# Data Mining and analytics
# Section No.3

2023/2024

## Numpy

- NumPy is a Python library.
- NumPy is used for working with arrays.
- NumPy is short for "Numerical Python".

In [1]:
```
import numpy as np
# From numpy import * ==> All Classes and variabels in numpy
A = np.array([1, 2, 3])
print(A)
```

```
[1 2 3]
```


In [2]:
```
# type(): This built-in Python function tells us the type of the object passed to it.
type(A)
```

Out[2]:
```
numpy.ndarray
```

In [3]:
```
# (dtype) that returns the data type of the array
A.dtype
```

Out[3]:
```
dtype('int32')
```

In [4]:
```
# create an array filled with 0's
np.zeros(10)
```

Out[4]:
```
array([0., 0., 0., 0., 0., 0., 0., 0., 0., 0.])
```

In [5]:
```
# Casting
np.zeros(10,int)
```

Out[5]:
```
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0])
```

In [6]:
```
# Create an array of Zeros 3X4
np.zeros((3,4))
```

Out[6]:
```
array([[0., 0., 0., 0.],
       [0., 0., 0., 0.],
       [0., 0., 0., 0.]])
```

In [7]:
```
# Create an array of Ones 3X4
np.ones((3,4))
```

Out[7]:
```
array([[1., 1., 1., 1.],
       [1., 1., 1., 1.],
       [1., 1., 1., 1.]])
```

In [8]:
```
# Create a constant array # (2 rows, 3 column),number in all array
np.full((2,3), 10)
```

Out[8]:
```
array([[10, 10, 10],
       [10, 10, 10]])
```

In [9]:
```
a=np.arange(1,21)
a
```

Out[9]:
```
array([ 1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16, 17,
       18, 19, 20])
```

In [10]:
```
fnumber=int(input("Enter First Number:"))
lnumber=int(input("Enter Last Number:"))
a=np.arange(fnumber,lnumber+1)
a
```

```
Enter First Number:5
Enter Last Number:30
```

Out[10]:
```
array([ 5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21,
       22, 23, 24, 25, 26, 27, 28, 29, 30])
```

In [11]:
```
# (Start, End, Step)
np.arange(5,30,0.5)
```

Out[11]:
```
array([ 5. ,  5.5,  6. ,  6.5,  7. ,  7.5,  8. ,  8.5,  9. ,  9.5, 10. ,
       10.5, 11. , 11.5, 12. , 12.5, 13. , 13.5, 14. , 14.5, 15. , 15.5,
       16. , 16.5, 17. , 17.5, 18. , 18.5, 19. , 19.5, 20. , 20.5, 21. ,
       21.5, 22. , 22.5, 23. , 23.5, 24. , 24.5, 25. , 25.5, 26. , 26.5,
       27. , 27.5, 28. , 28.5, 29. , 29.5])
```

In [12]:
```
# (Start, End, number of element in array)
np.linspace(5,30,5)
```

Out[12]:
```
array([ 5.  , 11.25, 17.5 , 23.75, 30.  ])
```

**Example : Create a 2-D array containing two arrays with the values 1,2,3 and 4,5,6:**

In [13]:
```
arr = np.array([[1, 2, 3], [4, 5, 6]])
print(arr)
```

```
[[1 2 3]
 [4 5 6]]
```

In [14]:
```
# 3-D array
arr = np.array([[[1, 2, 3], [4, 5, 6]], [[1, 2, 3], [4, 5, 6]]])
print(arr)
```

```
[[[1 2 3]
  [4 5 6]]

 [[1 2 3]
  [4 5 6]]]
```

In [15]:
```
# Check Number of Dimensions?
a = np.array(42)
b = np.array([1, 2, 3, 4, 5])
c = np.array([[1, 2, 3], [4, 5, 6]])
d = np.array([[[1, 2, 3], [4, 5, 6]], [[1, 2, 3], [4, 5, 6]]])
print(a.ndim)
print(b.ndim)
print(c.ndim)
print(d.ndim)
```

```
0
1
2
3
```

In [16]:
```
# Access Array Elements
arr = np.array([1, 2, 3, 4])
print(arr[0])
print(arr[1])
```

```
1
2
```

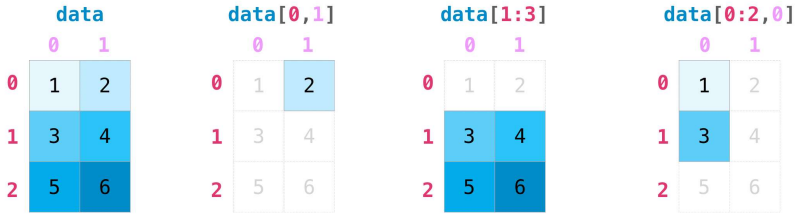
**Example: Get third and fourth elements from the following array and add them.**

In [17]:
```
print(arr[2] + arr[3])
```

```
7
```

In [18]:
```
# Access 2-D Arrays
data= np.array([[1, 2], [3, 4], [5, 6]])
print (data)
```

```
[[1 2]
 [3 4]
 [5 6]]
```


In [19]:
```
print('2nd element on 1st row: ', data[0, 1])
```

```
2nd element on 1st row:  2
```

In [20]:
```
data[1:3]
```

Out[20]:
```
array([[3, 4],
       [5, 6]])
```

In [21]:
```
data[0:2, 0]
```

Out[21]:
```
array([1, 3])
```

In [22]:
```
a=data[:,1]=0
a
```

Out[22]:
```
0
```

In [23]:
```
data
```

Out[23]:
```
array([[1, 0],
       [3, 0],
       [5, 0]])
```

In [24]:
```
a= data[:,1:3]=1
a
```

Out[24]:
```
1
```

In [25]:
```
data
```

Out[25]:
```
array([[1, 1],
       [3, 1],
       [5, 1]])
```

In [26]:
```
a= data[-1,:]=10
a
```

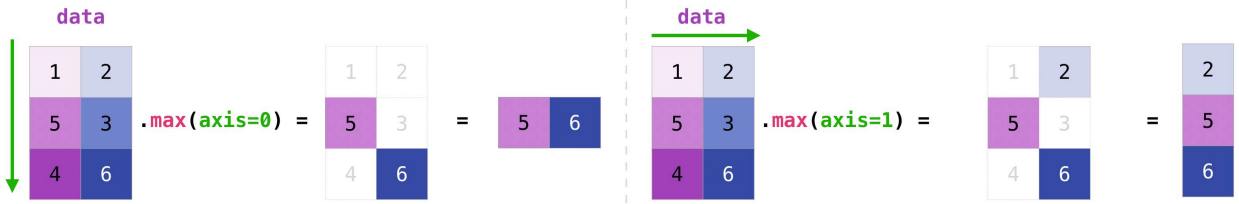
Out[26]:
```
10
```

In [27]:
```
data
```

Out[27]:
```
array([[ 1,  1],
       [ 3,  1],
       [10, 10]])
```

In [28]:
```
data= np.array([[1, 2], [3, 4], [5, 6]])
print (data)
```

```
[[1 2]
 [3 4]
 [5 6]]
```




In [29]:
```
data.max()
```

Out[29]: 6

In [30]:
```
data.max(axis=0)
```

Out[30]: array([5, 6])

In [31]:
```
data.max(axis=1)
```

Out[31]: array([2, 4, 6])

In [32]:
```
data.min()
```

Out[32]: 1

In [33]:
```
data.sum()
```

Out[33]: 21

In [34]:
```
# Access 3-D Arrays
arr2 = np.array([[[1, 2, 3], [4, 5, 6]], [[7, 8, 9], [10, 11, 12]]])
print (arr2)
```

```
[[[ 1  2  3]
  [ 4  5  6]]

 [[ 7  8  9]
  [10 11 12]]]
```

In [35]:
```
print(arr2[0 , 1 , 2])
# [0 # Number of array, 1 # Number of Row , 2 # Number of Column])
```

6

**How do you know the shape and size of an array?**

In [36]:
```
data2 = np.array([[1, 2, 3, 4, 5, 6]])
print(data2)
```

```
[[1 2 3 4 5 6]]
```

In [37]:
```
# shape of array
data2.shape
```

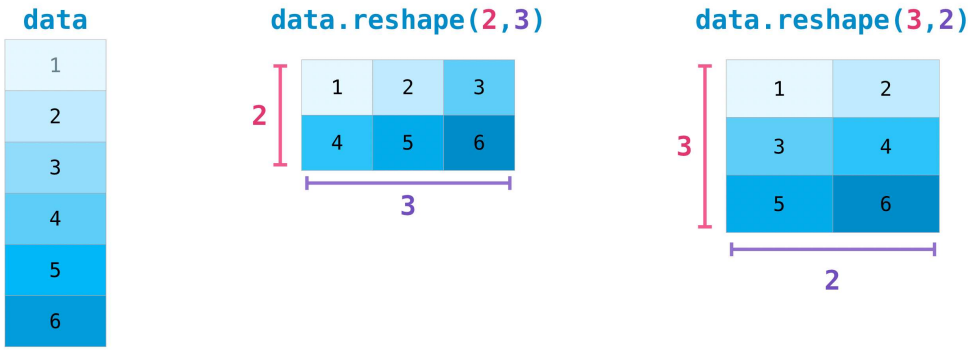
Out[37]: (1, 6)

In [38]:
```
# Number of element of array 2*5
data2.size
```

Out[38]: 6

In [39]:
```
# change size of element of array
np.resize(data2, (4,5))
```

Out[39]:
```
array([[1, 2, 3, 4, 5],
       [6, 1, 2, 3, 4],
       [5, 6, 1, 2, 3],
       [4, 5, 6, 1, 2]])
```




In [40]:
```
# give a new shape to an array without changing the data.
np.reshape(data2,(2,3))
```

Out[40]:
```
array([[1, 2, 3],
       [4, 5, 6]])
```

In [41]:
```
# give a new shape to an array without changing the data.
np.reshape(data2,(2,3))
```

Out[41]:
```
array([[1, 2, 3],
       [4, 5, 6]])
```

In [42]:
```
array = np.array([[[1, 2, 3, 4, 5], [6, 7, 8, 9, 10]]])
print(array)
```

```
[[[ 1  2  3  4  5]
  [ 6  7  8  9 10]]]
```

In [43]:
```
# Convert any dimantion to one dim
array.ravel()
```

Out[43]:
```
array([ 1,  2,  3,  4,  5,  6,  7,  8,  9, 10])
```

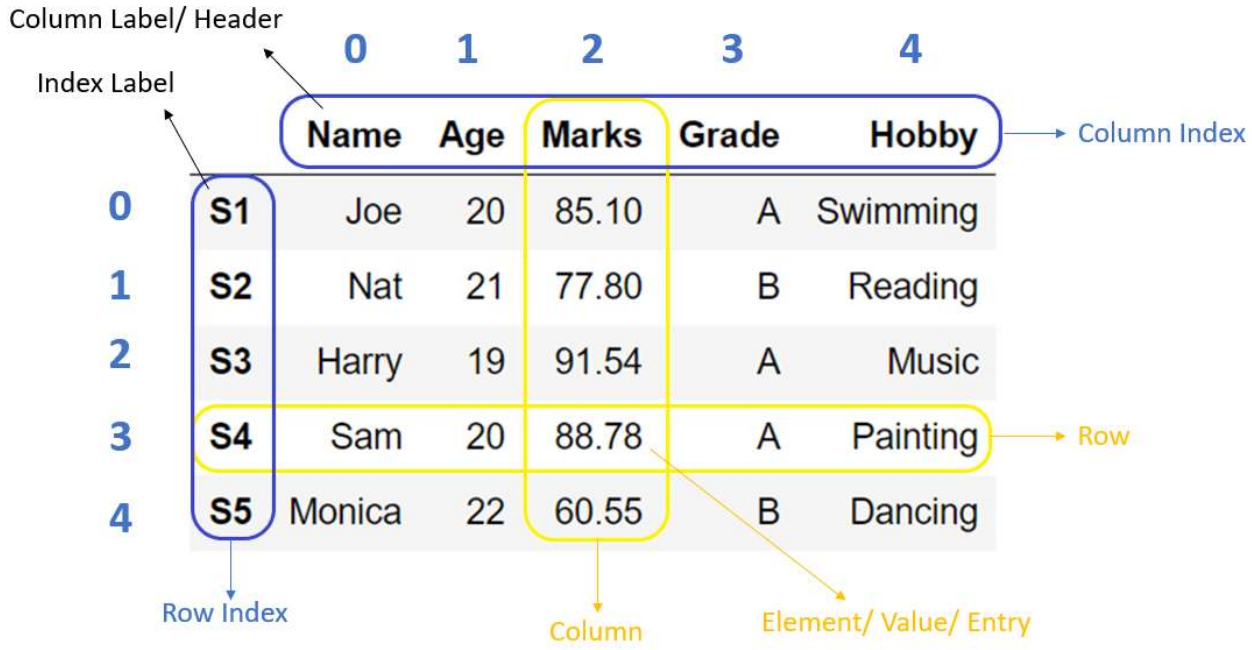
# Pandas

- Pandas is a Python library used for working with data sets.
- It has functions for analyzing, cleaning, exploring, and manipulating data.
- The name "Pandas" has a reference to both "Panel Data", and "Python Data Analysis" and was created by Wes McKinney in 2008.

Download Pandas: Pandas is included with Anaconda. If you don't already have Anaconda installed on your computer. Pandas Documentation: https://pandas.pydata.org/pandas-docs/stable/ (https://pandas.pydata.org/pandas-docs/stable/)

**Why Use pandas?** The recent success of machine learning algorithms is partly due to the huge amounts of data that we have available to train our algorithms on. However, when it comes to data, quantity is not the only thing that matters, the quality of your data is just as important. It often happens that large datasets don't come ready to be fed into your learning algorithms. More often than not, large datasets will often have missing values, outliers, incorrect values, etc... Having data with a lot of missing or bad values, for example, is not going to allow your machine learning algorithms to perform well. Therefore, one very important step in machine learning is to look at your data first and make sure it is well suited for your training algorithm by doing some basic data analysis. This is where Pandas come in. Pandas Series and DataFrames are designed for fast data analysis and manipulation, as well as being flexible and easy to use. Below are just a few features that makes Pandas an excellent package for data analysis:

- Allows the use of labels for rows and columns
- Can calculate rolling statistics on time series data
- Easy handling of NaN values
- Is able to load data of different formats into DataFrames
- Can join and merge different datasets together
- It integrates with NumPy and Matplotlib For these and other reasons, Pandas DataFrames have become one of the most commonly used Pandas object for data analysis in Python.




In [44]:
```
import pandas as pd
# From pandas import *
# From pandas import pd
```

In [45]:
```
# A Pandas Series is a one-dimensional array of indexed data
x = pd.Series([20, 35, 25, 33, 40], name="Age")
x
```

Out[45]:
```
0    20
1    35
2    25
3    33
4    40
Name: Age, dtype: int64
```

In [46]:
```
print(x[0])
print(x[3])
```

```
20
33
```

In [47]:
```
x.index
```

Out[47]:
```
RangeIndex(start=0, stop=5, step=1)
```

In [48]:
```
# With the (index) argument, you can name your own labels.
y = pd.Series([20, 35, 25, 33, 40], name ='age', index=['a', 'b', 'c', 'd', 'e'])
print(y)
```

```
a    20
b    35
c    25
d    33
e    40
Name: age, dtype: int64
```

In [49]:
```
print( y['a'])
```

```
20
```

In [50]:
```
y.index
```

Out[50]:
```
Index(['a', 'b', 'c', 'd', 'e'], dtype='object')
```

In [51]:
```
# Series-as-dictionary
population = pd.Series({'California': 38332521,
                        'Texas': 26448193,
                        'New York': 19651127,
                        'Florida': 19552860,
                        'Illinois': 12882135})
population
```

Out[51]:
```
California    38332521
Texas         26448193
New York      19651127
Florida       19552860
Illinois      12882135
dtype: int64
```

In [52]:
```
population['California']
```

Out[52]:
```
38332521
```

In [54]:
```
# A Pandas DataFrame
df = pd.DataFrame({"calories": [420, 380, 390, 400],
                   "duration": [50, 40, 45,30]})
print(df)
```

```
   calories  duration
0       420        50
1       380        40
2       390        45
3       400        30
```

## Accessors

A more efficient and more programmatically reusable method of accessing data in a DataFrame is by using accessors

- loc - accesses using lables
- iloc - accesses using index positions

In [55]:
```
# Using the .loc accessor
df.loc[1, 'calories']
```

Out[55]:
```
380
```

In [56]:
```
df.iloc[0] # First row of a data frame
```

Out[56]:
```
calories    420
duration     50
Name: 0, dtype: int64
```

In [57]:
```
df.iloc[-1] # Last row
```

Out[57]:
```
calories    400
duration     30
Name: 3, dtype: int64
```

In [58]:
```
df.iloc[:, 0] # First column
```

Out[58]:
```
0    420
1    380
2    390
3    400
Name: calories, dtype: int64
```

In [59]:
```
df.iloc[:, -1] # Last column
```

Out[59]:
```
0    50
1    40
2    45
3    30
Name: duration, dtype: int64
```

In [60]:
```
df.iloc[0:2] #First 2 rows
```

Out[60]:

| | calories | duration |
|---|---|---|
| **0** | 420 | 50 |
| **1** | 380 | 40 |

In [61]:
```
df.iloc[:, 0:2] # First 2 columns
```

Out[61]:

| | calories | duration |
|---|---|---|
| **0** | 420 | 50 |
| **1** | 380 | 40 |
| **2** | 390 | 45 |
| **3** | 400 | 30 |

In [62]:
```
df.iloc[1:3, 0:2] # Second through third rows and first 2 columns
```

Out[62]:

| | calories | duration |
|---|---|---|
| **1** | 380 | 40 |
| **2** | 390 | 45 |

**Read the CSV file into a pandas DataFrame**

In [63]:
```
import pandas as pd

# Read the CSV file into a Pandas DataFrame
dfff = pd.read_csv(r"F:\Job\ANU\Data Mining\cargame.csv")

# Display the DataFrame
print(dfff)
```

```
    Gender Name of Car  Pull Distance  Distance  Time
0        F         Car           11.8      53.0  3.57
1        F         Car            1.9      15.7  2.65
2        F         Car            3.9      31.0  2.30
3        F         Car           10.2      22.8  5.20
4        F         Car            5.0       6.0  2.53
..     ...         ...            ...       ...   ...
320      M       Racer            7.0      86.0  3.45
321      M       Racer            9.0      74.0  2.73
322      M       Racer           14.0     101.0  3.82
323      M       Racer           13.0      70.0  3.46
324      M       Racer           13.0      58.0  2.21

[325 rows x 5 columns]
```

## Descriptive Statistics

Let us now understand the functions under Descriptive Statistics in Python Pandas. The following table list down the important functions.

1. count() Number of non-null observations
2. sum() Sum of values
3. mean() Mean of Values
4. median() Median of Values
5. mode() Mode of values
6. std() Standard Deviation of the Values
7. min() Minimum Value
8. max() Maximum Value
9. abs() Absolute Value
10. prod() Product of Values
11. cumsum() Cumulative Sum
12. cumprod() Cumulative Product
13. quantile() function return values at the given quantile over requested axis

## Pandas - Analyzing DataFrames

In [64]:
```
# if the number of rows is not specified, the head() method will return the top 5 rows.
print(dfff.head())
```

```
  Gender Name of Car  Pull Distance  Distance  Time
0      F         Car           11.8      53.0  3.57
1      F         Car            1.9      15.7  2.65
2      F         Car            3.9      31.0  2.30
3      F         Car           10.2      22.8  5.20
4      F         Car            5.0       6.0  2.53
```

In [65]:
```
print(dfff.head(10))
```

```
  Gender Name of Car  Pull Distance  Distance  Time
0      F         Car           11.8      53.0  3.57
1      F         Car            1.9      15.7  2.65
2      F         Car            3.9      31.0  2.30
3      F         Car           10.2      22.8  5.20
4      F         Car            5.0       6.0  2.53
5      F         Car           10.2      26.0  2.93
6      F         Car            9.0      49.0  3.10
7      F         Car           12.0      27.3  3.21
8      M       Racer            6.0      35.0  4.40
9      M       Racer            8.0      50.0  3.02
```

In [66]:
```
# Print the last 5 rows of the DataFrame
print(dfff.tail())
```

```
    Gender Name of Car  Pull Distance  Distance  Time
320      M       Racer            7.0      86.0  3.45
321      M       Racer            9.0      74.0  2.73
322      M       Racer           14.0     101.0  3.82
```

```
323      M       Racer           13.0       70.0  3.46
324      M       Racer           13.0       58.0  2.21
```

In [67]:
```
print(dfff.tail(10))
```

```
    Gender Name of Car  Pull Distance  Distance  Time
315      M       Racer           12.0      18.0  1.76
316      M       Racer           15.0      44.0  2.58
317      M       Racer           12.0      10.0  1.76
318      M       Racer           10.0      28.0  2.11
319      M       Racer            7.0      64.0  3.11
320      M       Racer            7.0      86.0  3.45
321      M       Racer            9.0      74.0  2.73
322      M       Racer           14.0     101.0  3.82
323      M       Racer           13.0      70.0  3.46
324      M       Racer           13.0      58.0  2.21
```

In [68]:
```
dfff.sample()
```

Out[68]:

| | Gender | Name of Car | Pull Distance | Distance | Time |
|---|---|---|---|---|---|
| **178** | M | Racer | 7.0 | 60.96 | 1.83 |

In [69]:
```
dfff.sample(5)
```

Out[69]:

| | Gender | Name of Car | Pull Distance | Distance | Time |
|---|---|---|---|---|---|
| **240** | M | Taxi | 5.0 | 19.0 | 2.10 |
| **313** | M | Racer | 11.0 | 115.0 | 3.86 |
| **44** | M | Taxi | 8.0 | 40.0 | 5.00 |
| **113** | F | Truck | 10.0 | 0.0 | 0.00 |
| **108** | F | Truck | 27.0 | 85.0 | 5.00 |

In [70]:
```
# The DataFrames object has a method called info(), that gives you more information about the data set.
print(dfff.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 325 entries, 0 to 324
Data columns (total 5 columns):
 #   Column         Non-Null Count  Dtype  
---  ------         --------------  -----  
 0   Gender         325 non-null    object 
 1   Name of Car    325 non-null    object 
 2   Pull Distance  325 non-null    float64
 3   Distance       325 non-null    float64
 4   Time           325 non-null    float64
dtypes: float64(3), object(2)
memory usage: 12.8+ KB
None
```

## Descriptive statistics with **describe()** function

Descriptive or summary statistics in python:– pandas, can be obtained by using the describe() function.

The describe() function gives us the count, mean, standard deviation(std), minimum, Q1(25%), median(50%), Q3(75%), IQR(Q3 - Q1) and maximum values.

In [71]:
```
print(dfff.describe())
```

```
       Pull Distance    Distance        Time
count     325.000000  325.000000  325.000000
mean       10.872308   56.269046    3.586154
std         5.625145   41.092132    1.576155
min         1.000000    0.000000    0.000000
25%         7.000000   29.000000    2.500000
50%        10.000000   51.000000    3.210000
75%        14.000000   78.000000    4.560000
max        40.000000  390.000000   11.000000
```

In [72]:
```
dfff.describe(include='number')
```

Out[72]:

| | Pull Distance | Distance | Time |
|---|---|---|---|
| **count** | 325.000000 | 325.000000 | 325.000000 |
| **mean** | 10.872308 | 56.269046 | 3.586154 |
| **std** | 5.625145 | 41.092132 | 1.576155 |
| **min** | 1.000000 | 0.000000 | 0.000000 |
| **25%** | 7.000000 | 29.000000 | 2.500000 |
| **50%** | 10.000000 | 51.000000 | 3.210000 |

| | Pull Distance | Distance | Time |
|---|---|---|---|
| **75%** | 14.000000 | 78.000000 | 4.560000 |
| **max** | 40.000000 | 390.000000 | 11.000000 |

In [73]:
```
dfff.describe(include=['object'])
```

Out[73]:

| | Gender | Name of Car |
|---|---|---|
| **count** | 325 | 325 |
| **unique** | 2 | 4 |
| **top** | M | Taxi |
| **freq** | 174 | 89 |

In [74]:
```
dfff.describe(include='all')
```

Out[74]:

| | Gender | Name of Car | Pull Distance | Distance | Time |
|---|---|---|---|---|---|
| **count** | 325 | 325 | 325.000000 | 325.000000 | 325.000000 |
| **unique** | 2 | 4 | NaN | NaN | NaN |
| **top** | M | Taxi | NaN | NaN | NaN |
| **freq** | 174 | 89 | NaN | NaN | NaN |
| **mean** | NaN | NaN | 10.872308 | 56.269046 | 3.586154 |
| **std** | NaN | NaN | 5.625145 | 41.092132 | 1.576155 |
| **min** | NaN | NaN | 1.000000 | 0.000000 | 0.000000 |
| **25%** | NaN | NaN | 7.000000 | 29.000000 | 2.500000 |
| **50%** | NaN | NaN | 10.000000 | 51.000000 | 3.210000 |
| **75%** | NaN | NaN | 14.000000 | 78.000000 | 4.560000 |
| **max** | NaN | NaN | 40.000000 | 390.000000 | 11.000000 |

# Data Cleaning

Data cleaning is the process of changing or eliminating garbage, incorrect, duplicate, corrupted, or incomplete data in a dataset.

Data cleaning plays an important part in developing reliable answers within the analytical process and is observed to be a basic feature of the info science basics.

In [75]:
```
df = pd.read_csv(r"F:\Job\ANU\Data Mining/data.csv")
df
```

Out[75]:

| | Duration | Pulse | Maxpulse | Calories | date |
|---|---|---|---|---|---|
| **0** | 60 | 110.0 | 130 | 409.1 | '2020/12/01' |
| **1** | 60 | 117.0 | 145 | 479.0 | '2020/12/01' |
| **2** | 60 | 117.0 | 145 | 479.0 | '2020/12/01' |
| **3** | 45 | 109.0 | 175 | 282.4 | '2020/12/01' |
| **4** | 45 | 117.0 | 148 | 406.0 | '2020/12/01' |
| **...** | ... | ... | ... | ... | ... |
| **71** | 60 | NaN | 153 | 387.6 | '2020/12/29' |
| **72** | 90 | 100.0 | 127 | 700.0 | '2020/12/29' |
| **73** | 150 | 97.0 | 127 | 953.2 | '2020/12/29' |
| **74** | 45 | 114.0 | 146 | 304.0 | '2020/12/29' |
| **75** | 90 | 98.0 | 125 | 563.2 | '2020/12/29' |

76 rows × 5 columns

In [76]:
```
# to print all dataset
print(df.to_string())
```

```
    Duration  Pulse  Maxpulse  Calories          date
0         60  110.0       130     409.1  '2020/12/01'
1         60  117.0       145     479.0  '2020/12/01'
2         60  117.0       145     479.0  '2020/12/01'
3         45  109.0       175     282.4  '2020/12/01'
```

```
4         45  117.0       148     406.0  '2020/12/01'
5        450  102.0       127     300.0  '2020/12/01'
6         60  110.0       136     374.0  '2020/12/01'
7         45  104.0       134     253.3  '2020/12/01'
8         30  109.0       133     195.1  '2020/12/01'
9         30  109.0       133     195.1  '2020/12/01'
10        60  103.0       147     329.3  '2020/12/01'
11        60  100.0       120     250.7  '2020/12/01'
12        60  106.0       128     345.3  '2020/12/01'
13        60  104.0       132     379.3  '2020/12/01'
14        60   98.0       123     275.0  '2020/12/01'
15        60   98.0       120     215.2  '2020/12/01'
16        60  100.0       120     300.0  '2020/12/01'
17        45   90.0       112       NaN  '2020/12/01'
18        60    NaN       123     323.0  '2020/12/01'
19        45   97.0       125     243.0  '2020/12/01'
20        60  108.0       131     364.2  '2020/12/01'
21        45  100.0       119     282.0  '2020/12/01'
22        60  130.0       101     300.0  '2020/12/01'
23        45  105.0       132     246.0  '2020/12/01'
24        60  102.0       126     334.5  '2020/12/01'
25        60  100.0       120     250.0  '2020/12/01'
26        60   92.0       118     241.0  '2020/12/01'
27        60  103.0       132       NaN  '2020/12/01'
28        60  100.0       132     280.0  '2020/12/01'
29        60  102.0       129     380.3  '2020/12/01'
30        60   92.0       115     243.0  '2020/12/01'
31        45   90.0       112     180.1  '2020/12/01'
32        60  101.0       124     299.0  '2020/12/01'
33        60   93.0       113     223.0  '2020/12/01'
34        60  107.0       136     361.0  '2020/12/01'
35        60  114.0       140     415.0  '2020/12/01'
36        60  102.0       127     300.0  '2020/12/01'
37        60  100.0       120     300.0  '2020/12/01'
38        60  100.0       120     300.0  '2020/12/01'
39        45  104.0       129     266.0  '2020/12/01'
40        45   90.0       112     180.1  '2020/12/01'
41        60   98.0       126     286.0  '2020/12/05'
42        60  100.0       122     329.4  '2020/12/06'
43        60  111.0       138     400.0  '2020/12/01'
44        60  111.0       131     397.0  '2020/12/01'
45        60   99.0       119     273.0  '2020/12/01'
46        60  109.0       153     387.6  '2020/12/01'
47        45  111.0       136     300.0  '2020/12/01'
48        45  108.0       129     298.0  '2020/12/01'
49        60  111.0       139     397.6  '2020/12/01'
50        60  107.0       136     380.2  '2020/12/01'
51        80  123.0       146     643.1  '2020/12/01'
52        60  106.0       130     263.0  '2020/12/01'
53        60  118.0       151     486.0  '2020/12/01'
54        30  136.0       175     238.0  '2020/12/03'
55        60  121.0       146     450.7  '2020/12/01'
56        60  118.0       121     413.0   2020/12/08'
57        45  115.0       144     305.0          NaN
58        20  153.0       172     226.4  '2020/12/08'
59        45  123.0       152     321.0  '2020/12/08'
60       210  108.0       160    1376.0     20201226
61       160    NaN       137    1034.4     20201226
62       160  109.0       135     853.0     20201226
63        45  118.0       141     341.0     20201226
64        20  110.0       130     131.4  '2020/12/08'
65       180   90.0       130     800.4  '2020/12/08'
66       150    NaN       135     873.4  '2020/12/08'
67       150  107.0       130     816.0  '2020/12/29'
68        20  106.0       136     110.4  '2020/12/29'
69       300  108.0       143    1500.2  '2020/12/29'
70       150   97.0       129    1115.0  '2020/12/29'
71        60    NaN       153     387.6  '2020/12/29'
72        90  100.0       127     700.0  '2020/12/29'
73       150   97.0       127     953.2  '2020/12/29'
74        45  114.0       146     304.0  '2020/12/29'
75        90   98.0       125     563.2  '2020/12/29'
```

In [77]:

```
# Checking for any missing value
df.isnull()
```

Out[77]:

| | Duration | Pulse | Maxpulse | Calories | date |
|---|---|---|---|---|---|
| **0** | False | False | False | False | False |
| **1** | False | False | False | False | False |
| **2** | False | False | False | False | False |
| **3** | False | False | False | False | False |
| **4** | False | False | False | False | False |
| **...** | ... | ... | ... | ... | ... |
| **71** | False | True | False | False | False |
| **72** | False | False | False | False | False |
| **73** | False | False | False | False | False |
| **74** | False | False | False | False | False |
| **75** | False | False | False | False | False |

76 rows × 5 columns

In [78]:
```
#or
df.isnull().sum()
```

Out[78]:
```
Duration    0
Pulse       4
Maxpulse    0
Calories    2
date        1
dtype: int64
```

In [79]:
```
# knowing how many missing values in the data
df.isnull().sum().sum()
```

Out[79]: 7

In [80]:
```
# Removing missing values from the data set

new_df = df.dropna()
#df.dropna(inplace = True)
```

In [81]:
```
new_df.isnull().sum()
```

Out[81]:
```
Duration    0
Pulse       0
Maxpulse    0
Calories    0
date        0
dtype: int64
```

In [82]:
```
### drop rows with null value
df.dropna(subset=['Pulse'],inplace=True)
```

In [83]:
```
df.isnull().sum()
```

Out[83]:
```
Duration    0
Pulse       0
Maxpulse    0
Calories    2
date        1
dtype: int64
```

In [84]:
```
### Filling the missing values
# Replace Only For Specified Columns
x_mean = df['Calories'].mean()

df['Calories'].fillna(x_mean,inplace=True)
```

In [85]:
```
df.isnull().sum()
```

Out[85]:
```
Duration    0
Pulse       0
Maxpulse    0
Calories    0
date        1
dtype: int64
```

In [86]:
```
#Replace For All Columns
df.fillna(0,inplace=True) # 0 or 'Unknown'
```

In [87]:
```
df.isnull().sum()
```

Out[87]:
```
Duration    0
Pulse       0
Maxpulse    0
Calories    0
date        0
dtype: int64
```

**Forward fill, fills the missing value with the values above it.**

- df.fillna(method="ffill")

**Back fill, fills the missing value with the values below it.**

- df.fillna(method="bfill")

**Interploation finds the average for the above and below value and uses the value to fill the missing value**

- df.interpolate()

In [88]:
```
## Duplicates
print(df.duplicated().to_string())
```

```
0     False
1     False
2      True
3     False
4     False
5     False
6     False
7     False
8     False
9      True
10    False
11    False
12    False
13    False
14    False
15    False
16    False
17    False
19    False
20    False
21    False
22    False
23    False
24    False
25    False
26    False
27    False
28    False
29    False
30    False
31    False
32    False
33    False
34    False
35    False
36    False
37     True
38     True
39    False
40     True
41    False
42    False
43    False
44    False
45    False
46    False
47    False
48    False
49    False
50    False
51    False
52    False
53    False
54    False
55    False
56    False
57    False
58    False
59    False
60    False
62    False
63    False
64    False
65    False
67    False
68    False
69    False
70    False
72    False
73    False
74    False
75    False
```

In [89]:
```
#Remove duplicate rows
df.drop_duplicates(inplace=True)
```

In [90]:
```
print(df.duplicated())
```

```
0     False
1     False
3     False
4     False
5     False
      ...
70    False
72    False
73    False
74    False
75    False
Length: 67, dtype: bool
```

# Wrong data

If you take a look at our data set, you can see that in row 7, the duration is 450, but for all theother rows the duration is between 30 and 60.

It doesn't have to be wrong, but taking in consideration that this is the data set of someone'sworkout sessions, we conclude with the fact that this person did not work out in 450 minutes.

In [91]:
```
# Replacing Values

#for small dataset (look & replace)
df.loc[7,'Duration']=45
```

set some boundaries for legal values, and replace any values that areoutside of the boundaries.

In [92]:
```
for x in df.index:
  if df.loc[x,"Duration"]>120:
    df.loc[x,"Duration"]=120
```

In [93]:
```
print(df.to_string())
```

```
    Duration  Pulse  Maxpulse     Calories          date
0         60  110.0       130   409.100000  '2020/12/01'
1         60  117.0       145   479.000000  '2020/12/01'
3         45  109.0       175   282.400000  '2020/12/01'
4         45  117.0       148   406.000000  '2020/12/01'
5        120  102.0       127   300.000000  '2020/12/01'
6         60  110.0       136   374.000000  '2020/12/01'
7         45  104.0       134   253.300000  '2020/12/01'
8         30  109.0       133   195.100000  '2020/12/01'
10        60  103.0       147   329.300000  '2020/12/01'
11        60  100.0       120   250.700000  '2020/12/01'
12        60  106.0       128   345.300000  '2020/12/01'
13        60  104.0       132   379.300000  '2020/12/01'
14        60   98.0       123   275.000000  '2020/12/01'
15        60   98.0       120   215.200000  '2020/12/01'
16        60  100.0       120   300.000000  '2020/12/01'
17        45   90.0       112   397.354286  '2020/12/01'
19        45   97.0       125   243.000000  '2020/12/01'
20        60  108.0       131   364.200000  '2020/12/01'
21        45  100.0       119   282.000000  '2020/12/01'
22        60  130.0       101   300.000000  '2020/12/01'
23        45  105.0       132   246.000000  '2020/12/01'
24        60  102.0       126   334.500000  '2020/12/01'
25        60  100.0       120   250.000000  '2020/12/01'
26        60   92.0       118   241.000000  '2020/12/01'
27        60  103.0       132   397.354286  '2020/12/01'
28        60  100.0       132   280.000000  '2020/12/01'
29        60  102.0       129   380.300000  '2020/12/01'
30        60   92.0       115   243.000000  '2020/12/01'
31        45   90.0       112   180.100000  '2020/12/01'
32        60  101.0       124   299.000000  '2020/12/01'
33        60   93.0       113   223.000000  '2020/12/01'
34        60  107.0       136   361.000000  '2020/12/01'
35        60  114.0       140   415.000000  '2020/12/01'
36        60  102.0       127   300.000000  '2020/12/01'
39        45  104.0       129   266.000000  '2020/12/01'
41        60   98.0       126   286.000000  '2020/12/05'
42        60  100.0       122   329.400000  '2020/12/06'
43        60  111.0       138   400.000000  '2020/12/01'
44        60  111.0       131   397.000000  '2020/12/01'
45        60   99.0       119   273.000000  '2020/12/01'
46        60  109.0       153   387.600000  '2020/12/01'
47        45  111.0       136   300.000000  '2020/12/01'
48        45  108.0       129   298.000000  '2020/12/01'
49        60  111.0       139   397.600000  '2020/12/01'
50        60  107.0       136   380.200000  '2020/12/01'
51        80  123.0       146   643.100000  '2020/12/01'
52        60  106.0       130   263.000000  '2020/12/01'
53        60  118.0       151   486.000000  '2020/12/01'
54        30  136.0       175   238.000000  '2020/12/03'
55        60  121.0       146   450.700000  '2020/12/01'
56        60  118.0       121   413.000000   2020/12/08'
57        45  115.0       144   305.000000             0
58        20  153.0       172   226.400000  '2020/12/08'
59        45  123.0       152   321.000000  '2020/12/08'
60       120  108.0       160  1376.000000      20201226
62       120  109.0       135   853.000000      20201226
63        45  118.0       141   341.000000      20201226
64        20  110.0       130   131.400000  '2020/12/08'
65       120   90.0       130   800.400000  '2020/12/08'
67       120  107.0       130   816.000000  '2020/12/29'
68        20  106.0       136   110.400000  '2020/12/29'
69       120  108.0       143  1500.200000  '2020/12/29'
70       120   97.0       129  1115.000000  '2020/12/29'
72        90  100.0       127   700.000000  '2020/12/29'
73       120   97.0       127   953.200000  '2020/12/29'
74        45  114.0       146   304.000000  '2020/12/29'
75        90   98.0       125   563.200000  '2020/12/29'
```

This way you do not have to find out what to replace them with, and there is a good chance you donot need them to do your analyses.

In [94]:
```
# Removing Rows

for x in df.index:
  if df.loc[x,"Duration"]>120:
    df.drop(x,inplace=True)
```

# Questions

1) What is the output of the following code?

import numpy as np

a = np.arange(10)

print(a[2:5])

- a) [ 2, 3, 4]
- b) [0, 1, 2]
- c) [5, 6, 7]
- d) [2, 4, 6 ]

2) What is the output of the following code?

import numpy as np

a = np.array([1, 2, 3])

b = np.array([4, 5, 6])

c = a + b

print(c)

- a) [1, 2, 3, 4, 5, 6 ]
- b) [[1, 4], [2, 5], [3, 6] ]
- c) [ 5, 7, 9]
- d) Error

3) What will be output of following code ?

import pandas as pd

data = [['Anuj',21],['Rama',25],['Kapil',22]]

df = pd.DataFrame(data,columns=['Name','Age'])

print (df)

a)

- Name Age
- 0 Anuj 21
- 1 Rama 25
- 2 Kapil 22

b)

- Name Age
- 0 Anuj 21
- 1 Kapil 22
- 2 Rama 25

c)

- Name Age
- 0 Kapil 22
- 1 Rama 25
- 2 Anuj 21

d)

- Name Age
- 0 Rama 25
- 1 Anuj 21
- 2 Kapil 22

# Coding

1) **Use arange method to create a Numpy array in the range through 0 to 100 (including 100). Then find the maximum element in this array.**

2) import numpy as np

arr = np.array([[1, 2, 3, 4, 5], [6, 7, 8, 9, 10]])

**Complete the code above to form the following output.**

[[2 3 4]

[7 8 9]]

3) **Write a Pandas program to compare the elements of the two Pandas Series**

Sample Series: [2, 4, 6, 8, 10], [1, 3, 5, 7, 10]

In [95]:
```
# Answer 1
import numpy as np
x= np.arange(0,101)
print(x)
maximum = max(x)
print("Maximum element is: ", maximum)
```

```
[  0   1   2   3   4   5   6   7   8   9  10  11  12  13  14  15  16  17
  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35
  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53
  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71
  72  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89
  90  91  92  93  94  95  96  97  98  99 100]
Maximum element is:  100
```

In [96]:
```
# Answer 2
import numpy as np
ar = np.array([[1,2,3,4,5],[6,7,8,9,10]])
y = ar[:,1:4]
print(y)
```

```
[[2 3 4]
 [7 8 9]]
```

In [97]:
```
# Answer 3
import pandas as pd

series1 = pd.Series([2, 4, 6, 8, 10])
series2 = pd.Series([1, 3, 5, 7, 10])


print(series1 == series2)
```

```
0    False
1    False
2    False
3    False
4     True
dtype: bool
```

# End