Module 2 Data Clening using Pandas Lab Practical

what is pandas?

- It is a package useful for data analysis and manipulation.
- Pandas provide an easy way to create, manipulate and wrangle the data.
- Pandas provide powerful and easy-to-use data structures, as well as the means to quickly perform operations on these structures.

Data scientists use Pandas for its following advantages:

- Easily handles missing data.
- It uses Series for one-dimensional data structure and DataFrame for multi-dimensional data structure.
- It provides an efficient way to slice the data.
- It provides a flexible way to merge, concatenate or reshape the data.

In [ ]:
```
1
```

Note : Sample data and datasets used in this Notebook will be available in below github url

https://github.com/kundetivamsi2001/Datasets_AI-ML (https://github.com/kundetivamsi2001/Datasets_AI-ML)

1. How Install and import pandas

In [ ]:
```
1 !pip install pandas
```

DATA STRUCTURE IN PANDAS

A data structure is a way to arrange the data in such a way that so it can be accessed quickly and we can perform various operation on this data like- retrieval, deletion, modification etc.

Pandas deals with 3 data structure-

1. Series
2. Data Frame
3. Panel

Series

Series-Series is a one-dimensional array like structure with homogeneous data, which can be used to handle and manipulate data. What makes it special is its index attribute, which has incredible functionality and is heavily mutable.

It has two parts-

1. **Data part (An array of actual data)**
2. **Associated index with data (associated array of indexes or data labels)**

e.g.-

Index	Data
0	10
1	15
2	18

- ✓ We can say that **Series** is a labeled *one-dimensional array which can hold any type of data.*
- ✓ Data of **Series** is *always mutable*, means it can be changed.
- ✓ But the size of Data of **Series** is *always immutable*, means it cannot be changed.
- ✓ **Series** may be considered as a **Data Structure with two arrays** out which **one array** works as *Index (Labels)* and the **second array** works as *original Data.*
- ✓ ***Row Labels*** in Series are called *Index*.

Syntax to create a Series:

```
<Series Object>=pandas.Series (data, index=idx (optional))
```

✓ **Where data may be** *python sequence (Lists)*, ndarray, scalar value **or a python dictionary.**

How to create Series with nd array

Program-

```
import pandas as pd
import numpy as np
arr=np.array([10,15,18,22])
s = pd.Series(arr)
print(s)
```

Default Index

```
Output-
0  10
1  15
2  18
3  22
```

Data

Here we create an array of 4 values.

In [190]:
```
#how to create series from arrays in pandas
import pandas as pd
import numpy as np
arr=np.array([10,20,30,40,9,0,7,6,5,4,3,2,12,34,8798])
s=pd.Series(arr)
s
```

Out[190]:
```
0       10
1       20
2       30
3       40
4        9
5        0
6        7
7        6
8        5
9        4
10       3
11       2
12      12
13      34
14    8798
dtype: int32
```

In [191]:
```
#create series with label indexing
import numpy as np
arr=np.array([1,2,3,4])
s=pd.Series(arr,index=['first','second','third','fourth'])
s
```

Out[191]:
```
first     1
second    2
third     3
fourth    4
dtype: int32
```

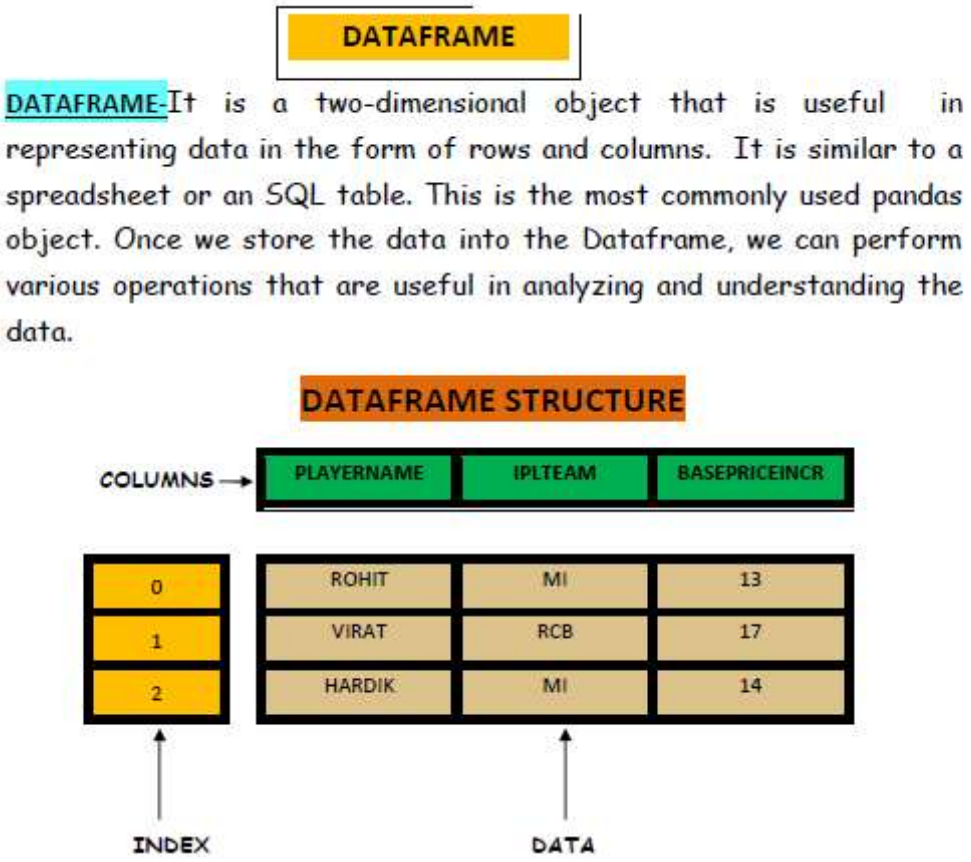
In [188]:
```
#Selection operator in series
s[1:4] # selecting data from index 1 to 3 i.e (n-1)
```

Out[188]:
```
second    2
third     3
fourth    4
dtype: int32
```

In [192]:
```
#Series can be created from list,tuple,dictionary
#create series from dictionary
dc={'Name':'Ram','Dept':'ECE', 'Marks':500,'Age':23}
sr=pd.Series(dc)
sr
```

Out[192]:
```
Name      Ram
Dept      ECE
Marks     500
Age        23
dtype: object
```

2. Most used Data structure in pandas is DataFrame



PROPERTIES OF DATAFRAME

1. A Dataframe has axes (indices)-
   - Row index (axis=0)
   - Column index (axes=1)
2. It is similar to a spreadsheet , whose row index is called index and column index is called column name.
3. A Dataframe contains Heterogeneous data.
4. A Dataframe Size is Mutable.
5. A Dataframe Data is Mutable.

A data frame can be created using any of the following-

1. Series
2. Lists
3. Dictionary
4. A numpy 2D array

How to create Empty Dataframe

```
: import pandas as pd
df=pd.DataFrame()
print(df)
```

```
Empty DataFrame
Columns: []
Index: []
```

In [193]:

```
#create dataframe from  series
#dc={'Name':'Ram','Dept':'ECE', 'Marks':500,'Age':23}
df=pd.DataFrame({'Name':['Ram','Raj','abc'],
                 'Dept':['ECE','CSE','me'],
                 'Marks':[500,550,89],
                 'Age':[20,19,23]})
df
```

Out[193]:

	Name	Dept	Marks	Age
0	Ram	ECE	500	20
1	Raj	CSE	550	19
2	abc	me	89	23

2.Creating of Dataframe ,row selction ,column selection

In [117]:
```
import pandas as pd
data = {
    'ID': ['101', '102', '103', '104'],  # Should be int
    'Price': ['$1,000', '$2,500', '$3,750', '$4,100'],  # Should be float
    'Date': ['2024-01-01', '2024-02-15', '2024-03-20', '2024-04-10'],  # Sho
    'Category': ['fashion', 'Furniture', 'Decor', 'Electricals']  # Should r
}

df=pd.DataFrame(data)
df.head()
```

Out[117]:

	ID	Price	Date	Category
0	101	$1,000	2024-01-01	fashion
1	102	$2,500	2024-02-15	Furniture
2	103	$3,750	2024-03-20	Decor
3	104	$4,100	2024-04-10	Electricals

2.2 Select operation in data frame

To access the column data ,we can mention the column name as subscript.

e.g. - df[Price] This can also be done by using df.Price.

To access multiple columns we can write as df[[col1, col2,---]]

In [120]:
```
# Access price column using select operator[]
#Note:Colunm name should be exactly same as in dataFrame(Case senstive)
df['Price']
```

Out[120]:
```
0    $1,000
1    $2,500
2    $3,750
3    $4,100
Name: Price, dtype: object
```

In [121]:
```
#Access category column using '.'(period)
df.Category
```

Out[121]:
```
0        fashion
1      Furniture
2          Decor
3    Electricals
Name: Category, dtype: object
```

3.Basic methods like head(),tail(),info() loc(),iloc(),describe(),dtypes,shape

In [61]:
```
# To check the basic information of the dataset
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4 entries, 0 to 3
Data columns (total 4 columns):
 #   Column    Non-Null Count  Dtype 
---  ------    --------------  ----- 
 0   ID        4 non-null      object
 1   Price     4 non-null      object
 2   Date      4 non-null      object
 3   Category  4 non-null      object
dtypes: object(4)
memory usage: 256.0+ bytes
```

In [62]:
```
#To check the No.of columns and rows in dataset
df.shape # Note: shape is attribute not a method
```

Out[62]:
```
(4, 4)
```

In [63]:
```
#To find the datatypes of all columns(variables) in DataFrame
df.dtypes # Note: dtypes is attribute not a method
```

Out[63]:
```
ID          object
Price       object
Date        object
Category    object
dtype: object
```

In [64]:
```
# to chcek the top 5 rows of dataframe
df.head()
```

Out[64]:

	ID	Price	Date	Category
0	101	$1,000	2024-01-01	fashion
1	102	$2,500	2024-02-15	Furniture
2	103	$3,750	2024-03-20	Decor
3	104	$4,100	2024-04-10	Electricals

In [65]:
```
#To chcek the last 5 rows of the dataframe
df.tail()
```

Out[65]:

	ID	Price	Date	Category
0	101	$1,000	2024-01-01	fashion
1	102	$2,500	2024-02-15	Furniture
2	103	$3,750	2024-03-20	Decor
3	104	$4,100	2024-04-10	Electricals

3. 1 Pandas provide loc() and iloc() methods to access the subset from a data frame using row/column.

Accessing the data frame through loc()

It is used to access a group of rows and columns.

Syntax- Df.loc[StartRow : EndRow, StartColumn : EndColumn]

Note -If we pass : in row or column part then pandas provide the entire rows or columns respectively.

In [66]:
```
#loc method take index for rows and col_names for columns
df.loc[2:5,'Price':'Category']# from 2to5 rows and Price to category column
```

Out[66]:

	Price	Date	Category
2	$3,750	2024-03-20	Decor
3	$4,100	2024-04-10	Electricals

Accessing the data frame through iloc()

It is used to access a group of rows and columns based on numeric index value.

Syntax- Df.loc[StartRowindexs : EndRowindex, StartColumnindex : EndColumnindex]

Note -If we pass : in row or column part then pandas provide the entire rows or columns respectively.

In [67]:
```
df.iloc[1:3,2:4]# from 1 to 3 rows and 2 to 4 columns(index based )
```

Out[67]:

	Date	Category
1	2024-02-15	Furniture
2	2024-03-20	Decor

4.Data Cleaning like type conversions,inconsident data fixing,

In [69]:
```
#convert ID colunm to Integer type
df['ID']=df['ID'].astype('int')
#Covert Price column to float datatype by removing some special charectors
df['Price']=df['Price'].replace({'\$': '',',':''},regex=True).astype(float)
#Convert Data column to Datatime datatype
df['Date']=pd.to_datetime(df['Date'])
#now the datatypes of columns are correctly fixed
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4 entries, 0 to 3
Data columns (total 4 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   ID        4 non-null      int32
 1   Price     4 non-null      float64
 2   Date      4 non-null      datetime64[ns]
 3   Category  4 non-null      object
dtypes: datetime64[ns](1), float64(1), int32(1), object(1)
memory usage: 240.0+ bytes
```

In [70]:
```
#to display the basic descriptive statistics of the dataframe for Numerical
df.describe()
```

Out[70]:

	ID	Price
count	4.000000	4.00000
mean	102.500000	2837.50000
std	1.290994	1404.38302
min	101.000000	1000.00000
25%	101.750000	2125.00000
50%	102.500000	3125.00000
75%	103.250000	3837.50000
max	104.000000	4100.00000

5. Concatination of Data Frames

In [71]:
```
df1 = pd.DataFrame({'ID': [1, 2,3,4,5],
                    'Name': ['Ali', 'Bobby','Ramesh','sakshi','sahil'],
                    'Age': [25, 30,23,20,24]})
df2 = pd.DataFrame({'ID': [6,7,8,9],
                    'Name': ['Cherry', 'Mahesh','Preethi','Santosh'],
                    'Age': [35, 30,26,28]})
df1.head()

```

Out[71]:

	ID	Name	Age
0	1	Ali	25
1	2	Bobby	30
2	3	Ramesh	23
3	4	sakshi	20
4	5	sahil	24

In [72]:
```
df2.head()
```

Out[72]:

	ID	Name	Age
0	6	Cherry	35
1	7	Mahesh	30
2	8	Preethi	26
3	9	Santosh	28

In [73]:
```
#combine 2 datafrmes row wise.
#df1 has 5 rows,df2 has 5 rows after concat total 10 rows
df_cat=pd.concat([df1,df2])# by default axis=0 (row wise)
df_cat

```

Out[73]:

	ID	Name	Age
0	1	Ali	25
1	2	Bobby	30
2	3	Ramesh	23
3	4	sakshi	20
4	5	sahil	24
0	6	Cherry	35
1	7	Mahesh	30
2	8	Preethi	26
3	9	Santosh	28

In [74]:
```
#combine data frames column wise
#if df1 has 3 columns and df2 has 2 columns after concat total has 5 columns
df3 = pd.DataFrame({'City': ['HYD', 'BEN','VIJ','CHE'],
                    'Salary': [50000, 60000,70000,34000]})
dff=pd.concat([df2,df3],axis=1)#column wise
dff
```

Out[74]:

	ID	Name	Age	City	Salary
0	6	Cherry	35	HYD	50000
1	7	Mahesh	30	BEN	60000
2	8	Preethi	26	VIJ	70000
3	9	Santosh	28	CHE	34000

6.Check for duplicate values and remove them

In [141]:
```
#Create dataframe with employee information
import numpy as np
data = {
    "ID": [1, 2, 2, 3, 4, 5, 6, 7, 8, 9],
    "Name": ["Amit", "Priya", "Priya", "Rahul", "Sneha", "Vikram", "Raj", "A
    "Age": [25, 30, 30, 34, -5, 40, 35, 29, 150, 28],
    "Salary (INR)": [500000, 540000, 540000, 620000, 720000, np.nan, 800000,
    "City": ["Mumbai", "Delhi", "Delhi", "Bangalore", "Chennai", "Chennai",
    "Joining Date": ["2022-01-15", "2021-08-20", "2021-08-20", "2020-06-30",
    "Department": ["HR", "Finance", "Finance", "IT", "HR", "HR", "IT", "Fina
}

df_emp=pd.DataFrame(data)
# make of copy of original dataframe
df_cp=df_emp.copy()
```

In [142]:
```
#methos to check no of duplicate records
df_emp.duplicated().sum()

```

Out[142]: 1

In [143]:
```
#method to remove duplicate records
df_emp=df_emp.drop_duplicates()
df_emp
```

Out[143]:

	ID	Name	Age	Salary (INR)	City	Joining Date	Department
0	1	Amit	25	500000.0	Mumbai	2022-01-15	HR
1	2	Priya	30	540000.0	Delhi	2021-08-20	Finance
3	3	Rahul	34	620000.0	Bangalore	2020-06-30	IT
4	4	Sneha	-5	720000.0	Chennai	2019-11-25	HR
5	5	Vikram	40	NaN	Chennai	2018-03-14	HR
6	6	Raj	35	800000.0	Kolkata	2017-09-10	IT
7	7	Ananya	29	920000.0	Pune	2016-07-04	Finance
8	8	Kiran	150	1030000.0	Hyderabad	2015-05-21	IT
9	9	Neha	28	1140000.0	Ahmedabad	2014-12-11	HR

7. apply() method in python

7.1 apply() method is used to perform any custom function on the dataframe or any part of dataframe like columns

In [144]:
```
# age column has negative values and also have values
#extreme  values like 150 years which is not correct in general
#lets fix that age values less than 0 and greater than 100 with its median
median_age = df_emp["Age"].median()

def age_correction(x):
    if x <= 0 or x > 100:
        return median_age
    return x

df_emp["Age"] = df_emp["Age"].apply(age_correction)
df_emp
```

Out[144]:

	ID	Name	Age	Salary (INR)	City	Joining Date	Department
0	1	Amit	25.0	500000.0	Mumbai	2022-01-15	HR
1	2	Priya	30.0	540000.0	Delhi	2021-08-20	Finance
3	3	Rahul	34.0	620000.0	Bangalore	2020-06-30	IT
4	4	Sneha	30.0	720000.0	Chennai	2019-11-25	HR
5	5	Vikram	40.0	NaN	Chennai	2018-03-14	HR
6	6	Raj	35.0	800000.0	Kolkata	2017-09-10	IT
7	7	Ananya	29.0	920000.0	Pune	2016-07-04	Finance
8	8	Kiran	30.0	1030000.0	Hyderabad	2015-05-21	IT
9	9	Neha	28.0	1140000.0	Ahmedabad	2014-12-11	HR

7.2 Lets see how to use lambda fucntion (anonymous function) using apply()

lambda is a anonymous function in handy without writing entire function description

In [145]:
```
#Make a copy of original dataframe first
df_cp=df_emp.copy()
#Now i want to increase salary by 10% for each employee using lamda function
df_cp['Salary (INR)']=df_cp['Salary (INR)'].apply(lambda x : x * 1.10)
df_cp
```

Out[145]:

	ID	Name	Age	Salary (INR)	City	Joining Date	Department
0	1	Amit	25.0	550000.0	Mumbai	2022-01-15	HR
1	2	Priya	30.0	594000.0	Delhi	2021-08-20	Finance
3	3	Rahul	34.0	682000.0	Bangalore	2020-06-30	IT
4	4	Sneha	30.0	792000.0	Chennai	2019-11-25	HR
5	5	Vikram	40.0	NaN	Chennai	2018-03-14	HR
6	6	Raj	35.0	880000.0	Kolkata	2017-09-10	IT
7	7	Ananya	29.0	1012000.0	Pune	2016-07-04	Finance
8	8	Kiran	30.0	1133000.0	Hyderabad	2015-05-21	IT
9	9	Neha	28.0	1254000.0	Ahmedabad	2014-12-11	HR

8. Check of missing values and handle them

In [146]:
```
#method to check for no of missing values
df_emp.isnull().sum()
```

Out[146]:
```
ID              0
Name            0
Age             0
Salary (INR)    1
City            0
Joining Date    0
Department      0
dtype: int64
```

We have 3 ways to handle missing values

1. Removing 2.Filling 3. Imputation

8.1 Remove or drop missing values

If the number of missing values are less when compared to size of dataframe we can remove them.If they are more we have to fill them or else we will lose information

Synatx : DataFrame.dropna(axis=0, how='any', thresh=None, subset=None, inplace=False)

df.dropna():Dropping rows with NaN values (default behavior)

df.dropna(axis=1):Dropping columns with NaN values

df.dropna(subset=col_name): Drop rows based on given column conatins Nan values

df.dropna(how=any/all):Drop if any row contain single Nan or all Nan values

df.dropna(thresh=2): drop rows with more than 2 missing values

In [147]:
```
#remove missing values
df_emp.dropna()
```

Out[147]:

	ID	Name	Age	Salary (INR)	City	Joining Date	Department
0	1	Amit	25.0	500000.0	Mumbai	2022-01-15	HR
1	2	Priya	30.0	540000.0	Delhi	2021-08-20	Finance
3	3	Rahul	34.0	620000.0	Bangalore	2020-06-30	IT
4	4	Sneha	30.0	720000.0	Chennai	2019-11-25	HR
6	6	Raj	35.0	800000.0	Kolkata	2017-09-10	IT
7	7	Ananya	29.0	920000.0	Pune	2016-07-04	Finance
8	8	Kiran	30.0	1030000.0	Hyderabad	2015-05-21	IT
9	9	Neha	28.0	1140000.0	Ahmedabad	2014-12-11	HR

8.2 Fill values with 0 or mean or median or mode

if you have more number of missing values we have to fill them,there are 2 ways to fill the missing values

1. Fill with '0'

2. Fill with mean or median or mode

In [148]:
```
#fill missing values with 0
df_cp['Salary (INR)'].fillna(0)
```

Out[148]:
```
0     550000.0
1     594000.0
3     682000.0
4     792000.0
5          0.0
6     880000.0
7    1012000.0
8    1133000.0
9    1254000.0
Name: Salary (INR), dtype: float64
```

In [149]:
```
#filling missing values with Median
df_emp['Salary (INR)']=df_emp['Salary (INR)'].fillna(df_emp['Salary (INR)']
                                                     .median())
df_emp
```

Out[149]:

	ID	Name	Age	Salary (INR)	City	Joining Date	Department
0	1	Amit	25.0	500000.0	Mumbai	2022-01-15	HR
1	2	Priya	30.0	540000.0	Delhi	2021-08-20	Finance
3	3	Rahul	34.0	620000.0	Bangalore	2020-06-30	IT
4	4	Sneha	30.0	720000.0	Chennai	2019-11-25	HR
5	5	Vikram	40.0	760000.0	Chennai	2018-03-14	HR
6	6	Raj	35.0	800000.0	Kolkata	2017-09-10	IT
7	7	Ananya	29.0	920000.0	Pune	2016-07-04	Finance
8	8	Kiran	30.0	1030000.0	Hyderabad	2015-05-21	IT
9	9	Neha	28.0	1140000.0	Ahmedabad	2014-12-11	HR

Type *Markdown* and LaTeX: α^2

7.3 Some advanced methods or algorithms like SimpleImputer or KNNImputer will be used to fill the missing values

Fill with SimpleImputer class

In [150]:
```
#imputation of missing values with SimpleImputer class
from sklearn.impute import SimpleImputer
#make sure to install sklearn package using command !pip install sklearn
import numpy as np
import pandas as pd

# lets create new sample data Sample Data
Sample = pd.DataFrame({
    "Age": [25, 30, np.nan, 40, 35],
    "Salary": [50000, np.nan, 60000, 65000, np.nan],
    "City": ["Delhi", "Mumbai", np.nan, "Chennai", "Delhi"]
})
#Note: np.nan is to say that it is missing value
Sample.info()


```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5 entries, 0 to 4
Data columns (total 3 columns):
 #   Column  Non-Null Count  Dtype  
---  ------  --------------  -----  
 0   Age     4 non-null      float64
 1   Salary  3 non-null      float64
 2   City    4 non-null      object 
dtypes: float64(2), object(1)
memory usage: 248.0+ bytes
```

In [151]:
```
#we have missing values in all 3 columns lets impute them with SimpleImputer
# Numerical columns filled with mean
sc = SimpleImputer(strategy="mean")
Sample[["Age","Salary"]]=num_imputer.fit_transform(Sample[["Age", "Salary"]]

# Categorical columns filled with mode
cat = SimpleImputer(strategy="most_frequent")
Sample[["City"]] = cat.fit_transform(Sample[["City"]])

Sample
```

Out[151]:

	Age	Salary	City
0	25.0	50000.000000	Delhi
1	30.0	58333.333333	Mumbai
2	32.5	60000.000000	Delhi
3	40.0	65000.000000	Chennai
4	35.0	58333.333333	Delhi

8. Sorting and Grouping of a dataframe

Take our employee dataset which is stored in df_emp and lets sort the employees and group the employess based on department and find the department wise average

In [152]:
```
#sort the employees from high to low salary
df_sorted=df_emp.sort_values(by='Salary (INR)',ascending=False)
df_sorted
```

Out[152]:

	ID	Name	Age	Salary (INR)	City	Joining Date	Department
9	9	Neha	28.0	1140000.0	Ahmedabad	2014-12-11	HR
8	8	Kiran	30.0	1030000.0	Hyderabad	2015-05-21	IT
7	7	Ananya	29.0	920000.0	Pune	2016-07-04	Finance
6	6	Raj	35.0	800000.0	Kolkata	2017-09-10	IT
5	5	Vikram	40.0	760000.0	Chennai	2018-03-14	HR
4	4	Sneha	30.0	720000.0	Chennai	2019-11-25	HR
3	3	Rahul	34.0	620000.0	Bangalore	2020-06-30	IT
1	2	Priya	30.0	540000.0	Delhi	2021-08-20	Finance
0	1	Amit	25.0	500000.0	Mumbai	2022-01-15	HR

Type *Markdown* and LaTeX: α^2

In [153]:
```
# group the data based on department department wise average salary
df_grp=df_emp.groupby(by='Department')['Salary (INR)'].mean()
df_grp
```

Out[153]:
```
Department
Finance    730000.000000
HR         780000.000000
IT         816666.666667
Name: Salary (INR), dtype: float64
```

Type *Markdown* and LaTeX: α^2

In [155]:
```
#lets check our final cleaned employee data
df_emp
```

Out[155]:

	ID	Name	Age	Salary (INR)	City	Joining Date	Department
0	1	Amit	25.0	500000.0	Mumbai	2022-01-15	HR
1	2	Priya	30.0	540000.0	Delhi	2021-08-20	Finance
3	3	Rahul	34.0	620000.0	Bangalore	2020-06-30	IT
4	4	Sneha	30.0	720000.0	Chennai	2019-11-25	HR
5	5	Vikram	40.0	760000.0	Chennai	2018-03-14	HR
6	6	Raj	35.0	800000.0	Kolkata	2017-09-10	IT
7	7	Ananya	29.0	920000.0	Pune	2016-07-04	Finance
8	8	Kiran	30.0	1030000.0	Hyderabad	2015-05-21	IT
9	9	Neha	28.0	1140000.0	Ahmedabad	2014-12-11	HR

In [156]:
```
#Save our employee data which cleaned in .csv file
df_emp.to_csv("emp_cleaned.csv", index=False)
# file will save in the current working folder
```

10. Check for outliers

Box plot is used to detect the outliers in your dataframe

In [1]:

```
#lets create one sample dataframe about house prices
import numpy as np
import pandas as pd


# Sample real-world-like dataset (House Prices)
data = {
"Price": [250000, 270000, 275000, 300000, 500000, 260000, 290000, 310000, 10
"Size_sqft": [1500, 1600, 1700, 1800, 2200, 1550, 1650, 1750, 5000, 1580, 18
"Bedrooms": [3, 3, 3, 4, 5, 13, 3, 4, 10, 3, 4, 4, 3, 12]
}

df_house = pd.DataFrame(data)

df_house
```

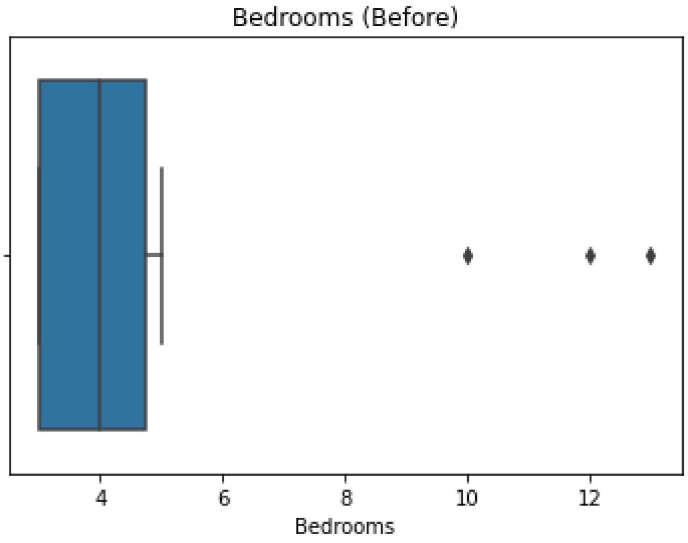
```
c:\users\vamsi2001\appdata\local\programs\python\python39\lib\site-packages\num
py\_distributor_init.py:30: UserWarning: loaded more than 1 DLL from .libs:
c:\users\vamsi2001\appdata\local\programs\python\python39\lib\site-packages\num
py\.libs\libopenblas.EL2C6PLE4ZYW3ECEVIV3OXXGRN2NRFM2.gfortran-win_amd64.dll
c:\users\vamsi2001\appdata\local\programs\python\python39\lib\site-packages\num
py\.libs\libopenblas64__v0.3.21-gcc_10_3_0.dll
  warnings.warn("loaded more than 1 DLL from .libs:"
```

Out[1]:

	Price	Size_sqft	Bedrooms
0	250000	1500	3
1	270000	1600	3
2	275000	1700	3
3	300000	1800	4
4	500000	2200	5
5	260000	1550	13
6	290000	1650	3
7	310000	1750	4
8	1000000	5000	10
9	270000	1580	3
10	320000	1850	4
11	340000	1900	4
12	255000	1520	3
13	6000000	7000	12

In [3]:
```
#Lets plot box plot to check for outliers
# Plot box plots before removing outliers
import matplotlib.pyplot as plt
import seaborn as sns

sns.boxplot(df_house["Bedrooms"])
plt.title("Bedrooms (Before)")
plt.show()
```

```
c:\users\vamsi2001\appdata\local\programs\python\python39\lib\site-packages\sea
born\_decorators.py:36: FutureWarning: Pass the following variable as a keyword
arg: x. From version 0.12, the only valid positional argument will be `data`, a
nd passing other arguments without an explicit keyword will result in an error
or misinterpretation.
  warnings.warn(
```



From the above box plot we can easily say that we have outliers in Bedrooms column of house dataframe, Lets handle that outliers using IQR

10.1 Handling outliers with IQR method

Interquartile Range (IQR)

IQR (Interquartile Range) is a statistical measure used to detect outliers in a dataset. It focuses on the middle 50% of the data and helps identify values that are significantly higher or lower than the rest.

7.1. Understanding Quartiles

To understand IQR, you need to know about quartiles. Quartiles divide sorted data into four equal parts:

- **Q1 (First Quartile - 25th Percentile)**: The median (middle) of the lower half of the data.
- **Q2 (Second Quartile - 50th Percentile or Median)**: The middle value of the dataset.

- **Q3 (Third Quartile - 75th Percentile)**: The median of the upper half of the data.

Example Data (Sorted)

```
[5, 7, 9, 12, 15, 18, 21, 25, 30, 35]
```

- **Q1 (25th percentile) = 9**
- **Q2 (50th percentile / median) = 15**
- **Q3 (75th percentile) = 25**

7.2. How to Calculate IQR

Formula:

[IQR = Q3 - Q1]

Using our example: [IQR = 25 - 9 = 16]

7.3. Detecting Outliers Using IQR

Outliers are values that are too far from the middle range. We define the lower and upper bounds to detect them:

Outlier Boundaries:

{Lower Bound} = Q1 - 1.5 * IQR

{Upper Bound} = Q3 + 1.5 * IQR

Applying it to Our Example:

{Lower Bound} = 9 - (1.5 * 16) = 9 - 24 = -15

{Upper Bound} = 25 + (1.5 * 16) = 25 + 24 = 49

Outliers:

Any value less than -15 or greater than 49 is considered an outlier.
Since our dataset `[5, 7, 9, 12, 15, 18, 21, 25, 30, 35]` has no values outside these bounds, there are **no outliers**.

7.4. Visualizing IQR with a Box Plot

A **box plot** (or **box-and-whisker plot**) is a graphical way to show IQR and outliers.

Box Plot Components:

- **Box** → Shows Q1, Q2 (Median), and Q3.

- **Whiskers** → Extend to the min and max values within the **IQR range**.
- **Dots (Outliers)** → Any values **outside** the lower and upper bounds.

Here's how we plot a box plot:

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

data = [5, 7, 9, 12, 15, 18, 21, 25, 30, 35, 100]  # 100 is an outlier
sns.boxplot(data=data)
plt.title("Box Plot Example")
plt.show()
```

7.5. Why Use IQR?

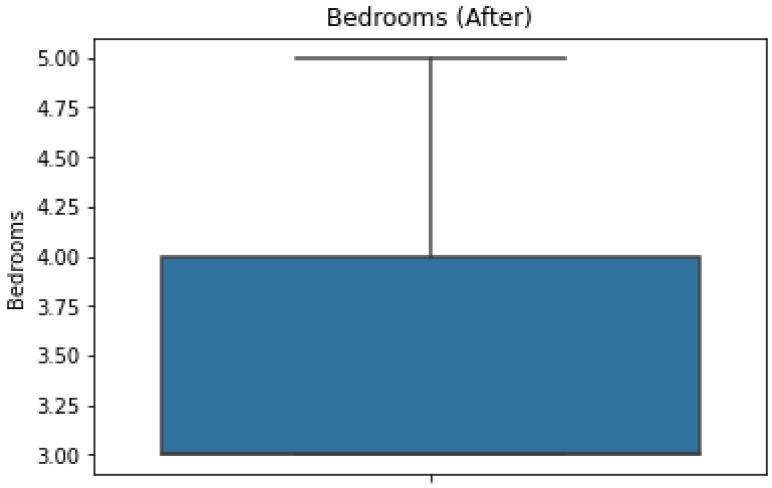
Better than Mean & Standard Deviation: Works well for **skewed** data and **non-normal distributions**.

Robust to Outliers: Unlike standard deviation, it **focuses on the middle 50%** of data, ignoring extreme values.

Widely Used in Data Science: Helps in **data cleaning**, **preprocessing**, and **anomaly detection**.

In [162]:
```
#lets remove outliers using IQR method
# Function to remove outliers using IQR
#lets create one sample dataframe about house prices
import numpy as np
import pandas as pd


# Sample real-world-like dataset (House Prices)
data = {
"Price": [250000, 270000, 275000, 300000, 500000, 260000, 290000, 310000, 10
"Size_sqft": [1500, 1600, 1700, 1800, 2200, 1550, 1650, 1750, 5000, 1580, 18
"Bedrooms": [3, 3, 3, 4, 5, 13, 3, 4, 10, 3, 4, 4, 3, 12]
}

df_house = pd.DataFrame(data)

df_house
Q1 = df_house['Bedrooms'].quantile(0.25)
Q3 = df_house['Bedrooms'].quantile(0.75)
IQR = Q3 - Q1

lb = Q1 - 1.5* IQR
ub = Q3 + 1.5 * IQR
df_house=df_house[(df_house['Bedrooms'] >= lb) &
                  (df_house['Bedrooms'] <= ub)]


```

In [163]:
```
# after handling again once check for outliers
#plt.subplot(1, 3, 3)
sns.boxplot(y=df_house["Bedrooms"])
plt.title("Bedrooms (After)")

#plt.tight_layout()
plt.show()

```

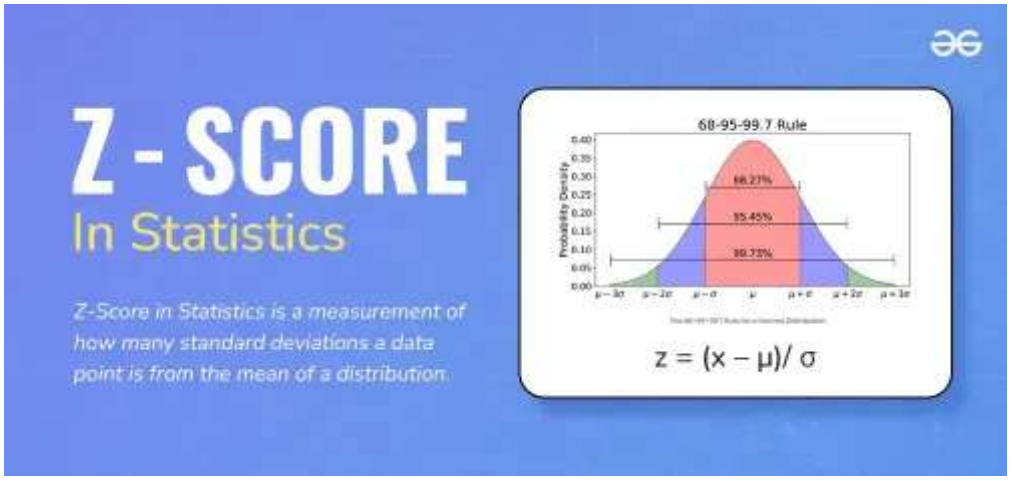


10.2 Automatic outlier detection techniques

There are some algorithms in python to detec and handle the outliers automatically like Z-score ,IsolationForest etc but as a beginner lets try with z-score method

Z-score is one of the technique used to detect and handle outliers automatically.

Z-Score tells how far a value is from the mean in units of standard deviation.



Values with |z| > 3 are usually considered outliers.

In [171]:
```
# make sure to install scipy package before importing
# command to install is !pip install scipy
from scipy.stats import zscore
df_house_cp = df_house.copy()

# Calculate z-scores only for numeric columns
z_scores = df_house_cp.apply(zscore)

# Mark outliers
df_house_cp["Outlier"] = (z_scores.abs() > 3).any(axis=1)

df_house_cp

```

Out[171]:

	Price	Size_sqft	Bedrooms	Outlier
0	250000	1500	3	False
1	270000	1600	3	False
2	275000	1700	3	False
3	300000	1800	4	False
4	500000	2200	5	False
6	290000	1650	3	False
7	310000	1750	4	False
9	270000	1580	3	False
10	320000	1850	4	False
11	340000	1900	4	False
12	255000	1520	3	False

In [ ]:
```
#As 5,8 records have outliers they got removed my z-score method ,now the
#outliers are removed
```

11. Merging and filtering of Dataframes

In [22]:
```
#Create 2 sample DataFrames
import pandas as pd

products = pd.DataFrame({
    "product_id": [101, 102, 103, 104],
    "product_name": ["Laptop", "Mobile", "Headphones", "Keyboard"],
    "category": ["Electronics", "Electronics", "Accessories", "Accessories"]
})

orders = pd.DataFrame({
    "order_id": [1, 2, 3, 4, 5],
    "product_id": [101, 102, 103, 101, 104],
    "quantity": [1, 2, 1, 1, 3],
    "order_value": [60000, 40000, 2000, 60000, 4500]
})
```

In [23]:
```
#join or merge the Dataframes
#Inner join means only common records in both the dataframes will be resultd
merged_df = pd.merge(orders,products,on="product_id",how="inner")

merged_df

```

Out[23]:

	order_id	product_id	quantity	order_value	product_name	category
0	1	101	1	60000	Laptop	Electronics
1	4	101	1	60000	Laptop	Electronics
2	2	102	2	40000	Mobile	Electronics
3	3	103	1	2000	Headphones	Accessories
4	5	104	3	4500	Keyboard	Accessories

In [24]:
```
#Filtering on data based on condition
ele_orders = merged_df[merged_df["category"] == "Electronics"]
ele_orders

```

Out[24]:

	order_id	product_id	quantity	order_value	product_name	category
0	1	101	1	60000	Laptop	Electronics
1	4	101	1	60000	Laptop	Electronics
2	2	102	2	40000	Mobile	Electronics

In [27]:
```
#Filtering based on one than one condition
filtered_orders = merged_df[
    (merged_df["category"] == "Accessories") &
    (merged_df["order_value"] >= 4000)
]

filtered_orders

```

Out[27]:

	order_id	product_id	quantity	order_value	product_name	category
4	5	104	3	4500	Keyboard	Accessories

12. Loading of datasets using pandas ,data can be available in diffrent file formates like .csv,.xls,.json...

In [177]:
```
#let us load iris( flower) data set .Make sure file avaible in
#current working folder or else give complete path
import pandas as pd
df_iris=pd.read_csv('iris.csv')
df_iris.head()
```

Out[177]:

	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5.0	3.6	1.4	0.2	setosa

In [178]:
```
#load employee data which is in excel format
emp=pd.read_excel('sample-staff.xlsx')
emp.head()

```

Out[178]:

	Emp ID	Name	Gender	Department	Salary	Start Date	FTE	Employee type	Work location
0	PR00147	Minerva Ricardot	Male	???	120000.00	12-Nov-18	1.0	Permanent	Remote
1	PR04686	Oona Donan	Female	Business Development	98000.00	2019-09-02 00:00:00	0.9	Permanent	Seattle, USA
2	SQ04612	Mick Spraberry	Female	Services	120000.00	2020-03-12 00:00:00	0.9	Permanent	Remote
3	VT01803	Freddy Linford	Female	Training	93128.34	Mar 5, 2018	1.0	Fixed Term	Seattle, USA
4	TN02749	Parasuramudu Jamakayala	Female	Training	57002.02	2-Apr-18	0.7	Permanent	Hyderabad, India

In [179]:
```
#let us load the json data using pandas
df_js=pd.read_json('sample.json')
df_js.head()
```

Out[179]:

	OrderID	CustomerName	ProductName	Category	Quantity	UnitPrice	TotalPrice	OrderDate	
0	1001	Rajesh Sharma	Smartphone	Electronics	1	20000	20000	2023-10-01	
1	1002	Priya Mehra	Laptop	Electronics	1	45000	45000	2023-10-05	
2	1003	Amit Trivedi	Office Chair	Furniture	2	3000	6000	2023-10-08	Be
3	1004	Sneha Iyer	Running Shoes	Sports	1	3500	3500	2023-10-10	
4	1005	Deepak Joshi	Refrigerator	Appliances	1	18000	18000	2023-10-12	

12.2 Loading of existing or inbuilt datasets using seaborn

Seaborn has inbuilt datasets for practice let us see how to load that kind of datasets like iris,tips,titanic,flights,mpg,penguins,taxis etc..

In [12]:
```
import seaborn as sns
dt=sns.load_dataset('tips')# instead of tips you can use any other datasets
dt.head()
```

Out[12]:

	total_bill	tip	sex	smoker	day	time	size
0	16.99	1.01	Female	No	Sun	Dinner	2
1	10.34	1.66	Male	No	Sun	Dinner	3
2	21.01	3.50	Male	No	Sun	Dinner	3
3	23.68	3.31	Male	No	Sun	Dinner	2
4	24.59	3.61	Female	No	Sun	Dinner	4

13. Encoding of a Variable (Feature)

Encoding of a varibel means converting categorical data into numerical data

Encoding can be done different ways based on the dataset

Two of the most used techniques are

1. One Hot encoding

2. LabelEncoding

In the above tips dataset Sex,Smoker,Day,time are categorical data so lets encode that variable

In [13]:
```
1 dt.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 244 entries, 0 to 243
Data columns (total 7 columns):
 #   Column      Non-Null Count  Dtype   
---  ------      --------------  -----   
 0   total_bill  244 non-null    float64 
 1   tip         244 non-null    float64 
 2   sex         244 non-null    category
 3   smoker      244 non-null    category
 4   day         244 non-null    category
 5   time        244 non-null    category
 6   size        244 non-null    int64   
dtypes: category(4), float64(2), int64(1)
memory usage: 7.4 KB
```

In [14]:
```
1 #1. One Hot Encoding
2 import pandas as pd
3 dt= pd.get_dummies(dt, columns=["sex", "smoker"])
4 dt.head()
5 
```

Out[14]:

	total_bill	tip	day	time	size	sex_Male	sex_Female	smoker_Yes	smoker_No
0	16.99	1.01	Sun	Dinner	2	0	1	0	1
1	10.34	1.66	Sun	Dinner	3	1	0	0	1
2	21.01	3.50	Sun	Dinner	3	1	0	0	1
3	23.68	3.31	Sun	Dinner	2	1	0	0	1
4	24.59	3.61	Sun	Dinner	4	0	1	0	1

But one Hot ending leads to more number of colums with zeros.One-Hot Encoding creates separate binary columns for each category. It leads to sparsity of the data. It is widely used for Nominal data . So we have to another type called Label Encoding

In [16]:
```
#2. LabelEncoding
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()

dt["day"] = le.fit_transform(dt["day"])
dt["time"] = le.fit_transform(dt["time"])

dt.head()

```

Out[16]:

	total_bill	tip	day	time	size	sex_Male	sex_Female	smoker_Yes	smoker_No
0	16.99	1.01	2	0	2	0	1	0	1
1	10.34	1.66	2	0	3	1	0	0	1
2	21.01	3.50	2	0	3	1	0	0	1
3	23.68	3.31	2	0	2	1	0	0	1
4	24.59	3.61	2	0	4	0	1	0	1

Label Encoder is used when you have ordinal data ,it is used when order of that variable matters

Pandas Excerice (DIY)

Every student should select one real time dataset and clean the dataset and save the cleaned dataset

You have to perform all possible and suitable techniques demonstrated above for your selected dataset

Real Time datasets can be available in the platforms like Kaggle,UCIRepository,data.gov etc..

@@@@@@@@@@@@ Thank You @@@@@@@@@@

@@@@@@@@@@ Happy Learning @@@@@@@@@@@

Type *Markdown* and LaTeX: α^2

In [ ]:
```

```