Python Pandas - Advanced

Agenda

- Concatenate and Join data frames (Pre-requisite)
- Merge data frames
- Reshape
- Pivot Tables and Cross Tables
- Check for Duplicates
- Dropping Rows and Columns
- Mapping and Replacing
- Group the Dataframe
- Summary Statistics and Skewness/Kurtosis
- Data Visualization using Matplotlib library

Concatenate the DataFrames

- A Pandas DataFrame is a two dimensional size-mutable, heterogeneous data structure with labeled rows and columns

- DataFrames can be concatenated vertically (column-wise) and horizontally (row-wise)

- The concat() and append() methods are used to concatenate the DataFrames

Join the DataFrames

- The join() method join the DataFrames based on index or key column
- Index of the first DataFrame should match to one of the column in the second DataFrame

Merge the DataFrames

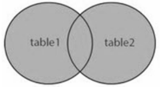
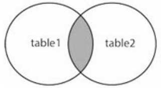
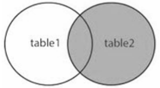
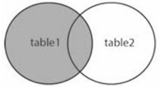
Merge the DataFrames

- The merge() method concatenates the DataFrames based on one or more keys

- If the column for join is not specified, the merge() method uses the overlapping column names as the keys

Types of merge

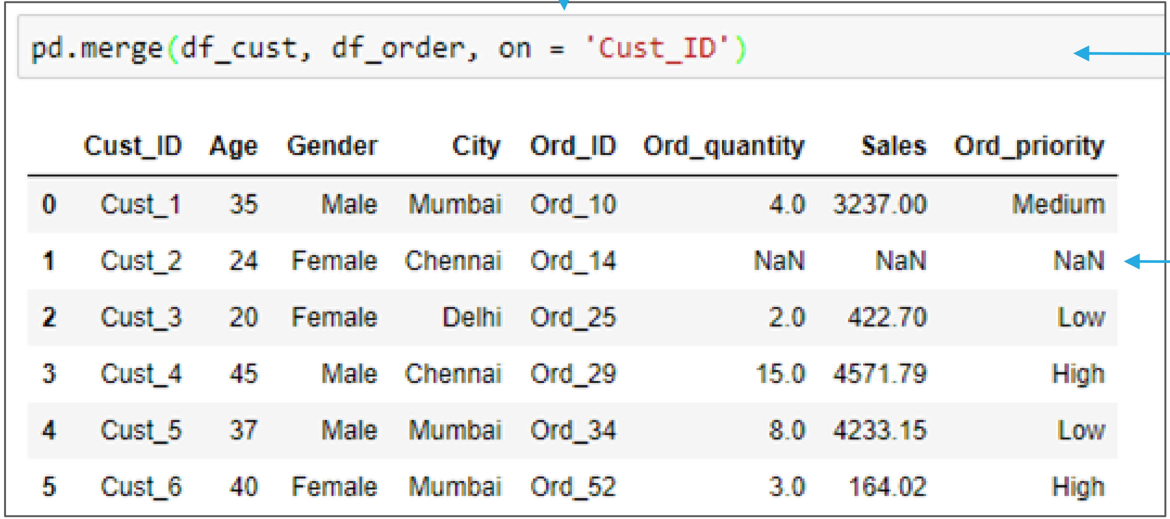
The merge types can be specified using the parameter, ‘how’

how = ‘Type’	Description	
outer	Use union of keys observed in both DataFrames	
inner	Use intersection of keys observed in both DataFrames	
right	Use only the keys found in the right DataFrame	
left	Use only the keys found in the left DataFrame	

If the type is not specified, by default it is ‘inner’

Inner merge

Merge both the DataFrames on common customer IDs



```
pd.merge(df_cust, df_order, on = 'Cust_ID')
```

	Cust_ID	Age	Gender	City	Ord_ID	Ord_quantity	Sales	Ord_priority
0	Cust_1	35	Male	Mumbai	Ord_10	4.0	3237.00	Medium
1	Cust_2	24	Female	Chennai	Ord_14	NaN	NaN	NaN
2	Cust_3	20	Female	Delhi	Ord_25	2.0	422.70	Low
3	Cust_4	45	Male	Chennai	Ord_29	15.0	4571.79	High
4	Cust_5	37	Male	Mumbai	Ord_34	8.0	4233.15	Low
5	Cust_6	40	Female	Mumbai	Ord_52	3.0	164.02	High

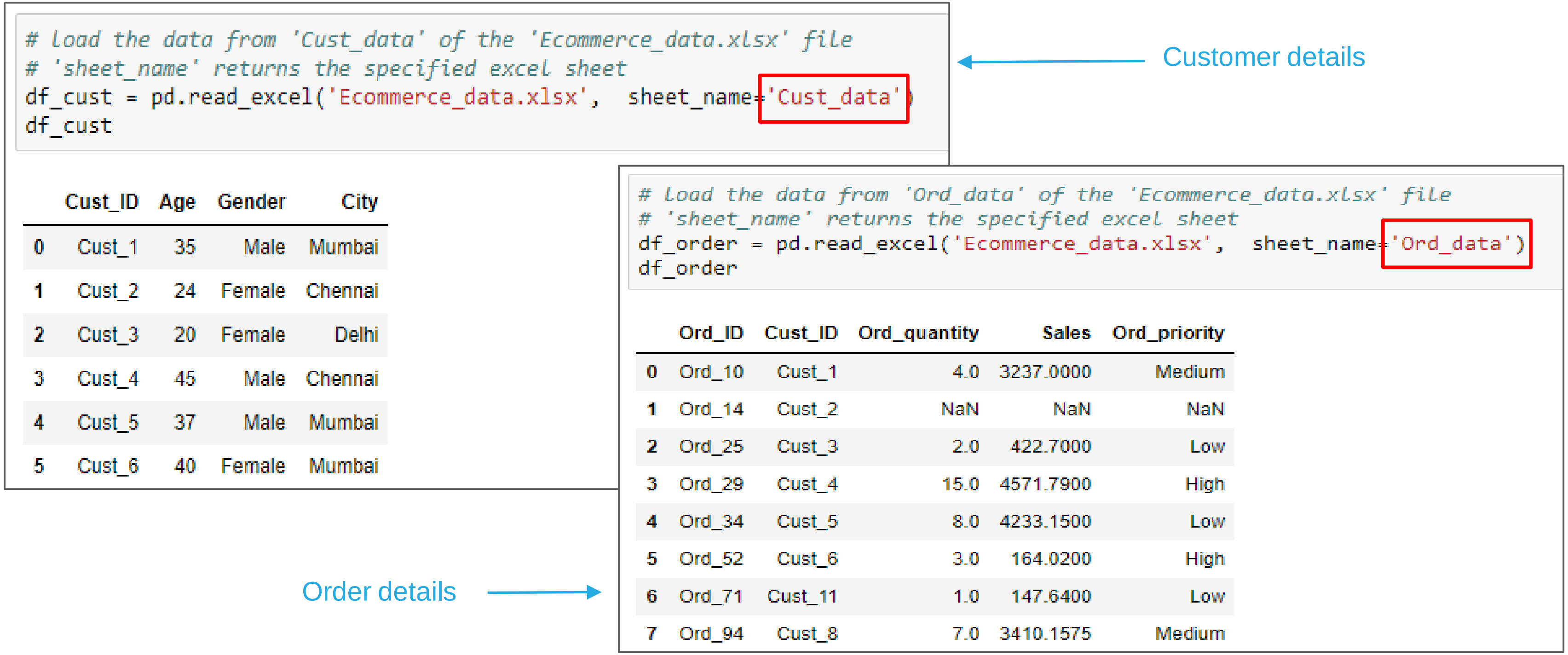
Merge on ‘Cust_ID’

Merge includes the common IDs in both the DataFrames

NaNs are printed where order details are not available

Read the DataFrames

Use the following DataFrames for further manipulations:



```
# load the data from 'Cust_data' of the 'Ecommerce_data.xlsx' file
# 'sheet_name' returns the specified excel sheet
df_cust = pd.read_excel('Ecommerce_data.xlsx',  sheet_name='Cust_data')
df_cust
```

	Cust_ID	Age	Gender	City
0	Cust_1	35	Male	Mumbai
1	Cust_2	24	Female	Chennai
2	Cust_3	20	Female	Delhi
3	Cust_4	45	Male	Chennai
4	Cust_5	37	Male	Mumbai
5	Cust_6	40	Female	Mumbai

Customer details

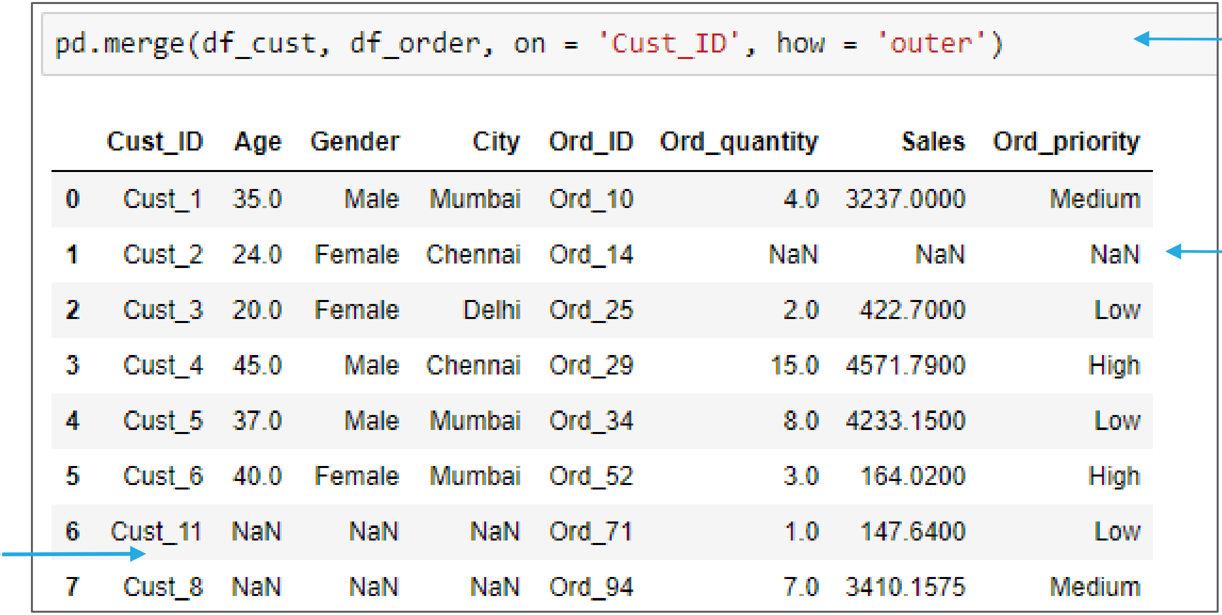
```
# load the data from 'Ord_data' of the 'Ecommerce_data.xlsx' file
# 'sheet_name' returns the specified excel sheet
df_order = pd.read_excel('Ecommerce_data.xlsx',  sheet_name='Ord_data')
df_order
```

	Ord_ID	Cust_ID	Ord_quantity	Sales	Ord_priority
0	Ord_10	Cust_1	4.0	3237.0000	Medium
1	Ord_14	Cust_2	NaN	NaN	NaN
2	Ord_25	Cust_3	2.0	422.7000	Low
3	Ord_29	Cust_4	15.0	4571.7900	High
4	Ord_34	Cust_5	8.0	4233.1500	Low
5	Ord_52	Cust_6	3.0	164.0200	High
6	Ord_71	Cust_11	1.0	147.6400	Low
7	Ord_94	Cust_8	7.0	3410.1575	Medium

Order details

Outer merge

Merge on ‘Cust_ID’



```
pd.merge(df_cust, df_order, on = 'Cust_ID', how = 'outer')
```

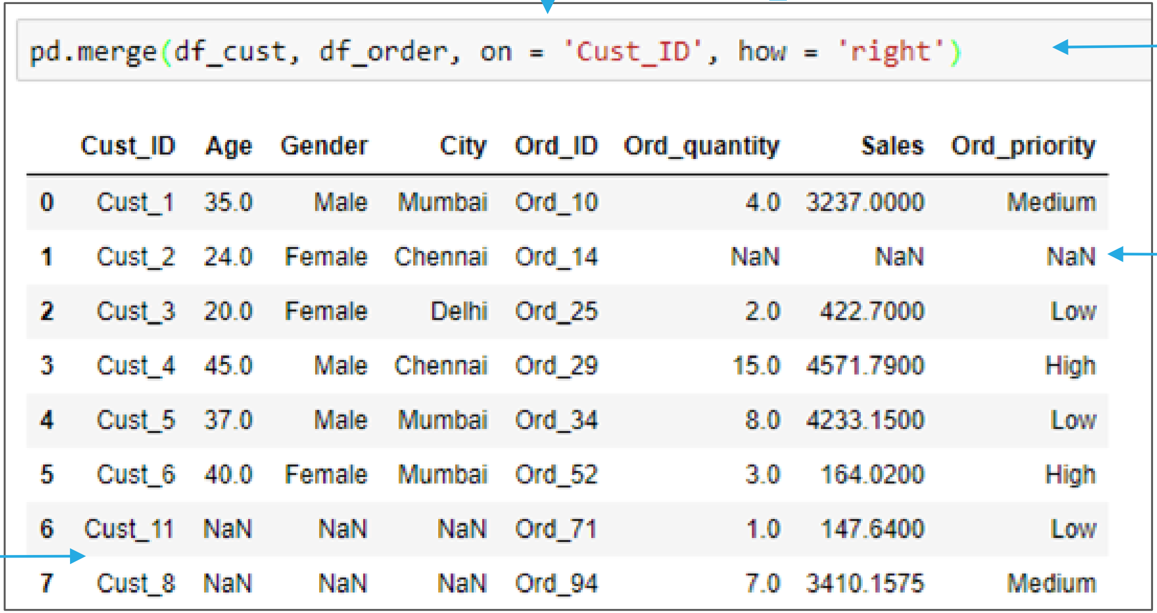
	Cust_ID	Age	Gender	City	Ord_ID	Ord_quantity	Sales	Ord_priority
0	Cust_1	35.0	Male	Mumbai	Ord_10	4.0	3237.0000	Medium
1	Cust_2	24.0	Female	Chennai	Ord_14	NaN	NaN	NaN
2	Cust_3	20.0	Female	Delhi	Ord_25	2.0	422.7000	Low
3	Cust_4	45.0	Male	Chennai	Ord_29	15.0	4571.7900	High
4	Cust_5	37.0	Male	Mumbai	Ord_34	8.0	4233.1500	Low
5	Cust_6	40.0	Female	Mumbai	Ord_52	3.0	164.0200	High
6	Cust_11	NaN	NaN	NaN	Ord_71	1.0	147.6400	Low
7	Cust_8	NaN	NaN	NaN	Ord_94	7.0	3410.1575	Medium

Outer merge includes the IDs in both DataFrames

NaNs are printed where order details are not available

NaNs are printed where customer details are not available

Right merge



```
pd.merge(df_cust, df_order, on = 'Cust_ID', how = 'right')
```

	Cust_ID	Age	Gender	City	Ord_ID	Ord_quantity	Sales	Ord_priority
0	Cust_1	35.0	Male	Mumbai	Ord_10	4.0	3237.0000	Medium
1	Cust_2	24.0	Female	Chennai	Ord_14	NaN	NaN	NaN
2	Cust_3	20.0	Female	Delhi	Ord_25	2.0	422.7000	Low
3	Cust_4	45.0	Male	Chennai	Ord_29	15.0	4571.7900	High
4	Cust_5	37.0	Male	Mumbai	Ord_34	8.0	4233.1500	Low
5	Cust_6	40.0	Female	Mumbai	Ord_52	3.0	164.0200	High
6	Cust_11	NaN	NaN	NaN	Ord_71	1.0	147.6400	Low
7	Cust_8	NaN	NaN	NaN	Ord_94	7.0	3410.1575	Medium

Merge on ‘Cust_ID’

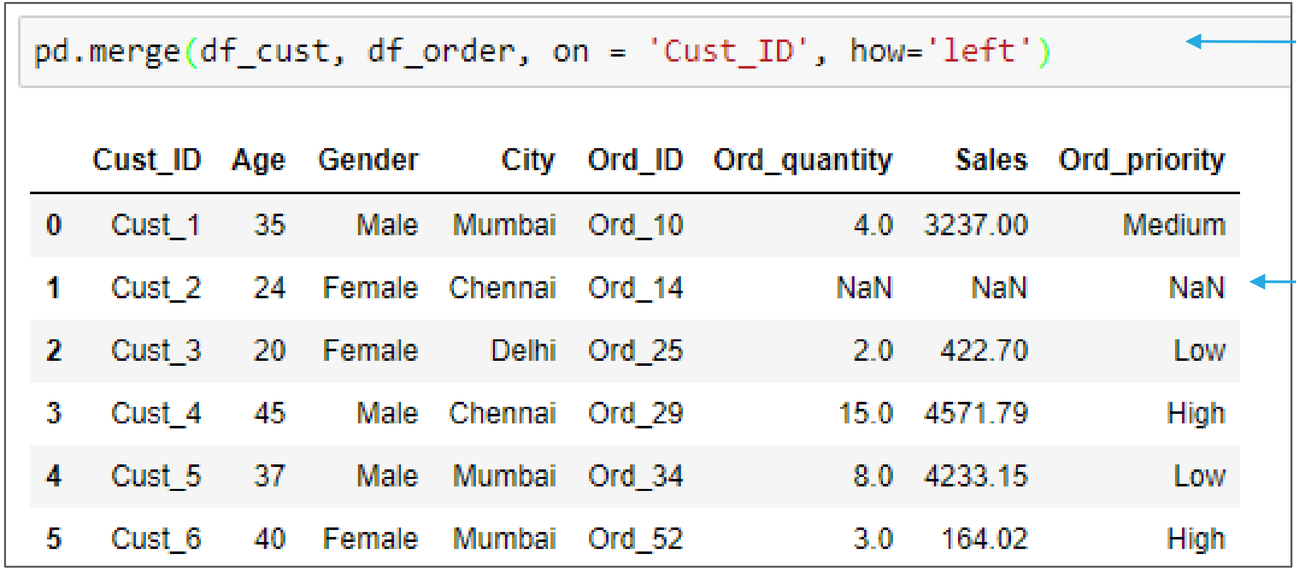
Merge includes all the IDs in ‘df_order’

NaNs are printed where order details are not available

NaNs are printed where customer details are not available

Left merge

Merge on ‘Cust_ID’



```
pd.merge(df_cust, df_order, on = 'Cust_ID', how='left')
```

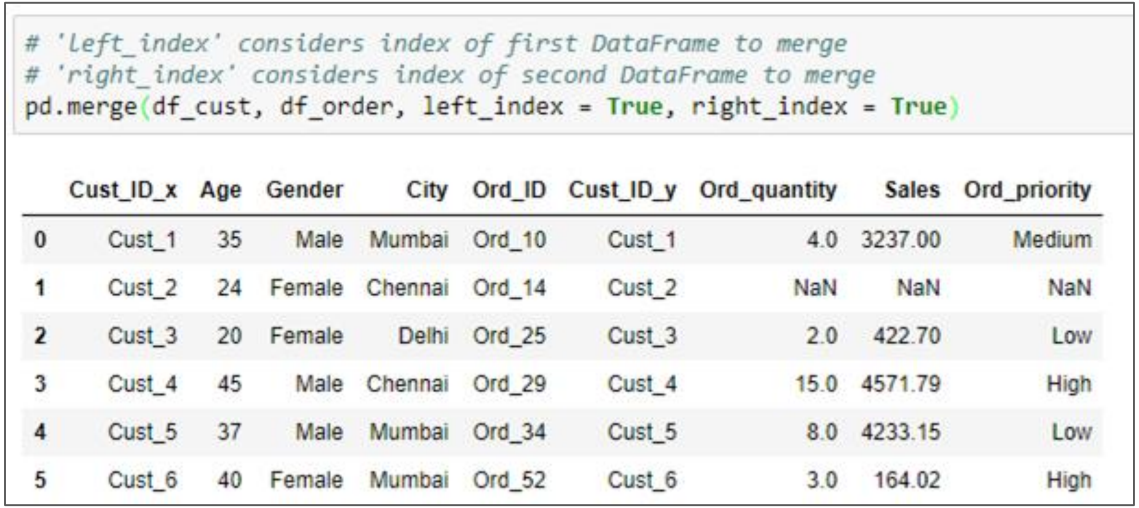
	Cust_ID	Age	Gender	City	Ord_ID	Ord_quantity	Sales	Ord_priority
0	Cust_1	35	Male	Mumbai	Ord_10	4.0	3237.00	Medium
1	Cust_2	24	Female	Chennai	Ord_14	NaN	NaN	NaN
2	Cust_3	20	Female	Delhi	Ord_25	2.0	422.70	Low
3	Cust_4	45	Male	Chennai	Ord_29	15.0	4571.79	High
4	Cust_5	37	Male	Mumbai	Ord_34	8.0	4233.15	Low
5	Cust_6	40	Female	Mumbai	Ord_52	3.0	164.02	High

Merge includes all the IDs in ‘df_cust’

NaNs are printed where order details are not available

Merge using index

- Merged DataFrame has the number of rows equal to that of the minimum of both the DataFrames. It includes rows from both DataFrames having same index
- This method is useful, only if the record have same index in both the DataFrames



```
# 'left_index' considers index of first DataFrame to merge
# 'right_index' considers index of second DataFrame to merge
pd.merge(df_cust, df_order, left_index = True, right_index = True)
```

	Cust_ID_x	Age	Gender	City	Ord_ID	Cust_ID_y	Ord_quantity	Sales	Ord_priority
0	Cust_1	35	Male	Mumbai	Ord_10	Cust_1	4.0	3237.00	Medium
1	Cust_2	24	Female	Chennai	Ord_14	Cust_2	NaN	NaN	NaN
2	Cust_3	20	Female	Delhi	Ord_25	Cust_3	2.0	422.70	Low
3	Cust_4	45	Male	Chennai	Ord_29	Cust_4	15.0	4571.79	High
4	Cust_5	37	Male	Mumbai	Ord_34	Cust_5	8.0	4233.15	Low
5	Cust_6	40	Female	Mumbai	Ord_52	Cust_6	3.0	164.02	High

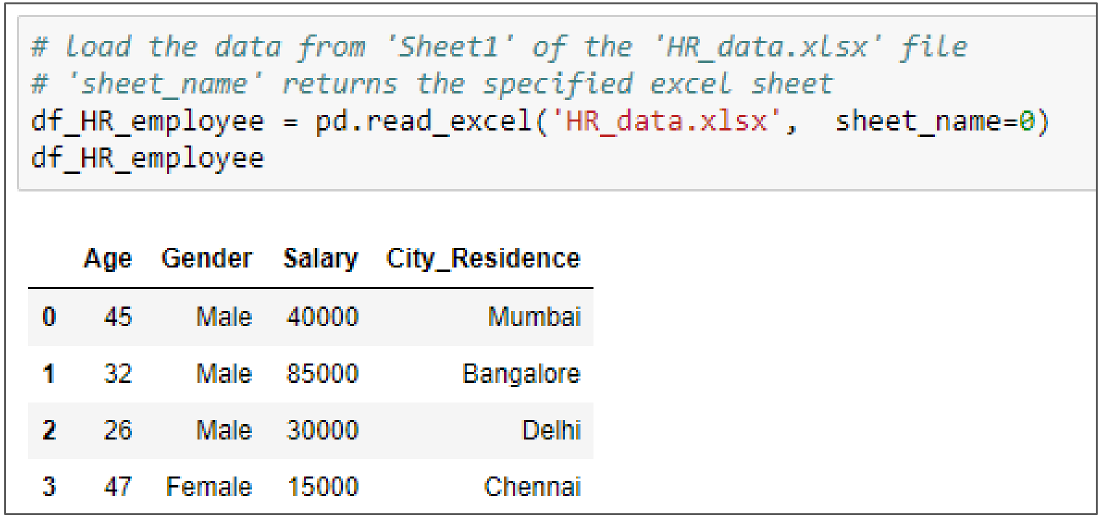
Merge vs. Join

Merge	Join
Joins one or more columns of the second DataFrame	Joins by the index of the second DataFrame
By default, performs ‘inner’ merge	By default, performs ‘Left’ join
Returns error if one tries to merge more than two DataFrames simultaneously	Joins multiple DataFrames by index

Reshape

Read the DataFrames

Use the following DataFrame for further manipulations:

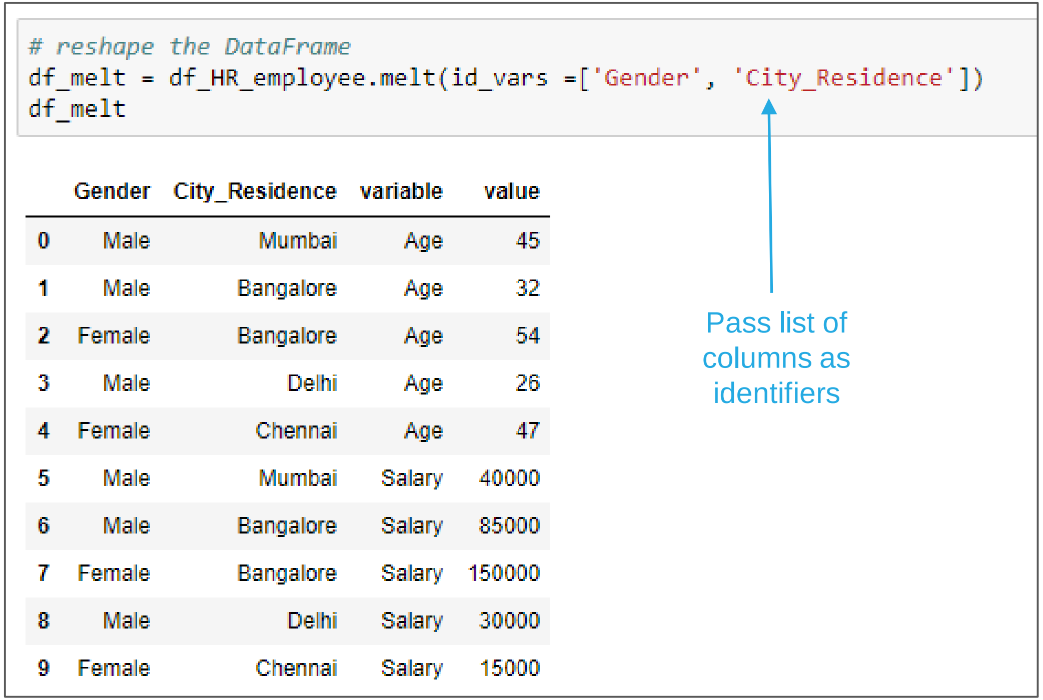


```
# load the data from 'Sheet1' of the 'HR_data.xlsx' file
# 'sheet_name' returns the specified excel sheet
df_HR_employee = pd.read_excel('HR_data.xlsx',  sheet_name=0)
df_HR_employee
```

	Age	Gender	Salary	City_Residence
0	45	Male	40000	Mumbai
1	32	Male	85000	Bangalore
2	26	Male	30000	Delhi
3	47	Female	15000	Chennai

Reshape

- The melt() method is used to change the DataFrame format from wide to long
- The column ‘variable’ contains all the columns except the identifiers and ‘value’ contains the values of corresponding column



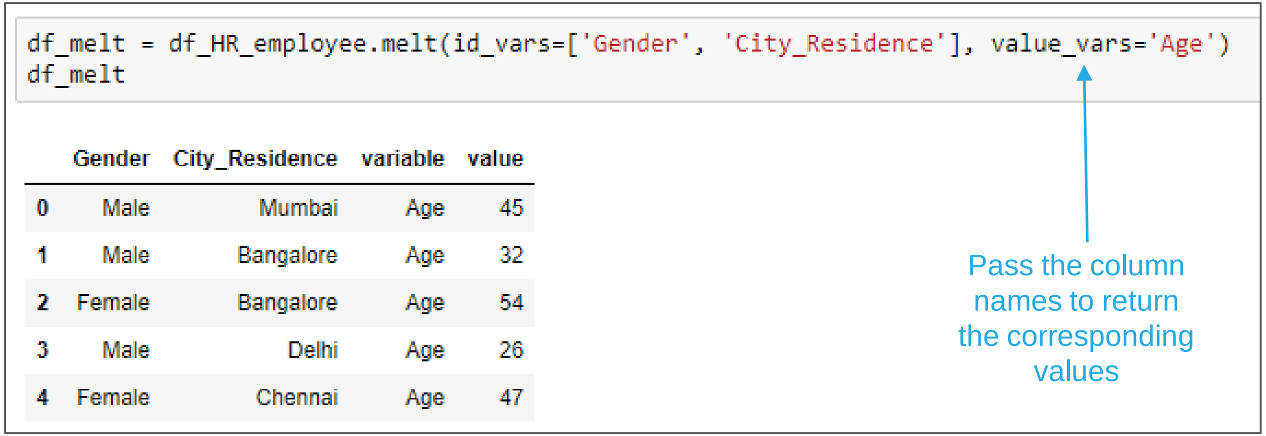
```
# reshape the DataFrame
df_melt = df_HR_employee.melt(id_vars =['Gender', 'City_Residence'])
df_melt
```

	Gender	City_Residence	variable	value
0	Male	Mumbai	Age	45
1	Male	Bangalore	Age	32
2	Female	Bangalore	Age	54
3	Male	Delhi	Age	26
4	Female	Chennai	Age	47
5	Male	Mumbai	Salary	40000
6	Male	Bangalore	Salary	85000
7	Female	Bangalore	Salary	150000
8	Male	Delhi	Salary	30000
9	Female	Chennai	Salary	15000

Pass list of columns as identifiers

Reshape

Assign the variables to the parameter, ‘value_vars’ to get the corresponding values for specified identifiers



```
df_melt = df_HR_employee.melt(id_vars=['Gender', 'City_Residence'], value_vars='Age')
df_melt
```

	Gender	City_Residence	variable	value
0	Male	Mumbai	Age	45
1	Male	Bangalore	Age	32
2	Female	Bangalore	Age	54
3	Male	Delhi	Age	26
4	Female	Chennai	Age	47

Pass the column names to return the corresponding values

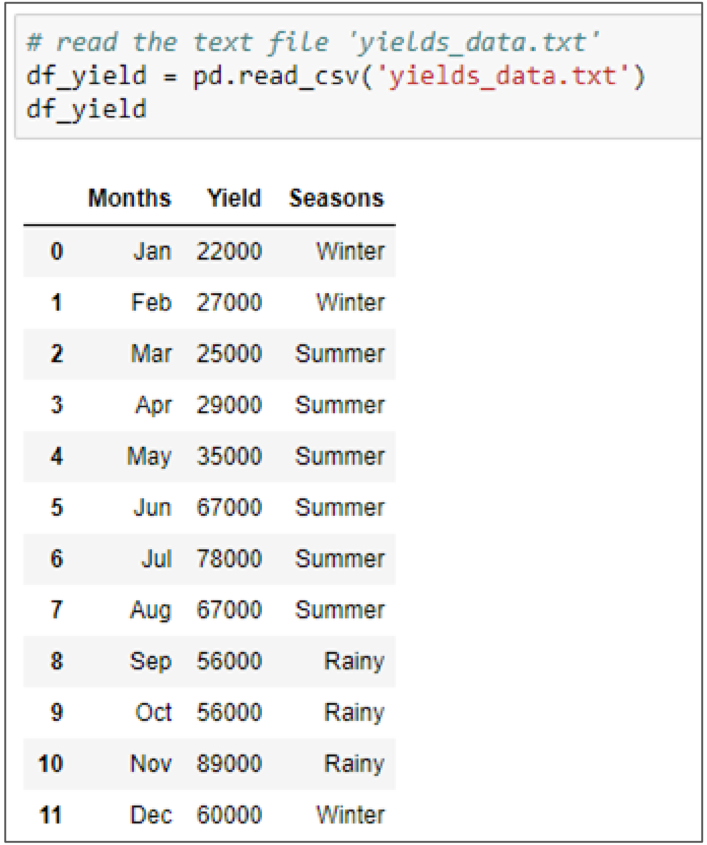
Pivot Tables

Pivot tables

- It has a DataFrame like structure
- It is used to display the data for the specified columns and index

Read the DataFrame

Use the following DataFrame to create a pivot table

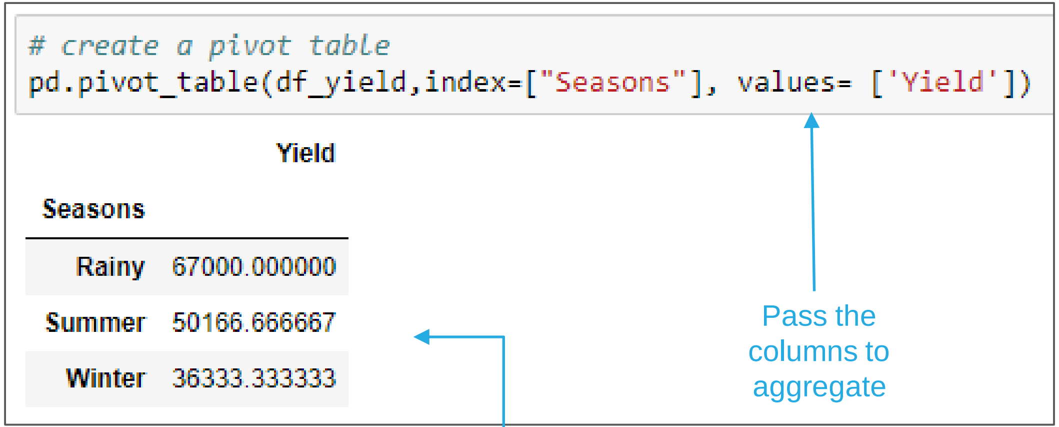


```
# read the text file 'yields_data.txt'
df_yield = pd.read_csv('yields_data.txt')
df_yield
```

	Months	Yield	Seasons
0	Jan	22000	Winter
1	Feb	27000	Winter
2	Mar	25000	Summer
3	Apr	29000	Summer
4	May	35000	Summer
5	Jun	67000	Summer
6	Jul	78000	Summer
7	Aug	67000	Summer
8	Sep	56000	Rainy
9	Oct	56000	Rainy
10	Nov	89000	Rainy
11	Dec	60000	Winter

Create a pivot table

- The pivot_table() method generates a pivot table for the given index
- By default, the aggregate function is ‘mean’, which aggregates the columns passed in the parameter, ‘values’



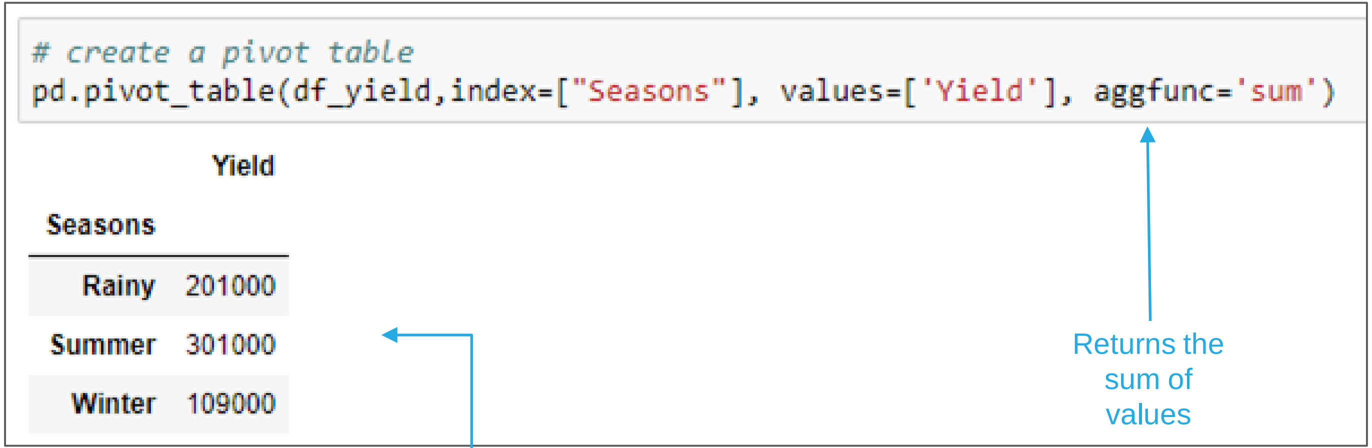
```
# create a pivot table
pd.pivot_table(df_yield,index=["Seasons"], values= ['Yield'])
```

```
                Yield
Seasons
  Rainy  67000.000000
 Summer  50166.666667
 Winter  36333.333333
```

Pass the columns to aggregate

Average yield per season

Create a pivot table



```
# create a pivot table
pd.pivot_table(df_yield,index=["Seasons"], values=['Yield'], aggfunc='sum')
```

```
          Yield
Seasons
  Rainy  201000
 Summer  301000
 Winter  109000
```

Returns the sum of values

Sum of yield per season

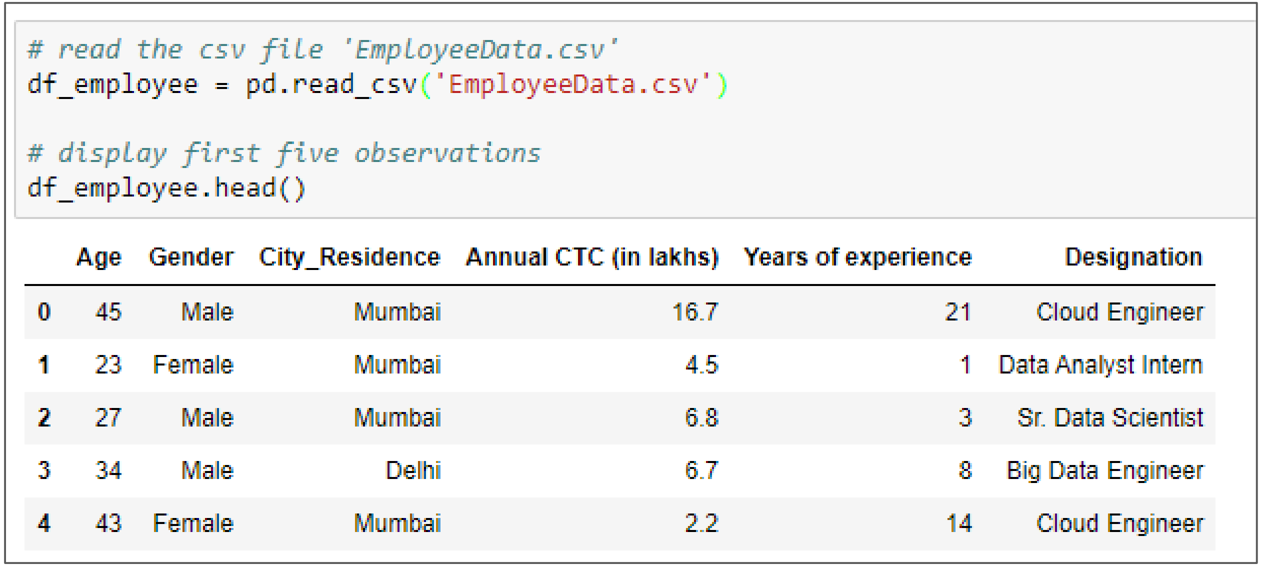
Cross Tables

Cross tables

- Cross tables are similar to pivot tables
- It computes a cross tabulation of two or more factors

Read the DataFrame

Read the csv file ‘EmployeeData’ and print the first five observations

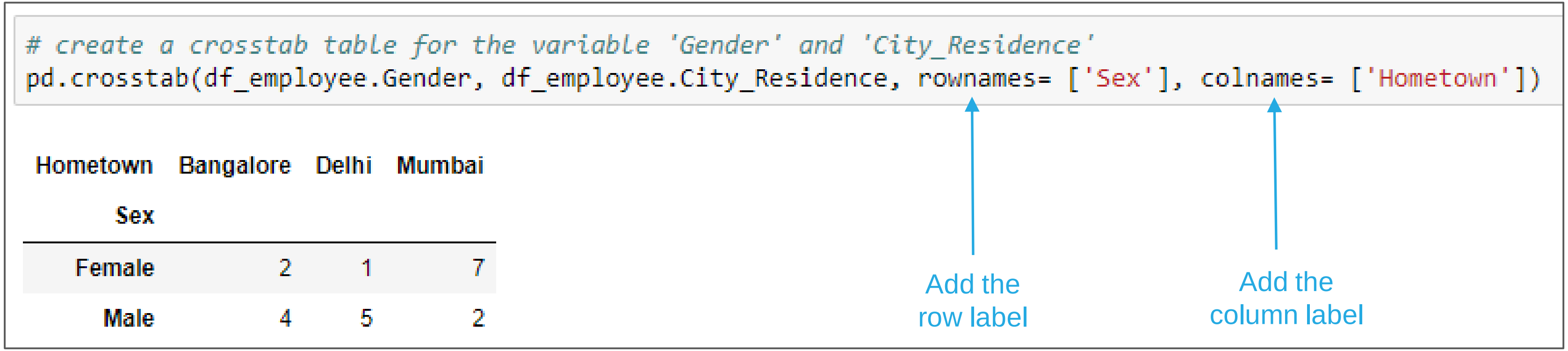


```python
# read the csv file 'EmployeeData.csv'
df_employee = pd.read_csv('EmployeeData.csv')

# display first five observations
df_employee.head()
```

	Age	Gender	City_Residence	Annual CTC (in lakhs)	Years of experience	Designation
0	45	Male	Mumbai	16.7	21	Cloud Engineer
1	23	Female	Mumbai	4.5	1	Data Analyst Intern
2	27	Male	Mumbai	6.8	3	Sr. Data Scientist
3	34	Male	Delhi	6.7	8	Big Data Engineer
4	43	Female	Mumbai	2.2	14	Cloud Engineer

Create a cross table

Find the city-wise gender count using the crosstab() method



```
# create a crosstab table for the variable 'Gender' and 'City_Residence'
pd.crosstab(df_employee.Gender, df_employee.City_Residence, rownames= ['Sex'], colnames= ['Hometown'])
```

Hometown	Bangalore	Delhi	Mumbai
Sex			
Female	2	1	7
Male	4	5	2

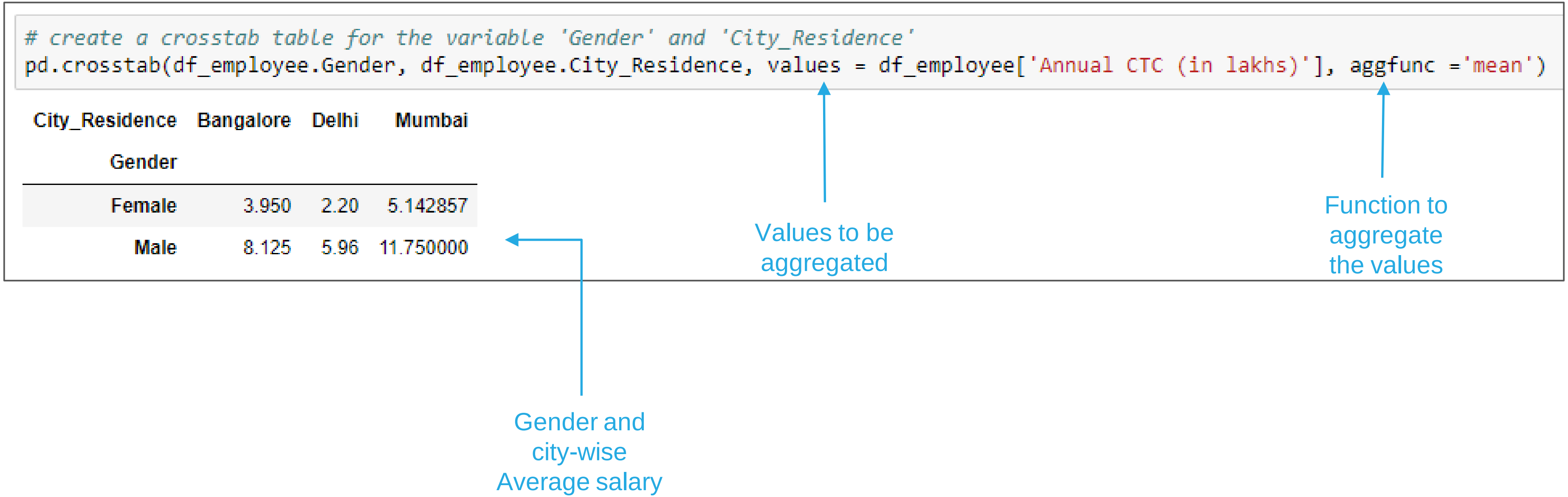
Add the row label

Add the column label

By default, the crosstab() method returns the frequency table of the variables

Create a cross table

Find the city-wise distribution of salary for different genders



```python
# create a crosstab table for the variable 'Gender' and 'City_Residence'
pd.crosstab(df_employee.Gender, df_employee.City_Residence, values = df_employee['Annual CTC (in lakhs)'], aggfunc ='mean')
```

City_Residence	Bangalore	Delhi	Mumbai
Gender			
Female	3.950	2.20	5.142857
Male	8.125	5.96	11.750000

Values to be aggregated

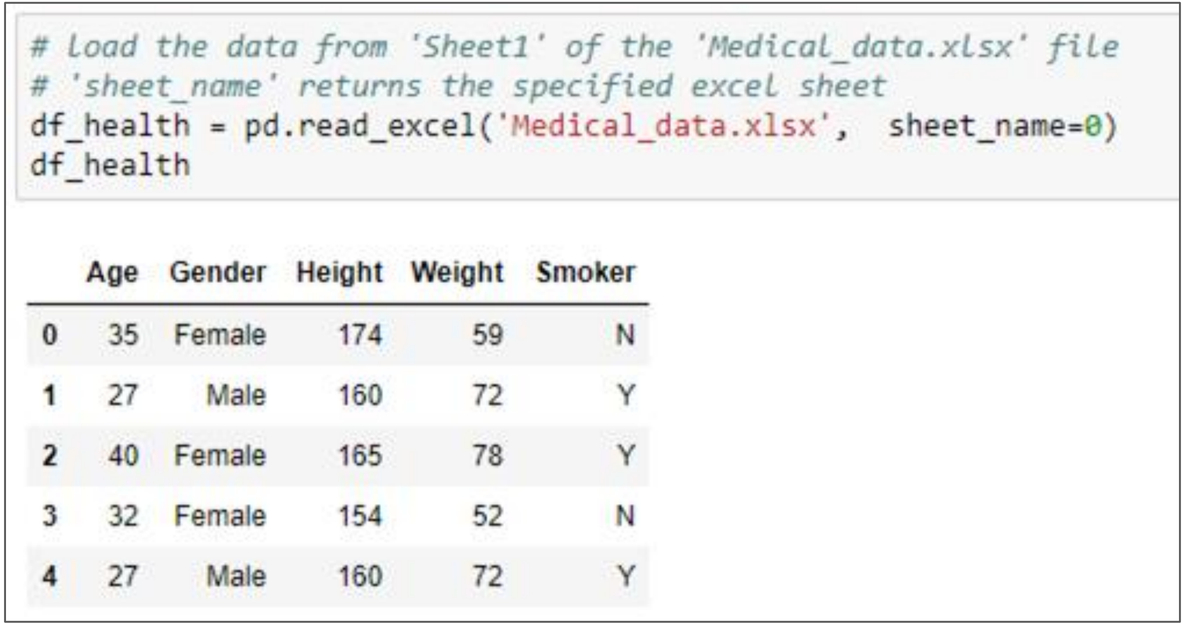
Function to aggregate the values

Gender and city-wise Average salary

Check for duplicates

Read the DataFrames

Use the below DataFrame for further manipulations:

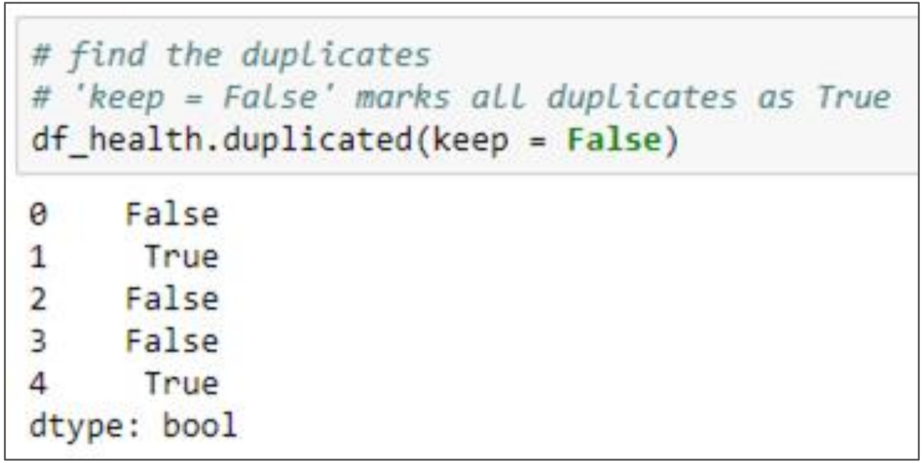


```
# Load the data from 'Sheet1' of the 'Medical_data.xlsx' file
# 'sheet_name' returns the specified excel sheet
df_health = pd.read_excel('Medical_data.xlsx',  sheet_name=0)
df_health
```

	Age	Gender	Height	Weight	Smoker
0	35	Female	174	59	N
1	27	Male	160	72	Y
2	40	Female	165	78	Y
3	32	Female	154	52	N
4	27	Male	160	72	Y

Check for duplicates

- Check the duplicate observations using the duplicated() method
- The second and last observation in the dataset is same

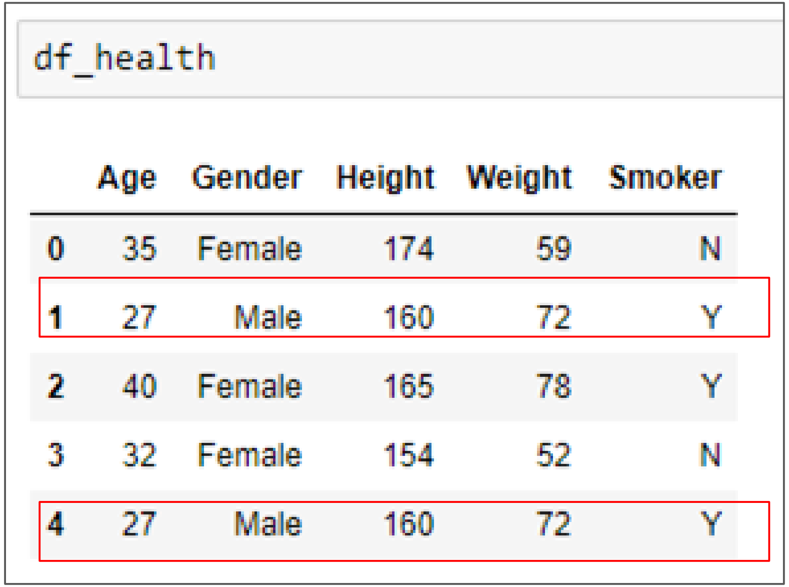


```
# find the duplicates
# 'keep = False' marks all duplicates as True
df_health.duplicated(keep = False)
```

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

Drop duplicates

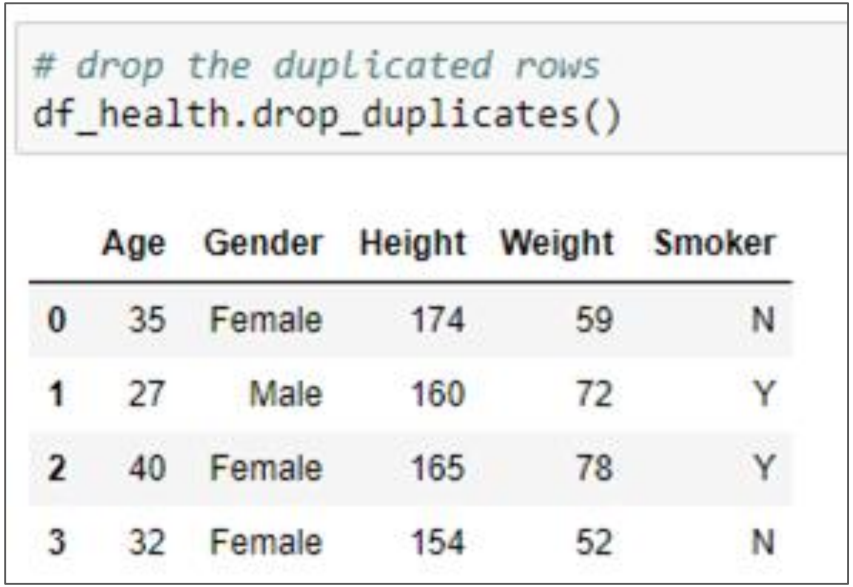
Use the drop_duplicates() method to drop the duplicated rows



```
df_health
```

	Age	Gender	Height	Weight	Smoker
0	35	Female	174	59	N
1	27	Male	160	72	Y
2	40	Female	165	78	Y
3	32	Female	154	52	N
4	27	Male	160	72	Y

Before



```
# drop the duplicated rows
df_health.drop_duplicates()
```

	Age	Gender	Height	Weight	Smoker
0	35	Female	174	59	N
1	27	Male	160	72	Y
2	40	Female	165	78	Y
3	32	Female	154	52	N

After

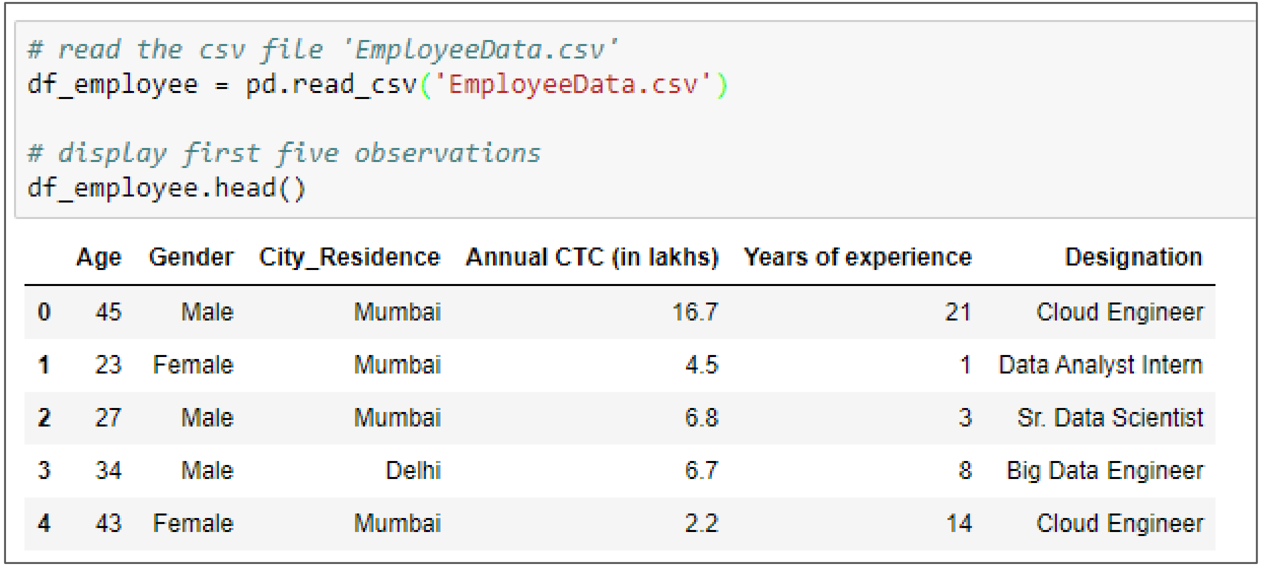
Dropping Rows and Columns

Drop the rows and columns

- The drop() method is used to drop the unwanted rows and columns from the data
- There are scenarios where we need to drop certain rows and/or columns which have missing values, or are redundant with respect to our analysis

Read the DataFrame

Read the csv file ‘EmployeeData’ and print the first five observations



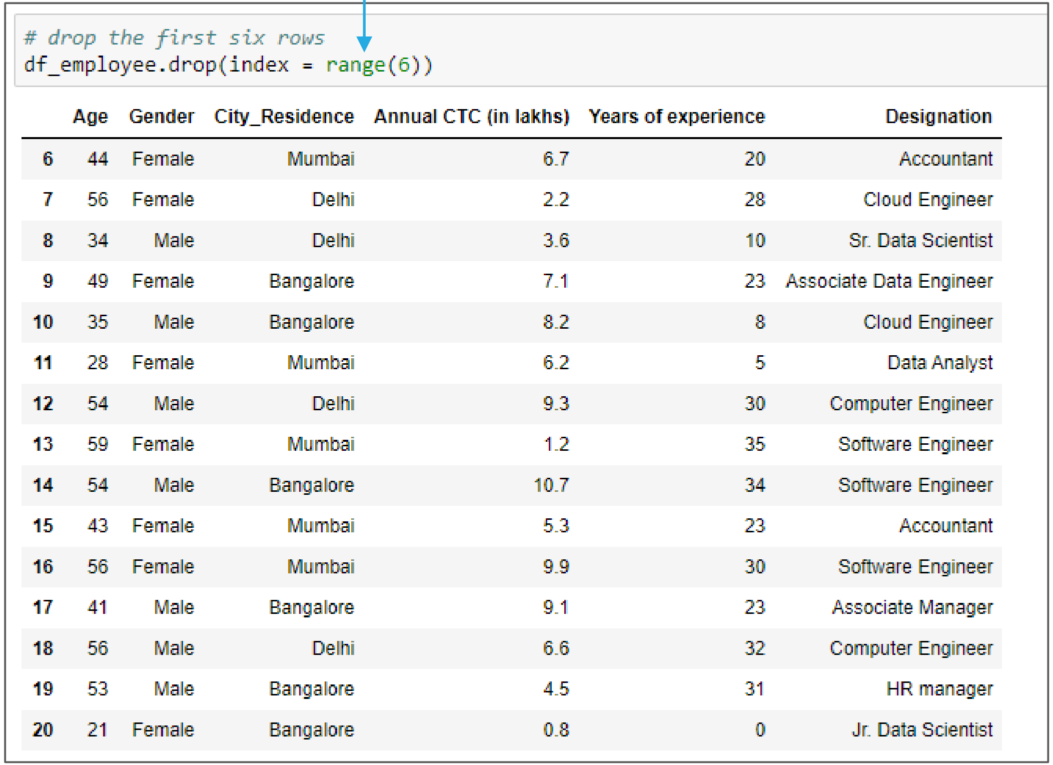
```
# read the csv file 'EmployeeData.csv'
df_employee = pd.read_csv('EmployeeData.csv')

# display first five observations
df_employee.head()
```

	Age	Gender	City_Residence	Annual CTC (in lakhs)	Years of experience	Designation
0	45	Male	Mumbai	16.7	21	Cloud Engineer
1	23	Female	Mumbai	4.5	1	Data Analyst Intern
2	27	Male	Mumbai	6.8	3	Sr. Data Scientist
3	34	Male	Delhi	6.7	8	Big Data Engineer
4	43	Female	Mumbai	2.2	14	Cloud Engineer

Drop the rows

- Use drop() method to drop the rows with index values
- Here ‘range(6)’ is used to drop the first six rows

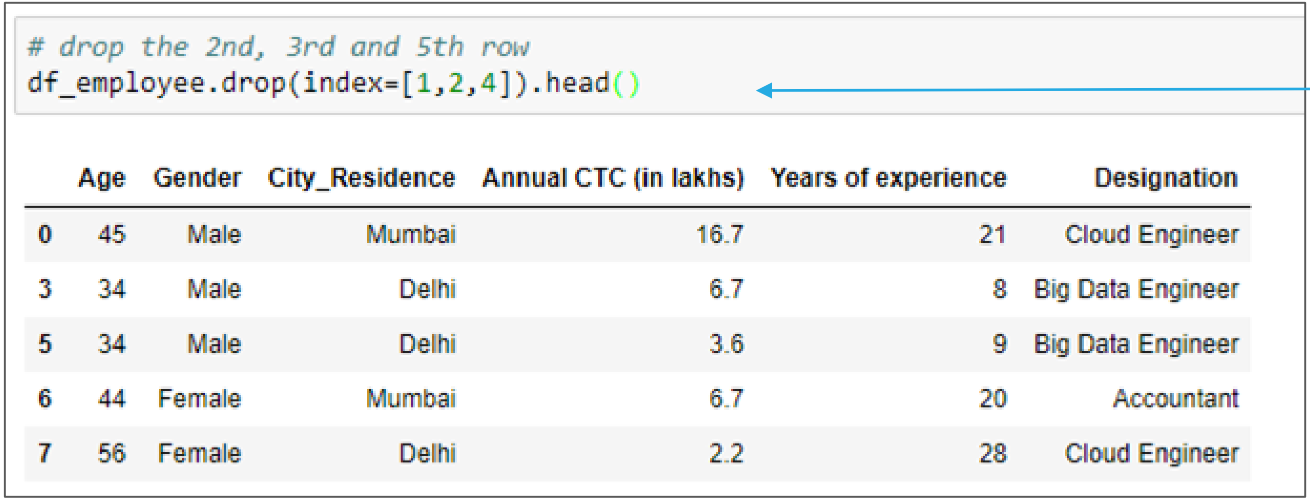
Pass the row indices to ‘index’



```
# drop the first six rows
df_employee.drop(index = range(6))
```

	Age	Gender	City_Residence	Annual CTC (in lakhs)	Years of experience	Designation
6	44	Female	Mumbai	6.7	20	Accountant
7	56	Female	Delhi	2.2	28	Cloud Engineer
8	34	Male	Delhi	3.6	10	Sr. Data Scientist
9	49	Female	Bangalore	7.1	23	Associate Data Engineer
10	35	Male	Bangalore	8.2	8	Cloud Engineer
11	28	Female	Mumbai	6.2	5	Data Analyst
12	54	Male	Delhi	9.3	30	Computer Engineer
13	59	Female	Mumbai	1.2	35	Software Engineer
14	54	Male	Bangalore	10.7	34	Software Engineer
15	43	Female	Mumbai	5.3	23	Accountant
16	56	Female	Mumbai	9.9	30	Software Engineer
17	41	Male	Bangalore	9.1	23	Associate Manager
18	56	Male	Delhi	6.6	32	Computer Engineer
19	53	Male	Bangalore	4.5	31	HR manager
20	21	Female	Bangalore	0.8	0	Jr. Data Scientist

Drop the rows

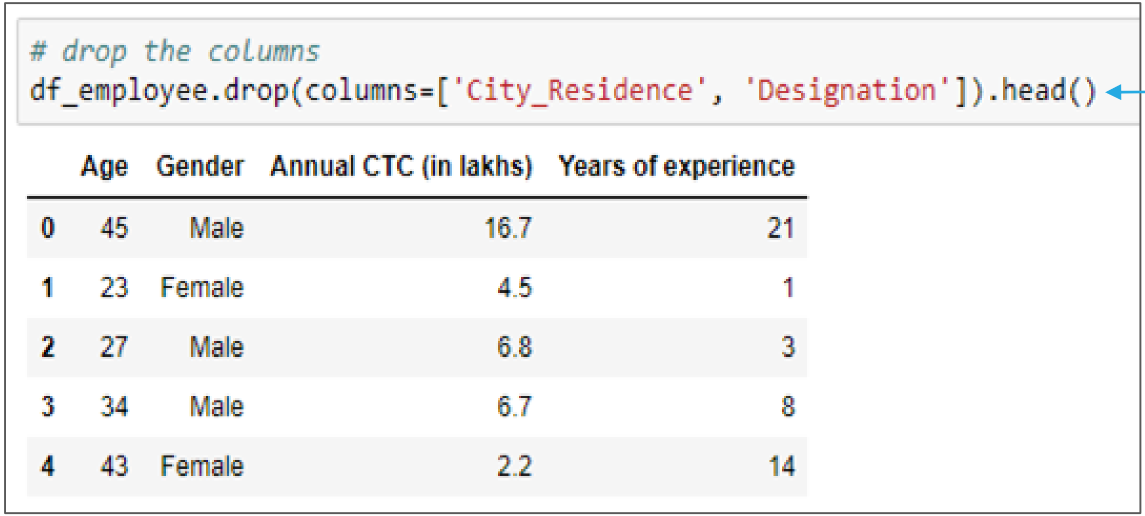


```
# drop the 2nd, 3rd and 5th row
df_employee.drop(index=[1,2,4]).head()
```

	Age	Gender	City_Residence	Annual CTC (in lakhs)	Years of experience	Designation
0	45	Male	Mumbai	16.7	21	Cloud Engineer
3	34	Male	Delhi	6.7	8	Big Data Engineer
5	34	Male	Delhi	3.6	9	Big Data Engineer
6	44	Female	Mumbai	6.7	20	Accountant
7	56	Female	Delhi	2.2	28	Cloud Engineer

Pass the list of row indices to drop the rows

Drop the columns



```python
# drop the columns
df_employee.drop(columns=['City_Residence', 'Designation']).head()
```

	Age	Gender	Annual CTC (in lakhs)	Years of experience
0	45	Male	16.7	21
1	23	Female	4.5	1
2	27	Male	6.8	3
3	34	Male	6.7	8
4	43	Female	2.2	14

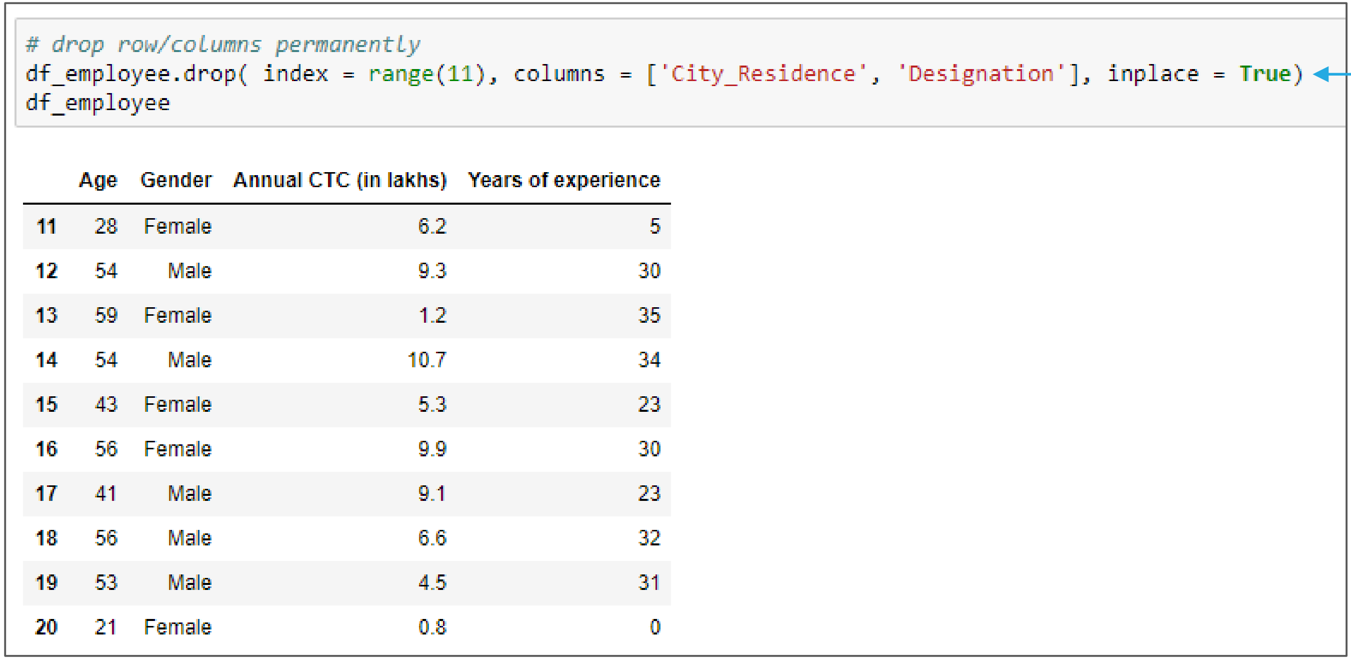
Pass the list of column names to drop the columns

Usage of inplace

- We saw how to drop the unwanted rows and column
- However, doing so does not delete it permanently
- To remove them permanently from the data, we use the parameter ‘inplace’ and set it to true
- By default, the value inplace takes is false

Usage of inplace

Drop the the first 11 rows and the variables ‘City_Residence’ and ‘Designation’



```
# drop row/columns permanently
df_employee.drop( index = range(11), columns = ['City_Residence', 'Designation'], inplace = True)
df_employee
```

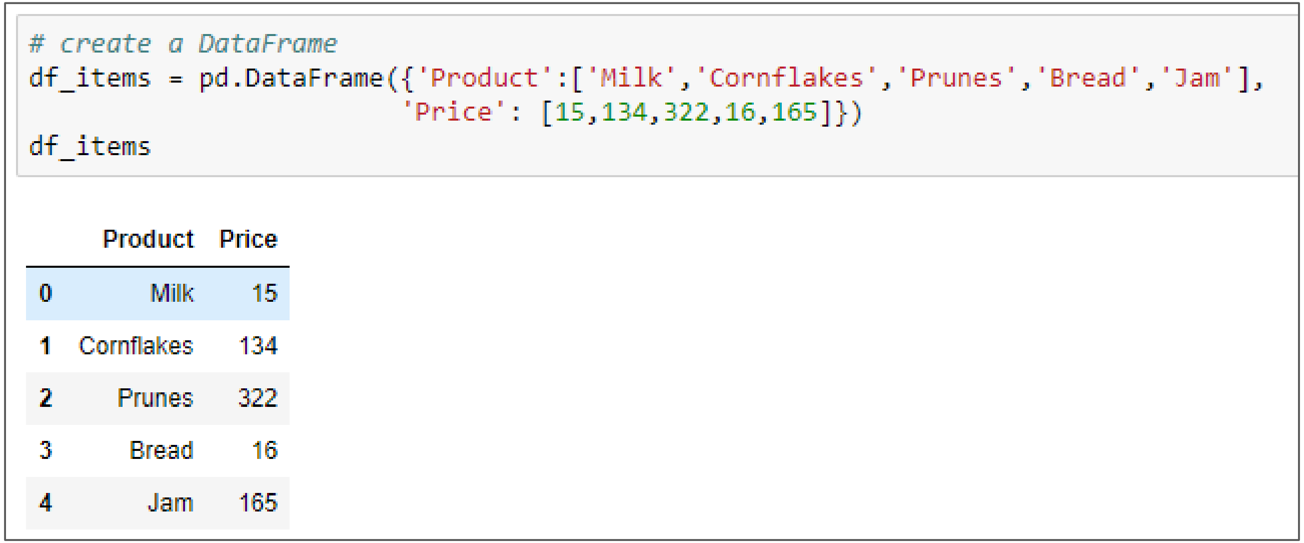
	Age	Gender	Annual CTC (in lakhs)	Years of experience
11	28	Female	6.2	5
12	54	Male	9.3	30
13	59	Female	1.2	35
14	54	Male	10.7	34
15	43	Female	5.3	23
16	56	Female	9.9	30
17	41	Male	9.1	23
18	56	Male	6.6	32
19	53	Male	4.5	31
20	21	Female	0.8	0

Removes the rows and columns from the original data

Mapping and Replacing

Create a DataFrame

Use the below DataFrame for further manipulations:

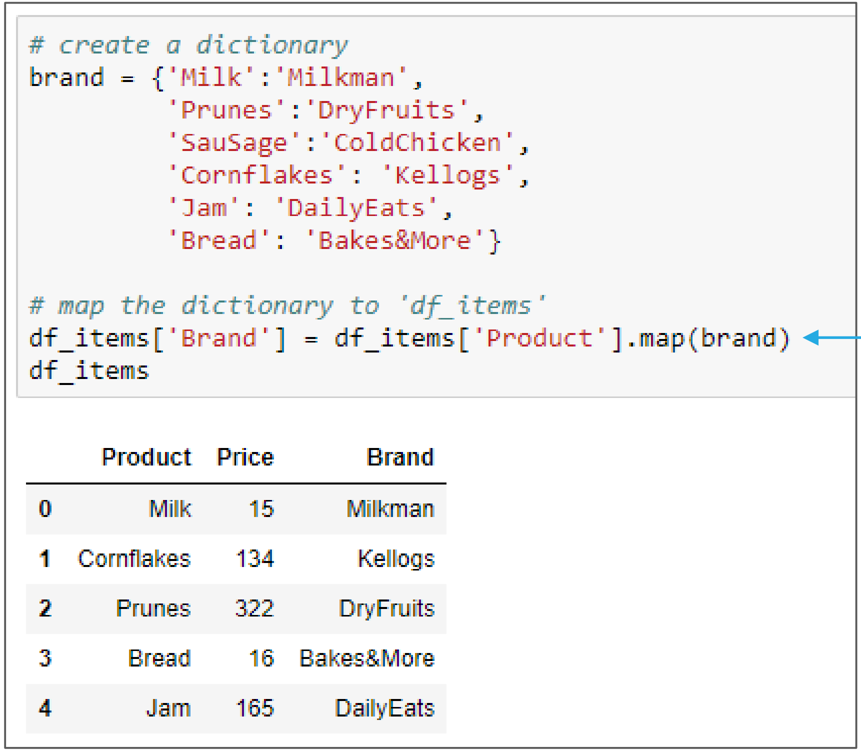


```
# create a DataFrame
df_items = pd.DataFrame({'Product':['Milk','Cornflakes','Prunes','Bread','Jam'],
                        'Price': [15,134,322,16,165]})
df_items
```

	Product	Price
0	Milk	15
1	Cornflakes	134
2	Prunes	322
3	Bread	16
4	Jam	165

Map the dictionary

Use the map() method to create a new column by mapping the DataFrame column values with the dictionary key



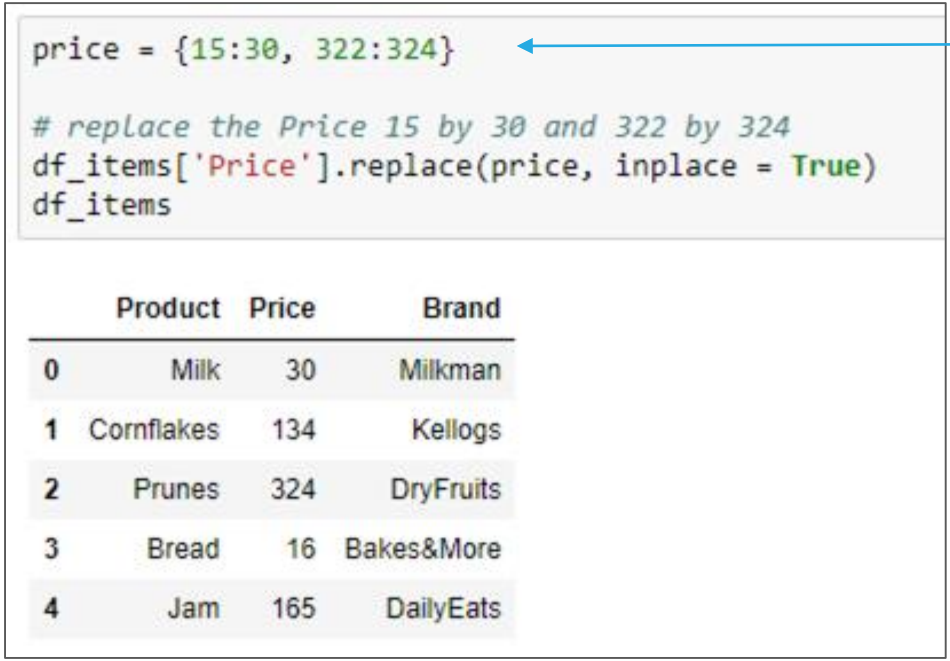
```python
# create a dictionary
brand = {'Milk':'Milkman',
         'Prunes':'DryFruits',
         'SauSage':'ColdChicken',
         'Cornflakes': 'Kellogs',
         'Jam': 'DailyEats',
         'Bread': 'Bakes&More'}

# map the dictionary to 'df_items'
df_items['Brand'] = df_items['Product'].map(brand)
df_items
```

	Product	Price	Brand
0	Milk	15	Milkman
1	Cornflakes	134	Kellogs
2	Prunes	322	DryFruits
3	Bread	16	Bakes&More
4	Jam	165	DailyEats

Map the dictionary to create a new column

Replace the values

The replace() method is used to replace the values in the DataFrame



```
price = {15:30, 322:324}

# replace the Price 15 by 30 and 322 by 324
df_items['Price'].replace(price, inplace = True)
df_items
```

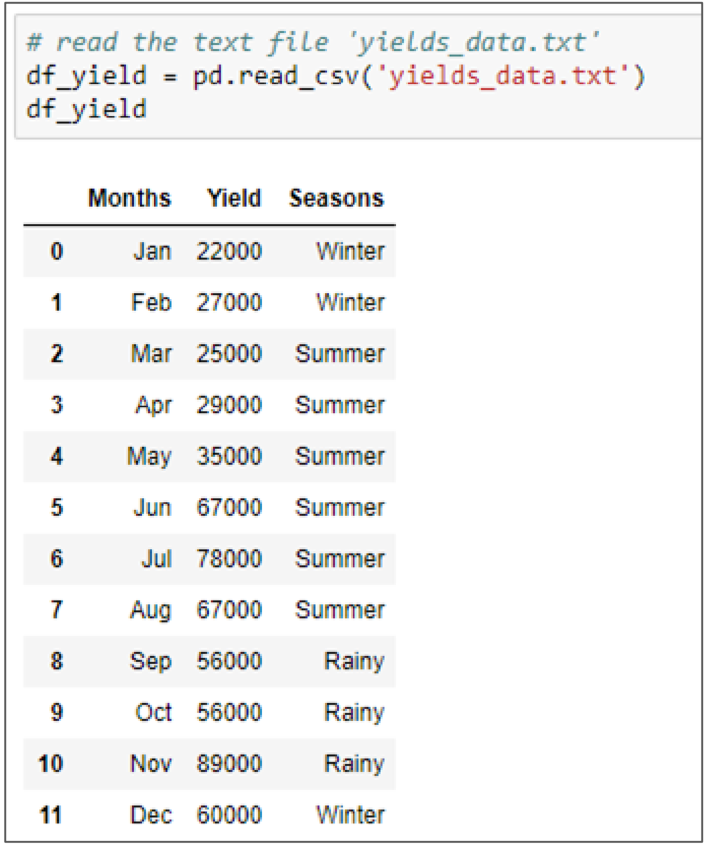
	Product	Price	Brand
0	Milk	30	Milkman
1	Cornflakes	134	Kellogs
2	Prunes	324	DryFruits
3	Bread	16	Bakes&More
4	Jam	165	DailyEats

Create a dictionary to replace the values

Group the DataFrame

Create a DataFrame

Use the following DataFrame for further manipulations:

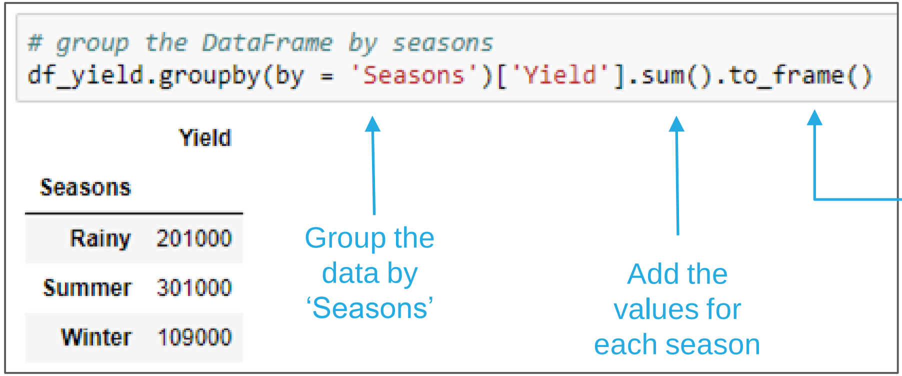


```
# read the text file 'yields_data.txt'
df_yield = pd.read_csv('yields_data.txt')
df_yield
```

	Months	Yield	Seasons
0	Jan	22000	Winter
1	Feb	27000	Winter
2	Mar	25000	Summer
3	Apr	29000	Summer
4	May	35000	Summer
5	Jun	67000	Summer
6	Jul	78000	Summer
7	Aug	67000	Summer
8	Sep	56000	Rainy
9	Oct	56000	Rainy
10	Nov	89000	Rainy
11	Dec	60000	Winter

Group the DataFrame

Use groupby() method to group the dataframe by the specific column(s)



```
# group the DataFrame by seasons
df_yield.groupby(by = 'Seasons')['Yield'].sum().to_frame()
```

```
           Yield
Seasons
  Rainy   201000
 Summer   301000
 Winter   109000
```

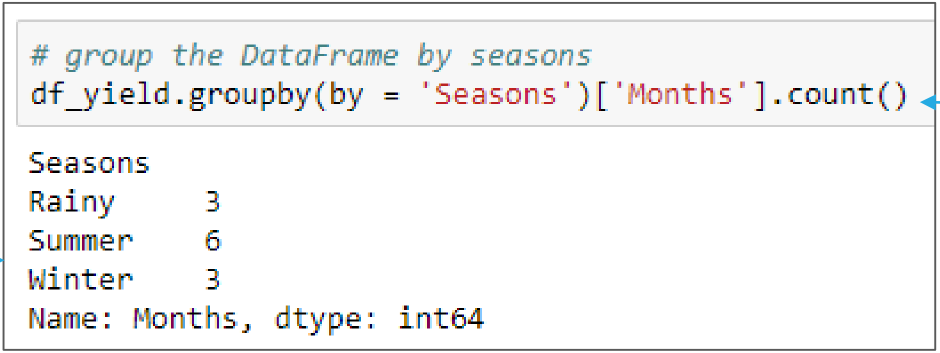
Group the data by 'Seasons'

Add the values for each season

Converts the series to DataFrame

Group the DataFrame

Get the number of months for each season



```
# group the DataFrame by seasons
df_yield.groupby(by = 'Seasons')['Months'].count()
```

```
Seasons
Rainy     3
Summer    6
Winter    3
Name: Months, dtype: int64
```

Returns the number of months per season

Output as a series

Visualization using Matplotlib

Data visualization

- Representation of the data in a pictorial or graphical format
- First step of data analysis
- Allow us to get the intuitive understanding of the data
- Helps to visualize the patterns in the data

Introduction to matplotlib

- It is a Python’s 2D plotting library
- ‘pyplot’ is a subpackage of matplotlib that provides a MATLAB-like way of plotting
- Provides a simple way of plotting the various plots like histogram, bar plot, scatter plot

Installation

Open terminal program (for Mac user) or command line (for Windows) and install the matplotlib using the command:

```
conda install matplotlib
```

Or

```
pip install matplotlib
```

Installation

- Alternatively, you can install matplotlib in a jupyter notebook using below code:

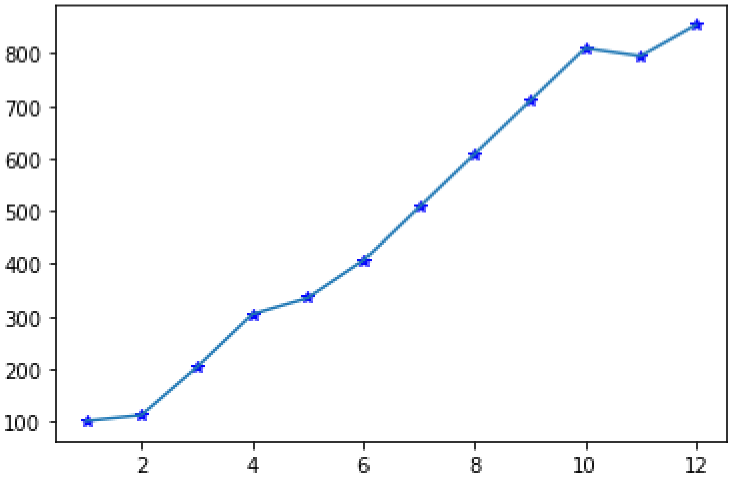
```
!pip install
matplotlib
```

- To import subpackage ‘pyplot’, use the command:

```
import
matplotlib.pyplot as
plt
```

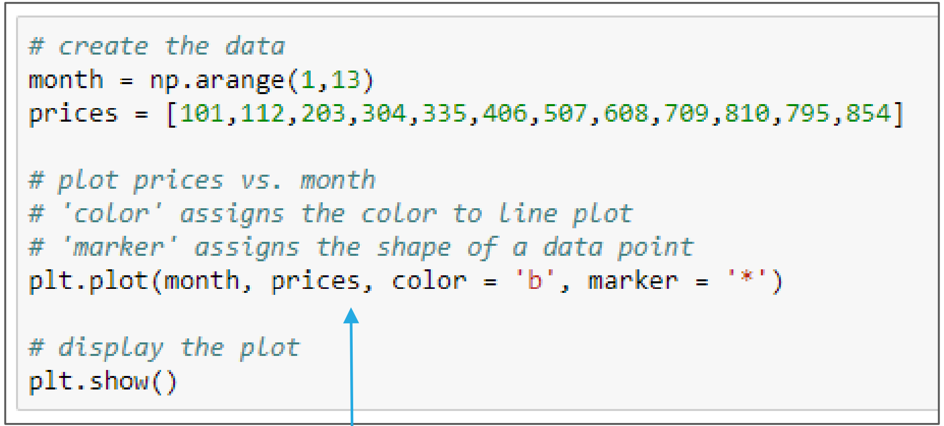
Line plot

It is a simple plot that displays the relationship between two variables



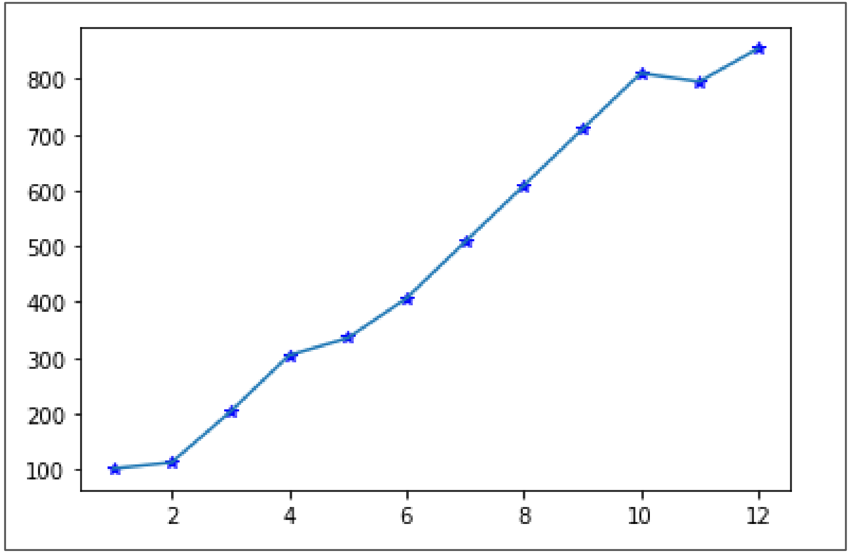
Plot a line plot from a list

Plot a line plot to visualize the price trend of a product over a year

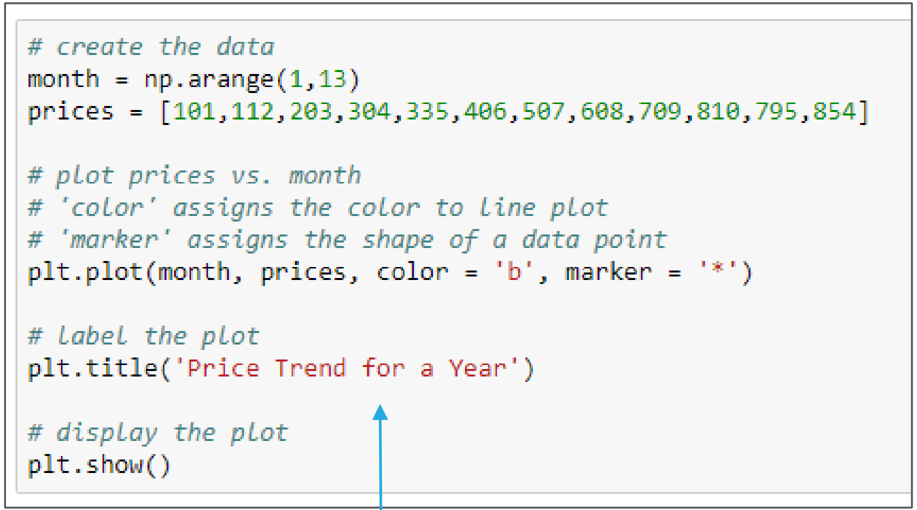


```
# create the data
month = np.arange(1,13)
prices = [101,112,203,304,335,406,507,608,709,810,795,854]

# plot prices vs. month
# 'color' assigns the color to line plot
# 'marker' assigns the shape of a data point
plt.plot(month, prices, color = 'b', marker = '*')

# display the plot
plt.show()
```

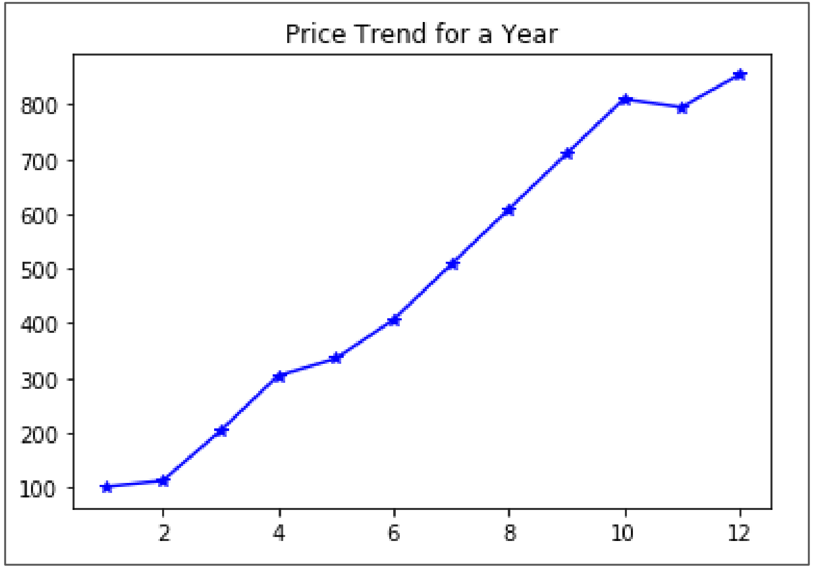
Plot the line plot using the plot() method



Add title of the graph

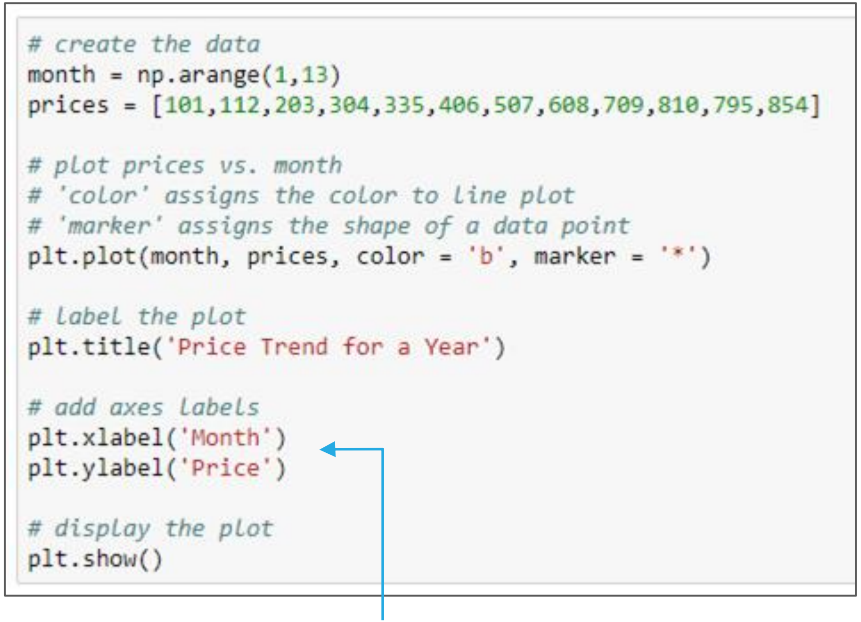


```
# create the data
month = np.arange(1,13)
prices = [101,112,203,304,335,406,507,608,709,810,795,854]

# plot prices vs. month
# 'color' assigns the color to line plot
# 'marker' assigns the shape of a data point
plt.plot(month, prices, color = 'b', marker = '*')

# label the plot
plt.title('Price Trend for a Year')

# display the plot
plt.show()
```

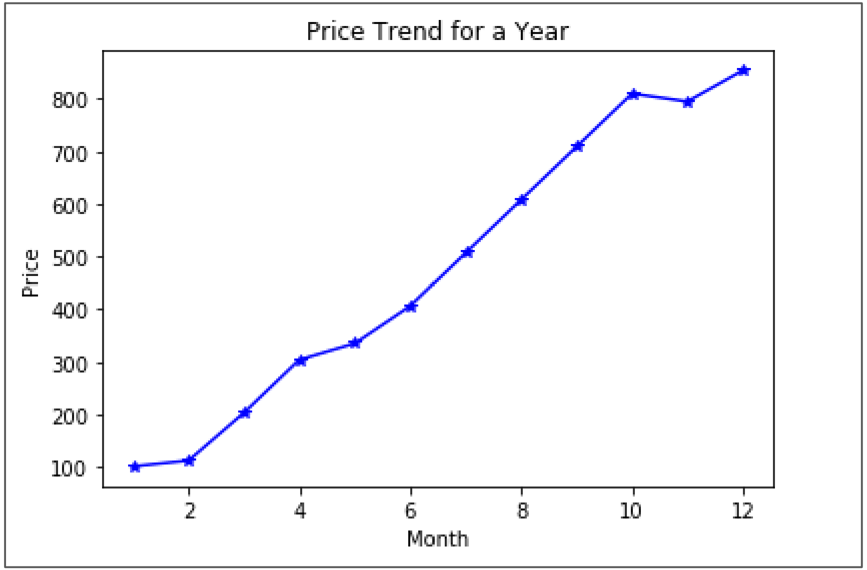


Put a title to the plot

Add axes labels

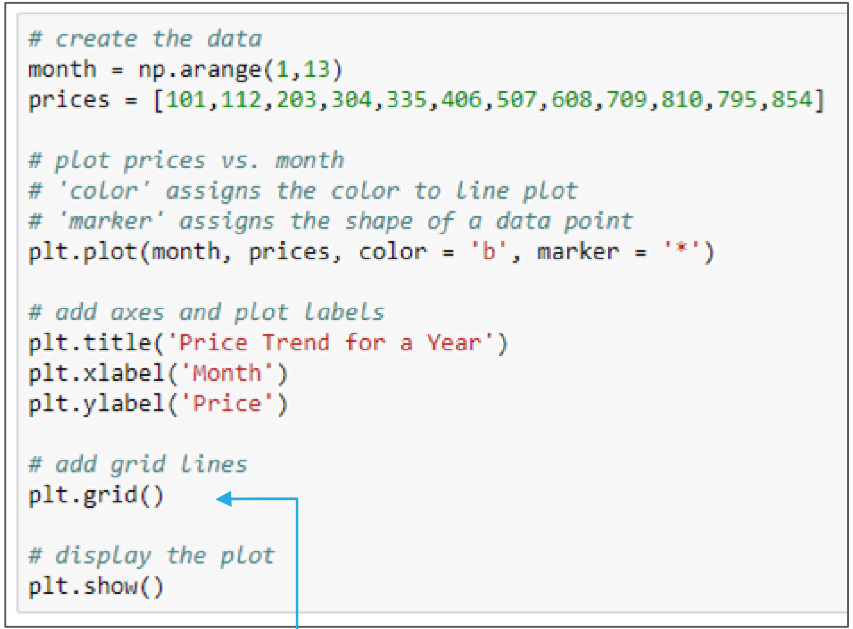


```
# create the data
month = np.arange(1,13)
prices = [101,112,203,304,335,406,507,608,709,810,795,854]

# plot prices vs. month
# 'color' assigns the color to line plot
# 'marker' assigns the shape of a data point
plt.plot(month, prices, color = 'b', marker = '*')

# label the plot
plt.title('Price Trend for a Year')

# add axes labels
plt.xlabel('Month')
plt.ylabel('Price')

# display the plot
plt.show()
```



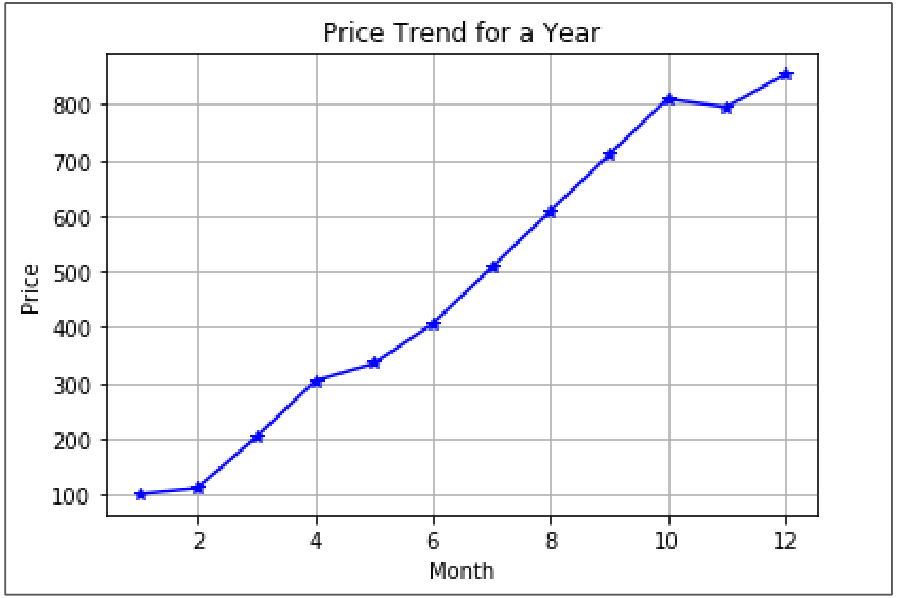
Add labels to x and y axis

Add grid lines to the plot

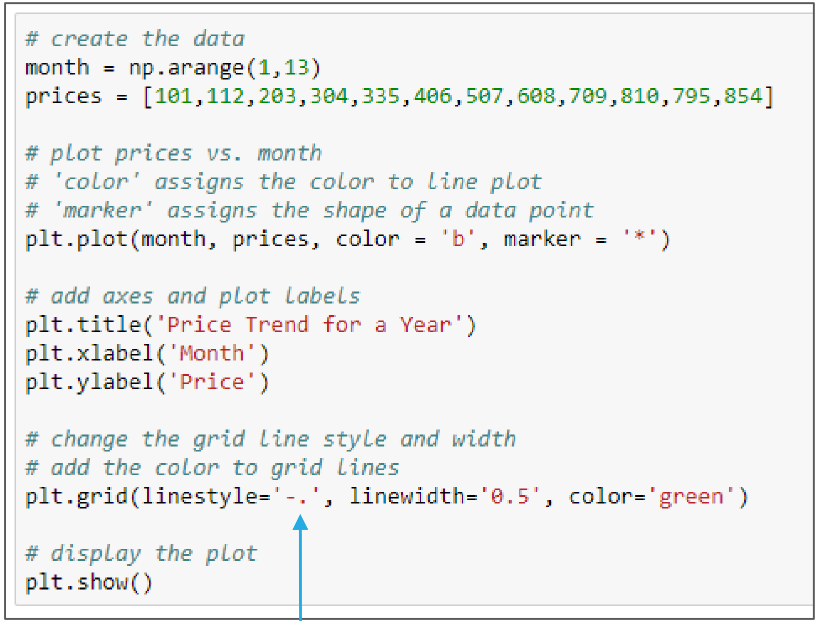


```
# create the data
month = np.arange(1,13)
prices = [101,112,203,304,335,406,507,608,709,810,795,854]

# plot prices vs. month
# 'color' assigns the color to line plot
# 'marker' assigns the shape of a data point
plt.plot(month, prices, color = 'b', marker = '*')

# add axes and plot labels
plt.title('Price Trend for a Year')
plt.xlabel('Month')
plt.ylabel('Price')

# add grid lines
plt.grid()

# display the plot
plt.show()
```

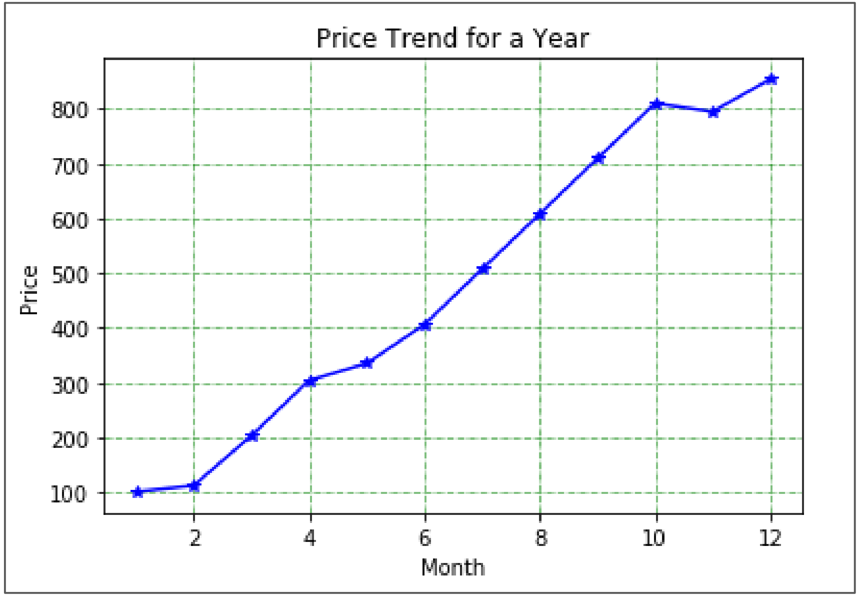
Add grid lines



Customize the grid lines



```
# create the data
month = np.arange(1,13)
prices = [101,112,203,304,335,406,507,608,709,810,795,854]

# plot prices vs. month
# 'color' assigns the color to line plot
# 'marker' assigns the shape of a data point
plt.plot(month, prices, color = 'b', marker = '*')

# add axes and plot labels
plt.title('Price Trend for a Year')
plt.xlabel('Month')
plt.ylabel('Price')

# change the grid line style and width
# add the color to grid lines
plt.grid(linestyle='-.', linewidth='0.5', color='green')

# display the plot
plt.show()
```



Change style, width and color of grid lines

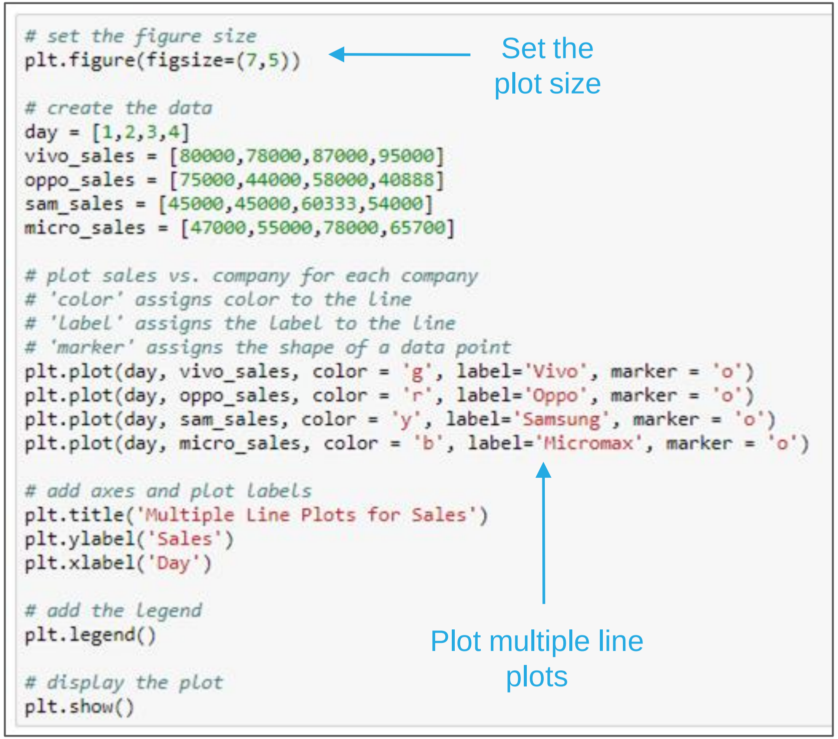
Multiple line plots

Plot the multiple line plots to represent the sales of each company recorded on the four different days. Use the data below to plot a graph:

Day	Sales			
	Vivo	Oppo	Samsung	Micromax
Day1	80000	75000	45000	47000
Day2	78000	44000	45000	55000
Day3	87000	58000	60333	78000
Day4	95000	40888	54000	65700

Multiple line plots

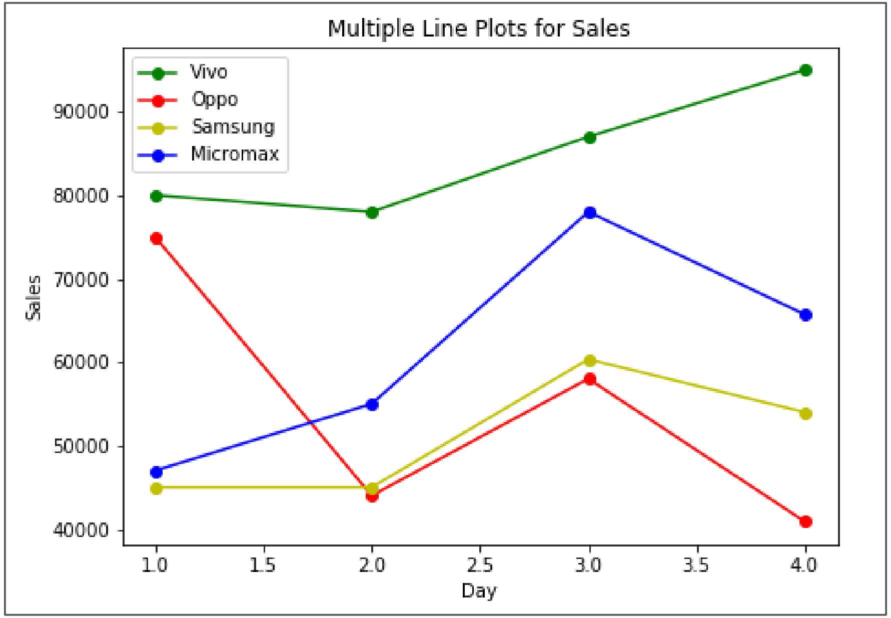
Each line represents the sales of a company for four days



```
# set the figure size
plt.figure(figsize=(7,5))

# create the data
day = [1,2,3,4]
vivo_sales = [80000,78000,87000,95000]
oppo_sales = [75000,44000,58000,40888]
sam_sales = [45000,45000,60333,54000]
micro_sales = [47000,55000,78000,65700]

# plot sales vs. company for each company
# 'color' assigns color to the line
# 'label' assigns the label to the line
# 'marker' assigns the shape of a data point
plt.plot(day, vivo_sales, color = 'g', label='Vivo', marker = 'o')
plt.plot(day, oppo_sales, color = 'r', label='Oppo', marker = 'o')
plt.plot(day, sam_sales, color = 'y', label='Samsung', marker = 'o')
plt.plot(day, micro_sales, color = 'b', label='Micromax', marker = 'o')

# add axes and plot labels
plt.title('Multiple Line Plots for Sales')
plt.ylabel('Sales')
plt.xlabel('Day')

# add the legend
plt.legend()

# display the plot
plt.show()
```

Set the plot size

Plot multiple line plots

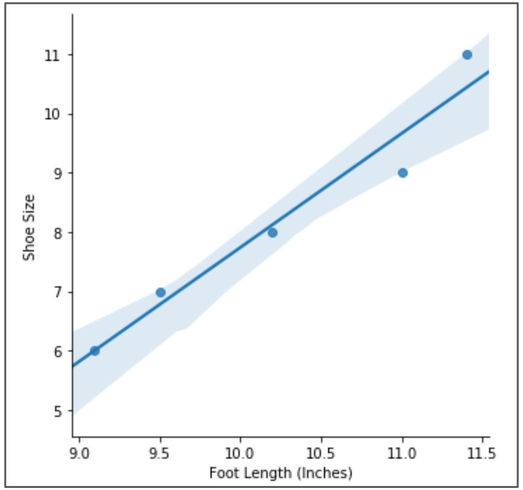


Scatter plot

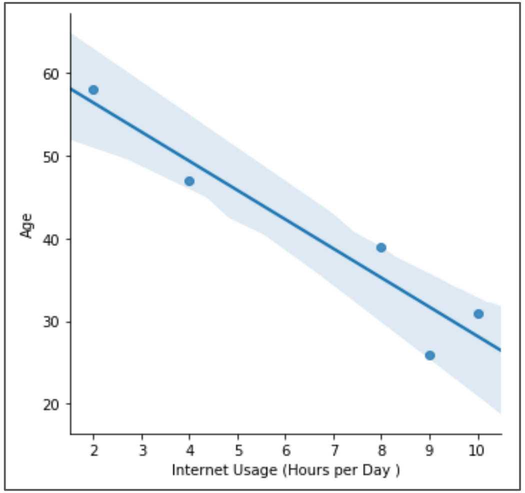
- It is used to display the relationship between two numeric variables
- Used to represent the extent of correlation between two variables
- Used to detect the extreme points in the data

Scatter plot

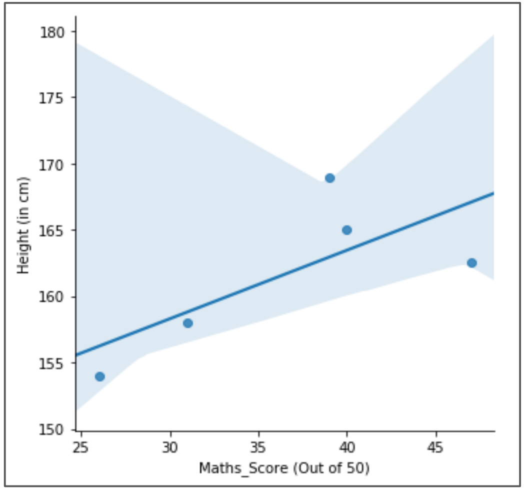
Scatter plots explaining the different types of correlation between the variables:



Positive Correlation
(ρ = 0.97167252)



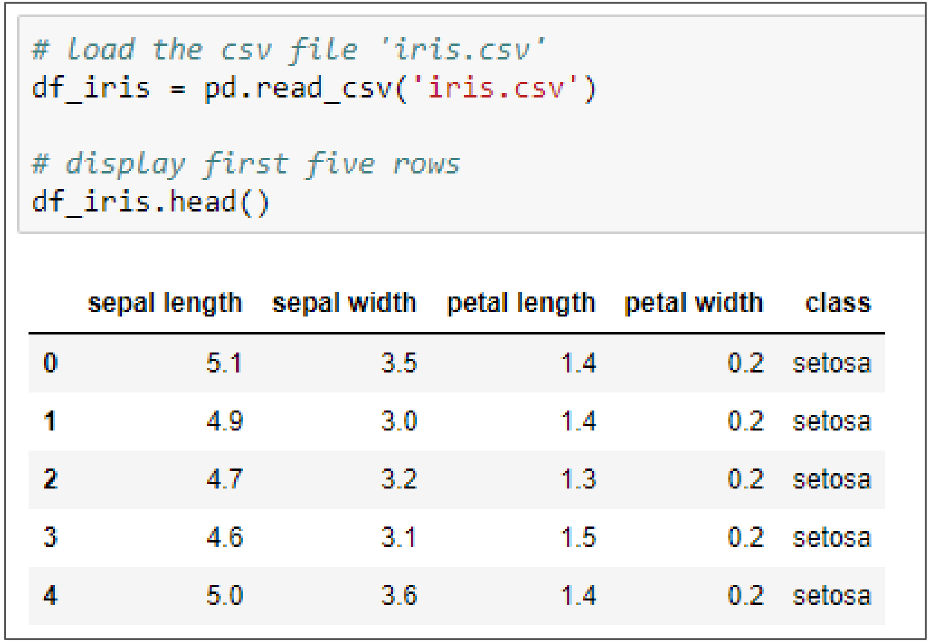
Negative Correlation
(ρ = -0.95056151)



No Correlation
(ρ = 0.09919779
)

Read the data

Use the iris data to create the scatter plot

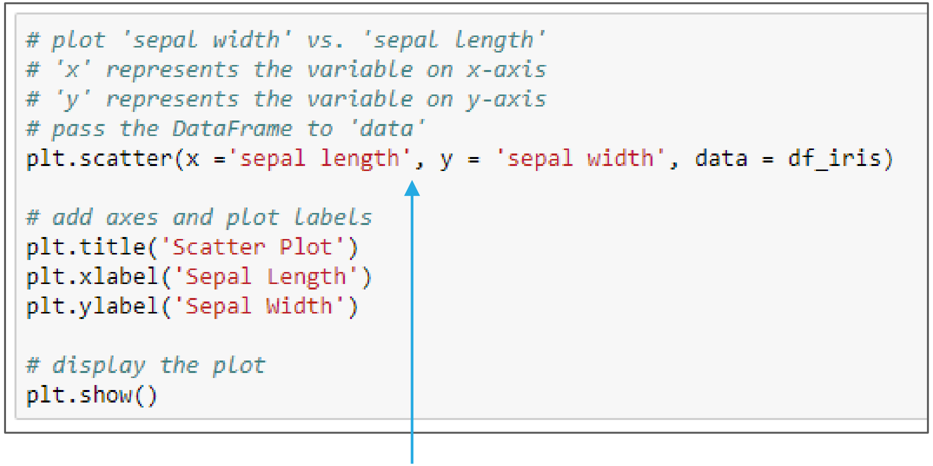


```
# load the csv file 'iris.csv'
df_iris = pd.read_csv('iris.csv')

# display first five rows
df_iris.head()
```

	sepal length	sepal width	petal length	petal width	class
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

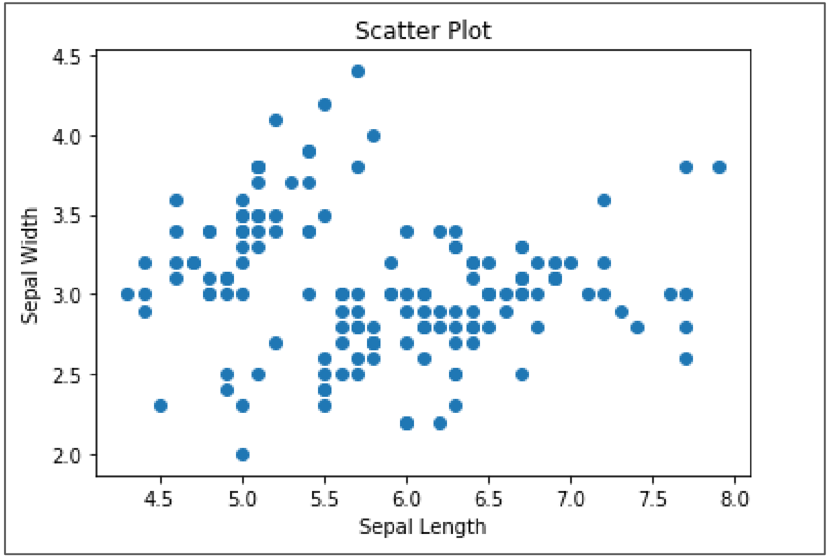
Scatter plot

Use the scatter() method to create scatter plot in matplotlib



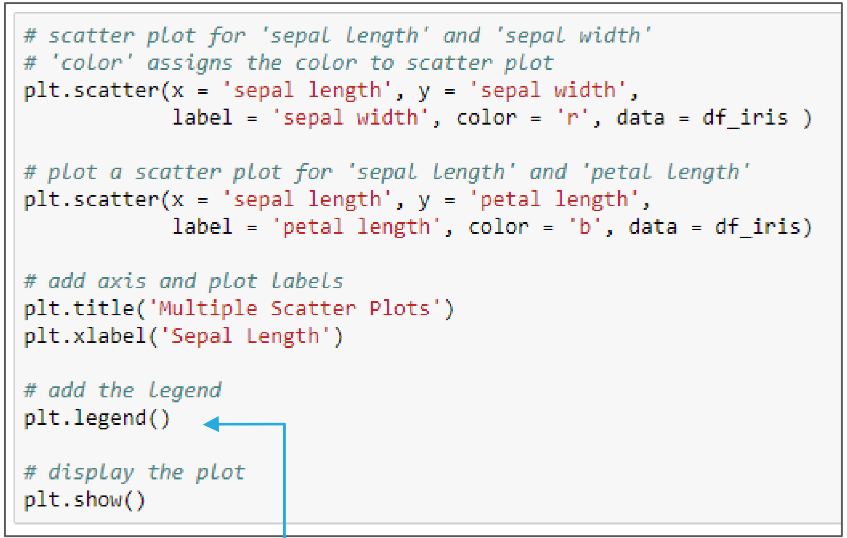
```
# plot 'sepal width' vs. 'sepal length'
# 'x' represents the variable on x-axis
# 'y' represents the variable on y-axis
# pass the DataFrame to 'data'
plt.scatter(x ='sepal length', y = 'sepal width', data = df_iris)

# add axes and plot labels
plt.title('Scatter Plot')
plt.xlabel('Sepal Length')
plt.ylabel('Sepal Width')

# display the plot
plt.show()
```

Set the variables on
x and y axis



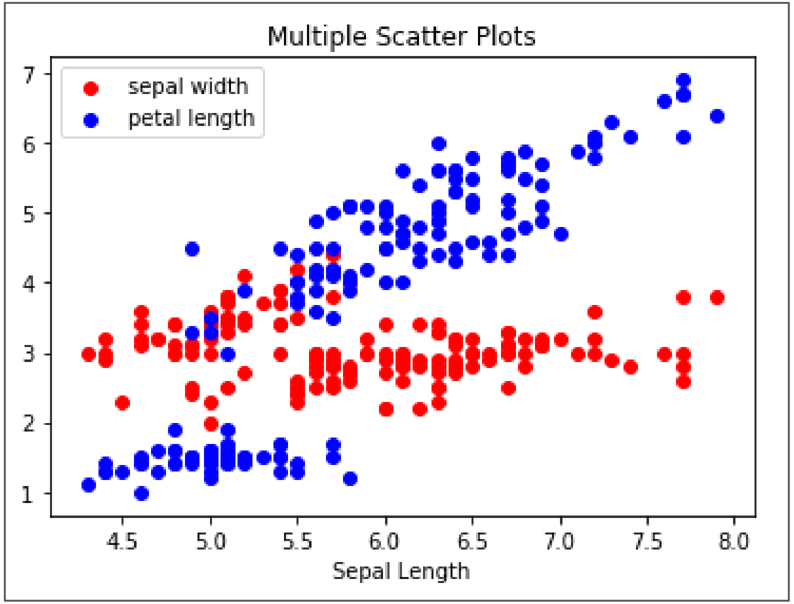
There is no significant correlation between ‘sepal length’ and ‘sepal width’

Multiple scatter plots



```
# scatter plot for 'sepal length' and 'sepal width'
# 'color' assigns the color to scatter plot
plt.scatter(x = 'sepal length', y = 'sepal width',
            label = 'sepal width', color = 'r', data = df_iris )

# plot a scatter plot for 'sepal length' and 'petal length'
plt.scatter(x = 'sepal length', y = 'petal length',
            label = 'petal length', color = 'b', data = df_iris)

# add axis and plot labels
plt.title('Multiple Scatter Plots')
plt.xlabel('Sepal Length')

# add the legend
plt.legend()

# display the plot
plt.show()
```

Add the legend



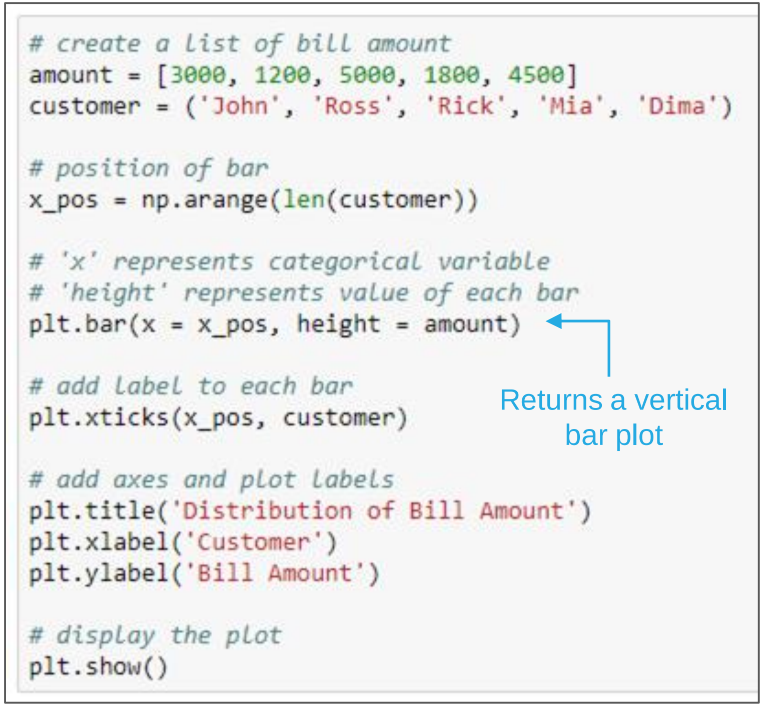
The plot shows the positive relationship between ‘sepal length’ and ‘petal length’

Bar plot

- It is used to display the categorical data with bars of lengths proportional to the values that they represent
- Used to compare the different categories of the categorical variable
- One axis displays the categorical variable and another displays the value for each category

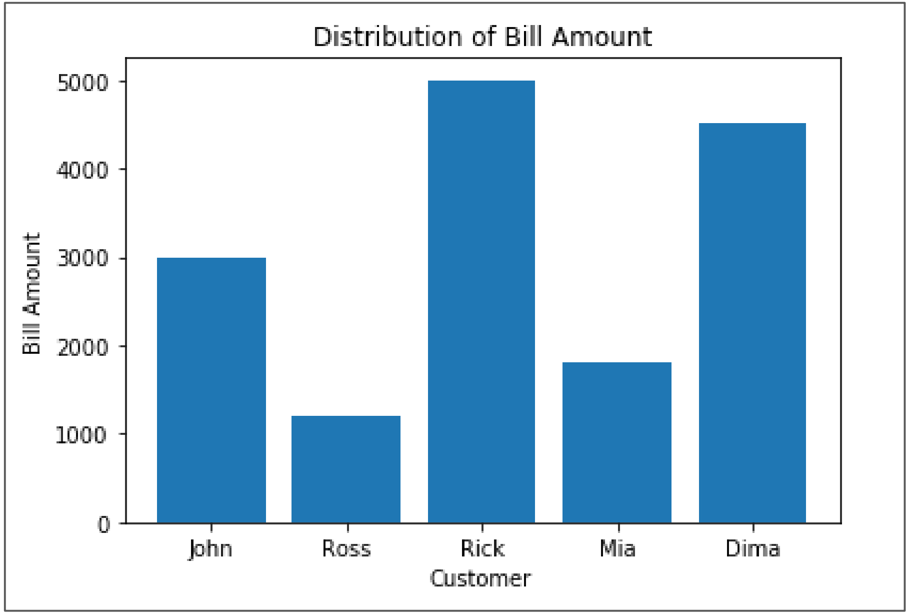
Bar plot

The bar displays the bill amount by customer



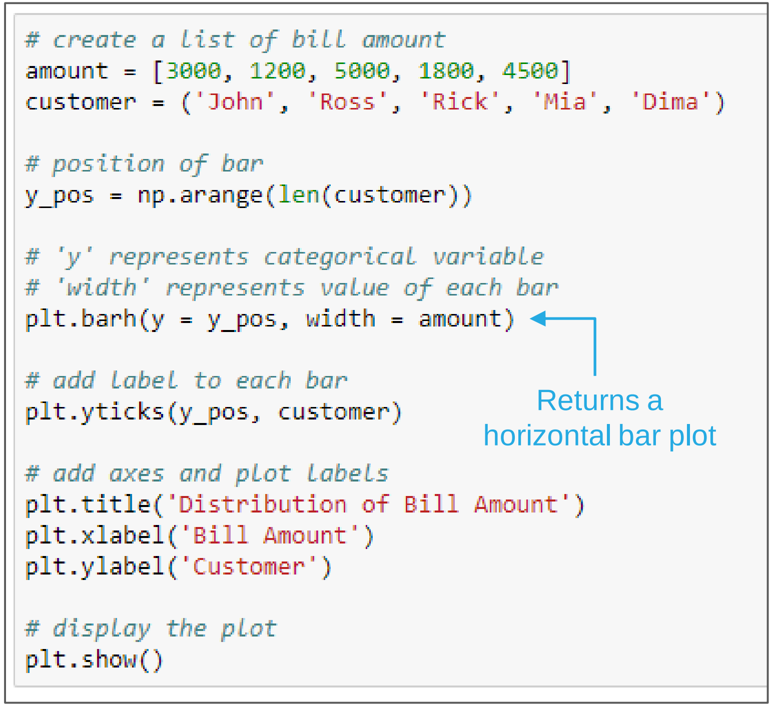
```
# create a list of bill amount
amount = [3000, 1200, 5000, 1800, 4500]
customer = ('John', 'Ross', 'Rick', 'Mia', 'Dima')

# position of bar
x_pos = np.arange(len(customer))

# 'x' represents categorical variable
# 'height' represents value of each bar
plt.bar(x = x_pos, height = amount)

# add label to each bar
plt.xticks(x_pos, customer)

# add axes and plot labels
plt.title('Distribution of Bill Amount')
plt.xlabel('Customer')
plt.ylabel('Bill Amount')

# display the plot
plt.show()
```

Returns a vertical bar plot



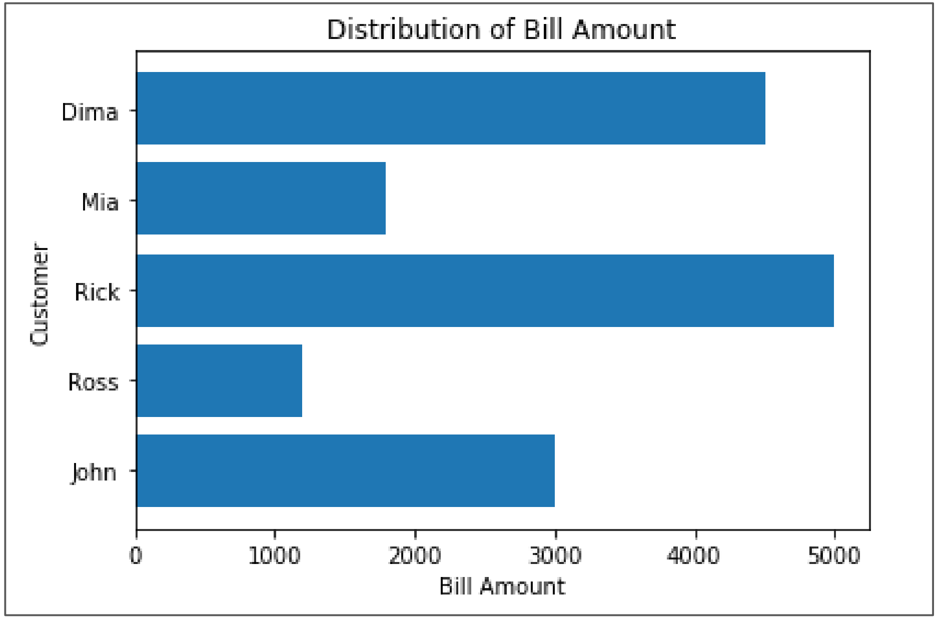
Horizontal bar plot

Plot the chart horizontally using the barh() method



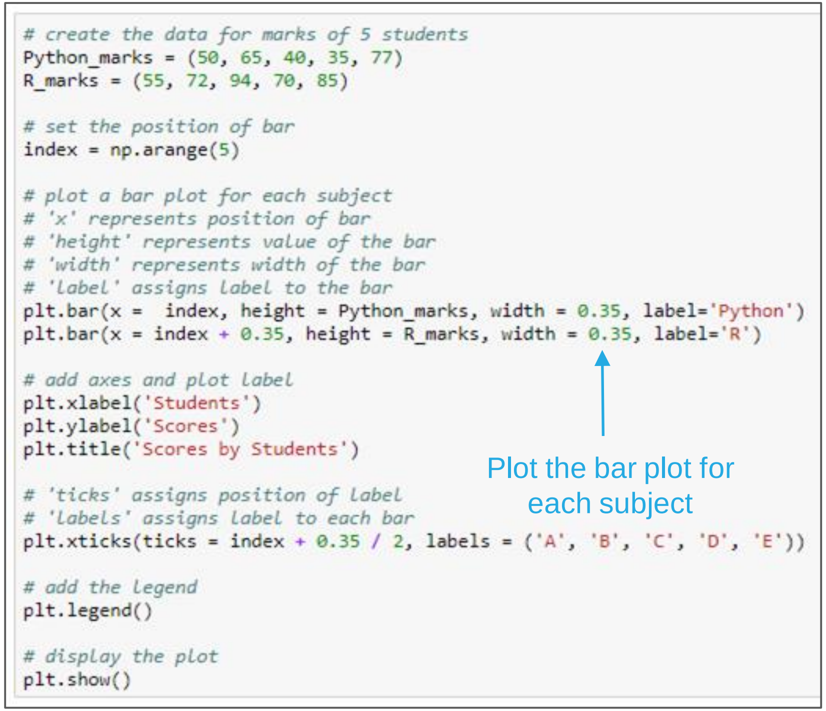
```
# create a list of bill amount
amount = [3000, 1200, 5000, 1800, 4500]
customer = ('John', 'Ross', 'Rick', 'Mia', 'Dima')

# position of bar
y_pos = np.arange(len(customer))

# 'y' represents categorical variable
# 'width' represents value of each bar
plt.barh(y = y_pos, width = amount)

# add label to each bar
plt.yticks(y_pos, customer)

# add axes and plot labels
plt.title('Distribution of Bill Amount')
plt.xlabel('Bill Amount')
plt.ylabel('Customer')

# display the plot
plt.show()
```

Returns a horizontal bar plot



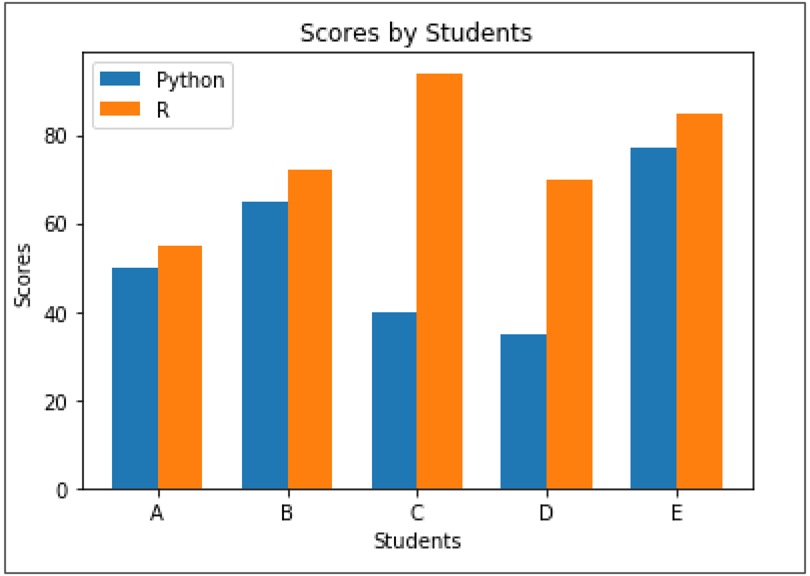
Grouped bar plot

Compare the marks of the students in R and Python

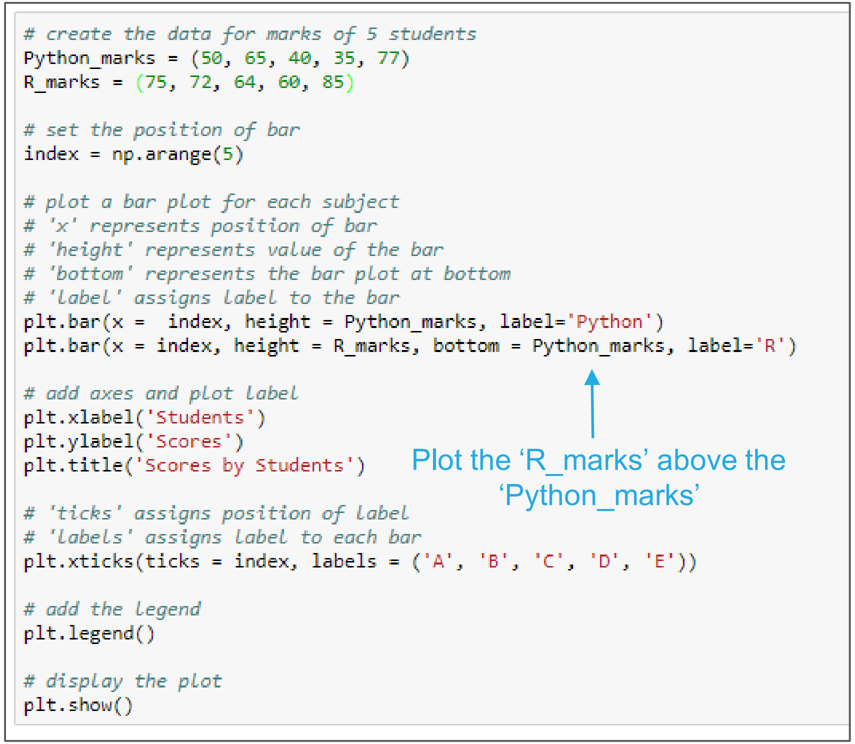


```
# create the data for marks of 5 students
Python_marks = (50, 65, 40, 35, 77)
R_marks = (55, 72, 94, 70, 85)

# set the position of bar
index = np.arange(5)

# plot a bar plot for each subject
# 'x' represents position of bar
# 'height' represents value of the bar
# 'width' represents width of the bar
# 'label' assigns label to the bar
plt.bar(x =  index, height = Python_marks, width = 0.35, label='Python')
plt.bar(x = index + 0.35, height = R_marks, width = 0.35, label='R')

# add axes and plot label
plt.xlabel('Students')
plt.ylabel('Scores')
plt.title('Scores by Students')

# 'ticks' assigns position of label
# 'labels' assigns label to each bar
plt.xticks(ticks = index + 0.35 / 2, labels = ('A', 'B', 'C', 'D', 'E'))

# add the legend
plt.legend()

# display the plot
plt.show()
```

Plot the bar plot for each subject

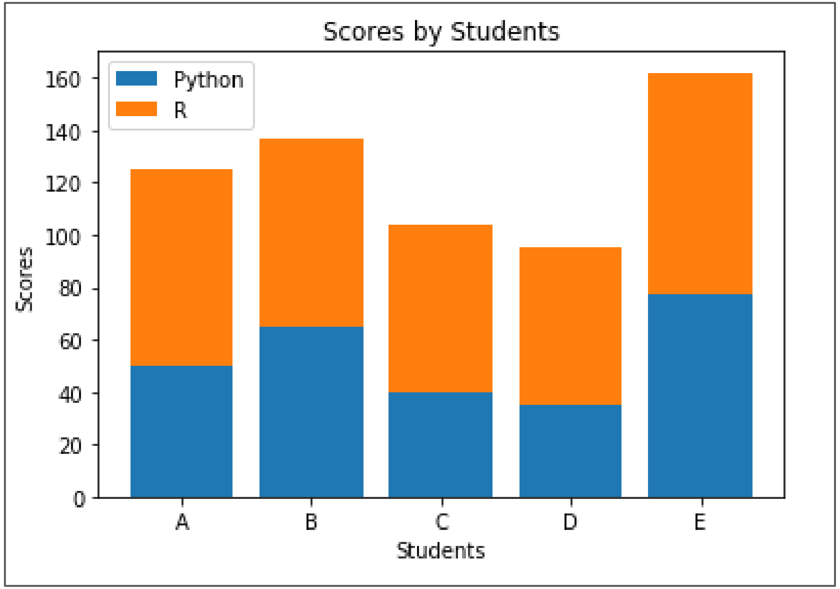


Stacked bar plot



```
# create the data for marks of 5 students
Python_marks = (50, 65, 40, 35, 77)
R_marks = (75, 72, 64, 60, 85)

# set the position of bar
index = np.arange(5)

# plot a bar plot for each subject
# 'x' represents position of bar
# 'height' represents value of the bar
# 'bottom' represents the bar plot at bottom
# 'label' assigns label to the bar
plt.bar(x =  index, height = Python_marks, label='Python')
plt.bar(x = index, height = R_marks, bottom = Python_marks, label='R')

# add axes and plot label
plt.xlabel('Students')
plt.ylabel('Scores')
plt.title('Scores by Students')

# 'ticks' assigns position of label
# 'labels' assigns label to each bar
plt.xticks(ticks = index, labels = ('A', 'B', 'C', 'D', 'E'))

# add the legend
plt.legend()

# display the plot
plt.show()
```

Plot the ‘R_marks’ above the ‘Python_marks’

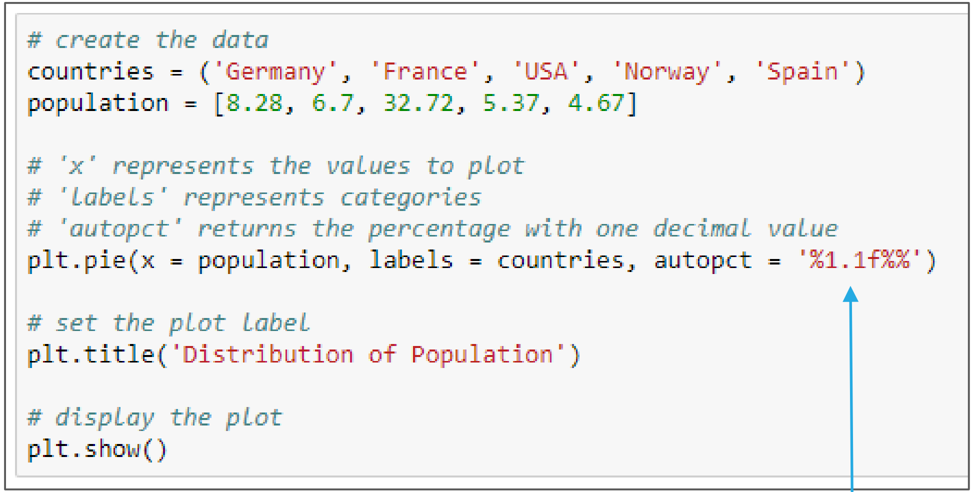


Pie plot

- It is a circular graph divided into sections displaying the numeric proportion
- It is used to display the univariate data
- Each section of the pie plot represents a single category in the data

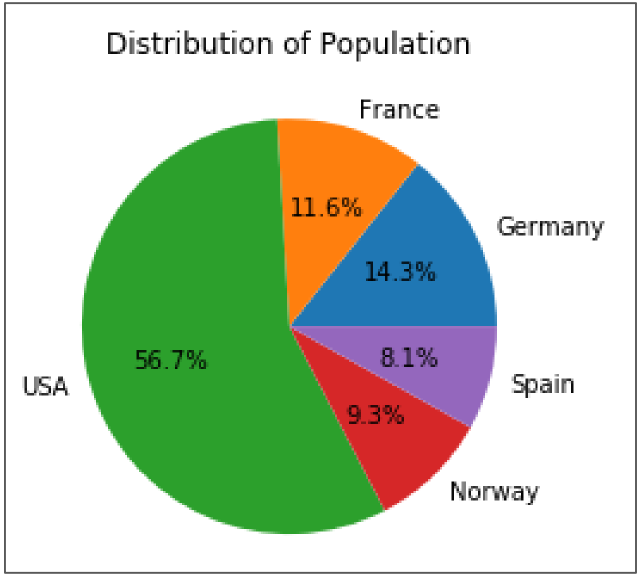
Pie plot

Plot a pie plot to study the population proportion for different countries



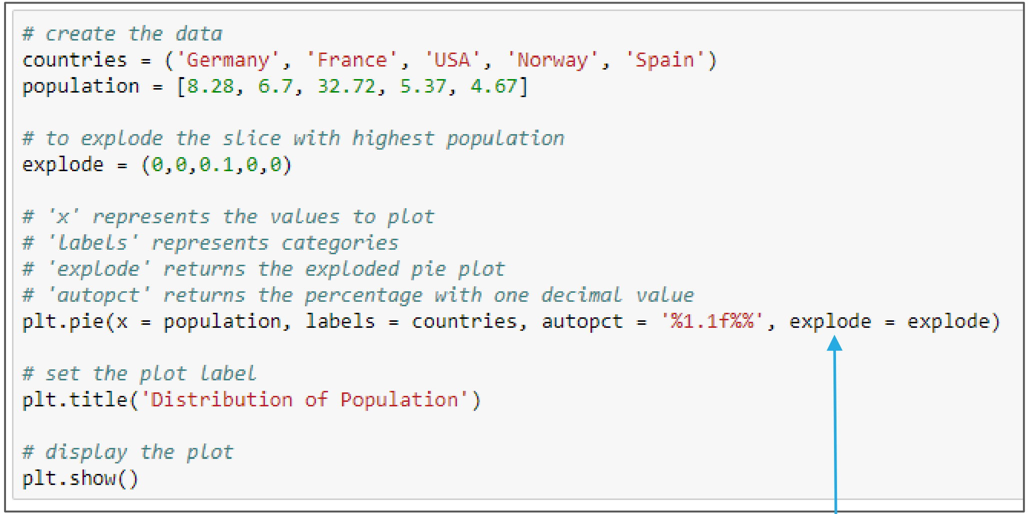
```
# create the data
countries = ('Germany', 'France', 'USA', 'Norway', 'Spain')
population = [8.28, 6.7, 32.72, 5.37, 4.67]

# 'x' represents the values to plot
# 'labels' represents categories
# 'autopct' returns the percentage with one decimal value
plt.pie(x = population, labels = countries, autopct = '%1.1f%%')

# set the plot label
plt.title('Distribution of Population')

# display the plot
plt.show()
```

Add the percentage with value to tenth place

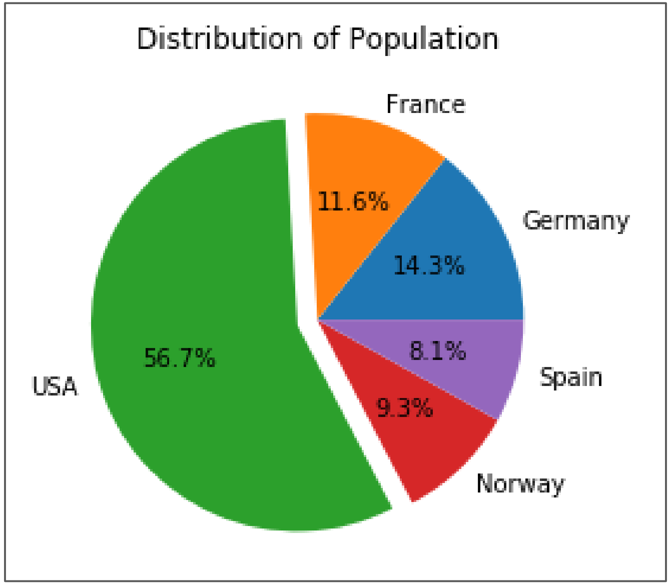


Exploded pie plot

It is a type of pie plot in which one or more sectors are separated from the disc



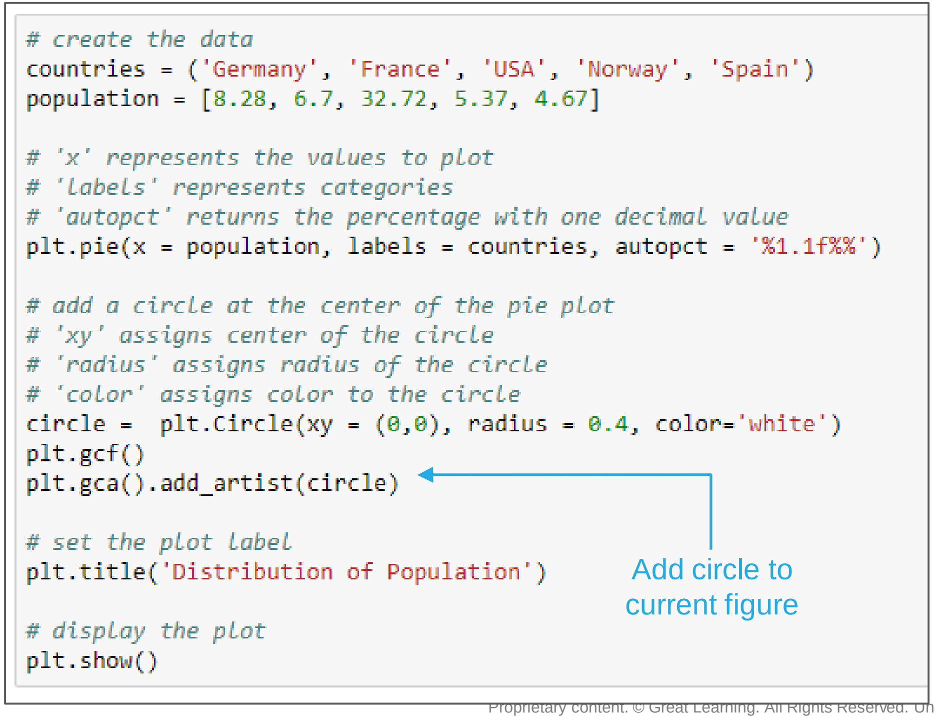
```
# create the data
countries = ('Germany', 'France', 'USA', 'Norway', 'Spain')
population = [8.28, 6.7, 32.72, 5.37, 4.67]

# to explode the slice with highest population
explode = (0,0,0.1,0,0)

# 'x' represents the values to plot
# 'labels' represents categories
# 'explode' returns the exploded pie plot
# 'autopct' returns the percentage with one decimal value
plt.pie(x = population, labels = countries, autopct = '%1.1f%%', explode = explode)

# set the plot label
plt.title('Distribution of Population')

# display the plot
plt.show()
```



Explode the country with highest population

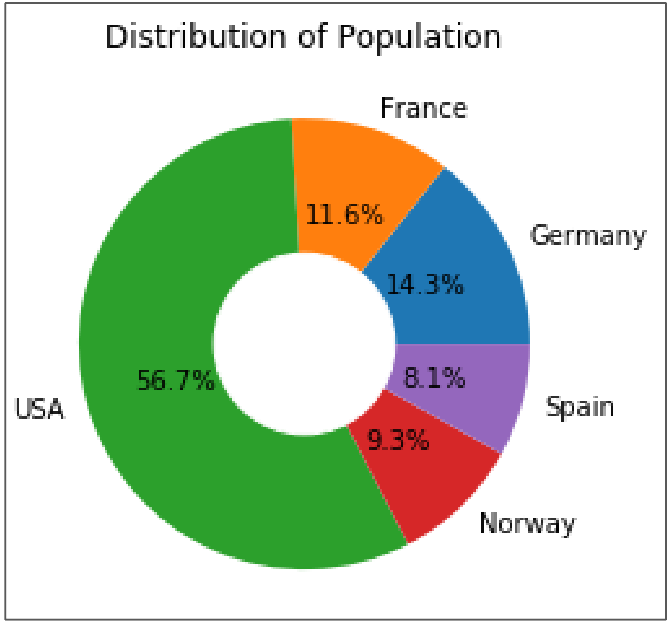
Donut pie plot

It is a type of pie plot with a hollow center representing a doughnut



```
# create the data
countries = ('Germany', 'France', 'USA', 'Norway', 'Spain')
population = [8.28, 6.7, 32.72, 5.37, 4.67]

# 'x' represents the values to plot
# 'labels' represents categories
# 'autopct' returns the percentage with one decimal value
plt.pie(x = population, labels = countries, autopct = '%1.1f%%')

# add a circle at the center of the pie plot
# 'xy' assigns center of the circle
# 'radius' assigns radius of the circle
# 'color' assigns color to the circle
circle =  plt.Circle(xy = (0,0), radius = 0.4, color='white')
plt.gcf()
plt.gca().add_artist(circle)

# set the plot label
plt.title('Distribution of Population')

# display the plot
plt.show()
```

Add circle to current figure

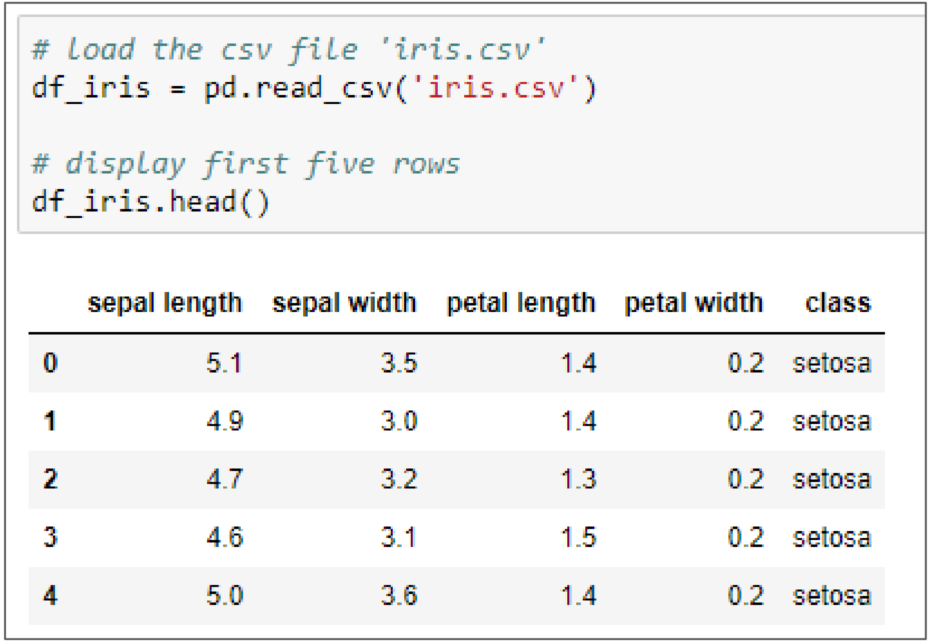


Histogram

- It is used to represent the distribution of the numeric variable
- It is an estimate of the probability distribution of a continuous data
- One axis represents the variable in the form of bars and another represents the frequency each bar
- There are no gaps between the bars of the histogram

Read the data

Use the iris data to create the histogram

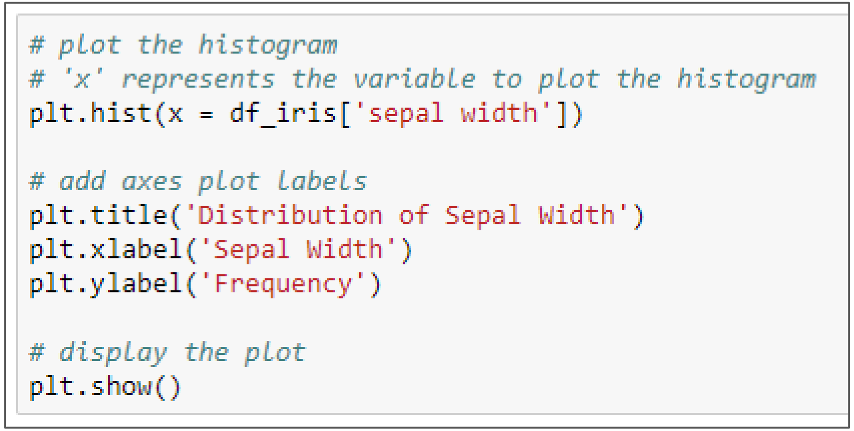


```
# load the csv file 'iris.csv'
df_iris = pd.read_csv('iris.csv')

# display first five rows
df_iris.head()
```

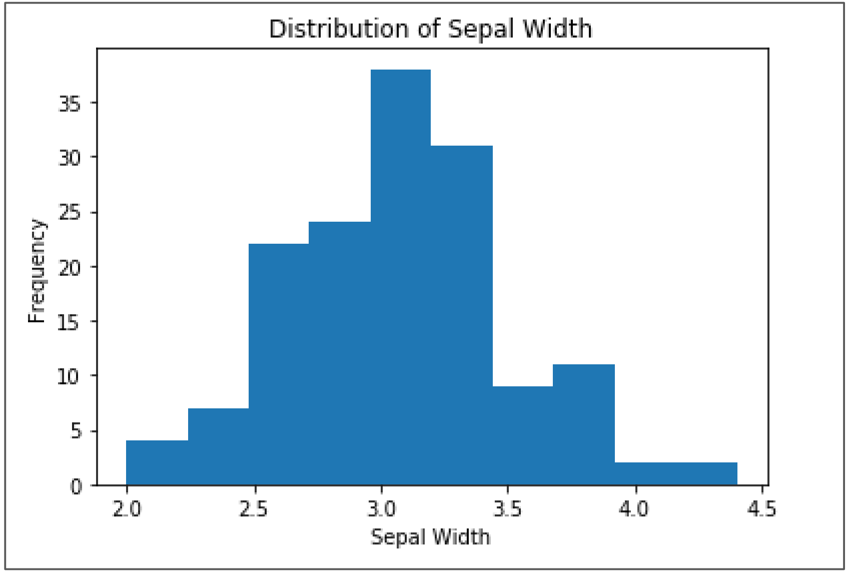
	sepal length	sepal width	petal length	petal width	class
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5.0	3.6	1.4	0.2	setosa

Histogram

Plot the histogram to check the distribution of the variable, ‘sepal width’



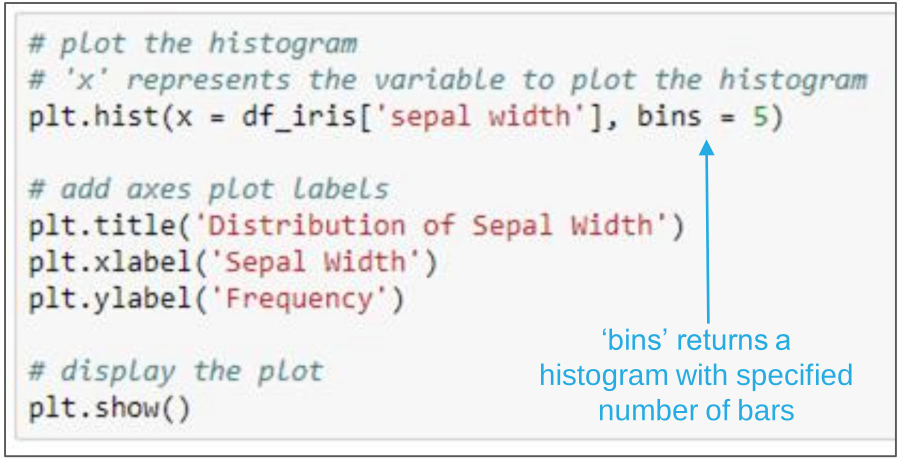
```
# plot the histogram
# 'x' represents the variable to plot the histogram
plt.hist(x = df_iris['sepal width'])

# add axes plot labels
plt.title('Distribution of Sepal Width')
plt.xlabel('Sepal Width')
plt.ylabel('Frequency')

# display the plot
plt.show()
```



Approximately 3 - 3.2 cm is the most occuring sepal width in the data

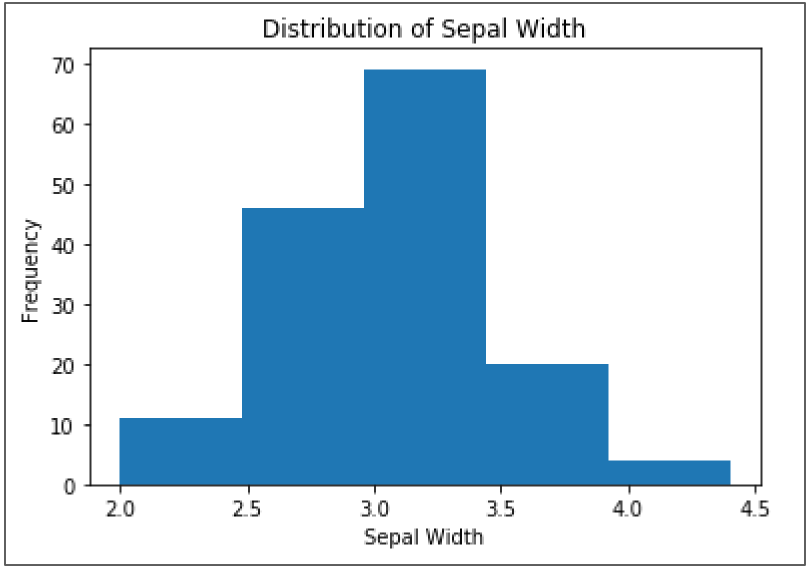
Histogram

Plot a histogram with 5 bins (bars)

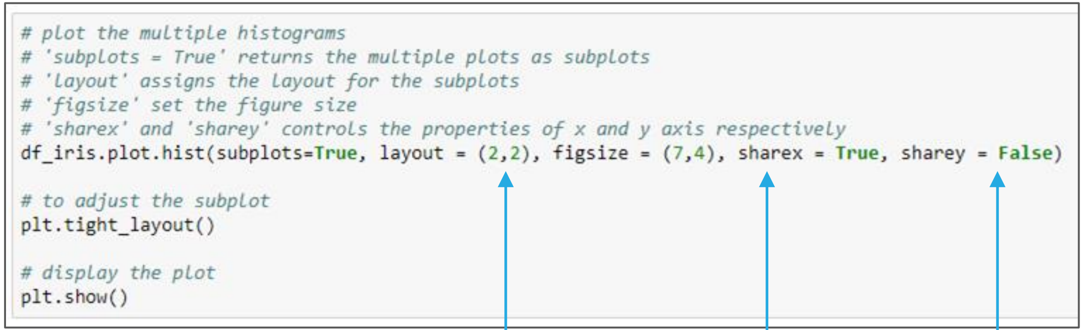


```
# plot the histogram
# 'x' represents the variable to plot the histogram
plt.hist(x = df_iris['sepal width'], bins = 5)

# add axes plot labels
plt.title('Distribution of Sepal Width')
plt.xlabel('Sepal Width')
plt.ylabel('Frequency')

# display the plot
plt.show()
```

‘bins’ returns a histogram with specified number of bars



Create multiple histograms

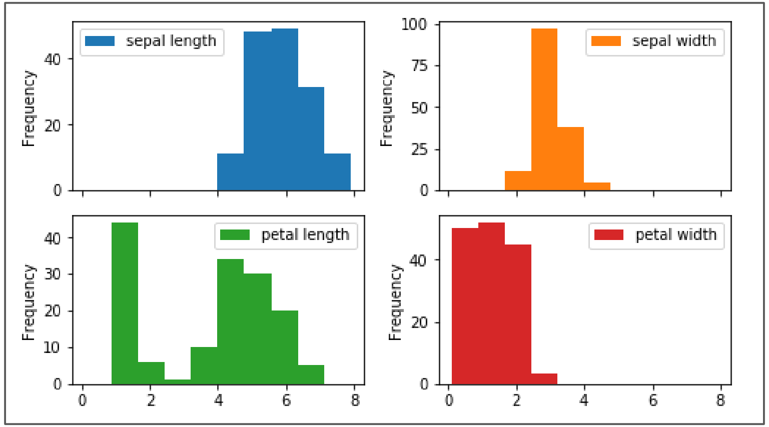


```
# plot the multiple histograms
# 'subplots = True' returns the multiple plots as subplots
# 'layout' assigns the layout for the subplots
# 'figsize' set the figure size
# 'sharex' and 'sharey' controls the properties of x and y axis respectively
df_iris.plot.hist(subplots=True, layout = (2,2), figsize = (7,4), sharex = True, sharey = False)

# to adjust the subplot
plt.tight_layout()

# display the plot
plt.show()
```

Number of rows and columns to plot the subplots

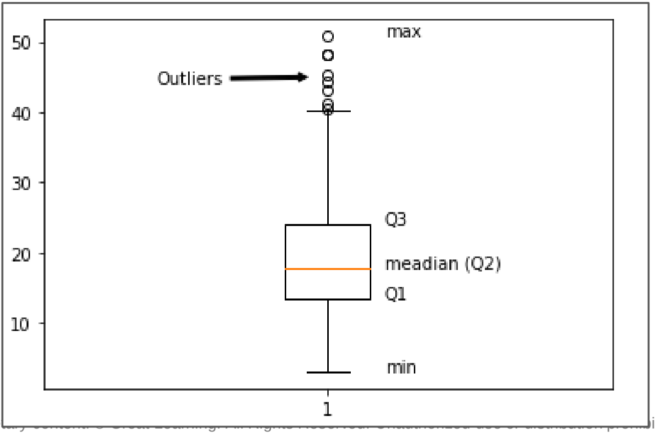
X-axis ticks are same for all subplots

Y-axis ticks are different for all subplots



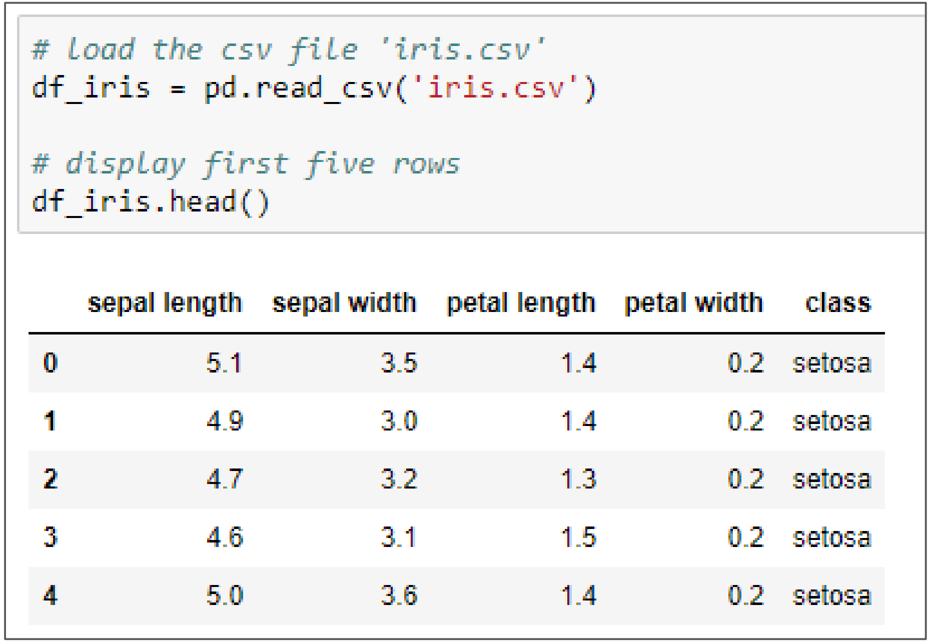
Box plot

- It is used to visualize the distribution of the numeric variable
- Represents the five number summary of the variable which includes the minimum, first quartile (Q1), second quartile (median), third quartile (Q3) and maximum of the variable
- Used to detect the outliers (extreme values) in the data



Read the data

Use the iris data to create the box plot

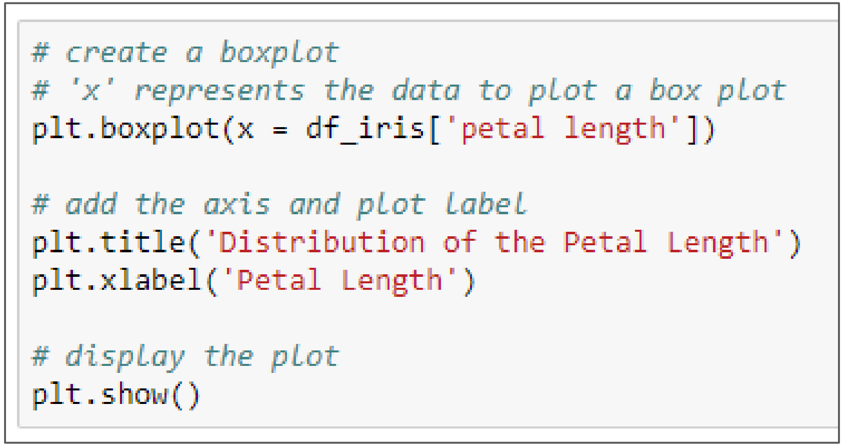


```
# load the csv file 'iris.csv'
df_iris = pd.read_csv('iris.csv')

# display first five rows
df_iris.head()
```

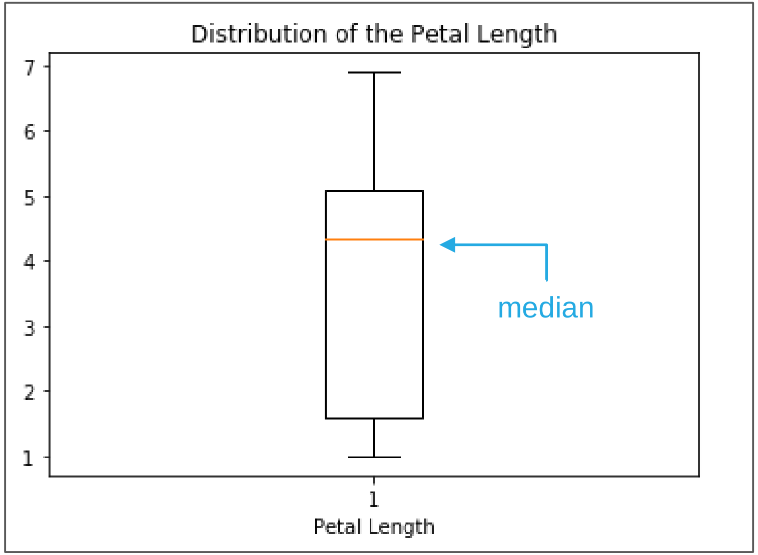
	sepal length	sepal width	petal length	petal width	class
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5.0	3.6	1.4	0.2	setosa

Box plot

Check the distribution of the variable ‘petal length’



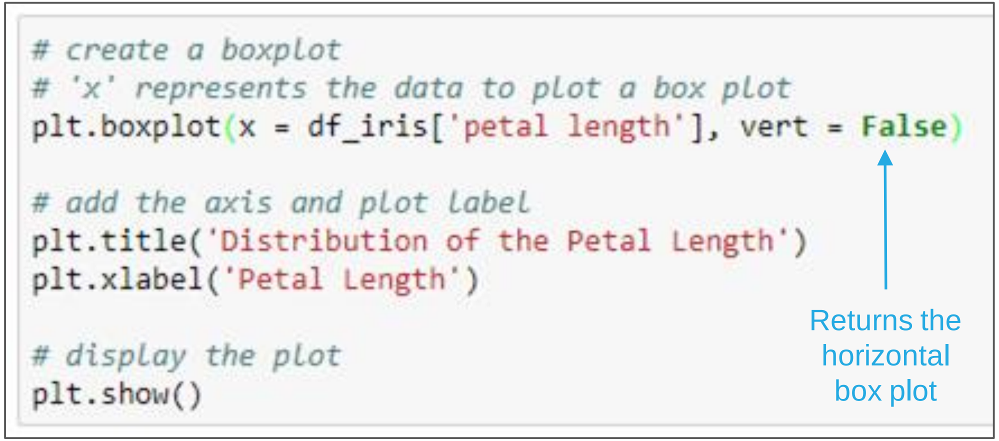
```
# create a boxplot
# 'x' represents the data to plot a box plot
plt.boxplot(x = df_iris['petal length'])

# add the axis and plot label
plt.title('Distribution of the Petal Length')
plt.xlabel('Petal Length')

# display the plot
plt.show()
```



The box plot shows that the variable ‘petal length’ is negatively skewed

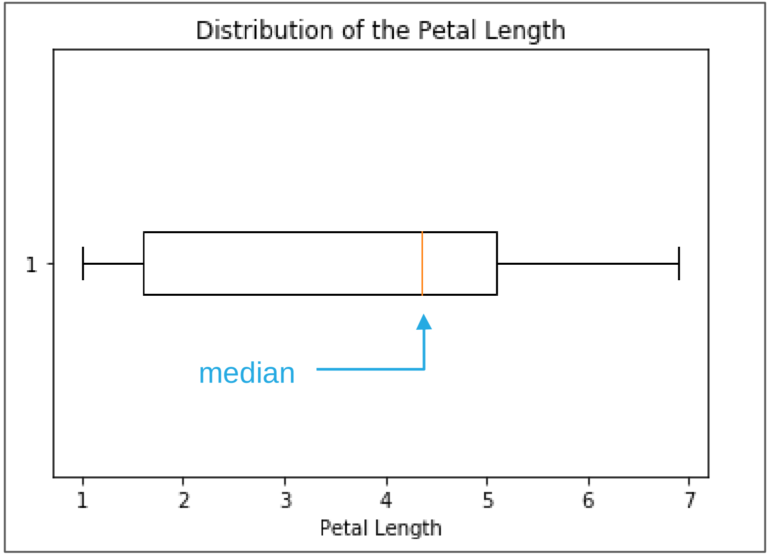
Horizontal box plot

Check the distribution of the variable ‘petal length’



```
# create a boxplot
# 'x' represents the data to plot a box plot
plt.boxplot(x = df_iris['petal length'], vert = False)

# add the axis and plot label
plt.title('Distribution of the Petal Length')
plt.xlabel('Petal Length')

# display the plot
plt.show()
```

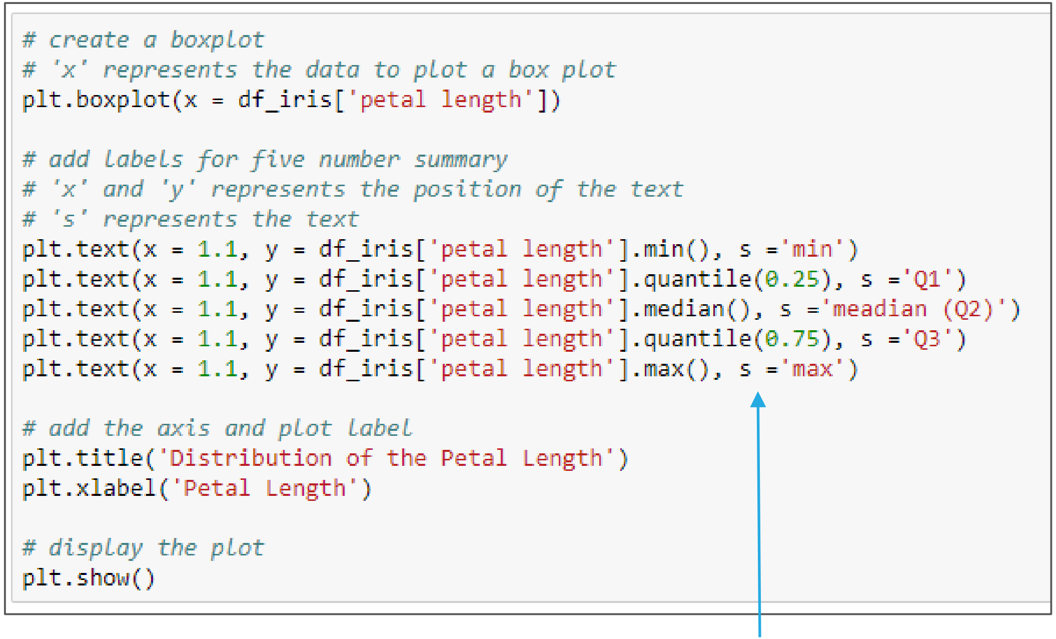
Returns the horizontal box plot



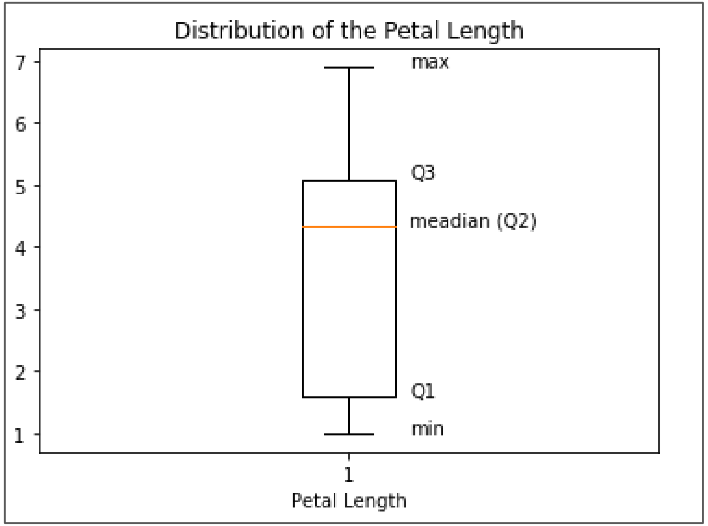
median

The box plot shows that the variable ‘petal length’ is negatively skewed

Add five number summary to box plot



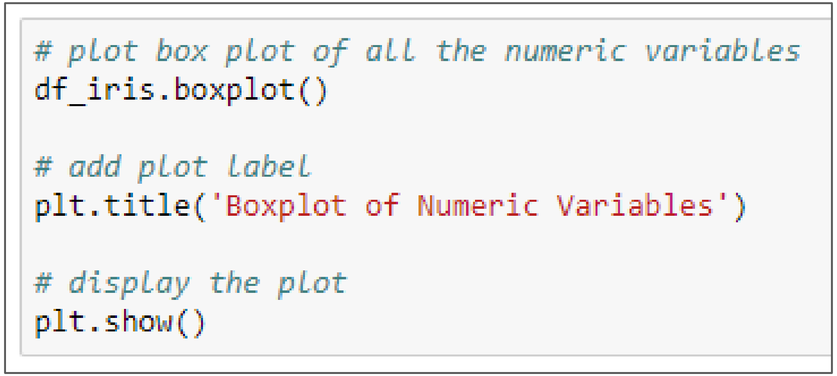
```
# create a boxplot
# 'x' represents the data to plot a box plot
plt.boxplot(x = df_iris['petal length'])

# add labels for five number summary
# 'x' and 'y' represents the position of the text
# 's' represents the text
plt.text(x = 1.1, y = df_iris['petal length'].min(), s ='min')
plt.text(x = 1.1, y = df_iris['petal length'].quantile(0.25), s ='Q1')
plt.text(x = 1.1, y = df_iris['petal length'].median(), s ='meadian (Q2)')
plt.text(x = 1.1, y = df_iris['petal length'].quantile(0.75), s ='Q3')
plt.text(x = 1.1, y = df_iris['petal length'].max(), s ='max')

# add the axis and plot label
plt.title('Distribution of the Petal Length')
plt.xlabel('Petal Length')

# display the plot
plt.show()
```



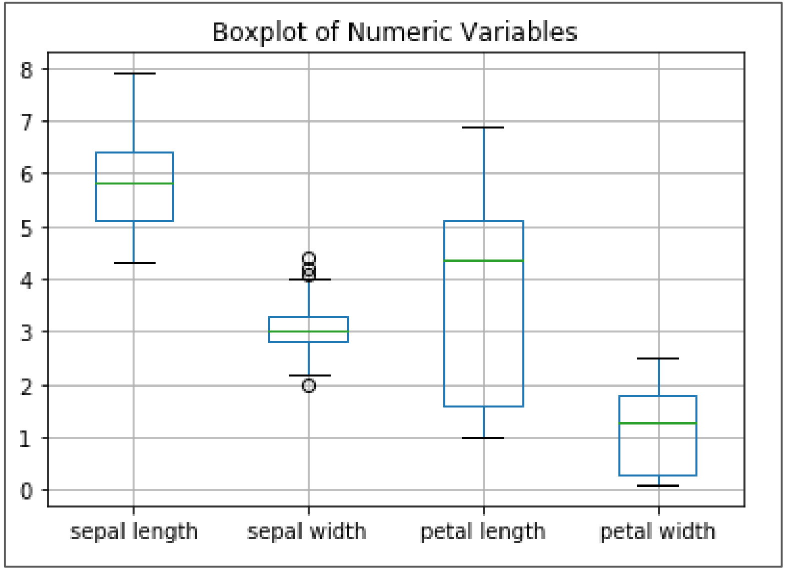
Add text to the plot

Box plot

Plot the boxplot of all the numeric variables in the data



```
# plot box plot of all the numeric variables
df_iris.boxplot()

# add plot label
plt.title('Boxplot of Numeric Variables')

# display the plot
plt.show()
```



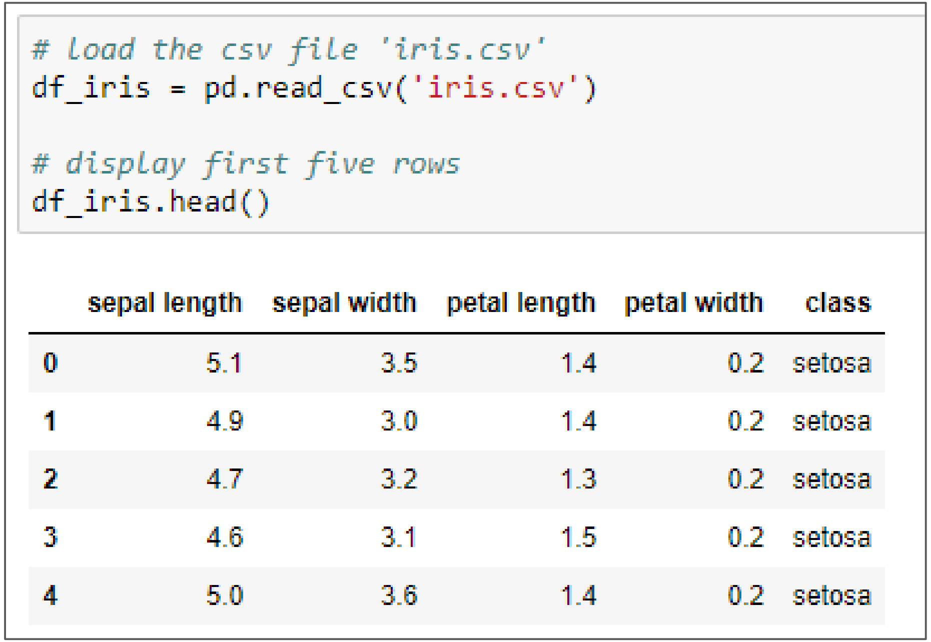
The boxplot of ‘sepal width’ shows the presence of outliers below and above the whiskers

Area plot

- It is similar to a line plot where the area under the line is shaded
- It is used to study the time series data

Read the data

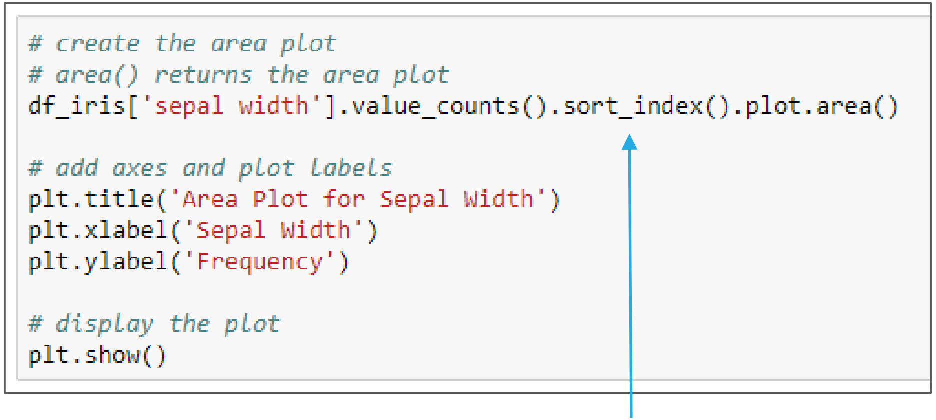
Use the iris data to create the area plot



```python
# load the csv file 'iris.csv'
df_iris = pd.read_csv('iris.csv')

# display first five rows
df_iris.head()
```

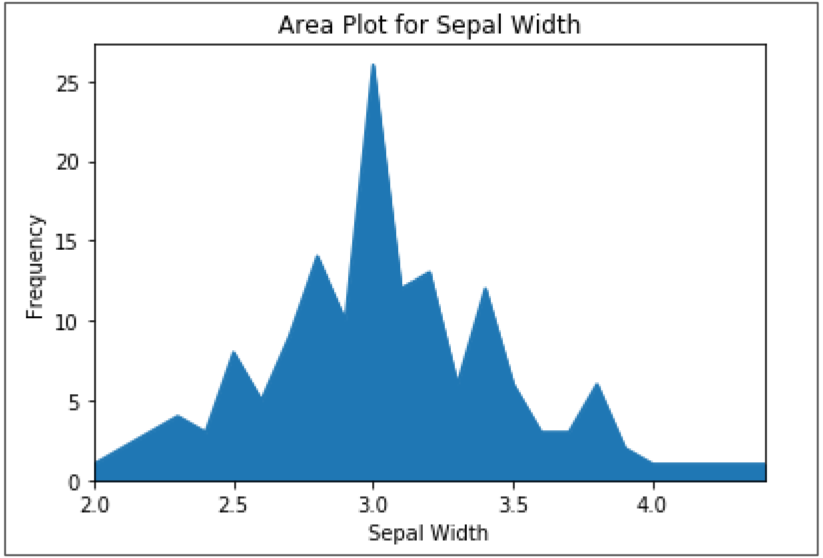
	sepal length	sepal width	petal length	petal width	class
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5.0	3.6	1.4	0.2	setosa

Area plot



```
# create the area plot
# area() returns the area plot
df_iris['sepal width'].value_counts().sort_index().plot.area()

# add axes and plot labels
plt.title('Area Plot for Sepal Width')
plt.xlabel('Sepal Width')
plt.ylabel('Frequency')

# display the plot
plt.show()
```

Calculate the frequency and sort the values to plot the area plot



Thank You