

AIM :-

```
import pandas as pd
from google.colab import files
uploaded=files.upload()
df=pd.read_csv('pokemon.csv')
df.head()
```

No file chosen

Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.

Saving pokemon.csv to pokemon.csv

	abilities	against_bug	against_dark	against_dragon	against_electric	against_fairy	against_fight	against_fire	against_ice	against_normal	against_water
0	['Overgrow', 'Chlorophyll']	1.0	1.0	1.0	0.5	0.5	0.5	0.5	2.0	1.0	1.0
1	['Overgrow', 'Chlorophyll']	1.0	1.0	1.0	0.5	0.5	0.5	0.5	2.0	1.0	1.0
2	['Overgrow', 'Chlorophyll']	1.0	1.0	1.0	0.5	0.5	0.5	0.5	2.0	1.0	1.0
3	['Blaze', 'Solar Power']	0.5	1.0	1.0	1.0	0.5	1.0	0.5	1.0	0.5	1.0
4	['Blaze', 'Solar Power']	0.5	1.0	1.0	1.0	0.5	1.0	0.5	1.0	0.5	1.0

5 rows × 41 columns

df.shape

(800, 13)

df.info()

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 800 entries, 0 to 799
Data columns (total 13 columns):
 #   Column      Non-Null Count  Dtype  
--- 
 0   #           800 non-null    int64  
 1   Name         800 non-null    object  
 2   Type 1      800 non-null    object  
 3   Type 2      414 non-null    object  
 4   Total        800 non-null    int64  
 5   HP           800 non-null    int64  
 6   Attack       800 non-null    int64  
 7   Defense      800 non-null    int64  
 8   Sp. Atk      800 non-null    int64  
 9   Sp. Def      800 non-null    int64  
 10  Speed         800 non-null    int64  
 11  Generation    800 non-null    int64  
 12  Legendary     800 non-null    bool   
dtypes: bool(1), int64(9), object(3)
memory usage: 75.9+ KB
```

df.describe()

	#	Total	HP	Attack	Defense	Sp. Atk	Sp. Def	Speed	Generation
count	800.000000	800.000000	800.000000	800.000000	800.000000	800.000000	800.000000	800.000000	800.000000
mean	362.813750	435.10250	69.258750	79.001250	73.842500	72.820000	71.902500	68.277500	3.32375
std	208.343798	119.96304	25.534669	32.457366	31.183501	32.722294	27.828916	29.060474	1.66129
min	1.000000	180.00000	1.000000	5.000000	5.000000	10.000000	20.000000	5.000000	1.00000
25%	184.750000	330.00000	50.000000	55.000000	50.000000	49.750000	50.000000	45.000000	2.00000
50%	364.500000	450.00000	65.000000	75.000000	70.000000	65.000000	70.000000	65.000000	3.00000
75%	539.250000	515.00000	80.000000	100.000000	90.000000	95.000000	90.000000	90.000000	5.00000
max	721.000000	780.00000	255.000000	190.000000	230.000000	194.000000	230.000000	180.000000	6.00000

```
df.isnull()
```

	#	Name	Type 1	Type 2	Total	HP	Attack	Defense	Sp. Atk	Sp. Def	Speed	Generation	Legendary
0	False	False	False	False	False	False	False	False	False	False	False	False	False
1	False	False	False	False	False	False	False	False	False	False	False	False	False
2	False	False	False	False	False	False	False	False	False	False	False	False	False
3	False	False	False	False	False	False	False	False	False	False	False	False	False
4	False	False	False	True	False	False	False	False	False	False	False	False	False
...
795	False	False	False	False	False	False	False	False	False	False	False	False	False
796	False	False	False	False	False	False	False	False	False	False	False	False	False
797	False	False	False	False	False	False	False	False	False	False	False	False	False
798	False	False	False	False	False	False	False	False	False	False	False	False	False
799	False	False	False	False	False	False	False	False	False	False	False	False	False

800 rows × 13 columns

```
df.isnull().sum()
```

	0
#	0
Name	0
Type 1	0
Type 2	386
Total	0
HP	0
Attack	0
Defense	0
Sp. Atk	0
Sp. Def	0
Speed	0
Generation	0
Legendary	0

```
dtype: int64
```

DEALING WITH MISSING VALUES

Deleting the missing values Imputing the missing values Imputing the missing values for categorical data

DELETING THE MISSING VALUES Delete the whole column

```
df2=df.drop(['Type 2'],axis=1)
df2.isnull().sum()
```

	0
#	0
Name	0
Type 1	0
Total	0
HP	0
Attack	0
Defense	0
Sp. Atk	0
Sp. Def	0
Speed	0
Generation	0
Legendary	0

dtype: int64

```
r=df.drop(labels=[1],axis=0) #r=df.drop(0) (both can be used)
r.head() #labels is used to delete a particular row , r
```

#	Name	Type 1	Type 2	Total	HP	Attack	Defense	Sp. Atk	Sp. Def	Speed	Generation	Legendary
0 1	Bulbasaur	Grass	Poison	318	45	49	49	65	65	45	1	False
2 3	Venusaur	Grass	Poison	525	80	82	83	100	100	80	1	False
3 3	VenusaurMega Venusaur	Grass	Poison	625	80	100	123	122	120	80	1	False
4 4	Charmander	Fire	Nan	309	39	52	43	60	50	65	1	False
5 5	Charmeleon	Fire	Nan	405	58	64	58	80	65	80	1	False

Deleting particular rows

```
#delete a range of rows - index values 10-20
r3=df.drop(labels=range(40,45),axis=0)
#45 excluded #r3=df.drop(range(10,20))
r3.head(20)
#Note that axis is 0 by default
```

#		Name	Type 1	Type 2	Total	HP	Attack	Defense	Sp. Atk	Sp. Def	Speed	Generation	Legendary
0	1	Bulbasaur	Grass	Poison	318	45	49	49	65	65	45	1	False
1	2	Ivysaur	Grass	Poison	405	60	62	63	80	80	60	1	False
2	3	Venusaur	Grass	Poison	525	80	82	83	100	100	80	1	False
3	3	VenusaurMega Venusaur	Grass	Poison	625	80	100	123	122	120	80	1	False
4	4	Charmander	Fire	0	309	39	52	43	60	50	65	1	False
5	5	Charmeleon	Fire	0	405	58	64	58	80	65	80	1	False
6	6	Charizard	Fire	Flying	534	78	84	78	109	85	100	1	False
7	6	CharizardMega Charizard X	Fire	Dragon	634	78	130	111	130	85	100	1	False
8	6	CharizardMega Charizard Y	Fire	Flying	634	78	104	78	159	115	100	1	False
9	7	Squirtle	Water	0	314	44	48	65	50	64	43	1	False
10	8	Wartortle	Water	0	405	59	63	80	65	80	58	1	False
11	9	Blastoise	Water	0	530	79	83	100	85	105	78	1	False
12	9	BlastoiseMega Blastoise	Water	0	630	79	103	120	135	115	78	1	False
13	10	Caterpie	Bug	0	195	45	30	35	20	20	45	1	False
14	11	Metapod	Bug	0	205	50	20	55	25	25	30	1	False
15	12	Butterfree	Bug	Flying	395	60	45	50	90	80	70	1	False
16	13	Weedle	Bug	Poison	195	40	35	30	20	20	50	1	False
17	14	Kakuna	Bug	Poison	205	45	25	50	25	25	35	1	False
18	15	Beedrill	Bug	Poison	395	65	90	40	45	80	75	1	False
19	15	BeedrillMega Beedrill	Bug	Poison	495	65	150	40	15	80	145	1	False

IMPUTING THE MISSING VALUE

FILL THE MISSING WITH ZERO

```
#replacing the missing values of the any column with 0.
#replacing the missing values with '0' using 'fillna' method
df['Type 2']=df['Type 2'].fillna(0)
df['Type 2'].isnull().sum()

np.int64(0)
```

FILL THE MISSING VALUE WITH MEAN

```
#replcaing with the mean
df['HP']=df['HP'].fillna(df['HP'].mean())
#here hp is not empty but if it would have been but it will be filled with it
```

Mean/Median/Mode Imputation : Replace missing entries with the average(mean),middle value(median),most frequent(mode)

REPLACING WITH MEDIAN

```
df['Attack']=df['Attack'].fillna(df['Attack'].median())
```

REPLACING WITH MODE

```
df['Speed']=df['Speed'].fillna(df['Speed'].mode())
```

DE-DUPLICATE

De-Duplicate means removing all duplicate values. There is no need for duplicate values in data analysis. These values only effect the accuracy and efficiency of the analysis result. To find duplicate values in the dataset, we will use a simple dataframe function i.e.,`duplicated()`

```
from google.colab import files
up=files.upload()
```

No file chosen Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.
Saving Cardetails.csv to Cardetails.csv

```
d=pd.read_csv('Cardetails.csv')
```

```
d.duplicated()
```

	0
0	False
1	False
2	False
3	False
4	False
...	...
8123	False
8124	False
8125	False
8126	True
8127	True

8128 rows × 1 columns

dtype: bool

```
print(d.duplicated().sum())
```

1202

Removing of duplicates `subset=None`: Specifies which columns to consider for identifying duplicates. If none, all columns are used. If a list of column names is provided, duplicates are detected based on those columns. `keep=default='first'`: Determines which duplicate row to keep.

'first': keeps the first occurrence and drops the rest. 'last': keeps the last occurrence and drop others.

`False`: Remove all duplicate rows. `inplace(defualt=False)`: If True, modifies the dataframe in place and return None. If false, return a new dataframe with duplicates removed. `ignore_index(default=False)`: If true, resets the index in the resulting dataframe

```
d.drop_duplicates(subset=None,keep='first',inplace=True,ignore_index=False)
#The drop_duplicates method with inplace=False returns a new DataFrame without modifying the original one.
```

```
print(d.duplicated().sum())
```

0

DEALING WITH OUTLIERS (Detection and Removal)

DETECTION

```
import sklearn
from sklearn.datasets import load_diabetes
import pandas as pd
```

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

```
diabetes=load_diabetes()
column_name=diabetes.feature_names
df_diabetics=pd.DataFrame(diabetes.data,columns=column_name)
#used to create a valid dataset for EDA
```

```
df_diabetics.head()
```

	age	sex	bmi	bp	s1	s2	s3	s4	s5	s6
0	0.038076	0.050680	0.061696	0.021872	-0.044223	-0.034821	-0.043401	-0.002592	0.019907	-0.017646
1	-0.001882	-0.044642	-0.051474	-0.026328	-0.008449	-0.019163	0.074412	-0.039493	-0.068332	-0.092204
2	0.085299	0.050680	0.044451	-0.005670	-0.045599	-0.034194	-0.032356	-0.002592	0.002861	-0.025930
3	-0.089063	-0.044642	-0.011595	-0.036656	0.012191	0.024991	-0.036038	0.034309	0.022688	-0.009362
4	0.005383	-0.044642	-0.036385	0.021872	0.003935	0.015596	0.008142	-0.002592	-0.031988	-0.046641

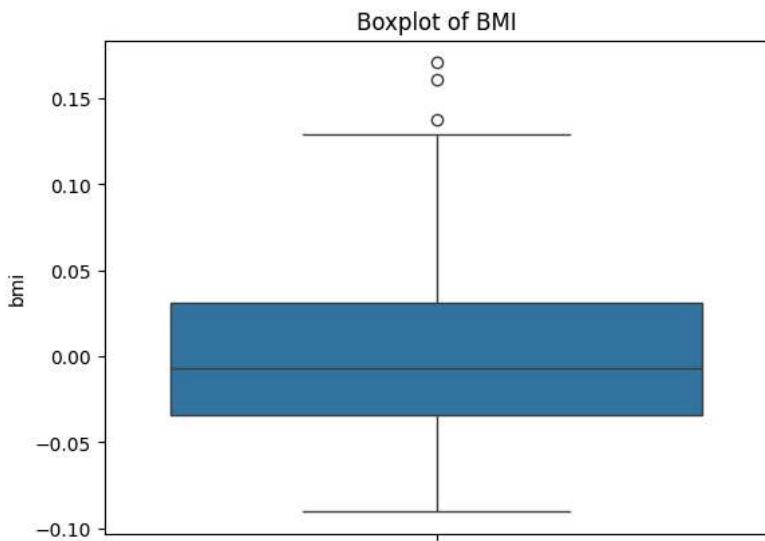
```
df_diabetics.shape
```

```
(442, 10)
```

```
df_diabetics.info()
```

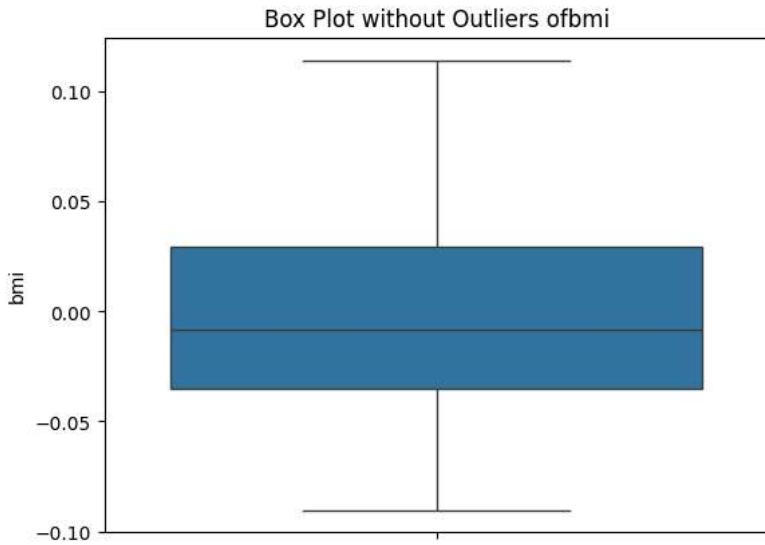
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 442 entries, 0 to 441
Data columns (total 10 columns):
 #   Column   Non-Null Count  Dtype  
 --- 
 0   age      442 non-null    float64
 1   sex      442 non-null    float64
 2   bmi      442 non-null    float64
 3   bp       442 non-null    float64
 4   s1       442 non-null    float64
 5   s2       442 non-null    float64
 6   s3       442 non-null    float64
 7   s4       442 non-null    float64
 8   s5       442 non-null    float64
 9   s6       442 non-null    float64
dtypes: float64(10)
memory usage: 34.7 KB
```

```
#For detection of outliers we used box plot
sns.boxplot(df_diabetics['bmi'])
plt.title('Boxplot of BMI')
plt.show()
#above the upper line the circles represent the outliers
```



REMOVAL

```
def removal_box_plot(df,column,threshold):
    removed_outliers=df[df[column]<=threshold]
    # a updated dataset with rows only which value is less than the threshold for
    sns.boxplot(removed_outliers[column])
    plt.title(f"Box Plot without Outliers of{column}")
    plt.show()
    return removed_outliers
threshold_value = 0.12
no_outliers=removal_box_plot(df_diabetics,'bmi',threshold_value)
```



TRANSFORMATION/SCALING Transformation or scaling is a technique used in data preprocessing to change the range or distribution of data.

Common scaling techniques include:

Min-Max Scaling: Scales the data to a fixed range, usually between 0 and 1.

Standardization (Z-score scaling): Transforms the data to have a mean of 0 and a standard deviation of 1.

```
#we do scaling so we can arrange all attribute to a particular range
#z-score=(X-mean)/standard deviation
from scipy import stats
import numpy as np
z=np.abs(stats.zscore(df_diabetics['age'])) #this finds zscore for each cell of age column and takes its absolute(age cant
print(z)
```

```
[0.80050009 0.03956713 1.79330681 1.87244107 0.11317236 1.94881082
 0.9560041 1.33508832 0.87686984 1.49059233 2.02518057 0.57139085
 0.34228161 0.11317236 0.95323959 1.1087436 0.11593688 1.48782782
 0.80326461 0.57415536 1.03237385 1.79607132 1.79607132 0.95323959
 1.33785284 1.41422259 2.25428981 0.49778562 1.10597908 1.41145807
 1.26148309 0.49778562 0.72413034 0.6477606 0.34228161 1.02960933
 0.26591186 0.19230663 0.03956713 0.03956713 0.11317236 2.10155031
 1.26148309 0.41865135 0.95323959 0.57139085 1.18511334 1.64333183
 1.41145807 0.87963435 0.72413034 1.25871858 1.1087436 0.19230663
 1.03237385 0.87963435 0.87963435 0.57415536 0.87686984 1.33508832
 1.49059233 0.87963435 0.57415536 0.72689486 1.41145807 0.9560041
 0.19230663 0.87686984 0.80050009 0.34228161 0.03956713 0.03956713
 1.33508832 0.26591186 0.26591186 0.19230663 0.65052511 2.02518057
 0.11317236 2.17792006 1.48782782 0.26591186 0.34504612 0.80326461
 0.03680262 0.95323959 1.49059233 0.95323959 1.1087436 0.9560041
 0.26591186 0.95323959 0.42141587 1.03237385 1.64333183 1.49059233
 1.18234883 0.57415536 0.03680262 0.03956713 0.34228161 0.34228161
 1.94881082 1.25871858 0.57415536 0.4950211 2.02518057 0.57139085
 0.41865135 0.80050009 0.87686984 0.41865135 1.79607132 0.41865135
 0.4950211 0.65052511 1.02960933 1.25871858 1.18511334 0.34228161
 1.03237385 1.33508832 1.02960933 0.11317236 0.11593688 0.11593688
 1.87244107 0.72413034 1.1087436 0.18954211 1.33785284 2.02518057
 0.34228161 0.87963435 1.56696208 0.11593688 1.94881082 0.11317236
 0.72413034 0.4950211 0.87686984 0.57415536 0.87686984 0.65052511
 0.6477606 0.87963435 0.65052511 1.18511334 1.26148309 1.03237385
 0.4950211 0.03680262 0.72689486 0.87686984 1.41145807 0.57415536]
```

0.34504612	0.03956713	0.26867637	0.11593688	0.19230663	0.9560041
1.1087436	0.34228161	0.95323959	0.87963435	1.18511334	1.48782782
0.03680262	0.03956713	0.4950211	0.42141587	0.87686984	1.33785284
0.34228161	1.41145807	0.95323959	1.02960933	0.87686984	0.49778562
0.80326461	1.02960933	0.95323959	0.95323959	0.34228161	1.56696208
1.71970158	1.41422259	0.11317236	0.03956713	0.18954211	0.11593688
1.18234883	0.18954211	1.41422259	0.57139085	0.49778562	1.02960933
1.1087436	0.87686984	1.18234883	0.72689486	1.71693706	0.03956713
2.32789504	0.65052511	0.03680262	0.18954211	0.6477606	0.80050009
0.18954211	1.9460463	1.41145807	0.03680262	0.6477606	0.57139085
0.26591186	1.56419757	0.87963435	1.87244187	0.4950211	0.9560041
0.49778562	2.10155031	0.57415536	0.6477606	2.17792006	1.41145807
1.1087436	0.57415536	0.80326461	0.18954211	0.26591186	1.41145807
0.95323959	1.41145807	0.57139085	1.18234883	0.72413034	0.4950211
1.02960933	0.6477606	2.17792006	0.34228161	1.26148309	0.57415536
0.87686984	1.71970158	0.87963435	0.26867637	1.41145807	1.1087436
0.11317236	1.71693706	0.6477606	0.03680262	1.03237385	0.57415536
1.51055731	0.75501105	0.07505001	1.07050022	0.34501512	1.55505720