IMARTICUS LEARNING

Python Programming

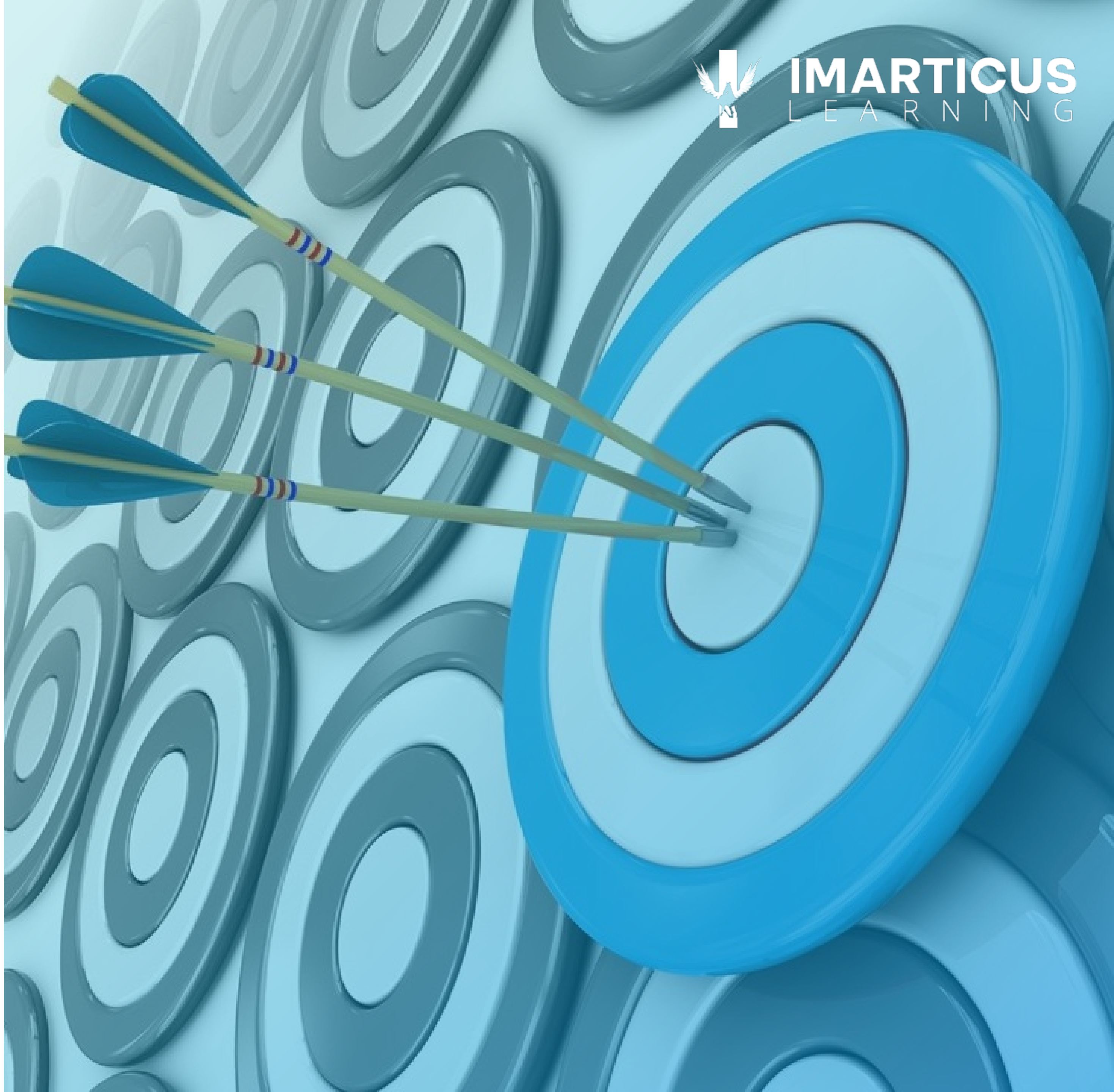
Visualization-Part1

IMARTICUS LEARNING

DISCLAIMER

The training content and delivery of this presentation is confidential, and cannot be recorded, or copied and distributed to any third party, without the written consent of Imarticus Learning Pvt. Ltd.

LEARNING OBJECTIVES



At the end of this session, you will learn:

- Visualization using Matplotlib
- Plot Styles & Settings
- Line Plot
- Multiline Plot
- Matplotlib Subplots
- Histogram
- Boxplot
- Pie Chart
- Scatter Plot

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

Visualization using Matplotlib

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

Plots in Matplotlib have a hierarchical structure that nests Python objects to create a tree-like structure

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

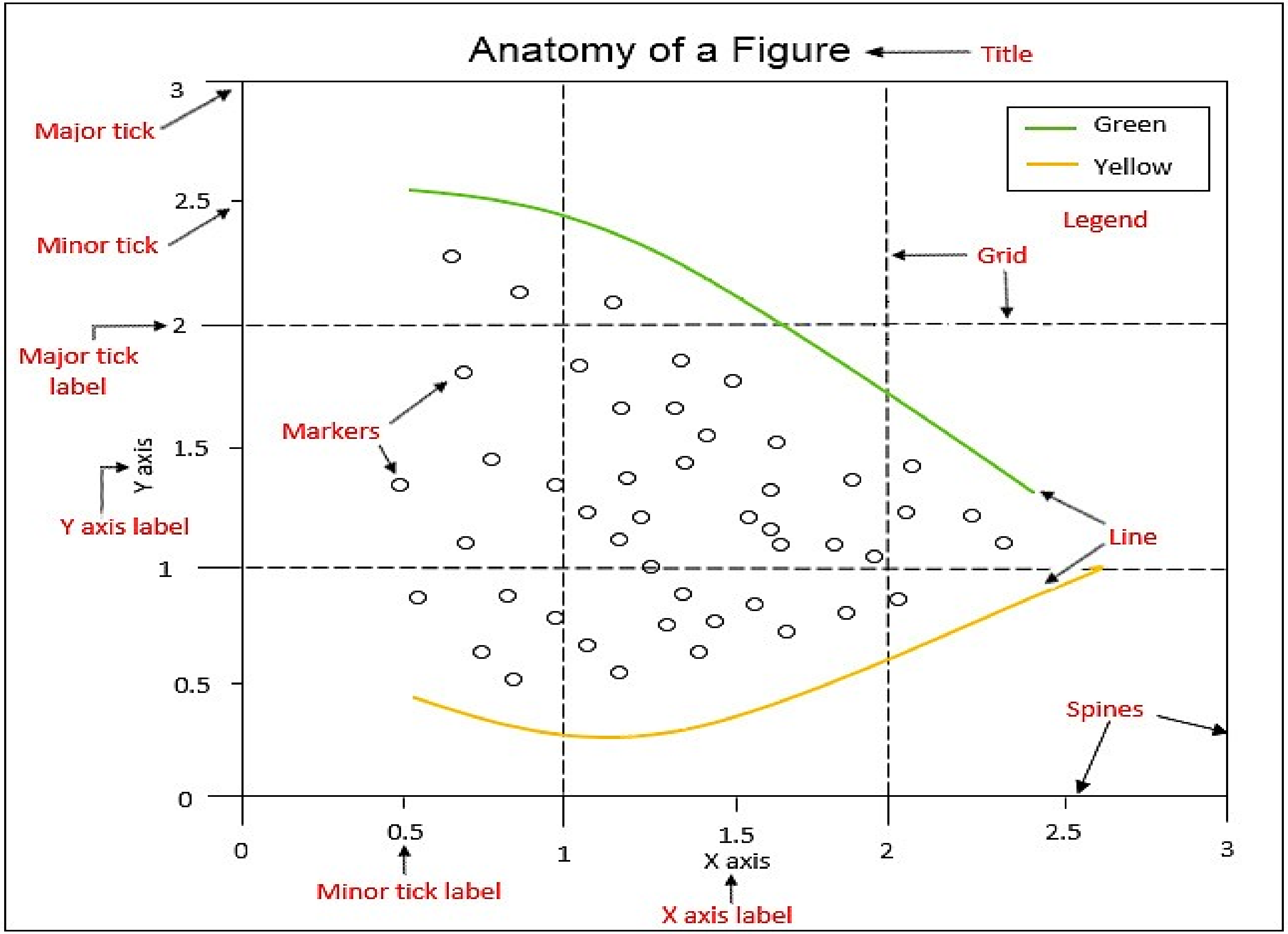
Alternatively, you can install matplotlib in a jupyter notebook using below code:

```
!pip install matplotlib
```

To import subpackage ‘pyplot’, use the command:

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

ANATOMY OF A MATPLOTIB



IMPORTANT COMMANDS OF MATPLOTIB

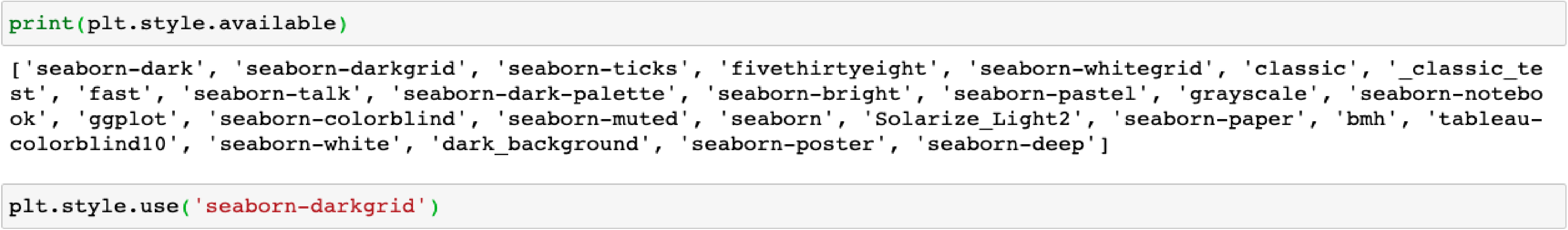
- **plt.plot()** - Create a plot object
- **plt.show()** - Explicit command required to display the plot object
- **plt.xlabel(), plt.ylabel()** - Specify labels for the X and Y axis respectively.
- **plt.title()** - Add a title to the plot object

Plot Styles and Settings

MATPLOTIB STYLES

- The style package adds support to change plotting styles
- There are a number of predefined styles
- To see list of available styles use **print(plt.style.available)**

MATPLOTIB STYLES

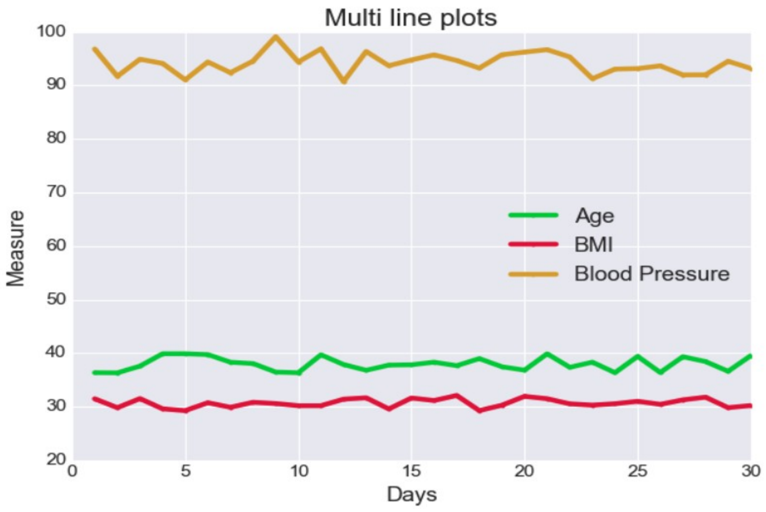


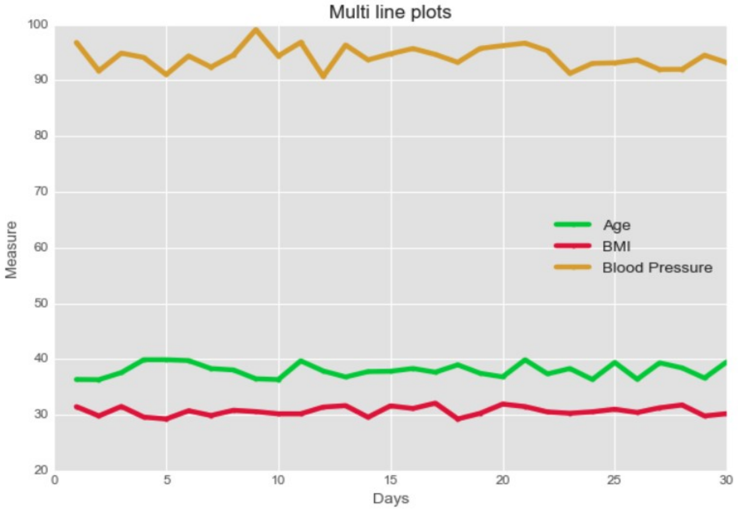
```
print(plt.style.available)
```

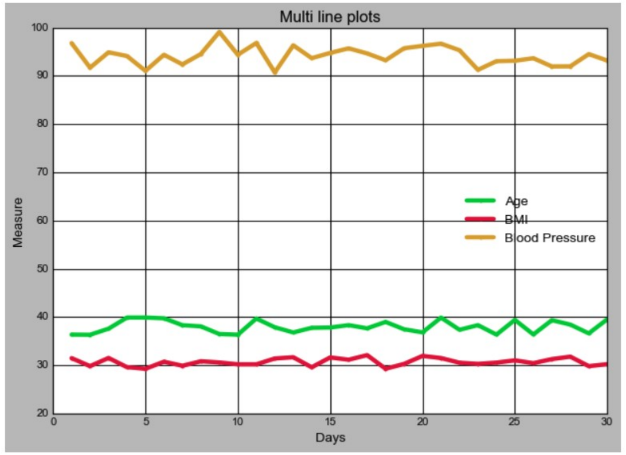
```
['seaborn-dark', 'seaborn-darkgrid', 'seaborn-ticks', 'fivethirtyeight', 'seaborn-whitegrid', 'classic', '_classic_te
st', 'fast', 'seaborn-talk', 'seaborn-dark-palette', 'seaborn-bright', 'seaborn-pastel', 'grayscale', 'seaborn-notebo
ok', 'ggplot', 'seaborn-colorblind', 'seaborn-muted', 'seaborn', 'Solarize_Light2', 'seaborn-paper', 'bmh', 'tableau-
colorblind10', 'seaborn-white', 'dark_background', 'seaborn-poster', 'seaborn-deep']
```

```
plt.style.use('seaborn-darkgrid')
```

Multi Line Plots



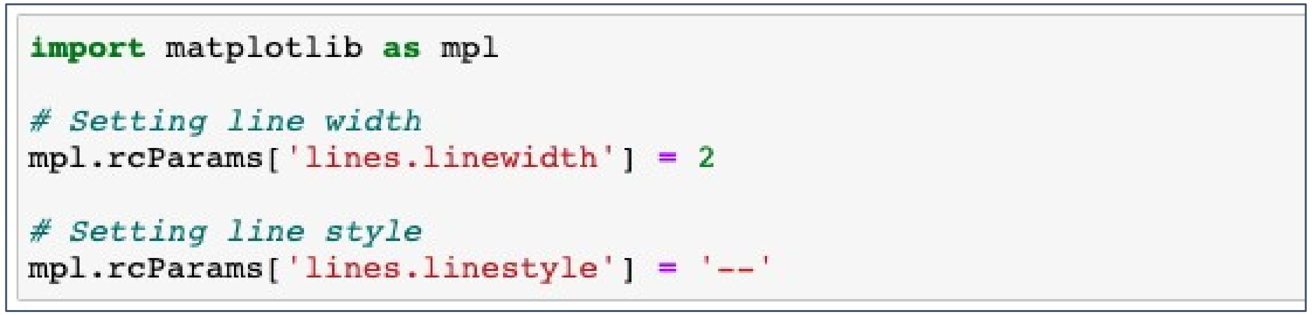




MATPLOTIB GLOBAL SETTINGS

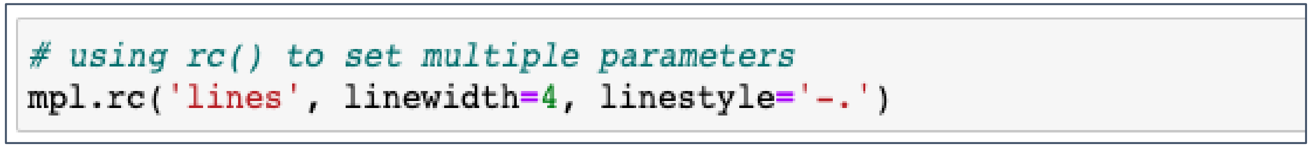
- The matplotlib.rcparams can be used to set global settings
- The matplotlib.rc() command can be used to modify multiple settings using keyword arguments
- https://matplotlib.org/3.1.1/tutorials/introductory/customizing.html

MATPLOTIB GLOBAL SETTINGS



```
import matplotlib as mpl

# Setting line width
mpl.rcParams['lines.linewidth'] = 2

# Setting line style
mpl.rcParams['lines.linestyle'] = '--'
```

Using rcParams[ ]



```
# using rc() to set multiple parameters
mpl.rc('lines', linewidth=4, linestyle='-.')
```

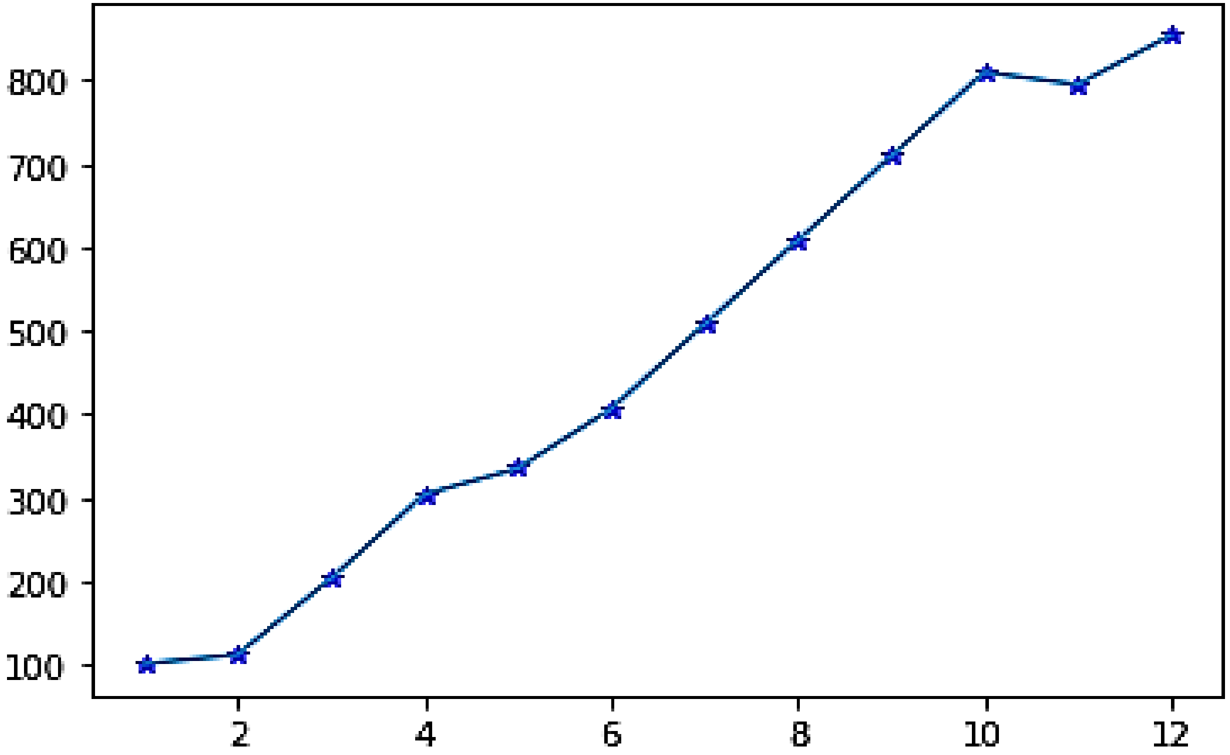
Using rc()

For complete documentation: https://matplotlib.org/3.1.1/tutorials/introductory/customizing.html

Line Plot

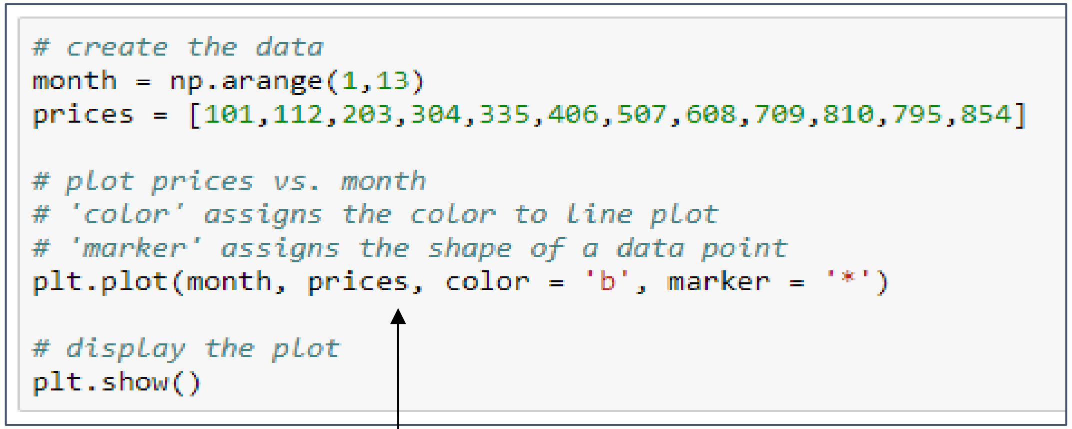
LINE PLOT

Line Plot is a simple plot that displays the relationship between two variables



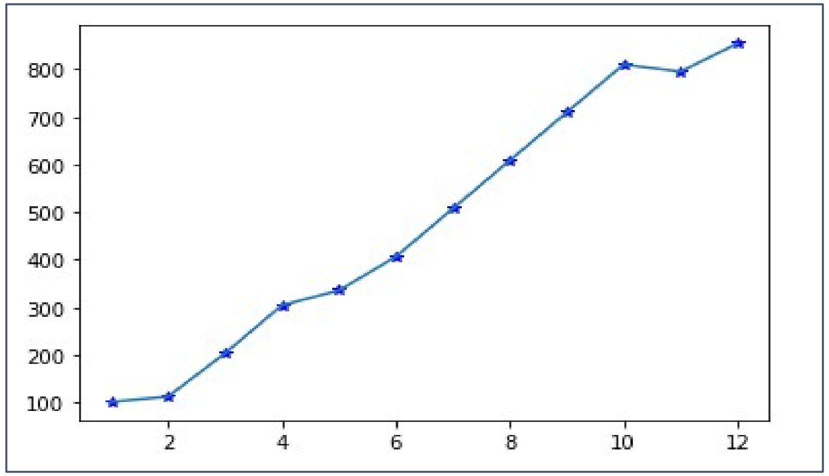
PLOT A LINE PLOT FROM A LIST

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

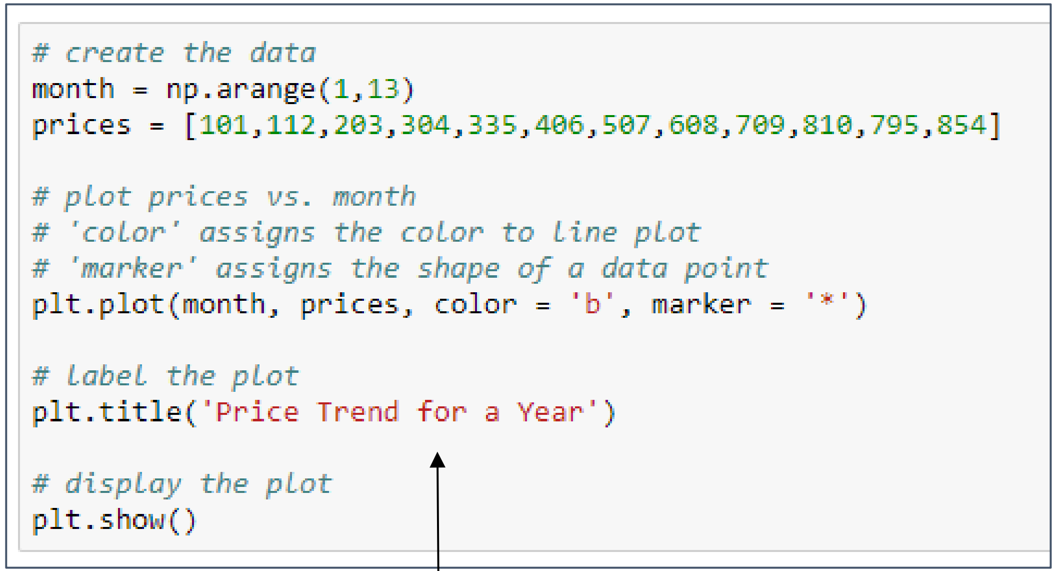


```
# create the data
month = np.arange(1,13)
prices = [101,112,203,304,335,406,507,608,709,810,795,854]

# 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

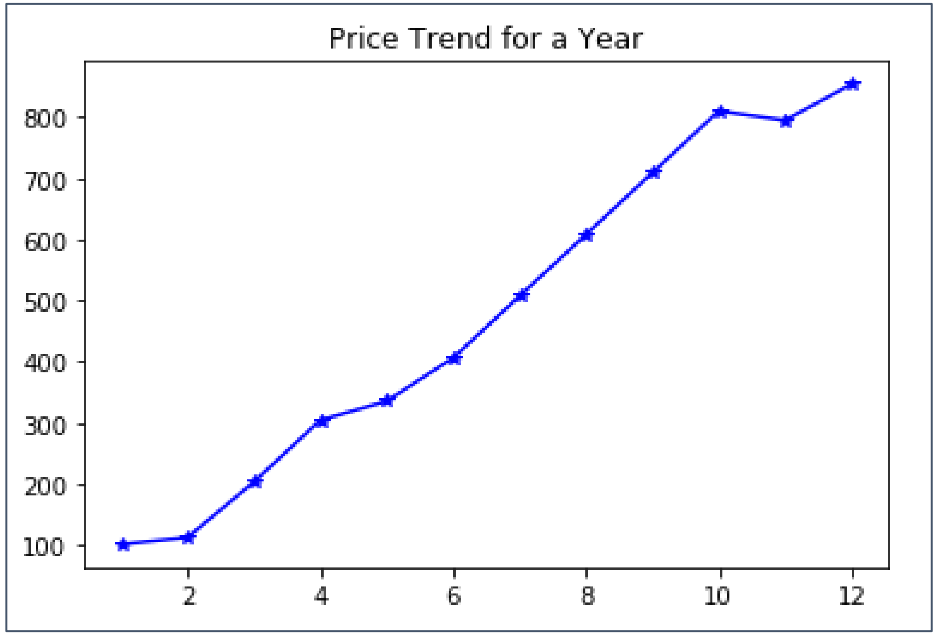


ADD TITLE TO THE GRAPH

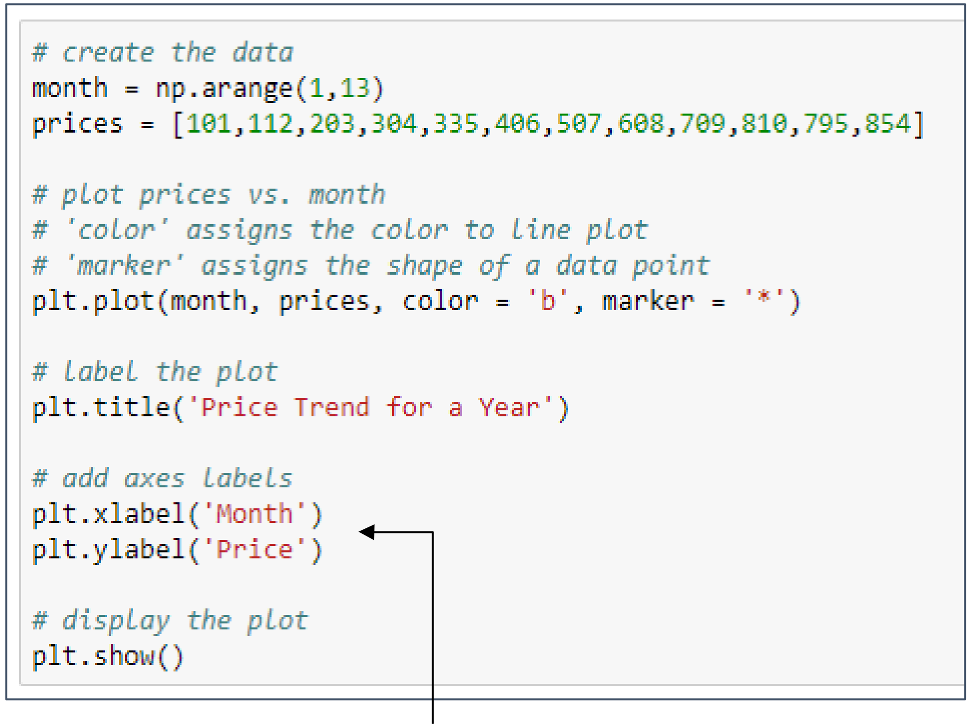


```
# create the data
month = np.arange(1,13)
prices = [101,112,203,304,335,406,507,608,709,810,795,854]

# 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()
```

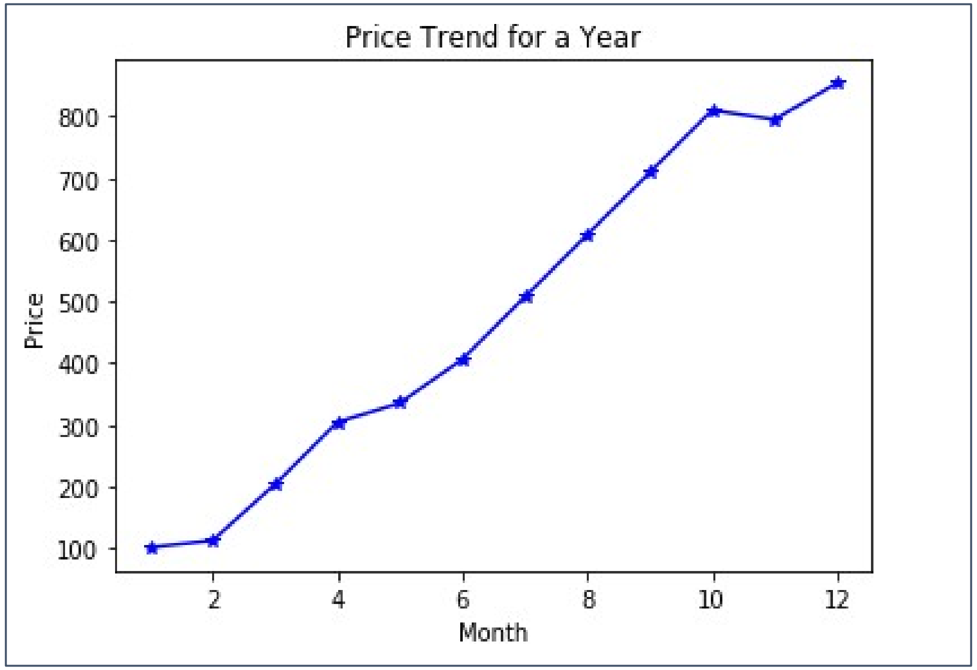
Put a title to the plot



ADD AXES LABELS

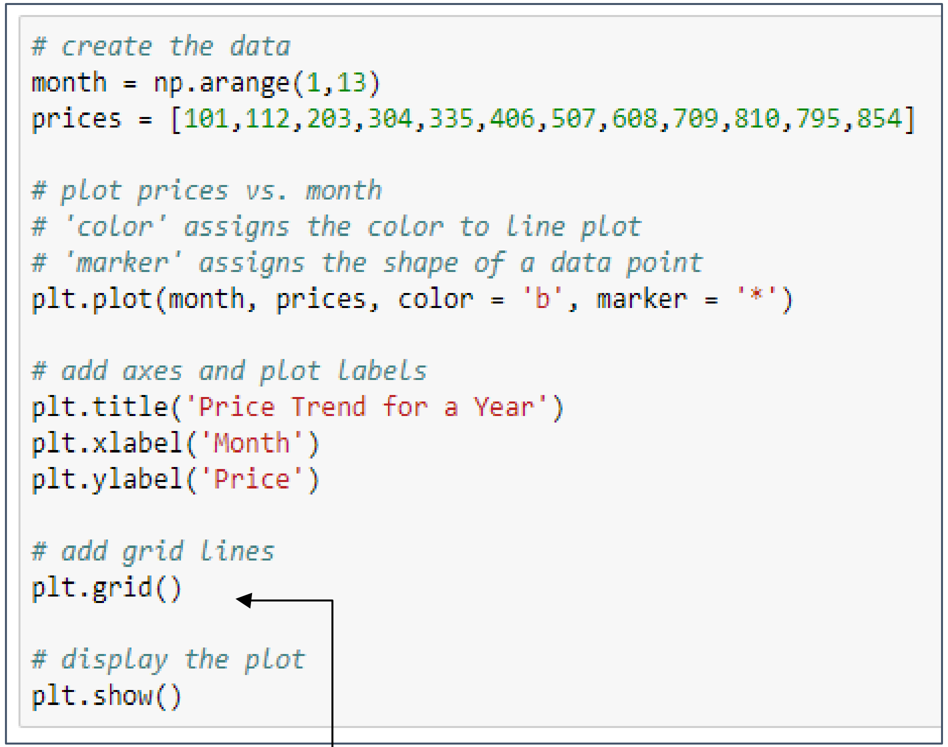


```
# create the data
month = np.arange(1,13)
prices = [101,112,203,304,335,406,507,608,709,810,795,854]

# 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()
```



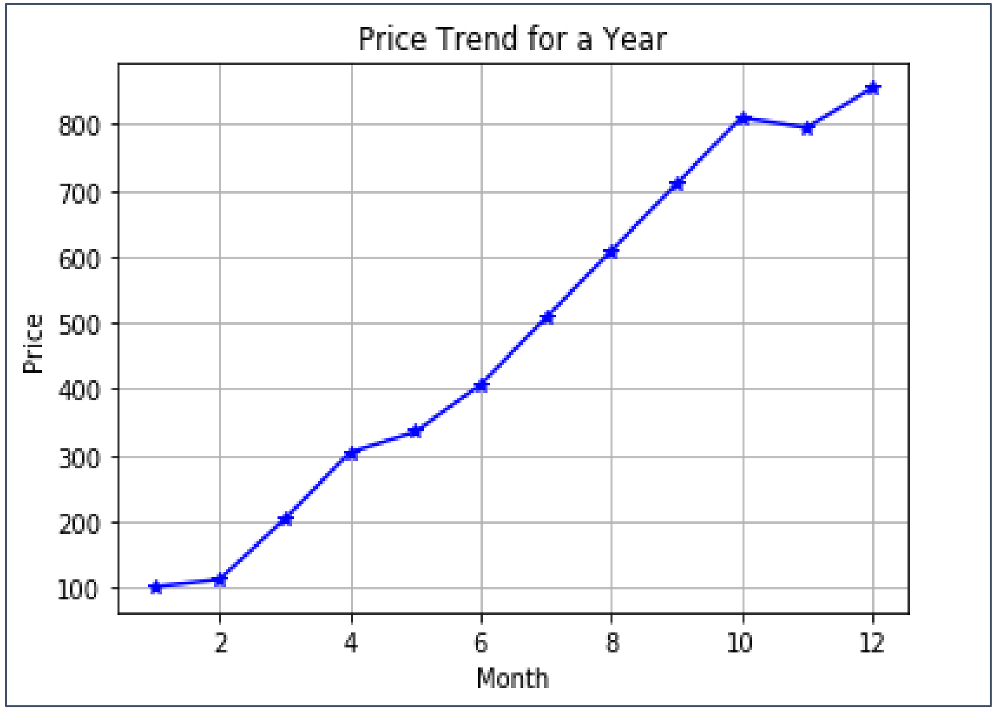
Add labels to x and y axis

ADD GRID LINES TO THE PLOT

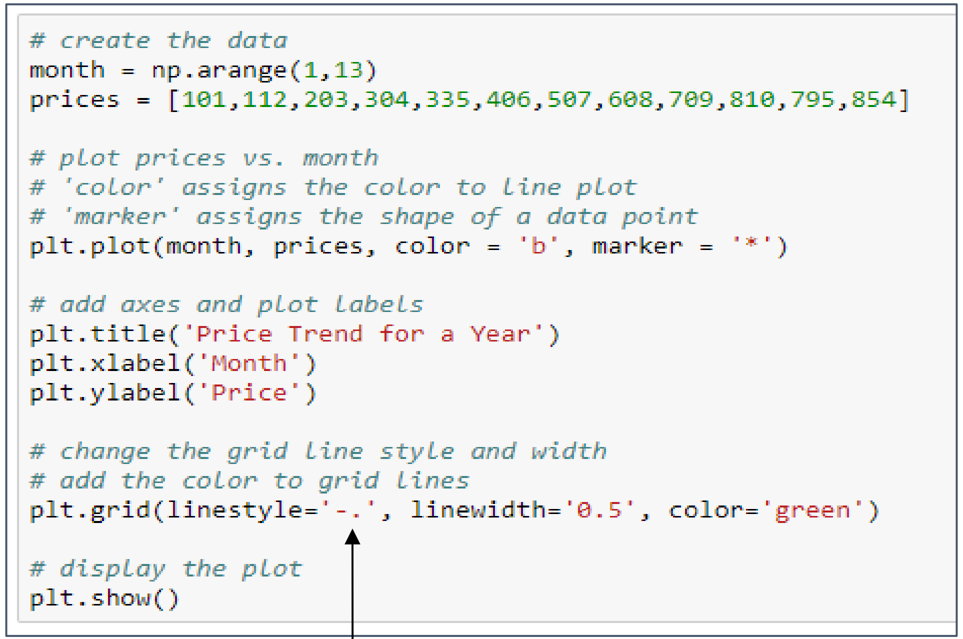


```
# create the data
month = np.arange(1,13)
prices = [101,112,203,304,335,406,507,608,709,810,795,854]

# 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()
```

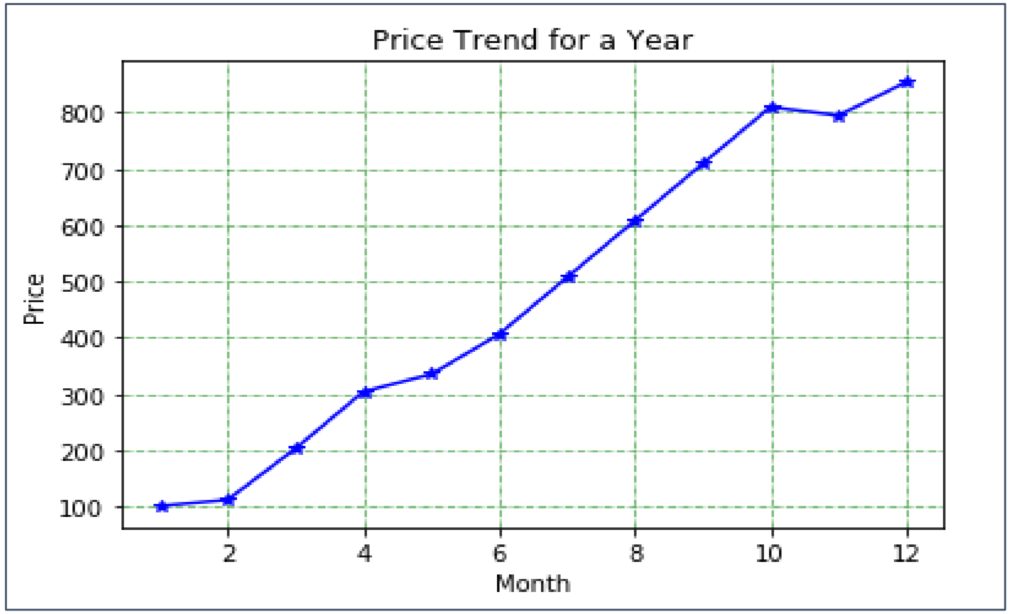
Add grid lines



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()
```

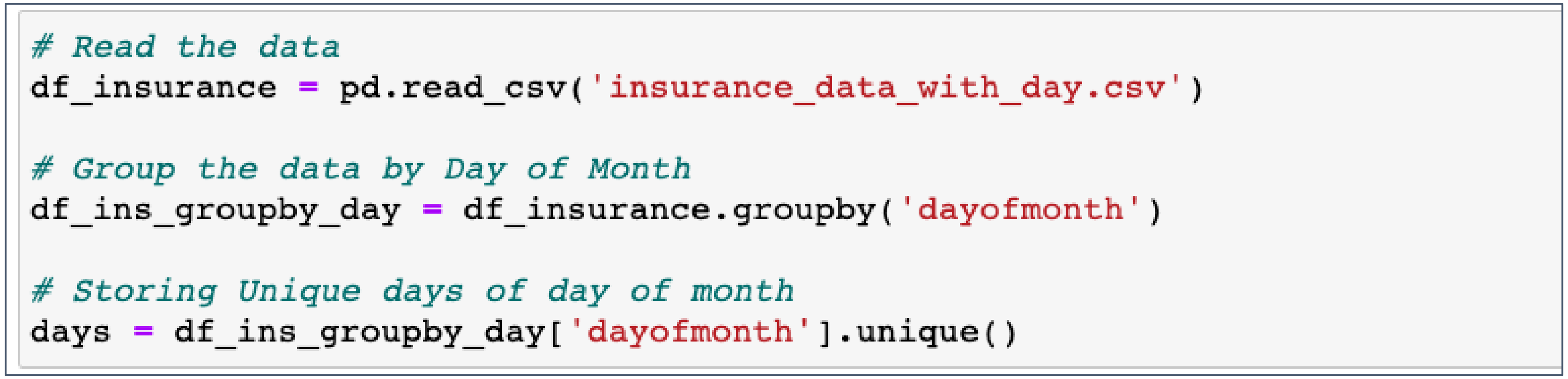


Change style, width and color of grid lines

Multiline Plot

MULTILINE PLOT

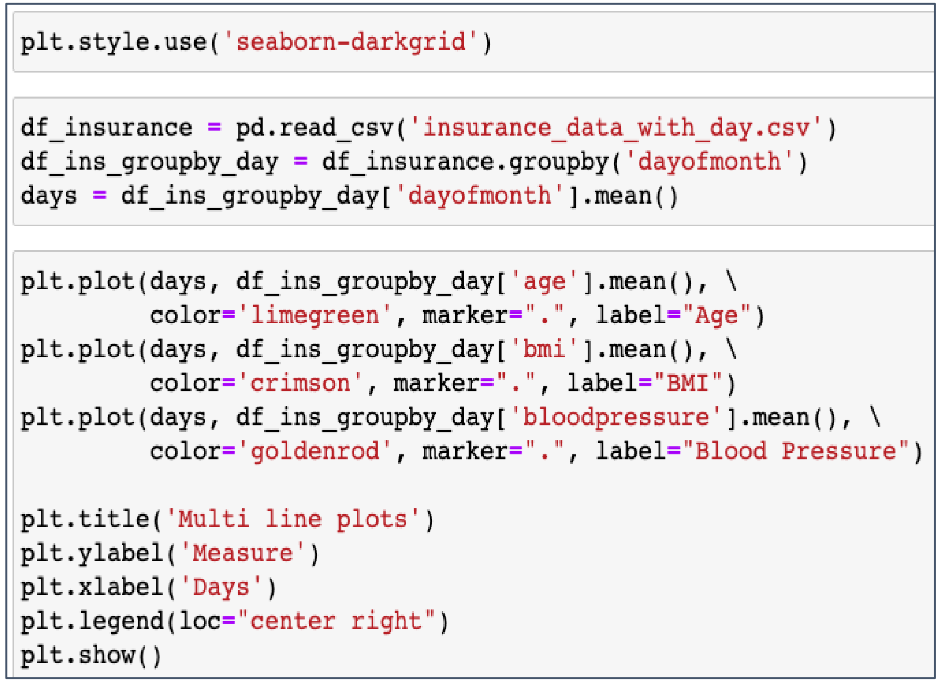
Read the Data, group the data by day of month



```
# Read the data
df_insurance = pd.read_csv('insurance_data_with_day.csv')

# Group the data by Day of Month
df_ins_groupby_day = df_insurance.groupby('dayofmonth')

# Storing Unique days of day of month
days = df_ins_groupby_day['dayofmonth'].unique()
```

MULTIPLE LINE PLOTS WITH MULTIPLE GROUPS

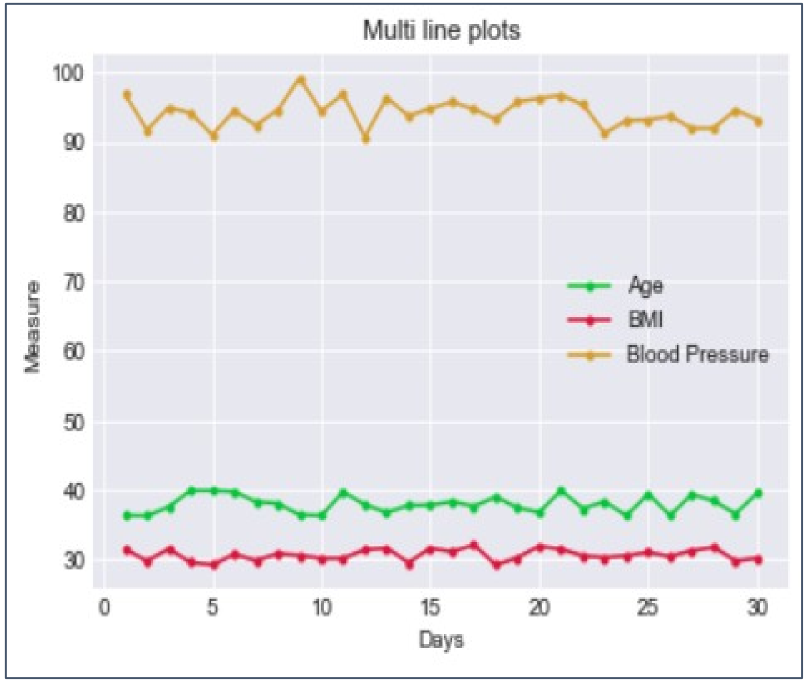
Each line represents different metrics of a patient like bmi, bp and age



```
plt.style.use('seaborn-darkgrid')
```

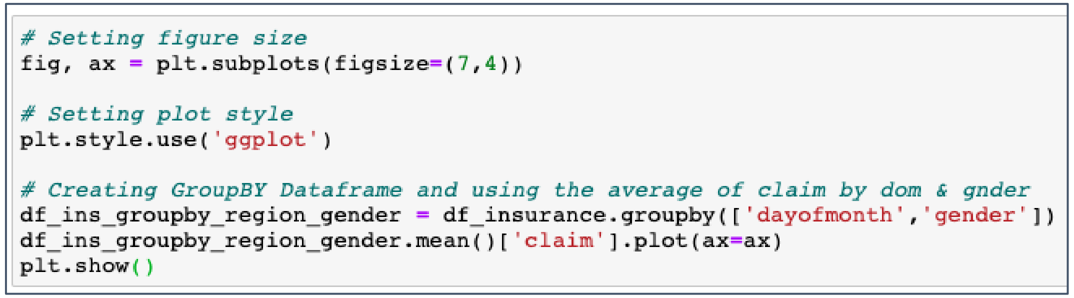
```
df_insurance = pd.read_csv('insurance_data_with_day.csv')
df_ins_groupby_day = df_insurance.groupby('dayofmonth')
days = df_ins_groupby_day['dayofmonth'].mean()
```

```
plt.plot(days, df_ins_groupby_day['age'].mean(), \
         color='limegreen', marker=".", label="Age")
plt.plot(days, df_ins_groupby_day['bmi'].mean(), \
         color='crimson', marker=".", label="BMI")
plt.plot(days, df_ins_groupby_day['bloodpressure'].mean(), \
         color='goldenrod', marker=".", label="Blood Pressure")

plt.title('Multi line plots')
plt.ylabel('Measure')
plt.xlabel('Days')
plt.legend(loc="center right")
plt.show()
```

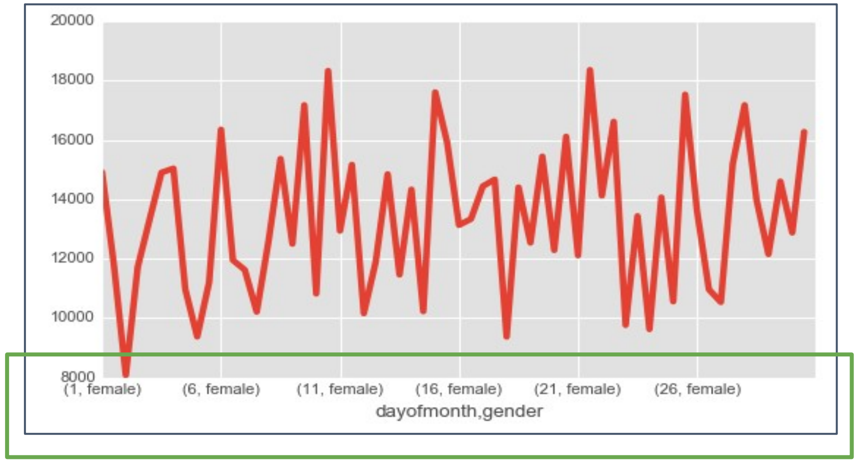


MULTIPLE LINE PLOTS

Each line represents day-wise average claim by gender



```
# Setting figure size
fig, ax = plt.subplots(figsize=(7,4))

# Setting plot style
plt.style.use('ggplot')

# Creating GroupBY Dataframe and using the average of claim by dom & gnder
df_ins_groupby_region_gender = df_insurance.groupby(['dayofmonth','gender'])
df_ins_groupby_region_gender.mean()['claim'].plot(ax=ax)
plt.show()
```

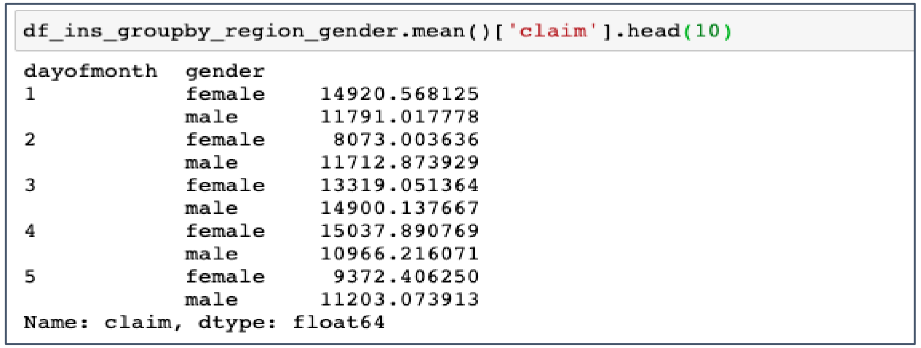


What went wrong? Looks like day of month & gender both appear in the x-axis

MULTIPLE LINE PLOTS

What went wrong?

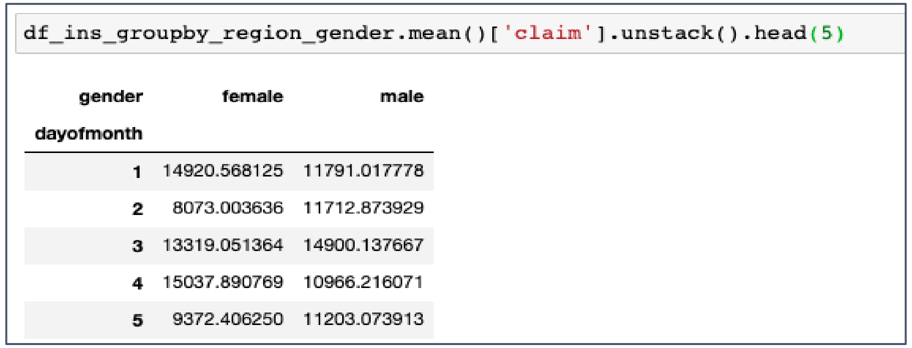
Dataframe after we applied groupby()



```
df_ins_groupby_region_gender.mean()['claim'].head(10)
```

```
dayofmonth  gender
1           female    14920.568125
            male      11791.017778
2           female     8073.003636
            male      11712.873929
3           female    13319.051364
            male      14900.137667
4           female    15037.890769
            male      10966.216071
5           female     9372.406250
            male      11203.073913
Name: claim, dtype: float64
```

Dataframe after we applied unstack() on groupby(). This pivots a level of row index to column axis

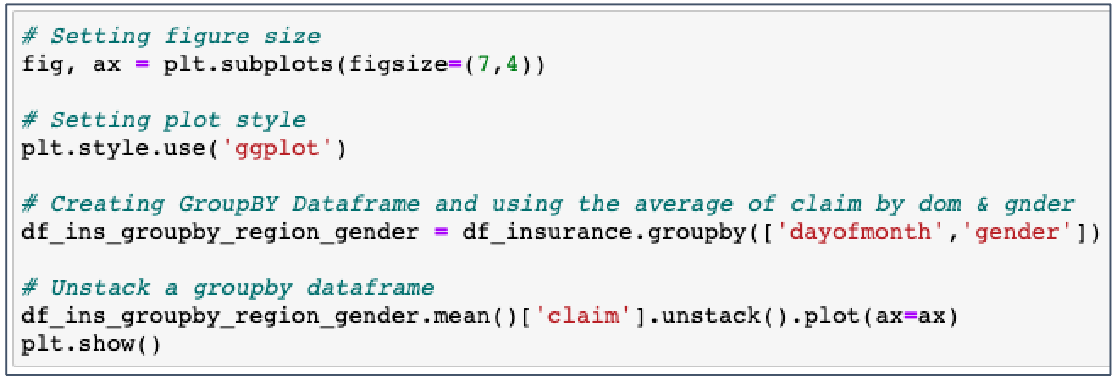


```
df_ins_groupby_region_gender.mean()['claim'].unstack().head(5)
```

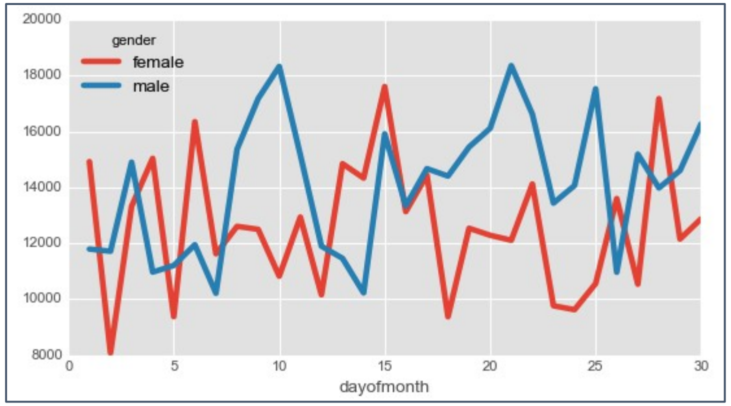
gender	female	male
dayofmonth		
1	14920.568125	11791.017778
2	8073.003636	11712.873929
3	13319.051364	14900.137667
4	15037.890769	10966.216071
5	9372.406250	11203.073913

MULTIPLE LINE PLOTS

After we use unstack() on groupby()



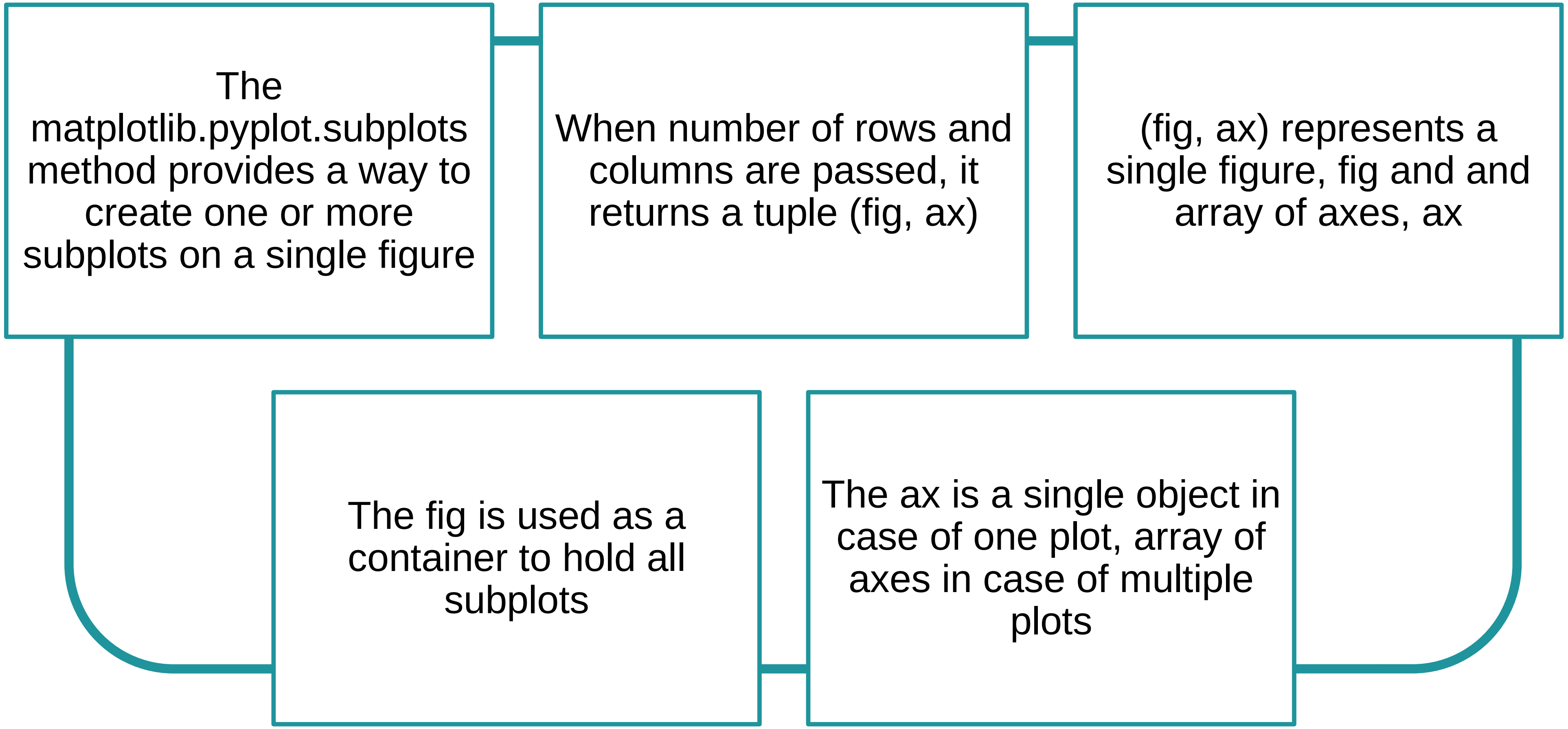
```
# Setting figure size
fig, ax = plt.subplots(figsize=(7,4))

# Setting plot style
plt.style.use('ggplot')

# Creating GroupBY Dataframe and using the average of claim by dom & gnder
df_ins_groupby_region_gender = df_insurance.groupby(['dayofmonth','gender'])

# Unstack a groupby dataframe
df_ins_groupby_region_gender.mean()['claim'].unstack().plot(ax=ax)
plt.show()
```



The unstack() pivots a level of row index to column axis.

Matplotlib Subplots

SUBPLOTS



The matplotlib.pyplot.subplots method provides a way to create one or more subplots on a single figure

When number of rows and columns are passed, it returns a tuple (fig, ax)

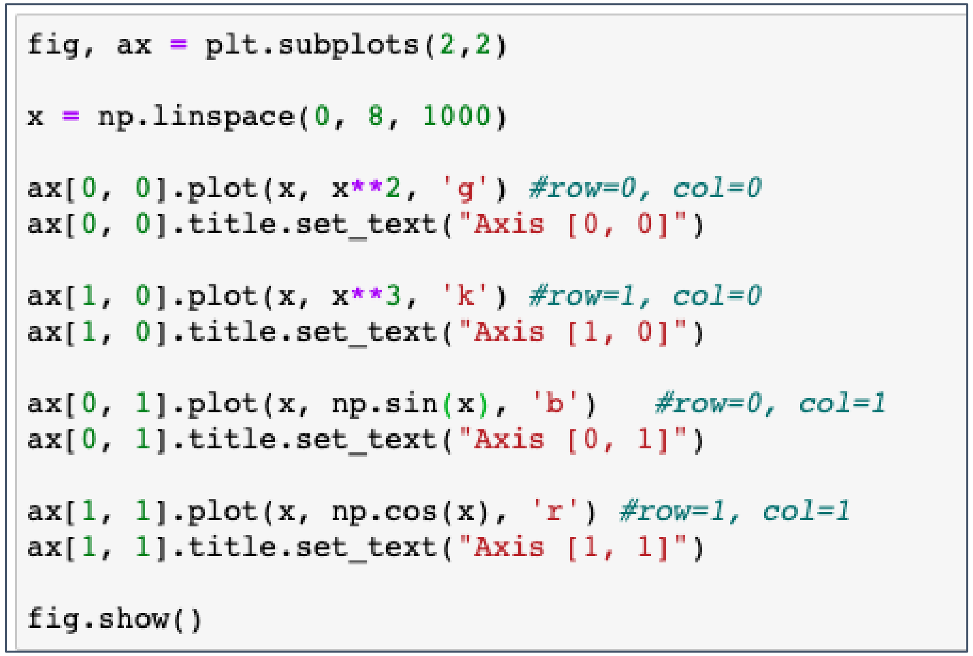
(fig, ax) represents a single figure, fig and and array of axes, ax

The fig is used as a container to hold all subplots

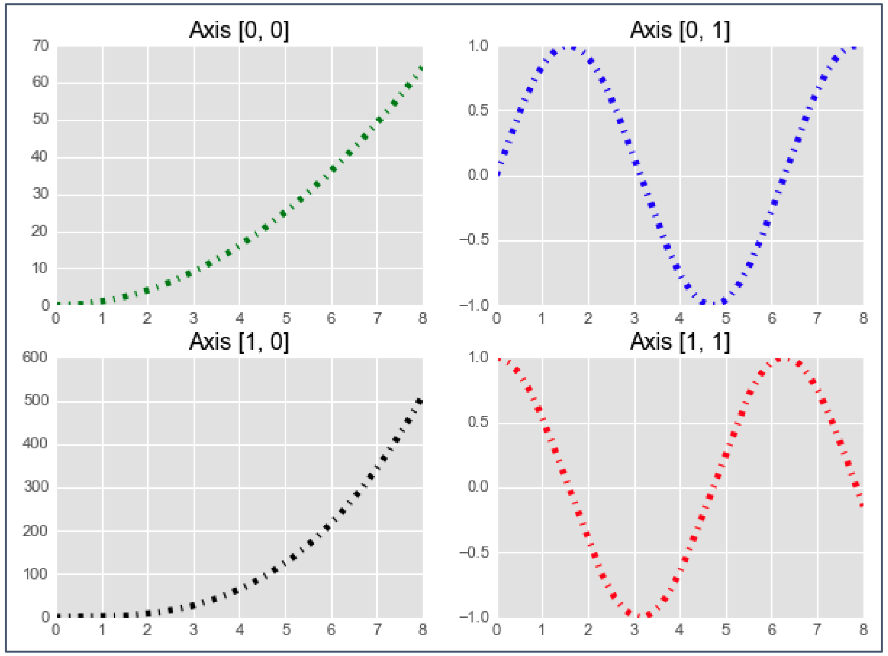
The ax is a single object in case of one plot, array of axes in case of multiple plots

SUBPLOTS

The subplots(2,2) generates 2 by 2 subplots



```
fig, ax = plt.subplots(2,2)

x = np.linspace(0, 8, 1000)

ax[0, 0].plot(x, x**2, 'g') #row=0, col=0
ax[0, 0].title.set_text("Axis [0, 0]")

ax[1, 0].plot(x, x**3, 'k') #row=1, col=0
ax[1, 0].title.set_text("Axis [1, 0]")

ax[0, 1].plot(x, np.sin(x), 'b')   #row=0, col=1
ax[0, 1].title.set_text("Axis [0, 1]")

ax[1, 1].plot(x, np.cos(x), 'r') #row=1, col=1
ax[1, 1].title.set_text("Axis [1, 1]")

fig.show()
```



Axis [0, 0]

Axis [0, 1]

Axis [1, 0]

Axis [1, 1]

Histogram

HISTOGRAM

It is used to represent the distribution of the numeric variable

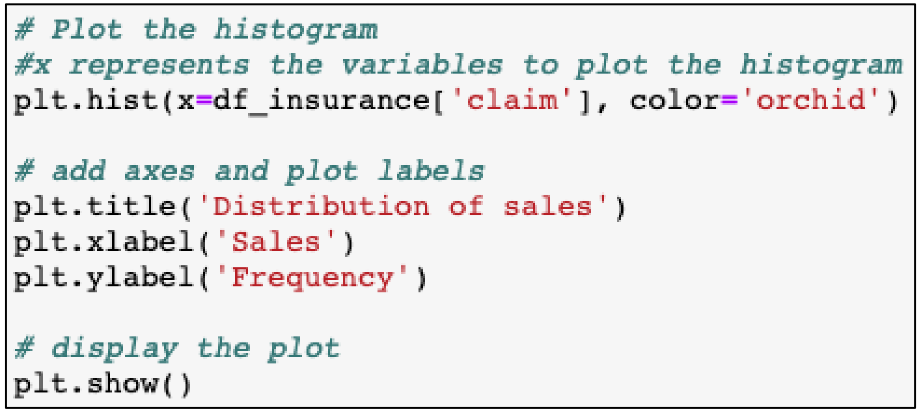
It is an estimate of the probability distribution of a continuous data

One axis represents the variable in the form of bars and another represents the frequency each bar

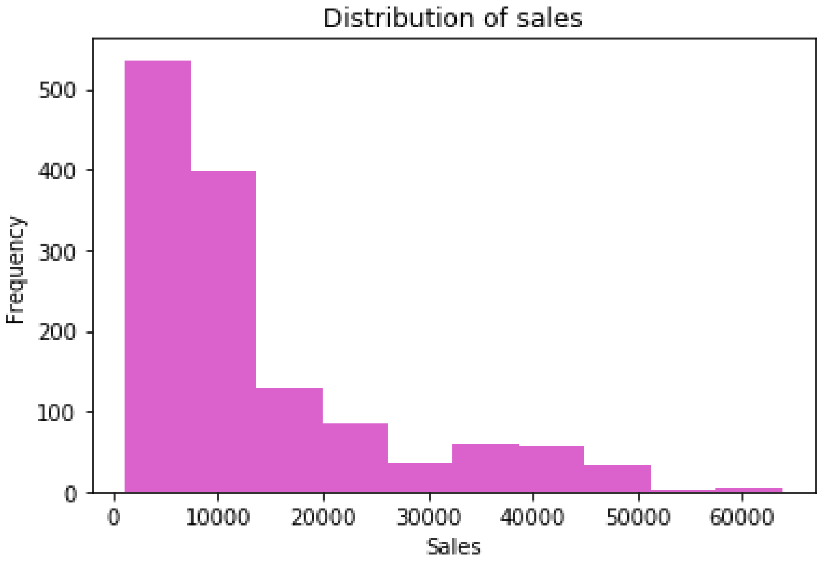
There are no gaps between the bars of the histogram

HISTOGRAM

Plot the histogram to check the distribution of the variable, ‘claim’



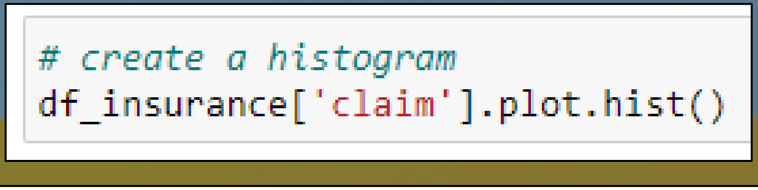
```
# Plot the histogram
#x represents the variables to plot the histogram
plt.hist(x=df_insurance['claim'], color='orchid')

# add axes and plot labels
plt.title('Distribution of sales')
plt.xlabel('Sales')
plt.ylabel('Frequency')

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



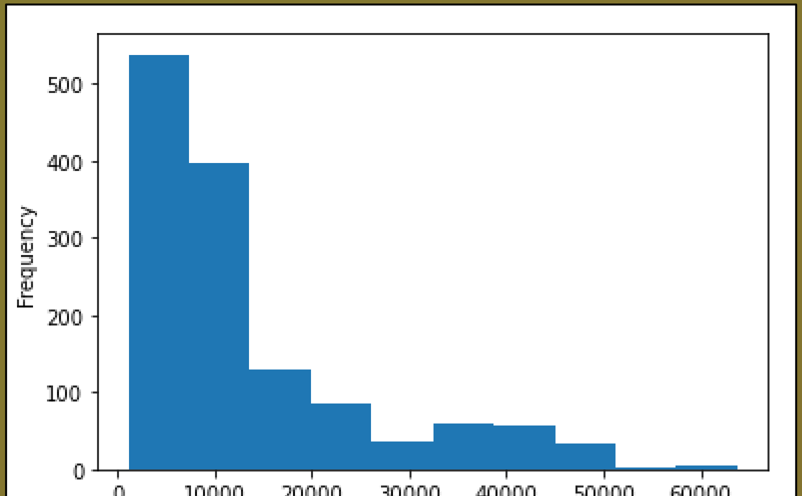
The histogram shows the variable ‘claim’ is right skewed

Did You Know?

Pandas has **tight integration** with matplotlib. You can plot data *directly* from your DataFrame using the *plot()* method



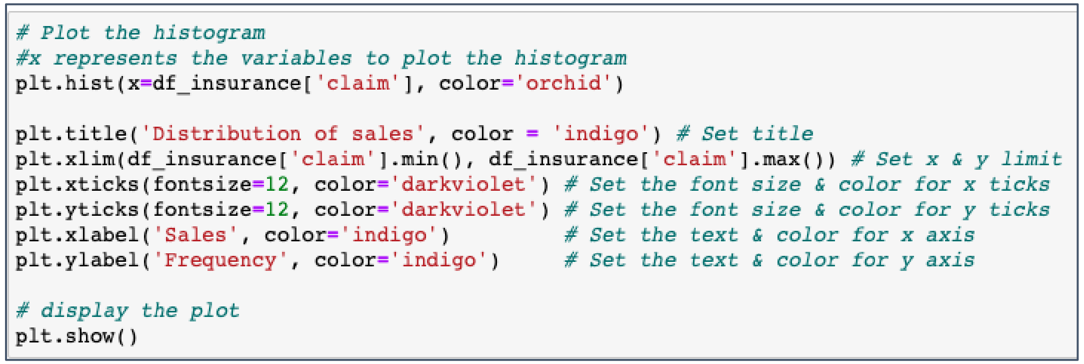
```
# create a histogram
df_insurance['claim'].plot.hist()
```



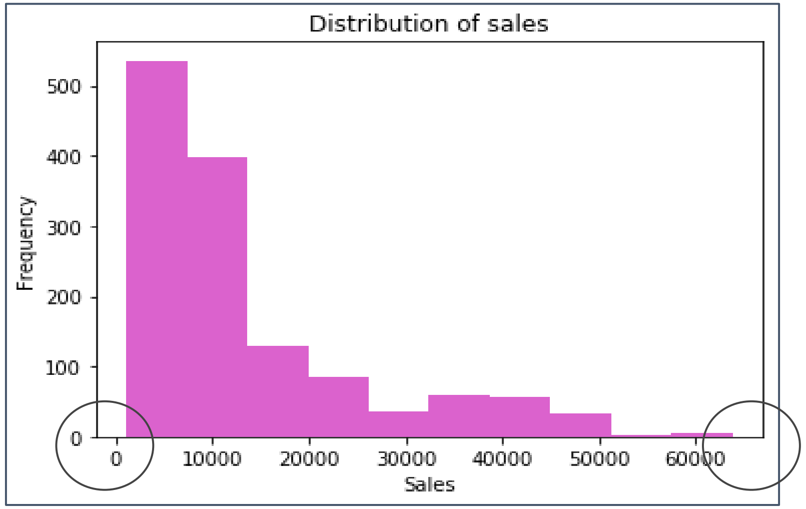
HISTOGRAM

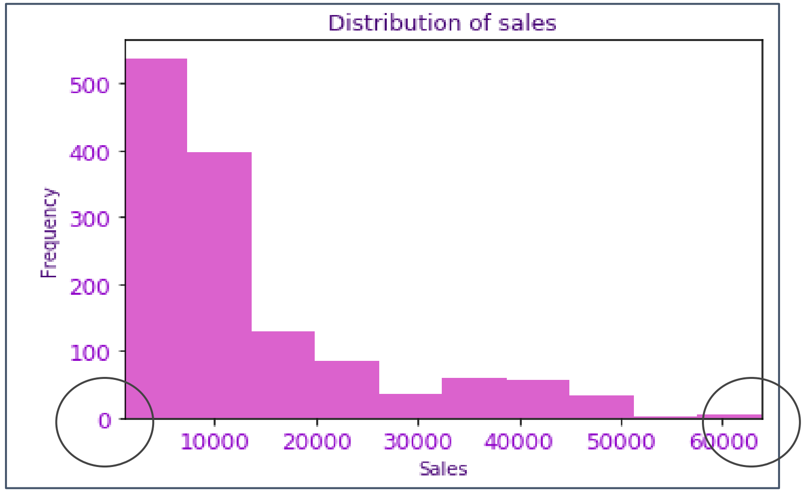
Histogram configurations:

- xlim
- ylim
- xticks
- yticks
- xlabel
- ylabel



```
# Plot the histogram
#x represents the variables to plot the histogram
plt.hist(x=df_insurance['claim'], color='orchid')

plt.title('Distribution of sales', color = 'indigo') # Set title
plt.xlim(df_insurance['claim'].min(), df_insurance['claim'].max()) # Set x & y limit
plt.xticks(fontsize=12, color='darkviolet') # Set the font size & color for x ticks
plt.yticks(fontsize=12, color='darkviolet') # Set the font size & color for y ticks
plt.xlabel('Sales', color='indigo')         # Set the text & color for x axis
plt.ylabel('Frequency', color='indigo')     # Set the text & color for y axis

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

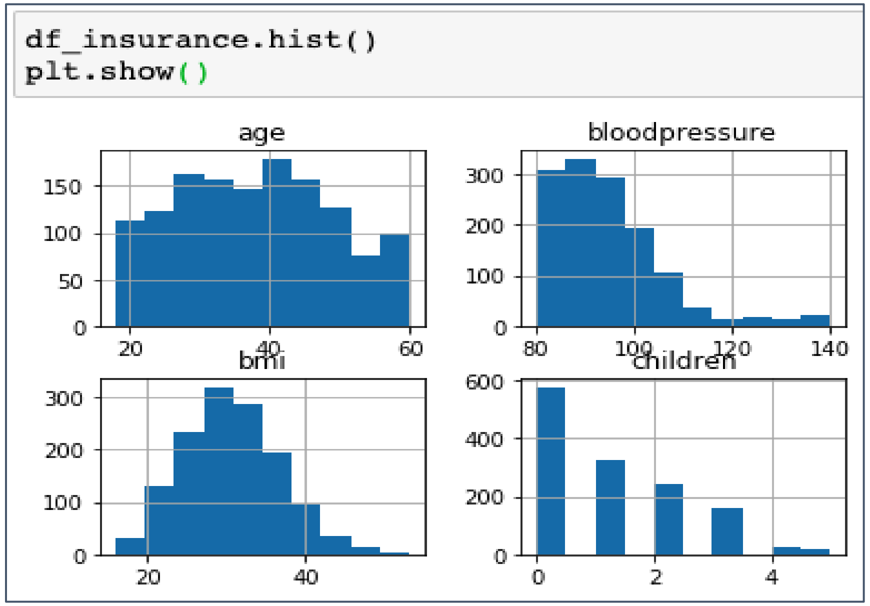




MULTIPLE LINE PLOTS

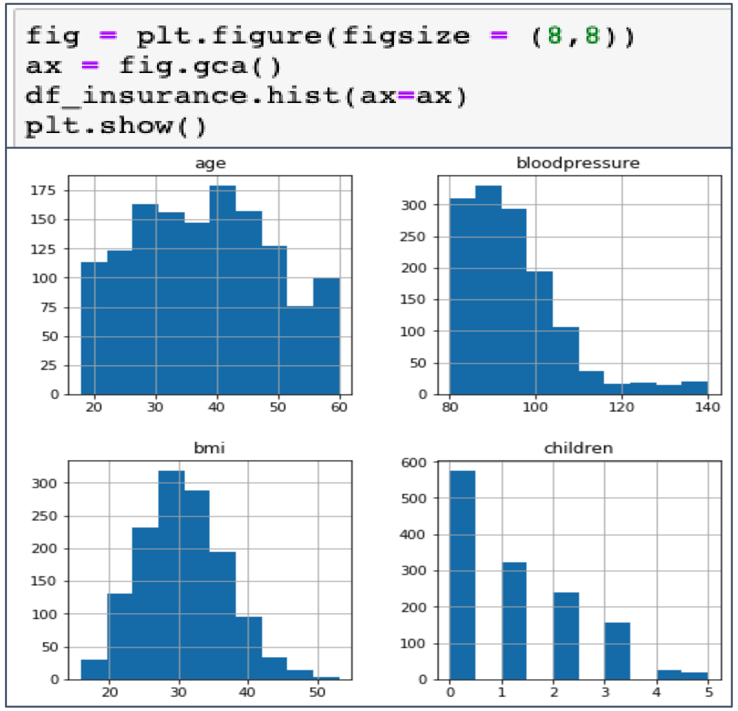
Plotting multiple histograms

Simple method



```
df_insurance.hist()
plt.show()
```

Using GCA (Get Current Axes)



```
fig = plt.figure(figsize = (8,8))
ax = fig.gca()
df_insurance.hist(ax=ax)
plt.show()
```

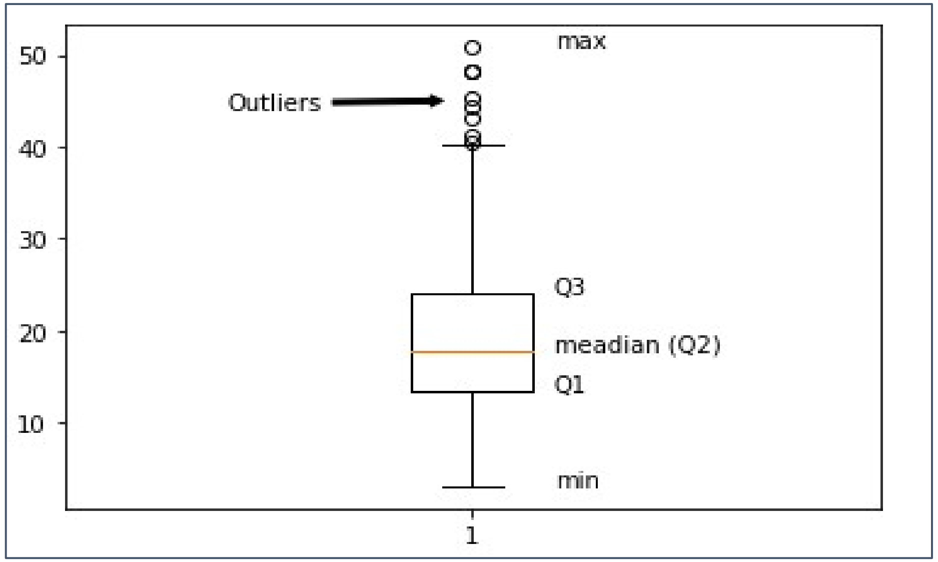
Boxplot

BOXPLOT

It is used to visualize the distribution of the numeric variable

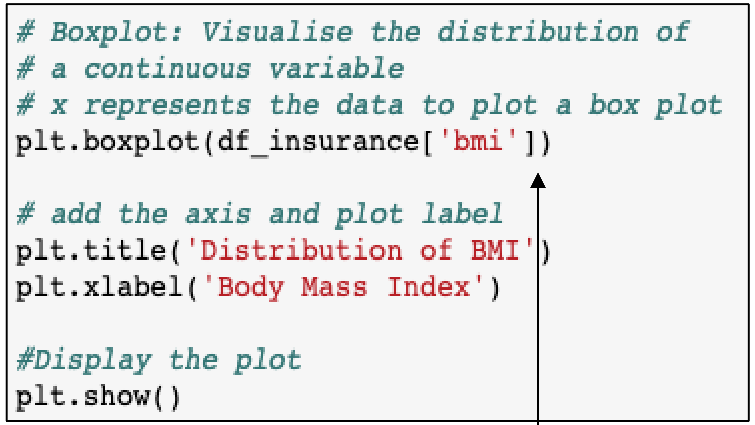
Represents the five number summary of the variable which includes the minimum, first quartile (Q1), second quartile (median), third quartile (Q3) and maximum of the variable

Used to detect the outliers (extreme values) in the data

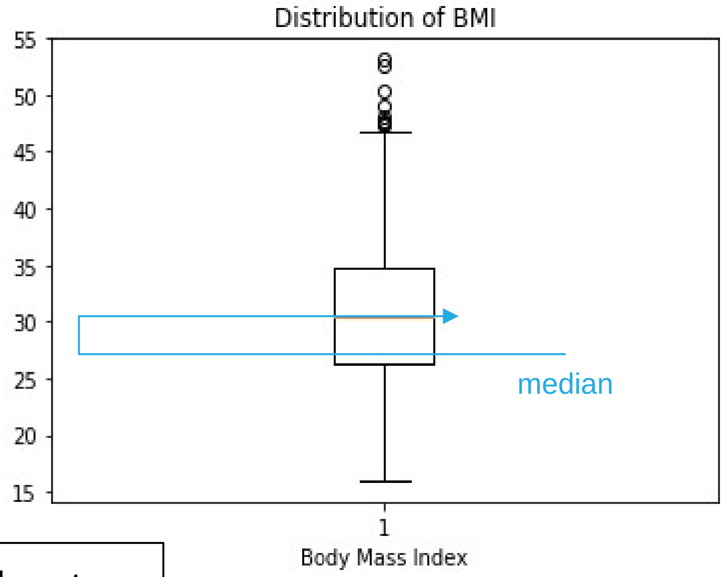


BOXPLOT

Check the distribution of the variable ‘bmi’



```
# Boxplot: Visualise the distribution of
# a continuous variable
# x represents the data to plot a box plot
plt.boxplot(df_insurance['bmi'])

# add the axis and plot label
plt.title('Distribution of BMI')
plt.xlabel('Body Mass Index')

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



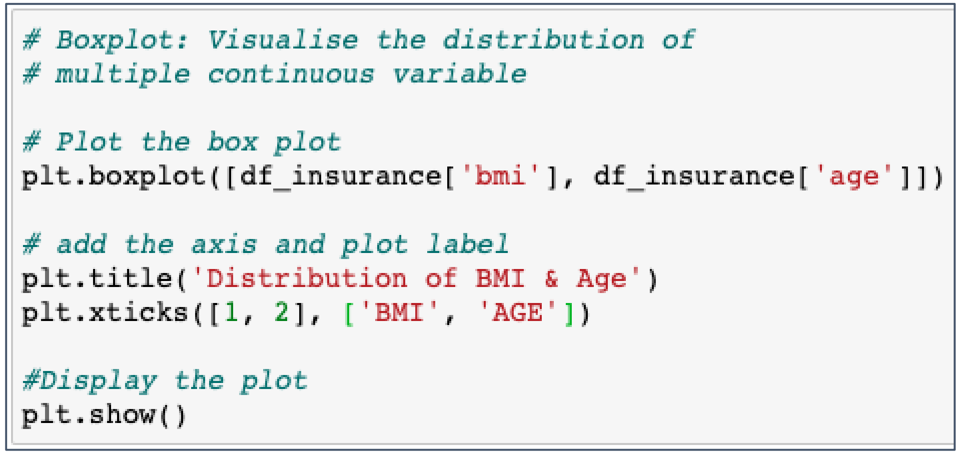
median

Use showfliers=False to view a boxplot without outliers

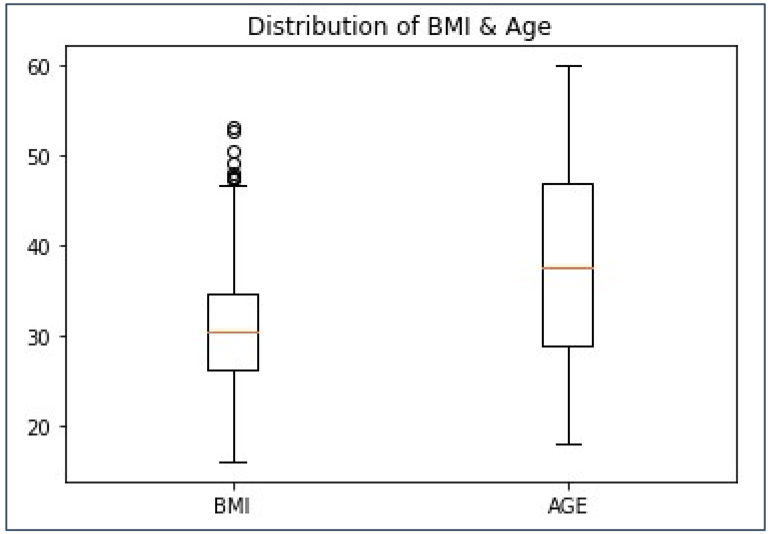
Use vert=False to view a horizontal boxplot

BOXPLOT

Boxplots for multiple numeric variables



```
# Boxplot: Visualise the distribution of
# multiple continuous variable

# Plot the box plot
plt.boxplot([df_insurance['bmi'], df_insurance['age']])

# add the axis and plot label
plt.title('Distribution of BMI & Age')
plt.xticks([1, 2], ['BMI', 'AGE'])

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

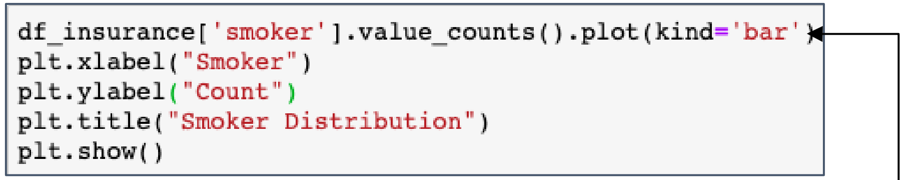


BAR PLOT

- 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

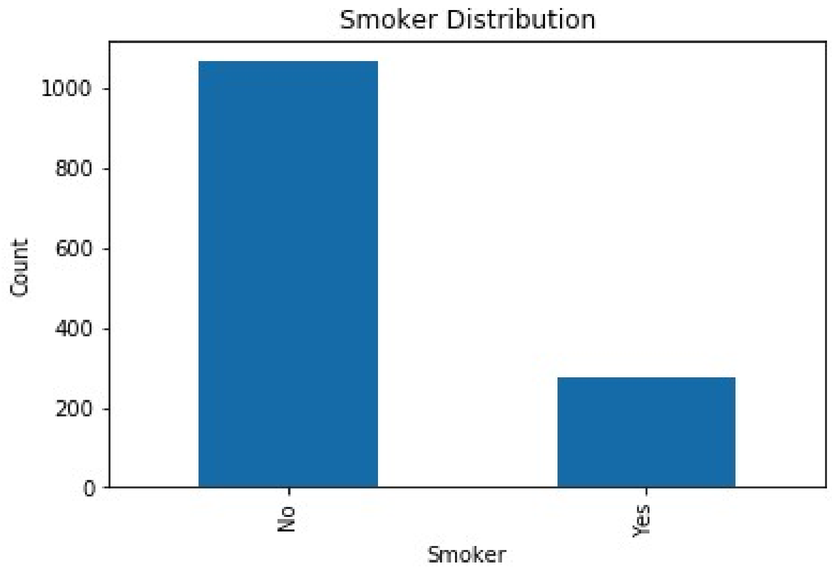
BAR PLOT

The bar plot displays the count of claims by smoker



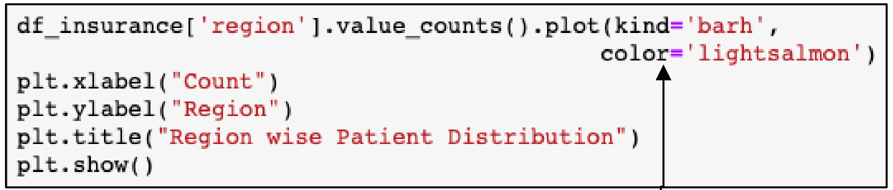
```
df_insurance['smoker'].value_counts().plot(kind='bar')
plt.xlabel("Smoker")
plt.ylabel("Count")
plt.title("Smoker Distribution")
plt.show()
```

Returns a vertical bar plot



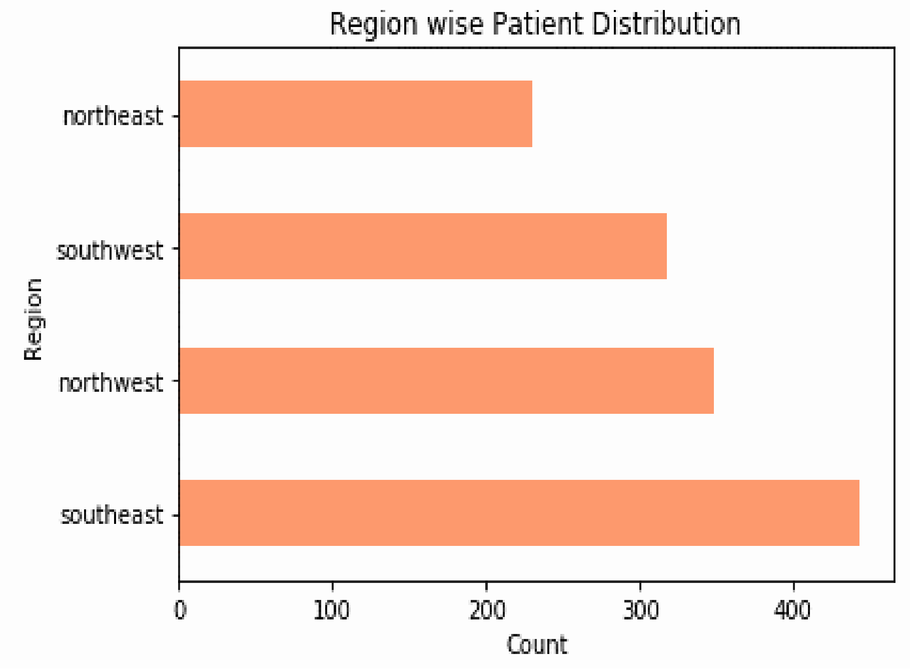
HORIZONTAL BAR PLOT

Plot the chart horizontally using the barh() method. This chart shows the count of patients from each region



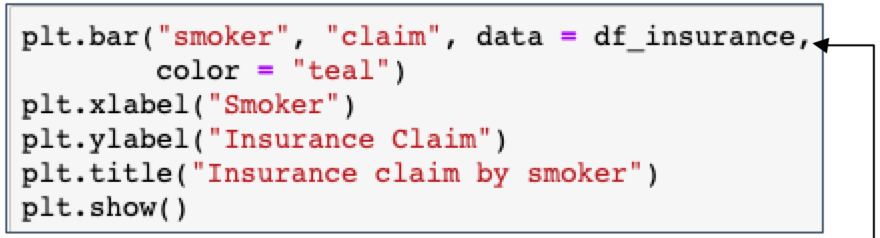
```
df_insurance['region'].value_counts().plot(kind='barh',
                                          color='lightsalmon')
plt.xlabel("Count")
plt.ylabel("Region")
plt.title("Region wise Patient Distribution")
plt.show()
```

Returns a horizontal bar plot



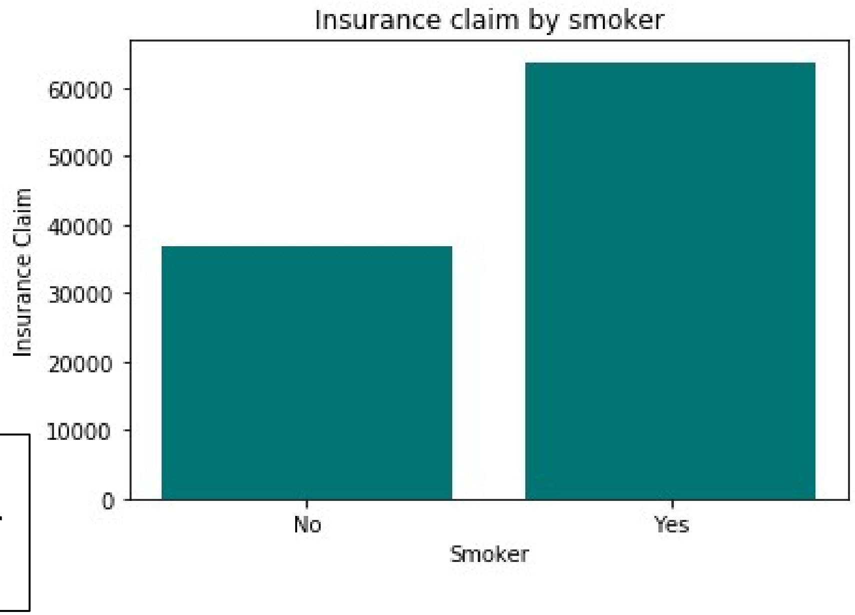
BAR PLOT

The bar plot displays the maximum insurance claim by smoker & non-smoker



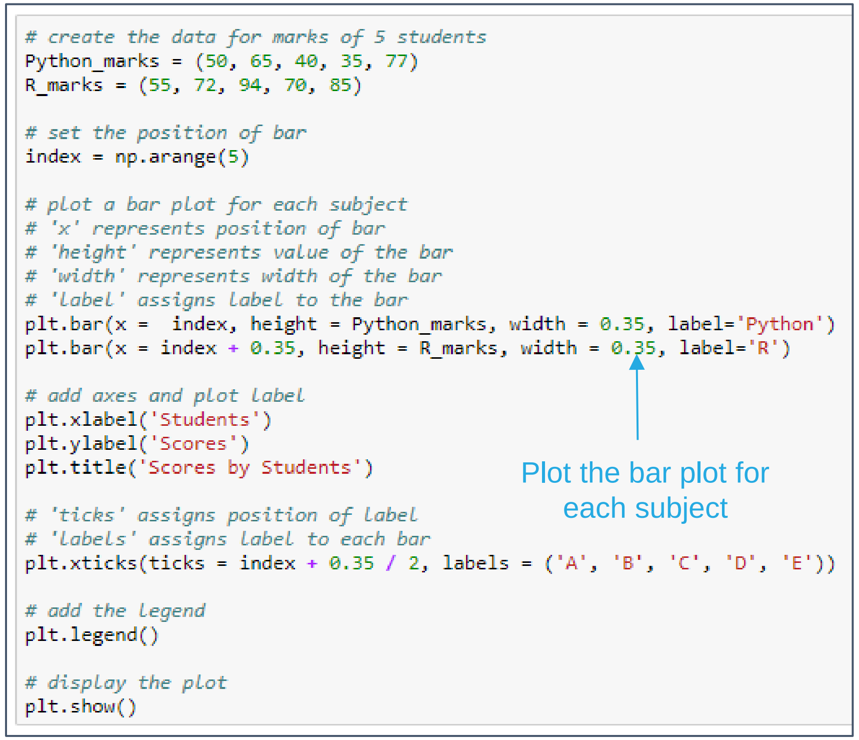
```
plt.bar("smoker", "claim", data = df_insurance,
        color = "teal")
plt.xlabel("Smoker")
plt.ylabel("Insurance Claim")
plt.title("Insurance claim by smoker")
plt.show()
```

Returns a vertical bar plot



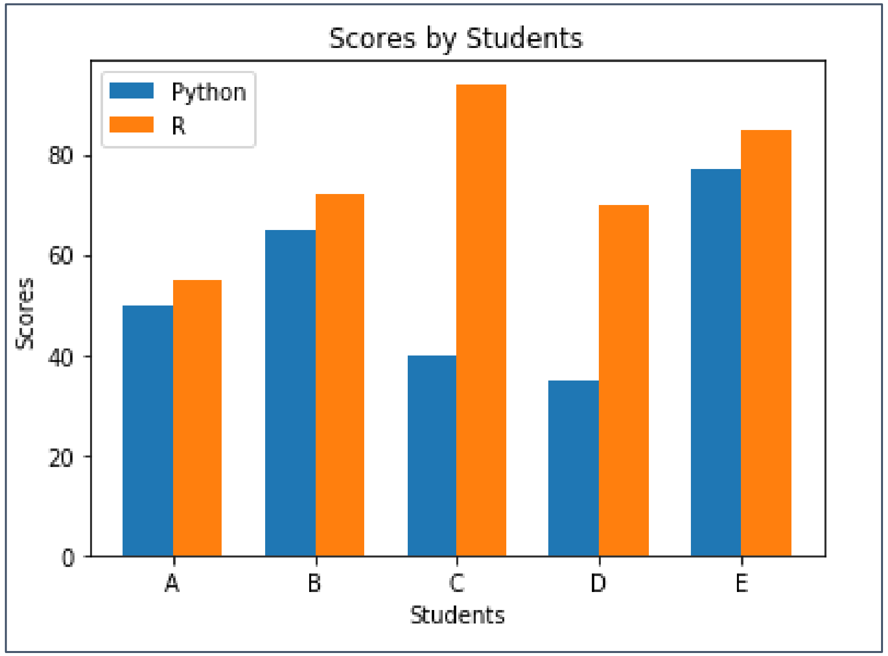
BAR PLOT

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()
```

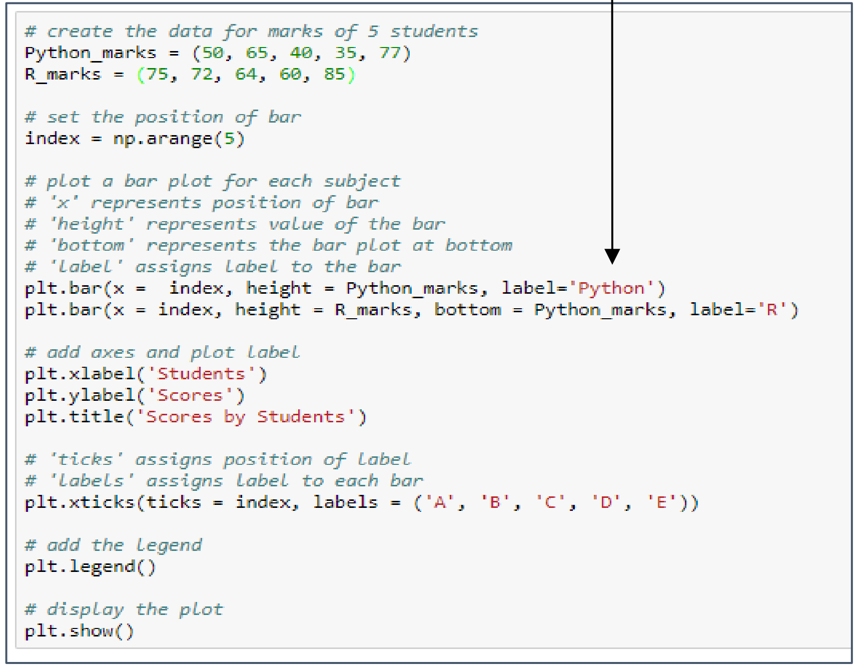
Plot the bar plot for each subject



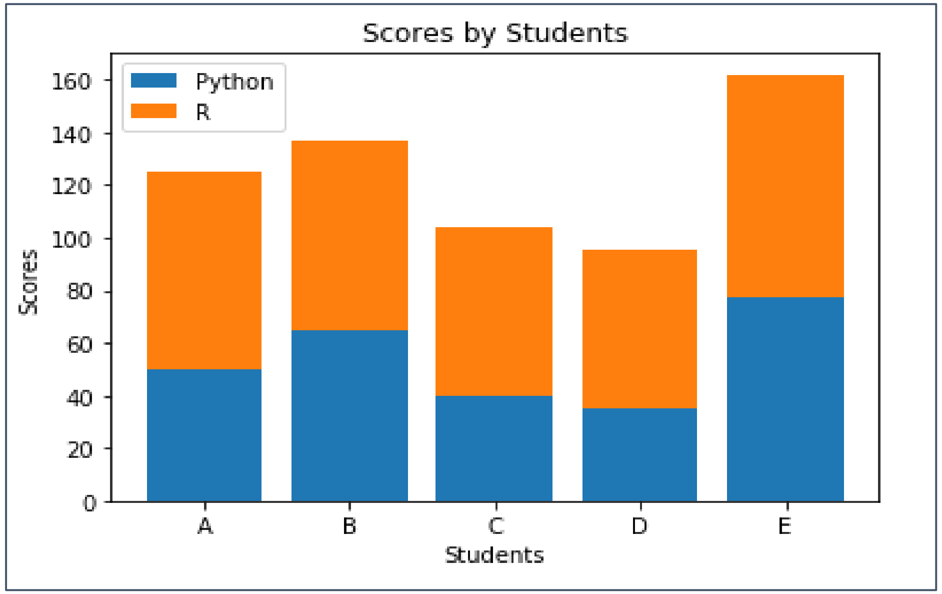
STACKED BAR PLOT

Compare the marks of the students in R and Python

Plot the ‘R_marks’ above the ‘Python_marks’



```
# 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()
```



Pie Chart

PIE PLOT

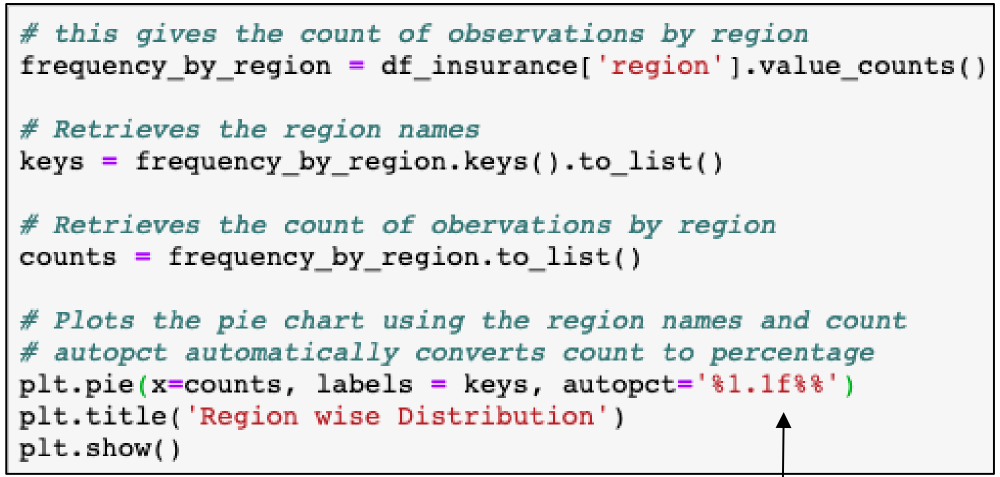
It is a circular graph divided into sections displaying the numeric proportion

It is used to display the univariate data

Each section of the pie plot represents a single

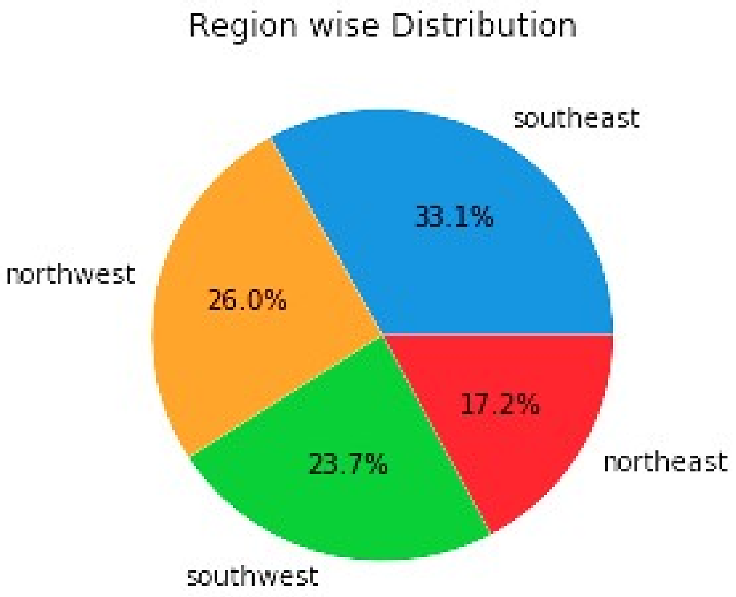
PIE PLOT

Plot a pie plot to study the region wise patient distribution



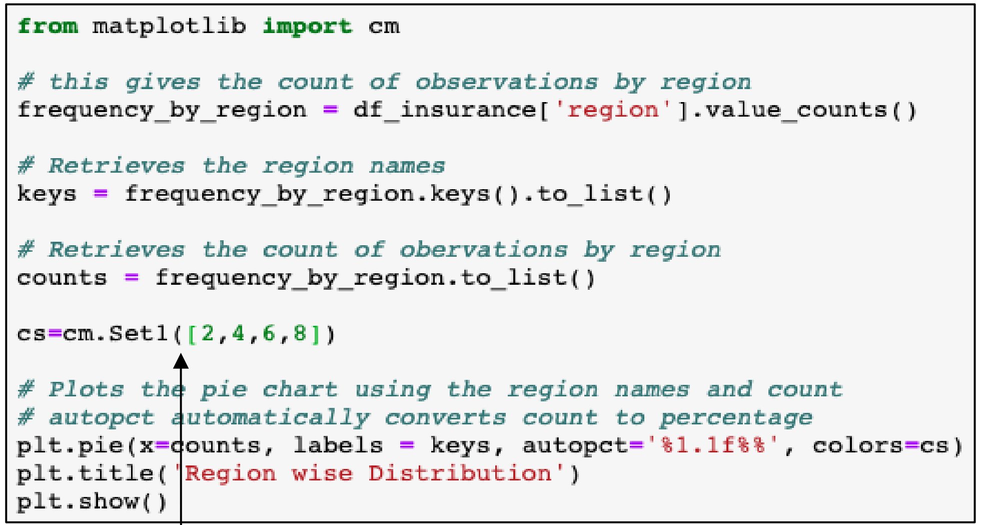
```
# this gives the count of observations by region
frequency_by_region = df_insurance['region'].value_counts()

# Retrieves the region names
keys = frequency_by_region.keys().to_list()

# Retrieves the count of obervations by region
counts = frequency_by_region.to_list()

# Plots the pie chart using the region names and count
# autopct automatically converts count to percentage
plt.pie(x=counts, labels = keys, autopct='%1.1f%%')
plt.title('Region wise Distribution')
plt.show()
```

Add the percentage with value to tenth place



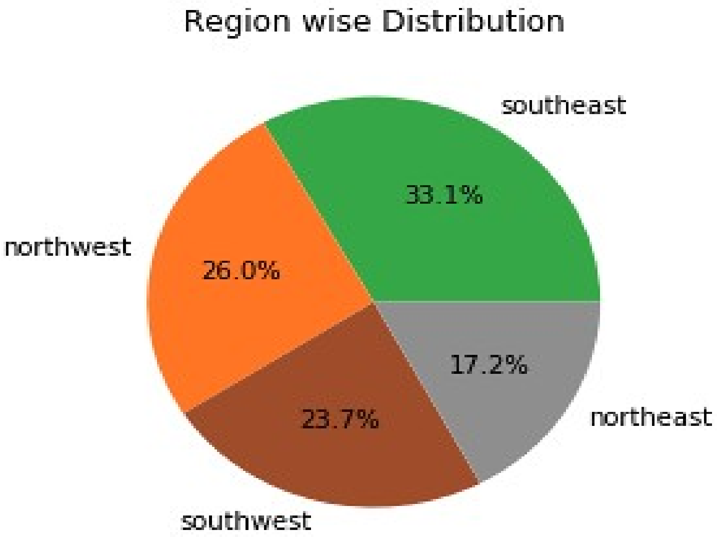
PIE PLOT

Adding colors to a pie plot using the cm module from matplotlib library



```
from matplotlib import cm

# this gives the count of observations by region
frequency_by_region = df_insurance['region'].value_counts()

# Retrieves the region names
keys = frequency_by_region.keys().to_list()

# Retrieves the count of obervations by region
counts = frequency_by_region.to_list()

cs=cm.Set1([2,4,6,8])

# Plots the pie chart using the region names and count
# autopct automatically converts count to percentage
plt.pie(x=counts, labels = keys, autopct='%1.1f%%', colors=cs)
plt.title('Region wise Distribution')
plt.show()
```

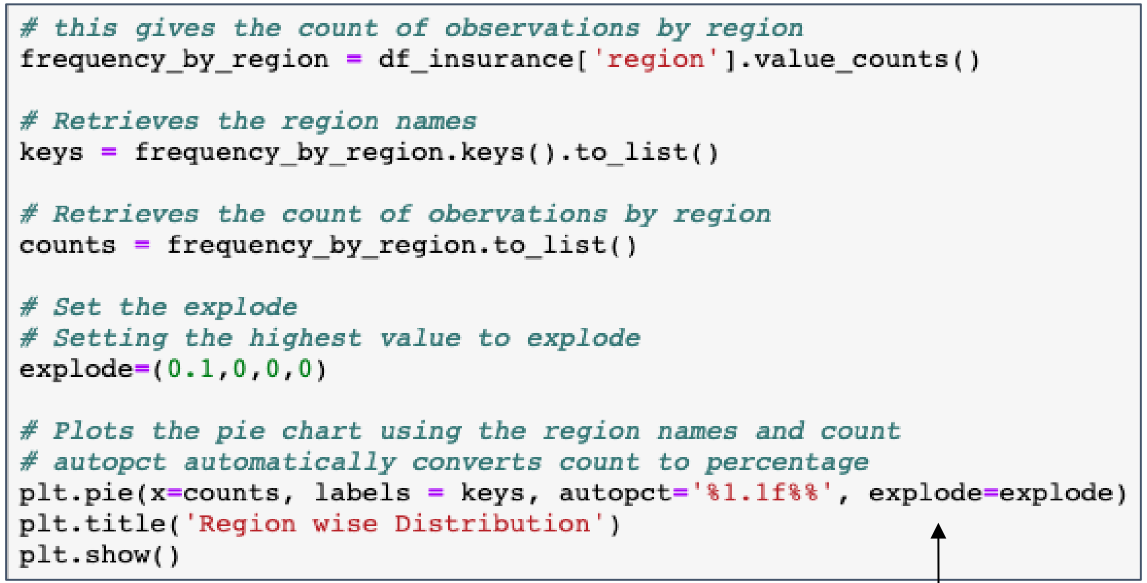
Add colors from Set1 palette.



More palettes are available at:
https://matplotlib.org/3.1.0/tutorials/colors/colormaps.html

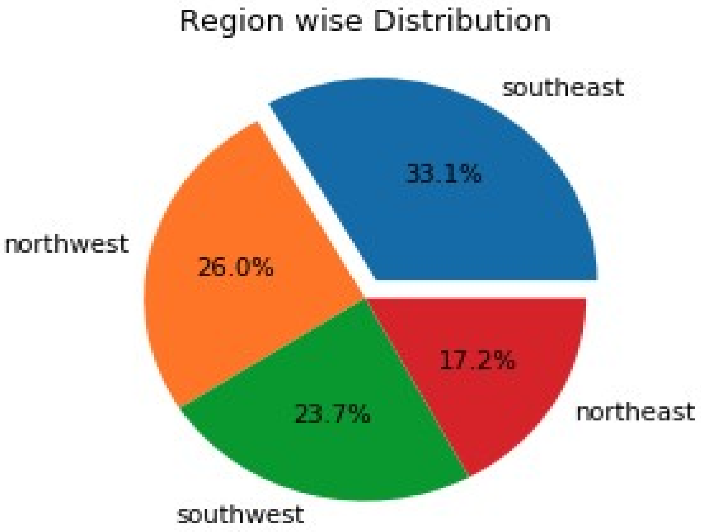
EXPLODED PIE PLOT

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



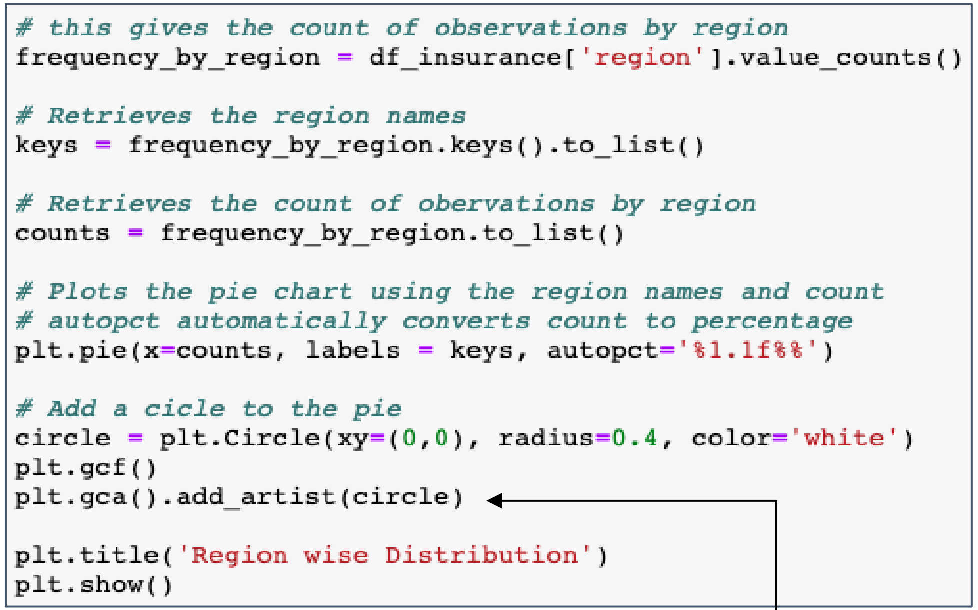
```
# this gives the count of observations by region
frequency_by_region = df_insurance['region'].value_counts()

# Retrieves the region names
keys = frequency_by_region.keys().to_list()

# Retrieves the count of obervations by region
counts = frequency_by_region.to_list()

# Set the explode
# Setting the highest value to explode
explode=(0.1,0,0,0)

# Plots the pie chart using the region names and count
# autopct automatically converts count to percentage
plt.pie(x=counts, labels = keys, autopct='%1.1f%%', explode=explode)
plt.title('Region wise Distribution')
plt.show()
```

Explode the region with highest count.

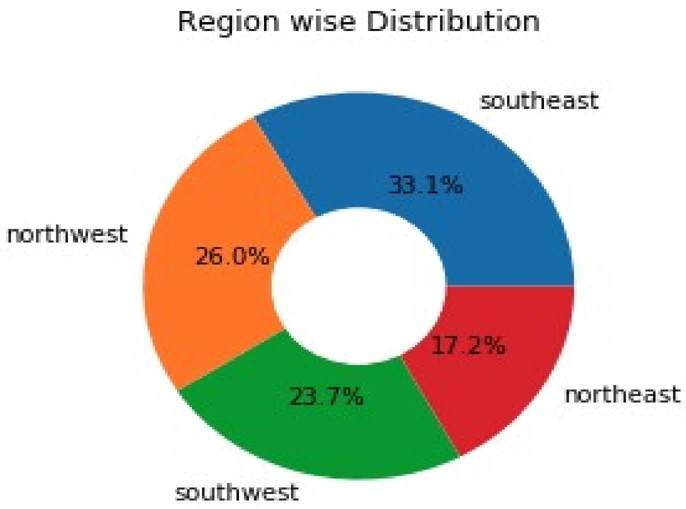


DONUT PIE PLOT

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



```
# this gives the count of observations by region
frequency_by_region = df_insurance['region'].value_counts()

# Retrieves the region names
keys = frequency_by_region.keys().to_list()

# Retrieves the count of obervations by region
counts = frequency_by_region.to_list()

# Plots the pie chart using the region names and count
# autopct automatically converts count to percentage
plt.pie(x=counts, labels = keys, autopct='%1.1f%%')

# Add a cicle to the pie
circle = plt.Circle(xy=(0,0), radius=0.4, color='white')
plt.gcf()
plt.gca().add_artist(circle)

plt.title('Region wise Distribution')
plt.show()
```



Add circle to current figure

Scatter Plot

SCATTER PLOT

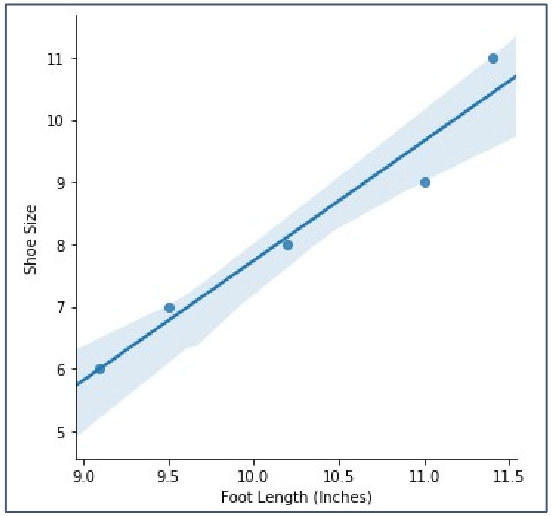
It is used to display the relationship between two numeric variables

Used to represent the extent of correlation between two variables

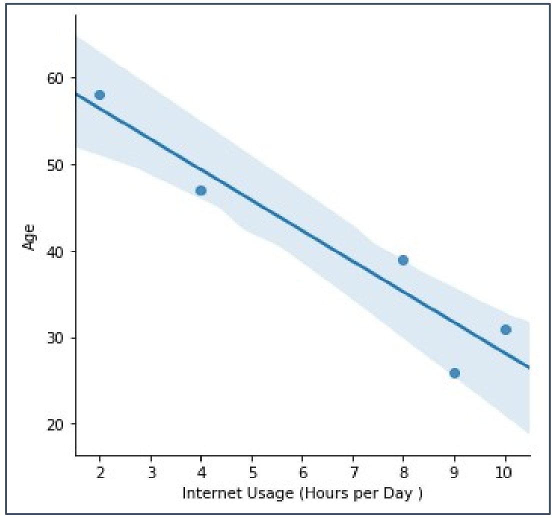
Used to detect the extreme points in the data

SCATTER PLOT

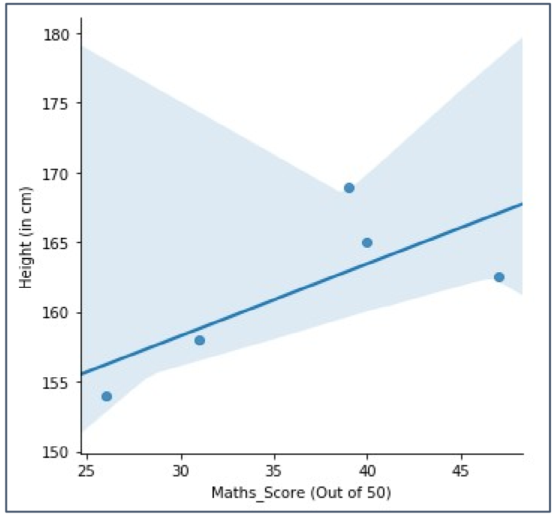
Scatter plots explaining the different types of correlation between the variables



Positive Correlation
(ρ = 0.97167252)



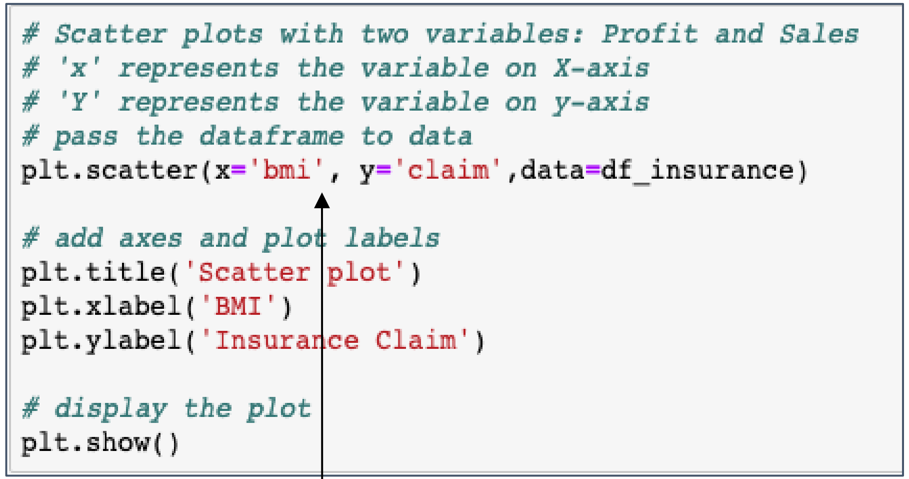
Negative Correlation
(ρ = -0.95056151)



No Correlation
(ρ = 0.09919779)

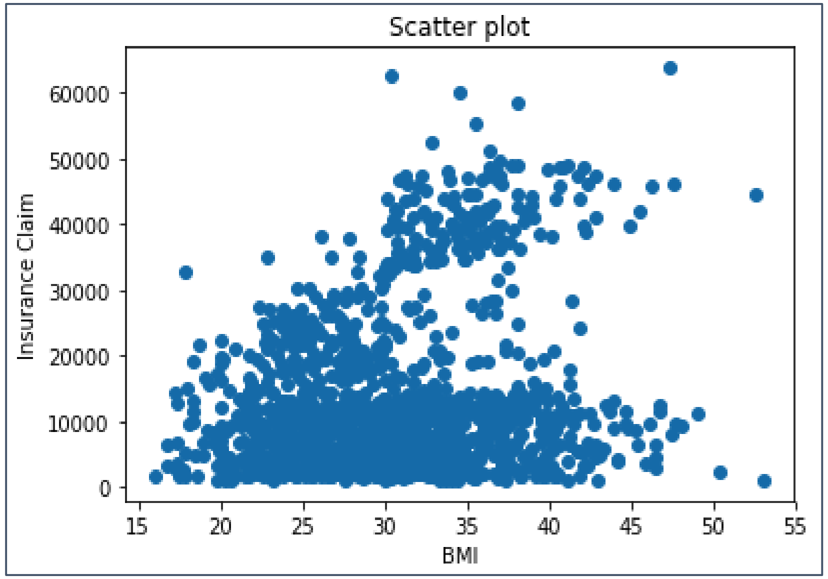
SCATTER PLOT

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



```
# Scatter plots with two variables: Profit and Sales
# 'x' represents the variable on X-axis
# 'Y' represents the variable on y-axis
# pass the dataframe to data
plt.scatter(x='bmi', y='claim',data=df_insurance)

# add axes and plot labels
plt.title('Scatter plot')
plt.xlabel('BMI')
plt.ylabel('Insurance Claim')

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

Set the variables on x and y axis


Scatter plot
Insurance Claim
BMI