Bad data could be:

1. Wrong data
2. Data in wrong format
3. Duplicates
4. Empty cells/missing values

Data Cleaning

In [1]:

```
import numpy as np
import pandas as pd
```

1. Wrong data

- Solution: Replace

In [2]:

```
df1 = pd.DataFrame({"Age":   [15,24,18,19.4,"20+"],
                    "Gender": ["male","female","female","female","male"]})
df1
```

Out[2]:

	Age	Gender
0	15	male
1	24	female
2	18	female
3	19.4	female
4	20+	male

In [3]:

```
df1["Age"].replace("20+", 20, inplace =True)
df1
```

Out[3]:

	Age	Gender
0	15.0	male
1	24.0	female
2	18.0	female
3	19.4	female
4	20.0	male

2.Wrong data type

- Solution: convert the datatype

In [4]:

```
df2 = pd.DataFrame({"Age":   [15,24,18,19.4,"20"],
                   "Gender":["male","female","female","female","male"]})
df2
```

Out[4]:

	Age	Gender
0	15	male
1	24	female
2	18	female
3	19.4	female
4	20	male

In [5]:

```
df2.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5 entries, 0 to 4
Data columns (total 2 columns):
 #   Column  Non-Null Count  Dtype 
---  ------  --------------  ----- 
 0   Age     5 non-null      object
 1   Gender  5 non-null      object
dtypes: object(2)
memory usage: 208.0+ bytes
```

In [6]:

```
df2["Age"] = df2["Age"].astype('int')
```

In [7]:

```
df2.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5 entries, 0 to 4
Data columns (total 2 columns):
 #   Column  Non-Null Count  Dtype 
---  ------  --------------  ----- 
 0   Age     5 non-null      int32 
 1   Gender  5 non-null      object
dtypes: int32(1), object(1)
memory usage: 188.0+ bytes
```

3. Duplicates

- Solution: Remove

In [8]:

```
df3 = pd.DataFrame({"Age":    [15,18,18,19,20],
                   "Gender":["male","female", "female","female","male"]})
df3
```

Out[8]:

	Age	Gender
0	15	male
1	18	female
2	18	female
3	19	female
4	20	male

In [9]:

```
#to check the duplicated records
df3.duplicated()
```

Out[9]:

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

In [10]:

```
# toremove the duplictes ---> we use df.drop_duplicates
df3.drop_duplicates()
```

Out[10]:

	Age	Gender
0	15	male
1	18	female
3	19	female
4	20	male

In [11]:

```
df3.drop_duplicates(inplace=True, ignore_index= True)
df3
```

Out[11]:

	Age	Gender
0	15	male
1	18	female
2	19	female
3	20	male

Missing values

- Solution: Either remove or replace

In [12]:

```
df = pd.DataFrame({"Age":   [15,np.nan,24,19,20],
                   "Gender":["male",np.nan,"female", "female","female"]})
df
```

Out[12]:

	Age	Gender
0	15.0	male
1	NaN	NaN
2	24.0	female
3	19.0	female
4	20.0	female

In [13]:

```
#to check the missing values records
df.isnull()
```

Out[13]:

	Age	Gender
0	False	False
1	True	True
2	False	False
3	False	False
4	False	False

In [14]:

```
# to check total missing values
df.isnull().sum()
```

Out[14]:

```
Age       1
Gender    1
dtype: int64
```

In [15]:

```
# total Missing values & there percentages for each variable

x=df.isnull().sum()
y=(df.isnull().sum()/len(df))*100
z={'Number of missing values':x,'Percentage of missing values':y}
missing = pd.DataFrame(z,columns=['Number of missing values','Percentage of missing values'])
missing
```

Out[15]:

	Number of missing values	Percentage of missing values
Age	1	20.0
Gender	1	20.0

In [16]:

```
#sorting of missing values based on there percentages
missing.sort_values(by='Percentage of missing values',ascending=False)
```

Out[16]:

	Number of missing values	Percentage of missing values
Age	1	20.0
Gender	1	20.0

Option 1. Remove the rows that contain missing values

In [17]:

```
df2 = df.dropna()
df2
```

Out[17]:

	Age	Gender
0	15.0	male
2	24.0	female
3	19.0	female
4	20.0	female

Option 2: Replace the nan values

- fill with value
- Continous Variables ---> Replace with either Mean or Median
- Discrete Variables ---> Replace with Mode

In [18]:

```
# replacing the 'age' column with value of 0
df['Age'].replace(np.nan, 0)
```

Out[18]:

```
0    15.0
1     0.0
2    24.0
3    19.0
4    20.0
Name: Age, dtype: float64
```

In [19]:

```
# replacing the 'age' column with mean
df['Age'].fillna(df["Age"].mean(),inplace=True)
df
```

Out[19]:

	Age	Gender
0	15.0	male
1	19.5	NaN
2	24.0	female
3	19.0	female
4	20.0	female

In [20]:

```
df["Gender"].fillna(df["Gender"].mode()[0],inplace=True)
df
```

Out[20]:

	Age	Gender
0	15.0	male
1	19.5	female
2	24.0	female
3	19.0	female
4	20.0	female

We can replace the missing values by using SimpleImputer() by using sklearn

In [21]:

```
df = pd.DataFrame({"Age":   [15,16,np.nan,24,19,20],
                   "Gender":["male",np.nan,"female", "female","female","male"]})
df
```

Out[21]:

	Age	Gender
0	15.0	male
1	16.0	NaN
2	NaN	female
3	24.0	female
4	19.0	female
5	20.0	male

In [22]:

```
from sklearn.impute import SimpleImputer
mean_imputer = SimpleImputer(strategy='mean')
df["Age"] = mean_imputer.fit_transform(df[["Age"]])
df
```

Out[22]:

	Age	Gender
0	15.0	male
1	16.0	NaN
2	18.8	female
3	24.0	female
4	19.0	female
5	20.0	male

In [23]:

```
from sklearn.impute import SimpleImputer
mode_imputer = SimpleImputer(strategy='most_frequent')
df["Gender"] = mode_imputer.fit_transform(df[["Gender"]])
df
```

Out[23]:

	Age	Gender
0	15.0	male
1	16.0	female
2	18.8	female
3	24.0	female
4	19.0	female
5	20.0	male

Outliers

- An outlier is a data point in a data set that is distant from all other observations, which is significantly different from the remaining data.
- A data point that lies outside the overall distribution of the dataset.

What are the impacts of having outliers in a dataset?

1. It causes various problems during our statistical analysis (It may cause a significant impact on the mean and the standard deviation) Statistics such as the mean and variance are very susceptible to outliers.
2. In addition, **some Machine Learning models are sensitive to outliers** which may decrease their performance. Thus, depending on which algorithm we wish to train, we often remove outliers from our variables.

Reasons for Outliers

1. Data Entry Errors (Ex: Entering salary as 1,00,000 instead of 10,000)
2. Measurement Errors (Ex: Measuring in meters instead of KM)
3. Instrumental Errors

Types of Outliers

1. Univariate Outliers --> Indentifing outlier for single variable
2. Bivariate Outliers --> Indentified as outlier by analyzing 2 variables at a time

Solution: 3R Technique

1. Remove (remove the outliers from our dataset)

2. Replace the ouliers
   - Rectify or Replace --> (data entry error) ---> Ask and confirm it from the Data Engineering team.
   - Replace with upper limit & lower limit based on IQR
3. Retain (consider for analysis) ---> Treat them separately

In [24]:

```
df= pd.DataFrame({"marks":[10,11,12,25,25,27,31,33,34,34,36,36,43,50,59]})
df
```

Out[24]:

	marks
0	10
1	11
2	12
3	25
4	25
5	27
6	31
7	33
8	34
9	34
10	36
11	36
12	43
13	50
14	59

Various ways of finding the outlier.

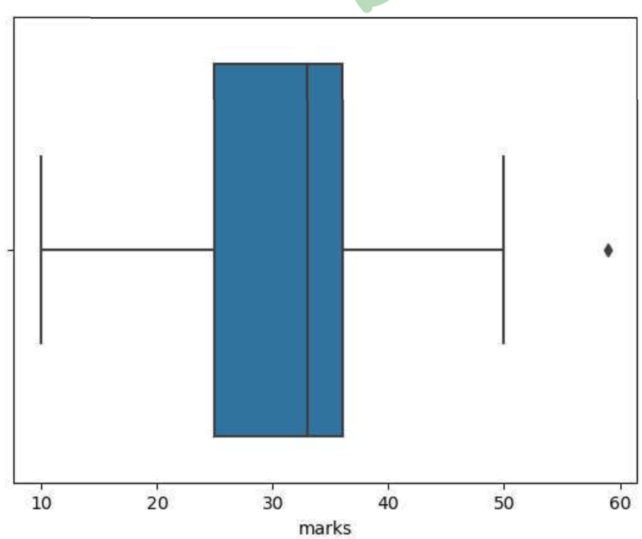
1. Boxplot
2. IQR

Identifying Outliers based on boxplot

In [25]:

```
import matplotlib.pyplot as plt
import seaborn as sns

sns.boxplot(x=df["marks"])
plt.show()
```



Identifying Outliers based on IQR

In [26]:

```python
#calculate Q1
Q1=df["marks"].quantile(0.25)
print("Q1:",Q1)

#calculate Q3
Q3=df["marks"].quantile(0.75)
print("Q3:",Q3)

#calculate IQR
IQR = Q3 - Q1
print("IQR:",IQR)

#Calculate lower limit of outlier
lower_limit = Q1 - (IQR * 1.5)
print("lower limit:",lower_limit)

#Calculate upper limit of outlier
upper_limit = Q3 + (IQR * 1.5)
print("upper limit:",upper_limit)
```

```
Q1: 25.0
Q3: 36.0
IQR: 11.0
lower limit: 8.5
upper limit: 52.5
```

Outliers Data

In [27]:

```python
df[(df["marks"]<lower_limit) | (df["marks"]>upper_limit)]
```

Out[27]:

	marks
14	59

Remove

In [28]:

```python
df.drop(index=14)
```

Out[28]:

	marks
0	10
1	11
2	12
3	25
4	25
5	27
6	31
7	33
8	34
9	34
10	36
11	36
12	43
13	50

Replace

> based on confirmation from data engineer team / based on research / based on domain expertise

In [29]:

```python
#pip install feature_engine
```

In [30]:

```python
from feature_engine.outliers import ArbitraryOutlierCapper

capper = ArbitraryOutlierCapper(max_capping_dict = {'marks':52},
                                min_capping_dict = {'marks':6})

capper.fit_transform(df[["marks"]])
```

Out[30]:

	marks
0	10
1	11
2	12
3	25
4	25
5	27
6	31
7	33
8	34
9	34
10	36
11	36
12	43
13	50
14	52

Replace

> based on iqr

In [31]:

```python
from feature_engine.outliers import Winsorizer

win = Winsorizer(capping_method='iqr', tail='both',fold=1.5)

win.fit_transform(df[["marks"]])
```

Out[31]:

	marks
0	10.0
1	11.0
2	12.0
3	25.0
4	25.0
5	27.0
6	31.0
7	33.0
8	34.0
9	34.0
10	36.0
11	36.0
12	43.0
13	50.0
14	52.5

In [32]:

```python
print(win.left_tail_caps_, win.right_tail_caps_)
```

```
{'marks': 8.5} {'marks': 52.5}
```