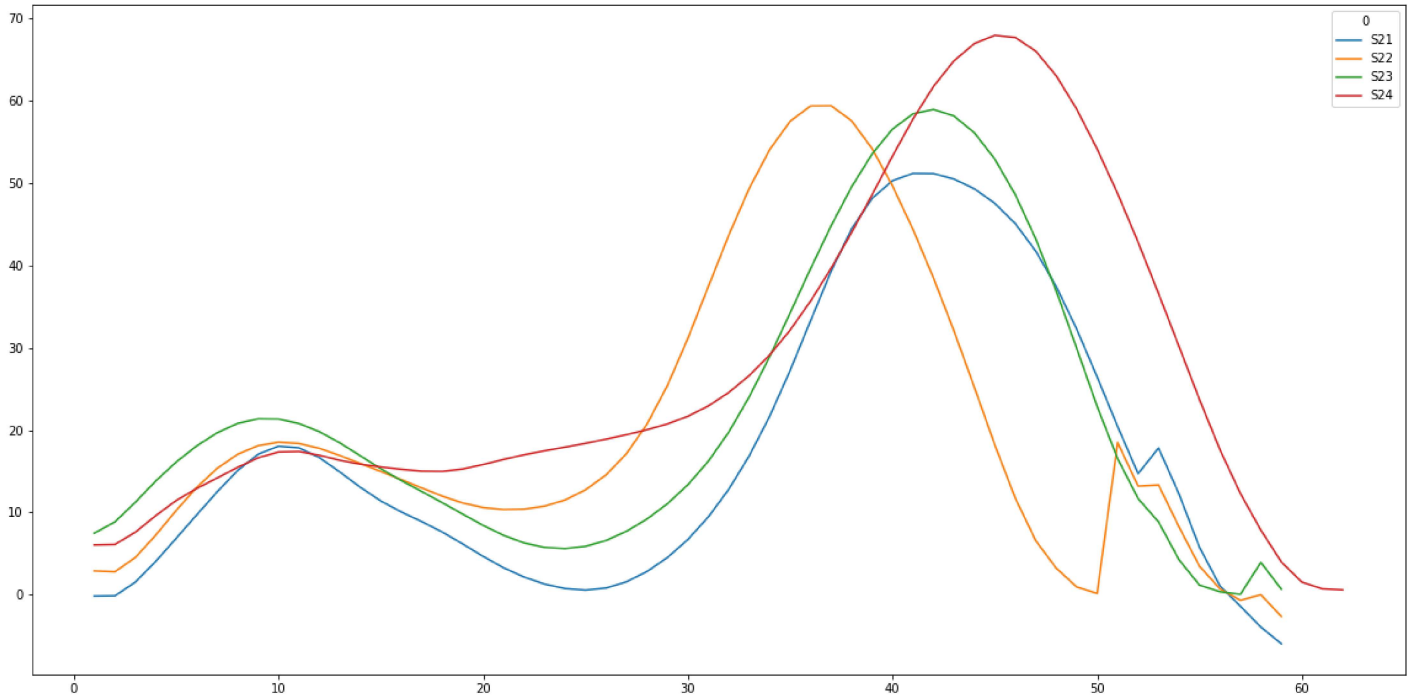
In [198]:

```
fig, axes = plt.subplots(1, 1, figsize=(20, 10))

ka_df = all_sheets[2]["df"].transpose().reset_index().drop(columns=['index'])
ka_df.columns = ka_df.iloc[0]
ka_df = ka_df.iloc[1:]
ka_df.plot(ax=axes)
```

Out[198]:

```
<AxesSubplot:>
```



Histograms of the attributes¶

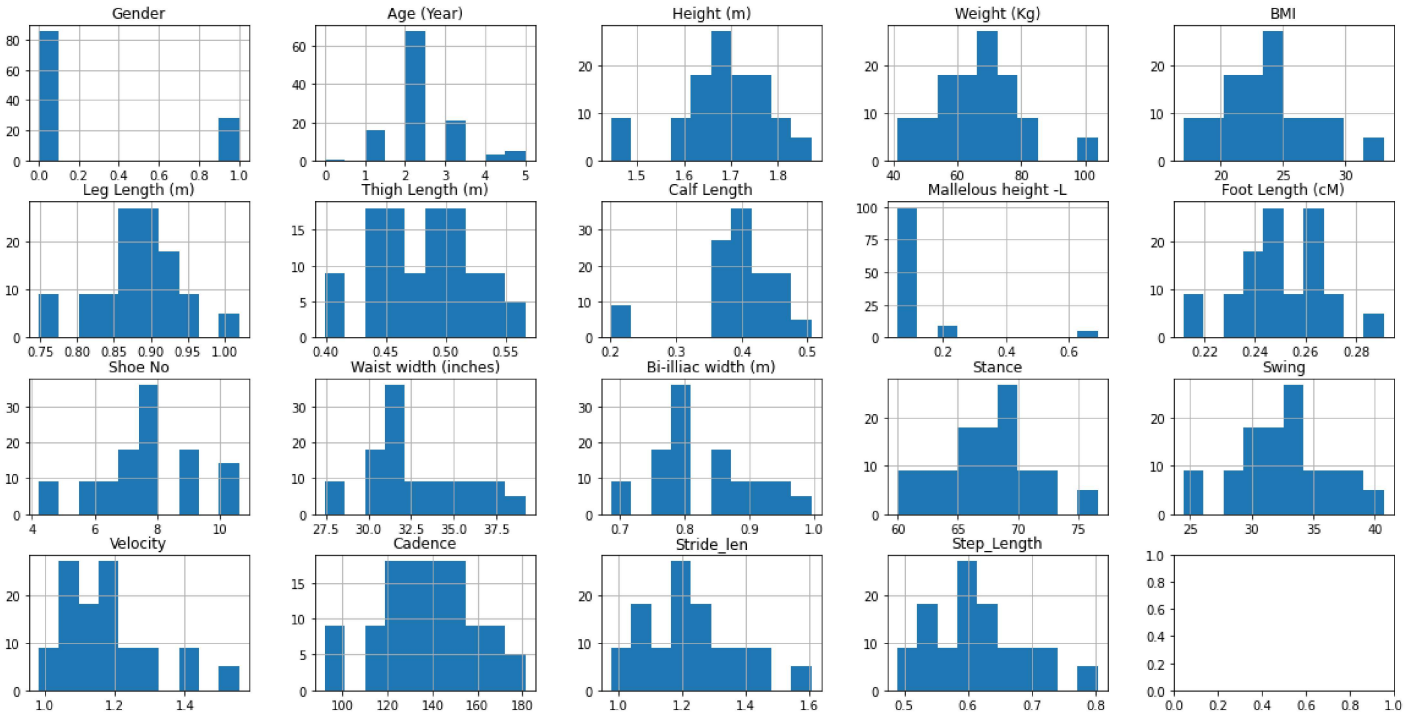
In [199]:

```
fig, axes = plt.subplots(4, 5, figsize=(20, 10))
merged_df.hist(ax=axes.flatten()[:19])
```

Out[199]:

```
array([<AxesSubplot:title={'center':'Gender'}>,
       <AxesSubplot:title={'center':'Age (Year)'}>,
       <AxesSubplot:title={'center':'Height (m)'}>,
       <AxesSubplot:title={'center':'Weight (Kg)'}>,
       <AxesSubplot:title={'center':'BMI'}>,
       <AxesSubplot:title={'center':'Leg Length (m)'}>,
       <AxesSubplot:title={'center':'Thigh Length (m)'}>,
       <AxesSubplot:title={'center':'Calf Length'}>,
       <AxesSubplot:title={'center':'Mallelous height -L'}>,
       <AxesSubplot:title={'center':'Foot Length (cM)'}>,
       <AxesSubplot:title={'center':'Shoe No'}>,
       <AxesSubplot:title={'center':'Waist width (inches)'}>,
       <AxesSubplot:title={'center':'Bi-illiac width (m)'}>,
       <AxesSubplot:title={'center':'Stance'}>,
       <AxesSubplot:title={'center':'Swing'}>,
       <AxesSubplot:title={'center':'Velocity'}>,
       <AxesSubplot:title={'center':'Cadence'}>,
```

```
<AxesSubplot:title={'center':'Stride_len'}>,
<AxesSubplot:title={'center':'Step_Length'}>], dtype=object)
```



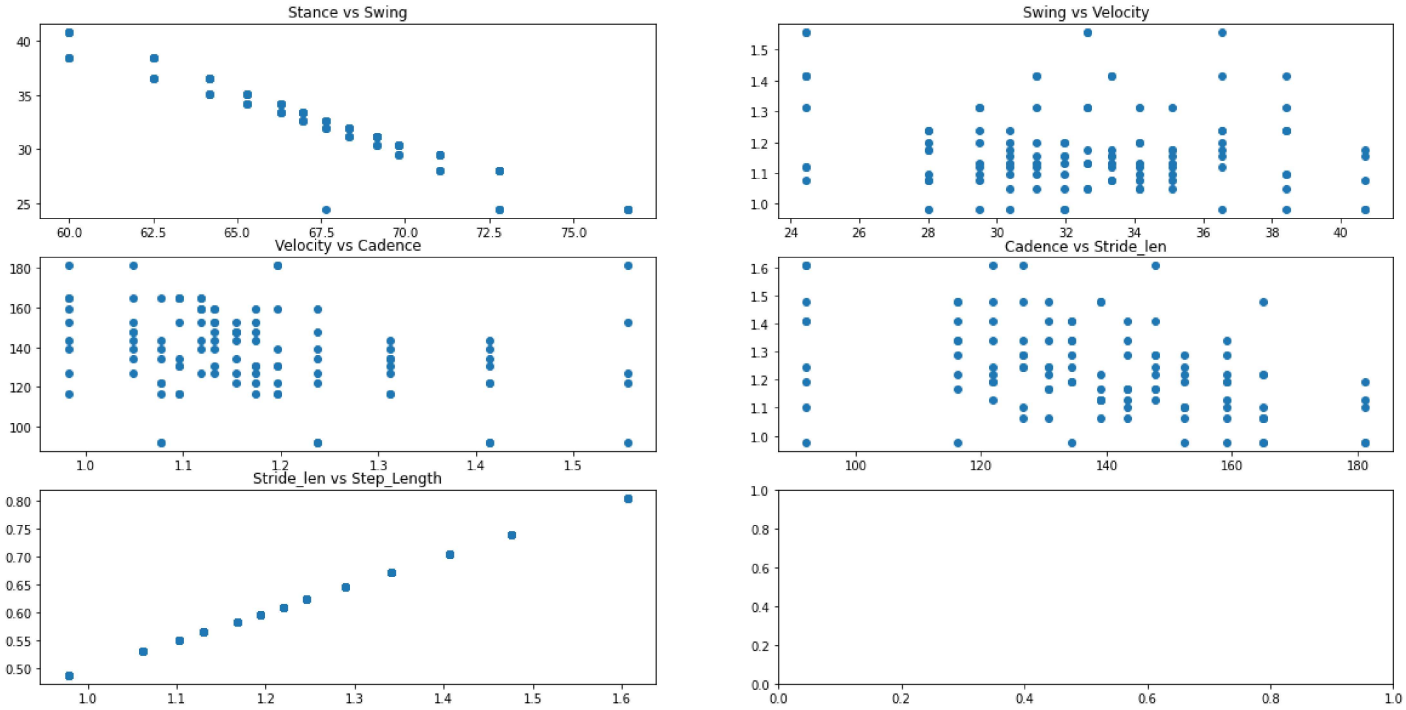
Scatter plots for Stance, Swing, Velocity, Cadence (Step/sec), Stride length , Step_ Length from Spatio-Temporal sheet

In [200]:

```
cols = ['Stance', 'Swing', 'Velocity', 'Cadence', 'Stride_len', 'Step_Length']

fig, axes = plt.subplots(3, 2, figsize=(20, 10))

for x in range(5):
    axes[int(x/2), x%2].scatter(merged_df[cols[x]], merged_df[cols[x+1]])
    axes[int(x/2), x%2].title.set_text(f"{cols[x]} vs {cols[x+1]}")
```



Boxplot for all the attributes :¶

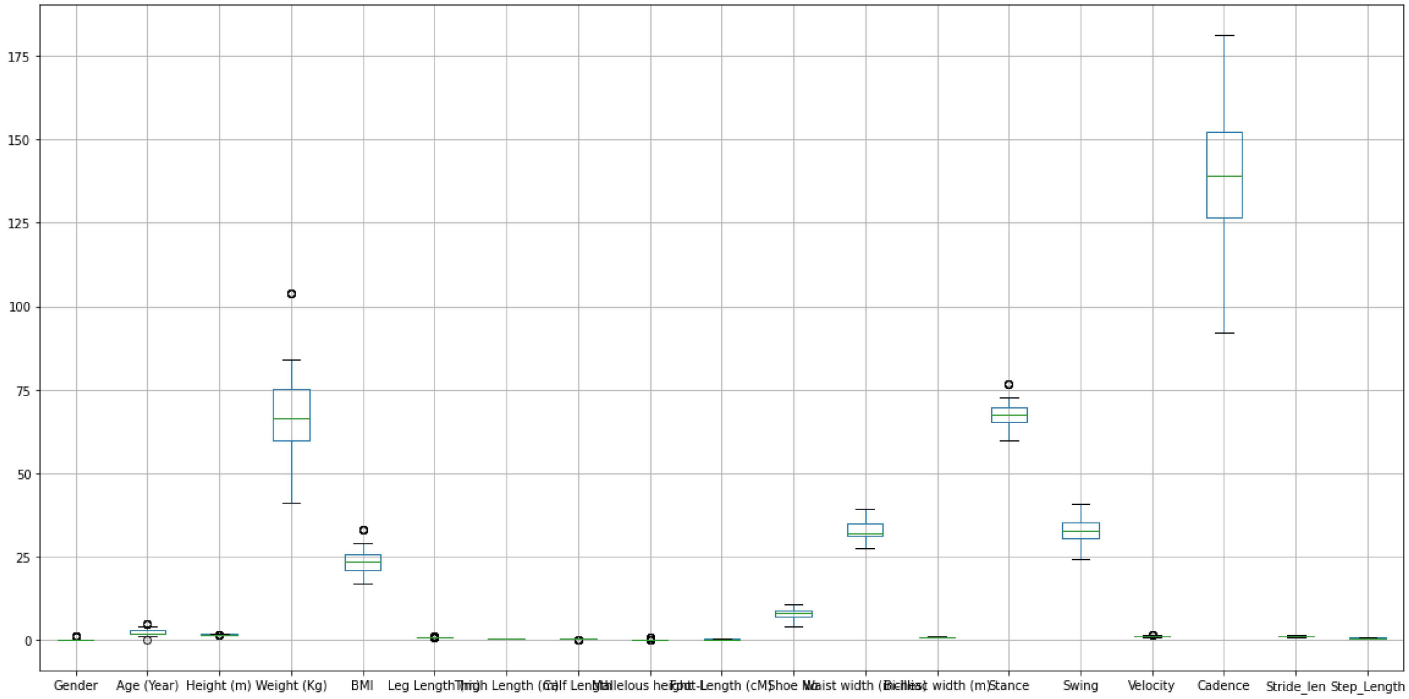
- it is a graphic display of five-number summary

In [201]:

```
fig, axes = plt.subplots(1, 1, figsize=(20, 10))
merged_df.boxplot()
```

Out[201]:

```
<AxesSubplot:>
```



In [202]:

```
## YOUR OBSERVATIONS :
# It is noticed that according to the data provided, men generally have more height, weight and leg l
# The cadence value and the weight are varying quite a lot compared to the other features
```

4.1.8 TASK 8: Data Integration¶

Consider that this dataset is result of joining 3 differnt related datasets.

Handling Redundancy in Data Integration

Redundant attributes may be able to be detected by

- Correlation analysis (X^2 Chi-square test) for Nominal Data
- Correlation coefficient for Numeric Data
- Covariance analysis for Numeric Data

  YOU can write your own function or use python functions

In [203]:

```
#Chi-Square function:
def chi_square(col1, col2):
    chi_val = stats.chisquare(col1, col2)[0]
    if not math.isinf(chi_val):
        return chi_val
    return float('NaN')
```

In [204]:

```
# Correlation analysis ( X^2 Chi-square test )
chi_df = pd.DataFrame(columns=merged_df.columns)
columns = merged_df.columns
for col1 in columns:
  for col2 in columns:
    chi_df.loc[col1,col2] = chi_square(merged_df[col1],merged_df[col2])
chi_df
```

```
c:\users\beeta\appdata\local\programs\python\python39\lib\site-packages\scipy\stats\stats.py:6125: Ru
  terms = (f_obs.astype(np.float64) - f_exp)**2 / f_exp
c:\users\beeta\appdata\local\programs\python\python39\lib\site-packages\scipy\stats\stats.py:6125: Ru
  terms = (f_obs.astype(np.float64) - f_exp)**2 / f_exp
```

Out[204]:

	Gender	Age (Year)	Height (m)	Weight (Kg)	BMI	Leg Length (m)	Thigh Length (
Gender	NaN	NaN	151.822377	7508.608558	2600.712444	77.454492	59.003824
Age (Year)	NaN	NaN	75.638409	7073.993769	2182.997542	327.476947	883.640009
Height (m)	NaN	NaN	0.000000	7189.114698	2289.495286	81.887000	343.363931
Weight (Kg)	NaN	NaN	295928.721564	0.000000	9339.136395	577771.473948	1081452.7721
BMI	NaN	NaN	33015.776467	3197.830246	0.000000	67275.018092	128610.16869
Leg Length (m)	NaN	NaN	42.763028	7365.549620	2459.397223	0.000000	39.165982
Thigh Length (m)	NaN	NaN	97.021972	7455.975669	2548.060823	21.000570	0.000000
Calf Length	NaN	NaN	111.614926	7475.355540	2567.210316	31.182302	3.035958
Mallelous height -L	NaN	NaN	166.909739	7539.111317	2630.557756	78.744809	36.257005
Foot Length (cM)	NaN	NaN	137.455531	7507.462716	2599.011718	51.302252	12.476590
Shoe No	NaN	NaN	2527.914570	5934.365627	1218.878681	6120.164766	12714.107998
Waist width (inches)	NaN	NaN	65858.308606	2010.633187	500.303646	131820.045269	249567.68885
Bi-illiac width (m)	NaN	NaN	48.670928	7376.943106	2470.478769	1.219148	31.376194
Stance	NaN	NaN	292899.336250	367.703737	9898.897462	570126.489428	1065707.0496
Swing	NaN	NaN	65156.410074	2109.152010	609.285133	130698.246593	247528.45828
Velocity	NaN	NaN	18.593989	7300.065266	2395.881295	14.358977	122.161645
Cadence	NaN	NaN	1281040.555856	10869.191518	67675.026343	2463384.166372	4573689.7771
Stride_len	NaN	NaN	15.242759	7287.698403	2383.979464	19.645730	142.313581
Step_Length	NaN	NaN	76.495749	7425.218443	2517.799433	10.100212	6.495871

In [205]:

```
# Correlation Coefficent for Numeric Data
merged_df.corr()
```

Out[205]:

	Gender	Age (Year)	Height (m)	Weight (Kg)	BMI	Leg Length (m)	Thigh Length (m)	Calf Le
Gender	1.000000	-0.270080	-0.631645	-0.387567	-0.114162	-0.411357	-0.202672	-0.4279
Age (Year)	-0.270080	1.000000	0.007530	0.148005	0.224277	0.143621	0.263945	0.0561
Height (m)	-0.631645	0.007530	1.000000	0.537966	0.078583	0.715751	0.455244	0.4455
Weight (Kg)	-0.387567	0.148005	0.537966	1.000000	0.842434	0.368756	0.248700	0.2618
BMI	-0.114162	0.224277	0.078583	0.842434	1.000000	0.094131	0.081957	0.1153
Leg Length (m)	-0.411357	0.143621	0.715751	0.368756	0.094131	1.000000	0.694860	0.5559
Thigh Length (m)	-0.202672	0.263945	0.455244	0.248700	0.081957	0.694860	1.000000	0.0228
Calf Length	-0.427918	0.056121	0.445532	0.261884	0.115311	0.555940	0.022896	1.0000
Mallelous height -L	-0.071567	-0.035997	0.158761	0.082764	0.006662	0.061773	0.048829	-0.0320
Foot Length (cM)	-0.672049	0.060137	0.777211	0.518527	0.181386	0.566508	0.339639	0.4059
Shoe No	-0.588998	-0.003352	0.692169	0.630143	0.365793	0.541575	0.274878	0.3890
Waist width (inches)	-0.330506	0.472595	0.334775	0.669885	0.623799	0.235515	0.197512	0.1649
Bi-illiac width (m)	-0.331314	0.457273	0.346511	0.676490	0.628473	0.245119	0.209913	0.1562
Stance	0.061310	0.061376	-0.100225	-0.011766	0.001665	0.053623	0.039460	0.0309
Swing	-0.049201	-0.105361	0.092023	-0.034015	-0.055549	-0.068879	-0.070801	-0.0091
Velocity	-0.184871	0.194703	-0.020872	0.068363	0.041475	0.013230	-0.033390	0.0164
Cadence	0.068847	-0.130261	-0.054918	-0.071334	-0.079305	-0.013389	-0.011975	-0.0655
Stride_len	-0.182213	0.071359	0.108507	0.010397	-0.054520	0.053677	0.047765	0.1615
Step_Length	-0.182214	0.071359	0.108508	0.010397	-0.054521	0.053677	0.047765	0.1615

In [206]:

```
# Covariance analysis for Numeric Data
merged_df.cov()
```

Out[206]:

	Gender	Age (Year)	Height (m)	Weight (Kg)	BMI	Leg Length (m)	Thigh Length (m)	Calf Le
Gender	0.188053	-0.106669	-0.027189	-2.300528	-0.186001	-0.010840	-0.003604	-0.012
Age (Year)	-0.106669	0.829488	0.000681	1.845109	0.767440	0.007949	0.009857	0.0034
Height (m)	-0.027189	0.000681	0.009853	0.730927	0.029307	0.004317	0.001853	0.0029
Weight (Kg)	-2.300528	1.845109	0.730927	187.361415	43.324190	0.306737	0.139591	0.2417
BMI	-0.186001	0.767440	0.029307	43.324190	14.115897	0.021492	0.012626	0.0292
Leg Length (m)	-0.010840	0.007949	0.004317	0.306737	0.021492	0.003693	0.001732	0.0022
Thigh Length (m)	-0.003604	0.009857	0.001853	0.139591	0.012626	0.001732	0.001681	0.0000
Calf Length	-0.012512	0.003446	0.002982	0.241702	0.029212	0.002278	0.000063	0.0045
Mallelous height -L	-0.004157	-0.004392	0.002111	0.151760	0.003353	0.000503	0.000268	-0.000
Foot Length (cM)	-0.005418	0.001018	0.001434	0.131958	0.012670	0.000640	0.000259	0.0005

	Gender	Age (Year)	Height (m)	Weight (Kg)	BMI	Leg Length (m)	Thigh Length (m)	Calf Le
Shoe No	-0.411961	-0.004923	0.110814	13.911721	2.216622	0.053082	0.018180	0.0423
Waist width (inches)	-0.421601	1.266122	0.097749	26.972572	6.894151	0.042100	0.023824	0.0327
Bi-illiac width (m)	-0.011071	0.032092	0.002650	0.713534	0.181951	0.001148	0.000663	0.0008
Stance	0.105127	0.221027	-0.039336	-0.636805	0.024740	0.012885	0.006398	0.0082
Swing	-0.085442	-0.384274	0.036579	-1.864516	-0.835764	-0.016762	-0.011626	-0.002
Velocity	-0.011018	0.024370	-0.000285	0.128603	0.021415	0.000110	-0.000188	0.0001
Cadence	0.632288	-2.512518	-0.115448	-20.678964	-6.310212	-0.017231	-0.010400	-0.093
Stride_len	-0.012474	0.010259	0.001700	0.022465	-0.032335	0.000515	0.000309	0.0017
Step_Length	-0.006237	0.005130	0.000850	0.011233	-0.016168	0.000257	0.000155	0.0008

4.1.9 TASK 9: Data Reduction Strategies¶

a) Principal Components Analysis (PCA)
b) Attributes Subset Selection

PCA Algorithm :¶

Given N data vectors from n-dimensions, find k □ n orthogonal vectors (principal components) that can be best used to represent data

- Normalize input data: Each attribute falls within the same range
- Compute k orthonormal (unit) vectors, i.e., principal components
- Each input data (vector) is a linear combination of the k principal component vectors
- The principal components are sorted in order of decreasing “significance” or strength
- Since the components are sorted, the size of the data can be reduced by eliminating the weak components, i.e., those with low variance (i.e., using the strongest principal components, it is possible to reconstruct a good approximation of the original data)

In [207]:

```
### Code for the PCA Analysis

#Normalizing the merged_df:
x = merged_df.values
min_max_scaler = preprocessing.MinMaxScaler()
x_scaled = min_max_scaler.fit_transform(x)
normalized_df = pd.DataFrame(x_scaled)
normalized_df.columns = merged_df.columns

#PCA:
pca = PCA(n_components=4)
principalComponents = pca.fit_transform(x_scaled)
principalDf = pd.DataFrame(data = principalComponents, columns = ['principal component 1', 'principal
```

```
#Displaying the principal components (19 cols -> 4cols)
principalDf
```

Out[207]:

	principal component 1	principal component 2	principal component 3	principal component 4
0	-0.181584	0.272377	-0.335334	-0.129696
1	-0.645300	-0.036137	-0.591320	-0.554696
2	0.840721	-0.566596	0.154273	0.250329
3	0.796097	0.251946	0.263341	0.475817
4	-0.102788	-0.495822	-0.266362	0.048043
...	...	...	...	...
108	-0.687696	-0.223543	0.515582	-0.518925
109	-0.152515	-0.082977	-0.276370	-0.355714
110	0.067603	-0.035744	-0.195851	-0.457838
111	-0.096577	-0.421158	-0.283810	-0.478322
112	-0.404317	0.291566	0.423652	-0.062814

113 rows × 4 columns

9 b). Attributes Subset Selection¶

- Redundant attributes
- - Duplicate much or all of the information contained in one or more other attributes
- - E.g., purchase price of a product and the amount of sales tax paid
- Irrelevant attributes
- - Contain no information that is useful for the data mining task at hand

In [208]:

```
#We can remove height and weight because we have BMI as a column that contains a better representatic
try:
    merged_df = merged_df.drop(columns=['Height (m)', 'Weight (Kg)'])
except:
    print("These columns have already been dropped.")
```

In [209]:

```
#We can also remove foot length because every subject is wearing a shoe and their foot length is not
try:
```

```
    merged_df = merged_df.drop(columns=['Foot Length (cM)'])
except:
    print("The column have already been dropped.")

merged_df
```

Out[209]:

	Gender	Age (Year)	BMI	Leg Length (m)	Thigh Length (m)	Calf Length	Mallelous height -L	Shoe No	Wais
0	0	2	20.507301	0.935556	0.518889	0.412778	0.075000	7.888889	31.1
1	0	2	19.543862	1.019000	0.566000	0.506000	0.080000	10.600000	32.0
2	1	2	25.667457	0.816111	0.452778	0.355000	0.690000	4.222222	34.0
3	1	2	24.078185	0.816111	0.491667	0.201111	0.072778	7.888889	30.0
4	0	1	26.926892	0.846667	0.474444	0.369444	0.080000	8.000000	33.2
...	...	...	...	...	...	...	...	...	...
108	0	2	25.667457	0.897778	0.474444	0.421111	0.062778	9.000000	35.8
109	0	2	21.870881	0.962778	0.491667	0.462222	0.070000	8.000000	31.1
110	0	2	23.517249	0.880000	0.491667	0.388889	0.070000	6.666667	30.0
111	0	5	22.661131	0.905556	0.486667	0.412778	0.054444	7.888889	32.0
112	0	4	21.079829	0.935556	0.537778	0.388889	0.062778	7.000000	35.8

113 rows × 16 columns

Attributes Creation¶

Add a column or Row¶

In [210]:

```
# There isn't any column that can be added from the given attributes because they're all fairly unrel
# that we cant find any relation among any two or more columns
```

4.1.10 TASK 10). Data Transformation Methods :¶

- Smoothing: Remove noise from data
- Attribute/feature construction -New attributes constructed from the given ones
- Aggregation: Summarization, data cube construction
- Normalization: Scaled to fall within a smaller, specified range
- - min-max normalization
- - Standard Normalization / Standardization or Z-score Normalization

- normalization by decimal scaling

In [211]:

```
# What is the need of Normalization / Standardizing the data ?
```

Data Normalization is only needed when the range of data or the scale of data is different among all the attributes of a given data set. Normalization makes every data within a certain scale, so it is easier to compare values when the scales are the same foa all the attributes. Comparison between different attributes become much more easier to visualize/analyse.

Apply Normalization on Knee_Angle for 4 subejcts in a single plot.¶

In [212]:

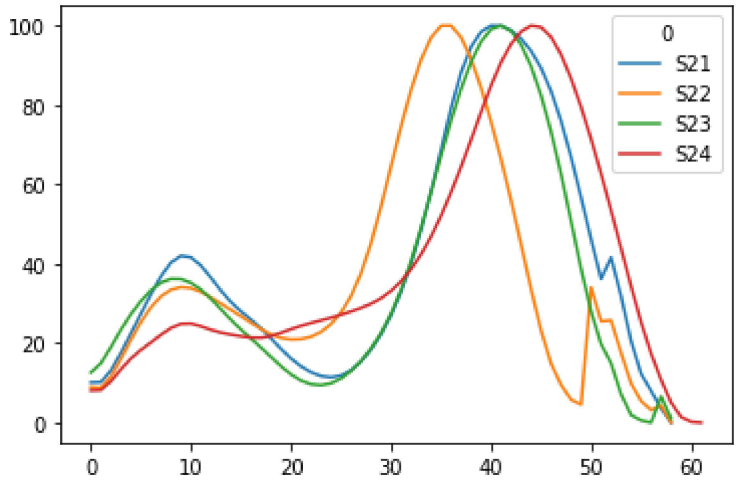
```
#Transposing the given knee_angle dataframe:
ka_df = all_sheets[2]["df"].transpose().reset_index().drop(columns=['index'])
columns = ka_df.iloc[0]
ka_df = ka_df.iloc[1:]

#Normalizing the dataframe:
x = ka_df.values #returns a numpy array
min_max_scaler = preprocessing.MinMaxScaler()
x_scaled = min_max_scaler.fit_transform(x)
ka_df = pd.DataFrame(x_scaled)
ka_df.columns = columns

#Multiplying the dataframe values by 100 to get within a range of 1-100 and displaying:
ka_df=ka_df*100
ka_df.plot()
```

Out[212]:

```
<AxesSubplot:>
```



In [213]:

```
#Displaying the knee_angle dataframe that was normalized to a range of 1-100
ka_df
```

Out[213]:

	S21	S22	S23	S24
0	10.131859	8.932573	12.610472	8.113273
1	10.189980	8.761040	14.928943	8.192551
2	13.105784	11.548713	18.976438	10.386934
3	17.530907	15.956283	23.379659	13.393438
4	22.440214	20.786234	27.298132	16.122458
...	...	...	...	...
57	3.550542	4.258916	6.567746	10.769579
58	0.000000	0.000000	1.039434	5.025882
59	NaN	NaN	NaN	1.410001
60	NaN	NaN	NaN	0.190607
61	NaN	NaN	NaN	0.000000

62 rows × 4 columns

Apply normalization on the attributes of the dataset¶

In [214]:

```
#Normalizing merged_df:
cols = merged_df.columns
x = merged_df.values #returns a numpy array
min_max_scaler = preprocessing.MinMaxScaler()
x_scaled = min_max_scaler.fit_transform(x)
merged_df = pd.DataFrame(x_scaled)
```

```
merged_df.columns = cols
merged_df
```

Out[214]:

	Gender	Age (Year)	BMI	Leg Length (m)	Thigh Length (m)	Calf Length	Mallelous height -L	Shoe No	Waist v
0	0.0	0.4	0.216171	0.691708	0.717145	0.694242	0.032343	0.574913	0.3166
1	0.0	0.4	0.156213	1.000000	1.000000	1.000000	0.040210	1.000000	0.3875
2	1.0	0.4	0.537300	0.250411	0.320213	0.504738	1.000000	0.000000	0.5623
3	1.0	0.4	0.438396	0.250411	0.553702	0.000000	0.028846	0.574913	0.2173
4	0.0	0.2	0.615678	0.363300	0.450300	0.552114	0.040210	0.592334	0.4962
...	...	...	...	...	...	...	...	...	...
108	0.0	0.4	0.537300	0.552135	0.450300	0.721574	0.013112	0.749129	0.7183
109	0.0	0.4	0.301030	0.792282	0.553702	0.856414	0.024476	0.592334	0.3166
110	0.0	0.4	0.403487	0.486453	0.553702	0.615889	0.024476	0.383275	0.2173
111	0.0	1.0	0.350209	0.580870	0.523682	0.694242	0.000000	0.574913	0.3875
112	0.0	0.8	0.251800	0.691708	0.830554	0.615889	0.013112	0.435540	0.7183

113 rows × 16 columns

5 Your Learning and observation :¶

Through this assignment I come to knowabout normalization , attribute subset selection, comparing dataset by histogram piechart and filling missing value etc