DS CAT2 ANSWERS

Unit-III

1. Illustrate numpy arrays

NumPy arrays are a fundamental data structure in the NumPy library for Python. They are used to store and manipulate large, homogeneous, and multidimensional arrays of data. NumPy arrays are similar to Python lists, but they are more efficient in terms of memory usage and computation speed.

NumPy arrays are created using the **numpy.array()** function and can be one-dimensional, two-dimensional, or multi-dimensional. They can be accessed and manipulated using indexing and slicing, and they support a wide range of mathematical and logical operations.

One of the key benefits of NumPy arrays is their ability to perform vectorized operations. Vectorization refers to the ability to perform operations on entire arrays at once, rather than on individual elements. This makes NumPy arrays much faster than performing the same operations on Python lists or arrays using for loops.

NumPy arrays are widely used in scientific computing, data analysis, and machine learning. They are often used to store and manipulate large datasets, such as images or sensor data, and to perform numerical operations on those datasets.

Example:

The following example shows how to initialize a NumPy array from a list.

Python3

```
import numpy as np



li = [1, 2, 3, 4]

numpyArr = np.array(li)

print(numpyArr)
```

Output:

```
[1 2 3 4]
```

2. Describe computation on arrays with an example

Computation on NumPy arrays is generally done element-wise, which means that mathematical and logical operations are performed on each element of the array individually. This is a fundamental feature of NumPy that makes it possible to perform efficient, vectorized operations on large arrays.

Here's an example of how to perform a computation on a NumPy array:

```
import numpy as np

# create an array of values
a = np.array([1, 2, 3, 4, 5])

# compute the square of each element in the array
b = a**2

print(b)
# output: [ 1  4  9 16 25]

# compute the sin of each element in the array
c = np.sin(a)

print(c)
# output: [ 0.84147098  0.90929743  0.14112001 -0.7568025  -0.95892427]
```

In this example, we create a NumPy array `a` and then use the `**` operator to compute the square of each element in the array. We store the result in a new array `b`. Similarly, we use the `np.sin()` function to compute the sine of each element in the array `a`, and store the result in a new array `c`.

These computations are done element-wise, which means that the operations are performed on each element in the array individually, and the result is stored in a new array. This makes it possible to perform efficient, vectorized computations on large arrays, which is a fundamental feature of NumPy.

3. Explain data manipulation with Pandas

Pandas is a powerful library for data manipulation and analysis in Python. It provides a flexible and easy-to-use data structure called a DataFrame, which is similar to a table in a database. Pandas allows you to load data from a wide range of sources, including CSV files, Excel spreadsheets, SQL databases, and more. Once the data is loaded, you can use pandas to clean, transform, and manipulate the data in a variety of ways.

Here are some common data manipulation tasks that can be performed using pandas:

Loading data: Pandas provides functions to load data from a variety of sources, including CSV files, Excel spreadsheets, SQL databases, and more. For example, you can load a CSV file using the `pandas.read_csv()` function.

Cleaning data: Data is often messy and may contain missing values, duplicate entries, or other issues. Pandas provides functions to clean data, such as `dropna()` to remove rows with missing values, and `drop_duplicates()` to remove duplicate rows.

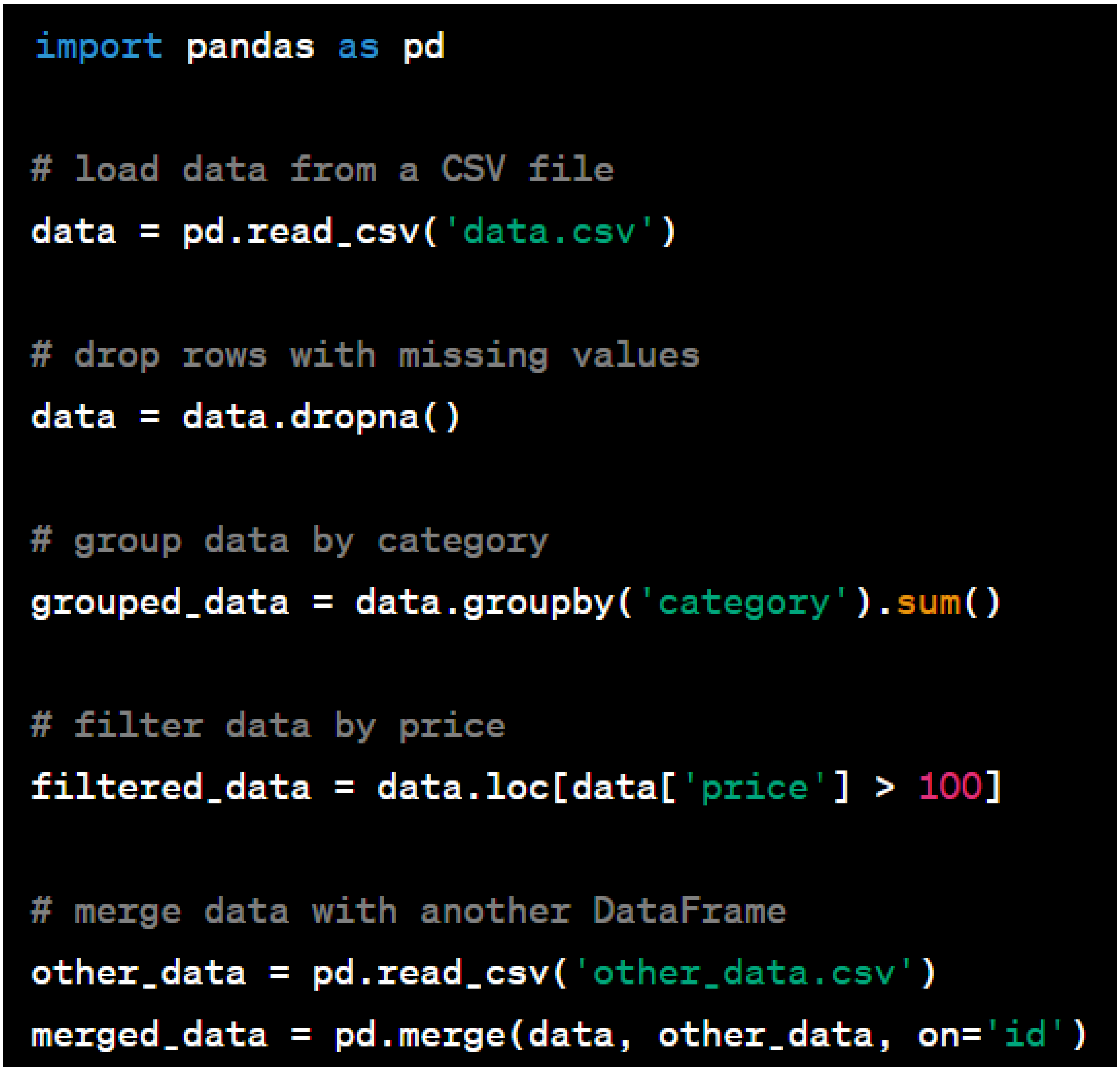
Transforming data: Pandas provides powerful tools for transforming data, such as `groupby()` to group data by a particular column, `pivot()` to pivot data based on a particular column, and `apply()` to apply a function to each row or column.

Filtering data: Pandas provides functions to filter data based on particular conditions, such as `loc[]` to filter rows based on a particular condition, and `isin()` to filter rows based on a set of values.

Merging data: Pandas provides functions to merge data from multiple sources, such as `merge()` to merge data based on a particular column.

Here's an example of how to use pandas to load and manipulate data:

In this example, we load data from a CSV file using pd.read_csv(), drop rows with missing values using dropna(), group the data by category using groupby(), filter the data by price using loc[], and merge the data with another DataFrame using merge(). These are just a few examples of the many powerful data manipulation tools available in pandas.



```python
import pandas as pd

# load data from a CSV file
data = pd.read_csv('data.csv')

# drop rows with missing values
data = data.dropna()

# group data by category
grouped_data = data.groupby('category').sum()

# filter data by price
filtered_data = data.loc[data['price'] > 100]

# merge data with another DataFrame
other_data = pd.read_csv('other_data.csv')
merged_data = pd.merge(data, other_data, on='id')
```

4. Summarize hierarchial indexing

Hierarchical indexing is a method of creating structured group relationships in the dataset. Data frames can have hierarchical indexes. To show this, let me create a dataset.

```
In [10]:
df=pd.DataFrame(
    np.arange(12).reshape(4,3),
    index=[["a","a","b","b"],
           [1,2,1,2]],
    columns=[["num","num","ver"],
             ["math","stat","geo"]])
df
```

```
Out[10]:
        num        ver
        math stat  geo
a  1     0     1     2
   2     3     4     5
b  1     6     7     8
   2     9    10    11
```

Notice that in this dataset, both row and column have hierarchical indexes. You can name hierarchical levels. Let’s show this.

```
In [11]:
df.index.names=["class","exam"]
df.columns.names=["field","lesson"]
df
```

```
Out[11]:
            field   num         ver
           lesson   math  stat  geo
class  exam
    a     1          0     1     2
          2          3     4     5
    b     1          6     7     8
          2          9    10    11
```

OR

Hierarchical indexing, also known as multi-indexing, is a feature of pandas that allows you to work with data that has multiple dimensions or levels. It provides a way to represent complex data sets in a structured and organized way, and makes it easier to manipulate and analyze the data.

In pandas, hierarchical indexing is achieved by creating a DataFrame with multiple index columns. Each index column represents a different level of the hierarchy, and can be used to select and manipulate subsets of the data.

Here's an example of how to create a DataFrame with hierarchical indexing in pandas:

```python
import pandas as pd

# create a DataFrame with hierarchical indexing
data = pd.DataFrame({
    'group': ['A', 'A', 'B', 'B'],
    'subgroup': ['X', 'Y', 'X', 'Y'],
    'value': [1, 2, 3, 4]
})

# set the hierarchical index
data = data.set_index(['group', 'subgroup'])

# display the DataFrame
print(data)
```

In this example, we create a DataFrame `data` with three columns: `group`, `subgroup`, and `value`. We then set the hierarchical index to be the `group` and `subgroup` columns, using the `set_index()` function. This creates a DataFrame with two levels of indexing: the `group` level and the `subgroup` level.

We can then use the hierarchical index to select and manipulate subsets of the data. For example, we can select all rows in the `group` "A" and `subgroup` "X" using the following code:

```python
# select rows with group "A" and subgroup "X"
subset = data.loc[('A', 'X')]

# display the subset
print(subset)
```

This will display a DataFrame with the rows that match the specified index values.

Hierarchical indexing can be especially useful for working with data that has multiple dimensions or levels, such as time series data or financial data with multiple layers of grouping. It provides a way to organize and structure the data in a way that makes it easier to manipulate and analyze.

5. Write a note on combining datasets.

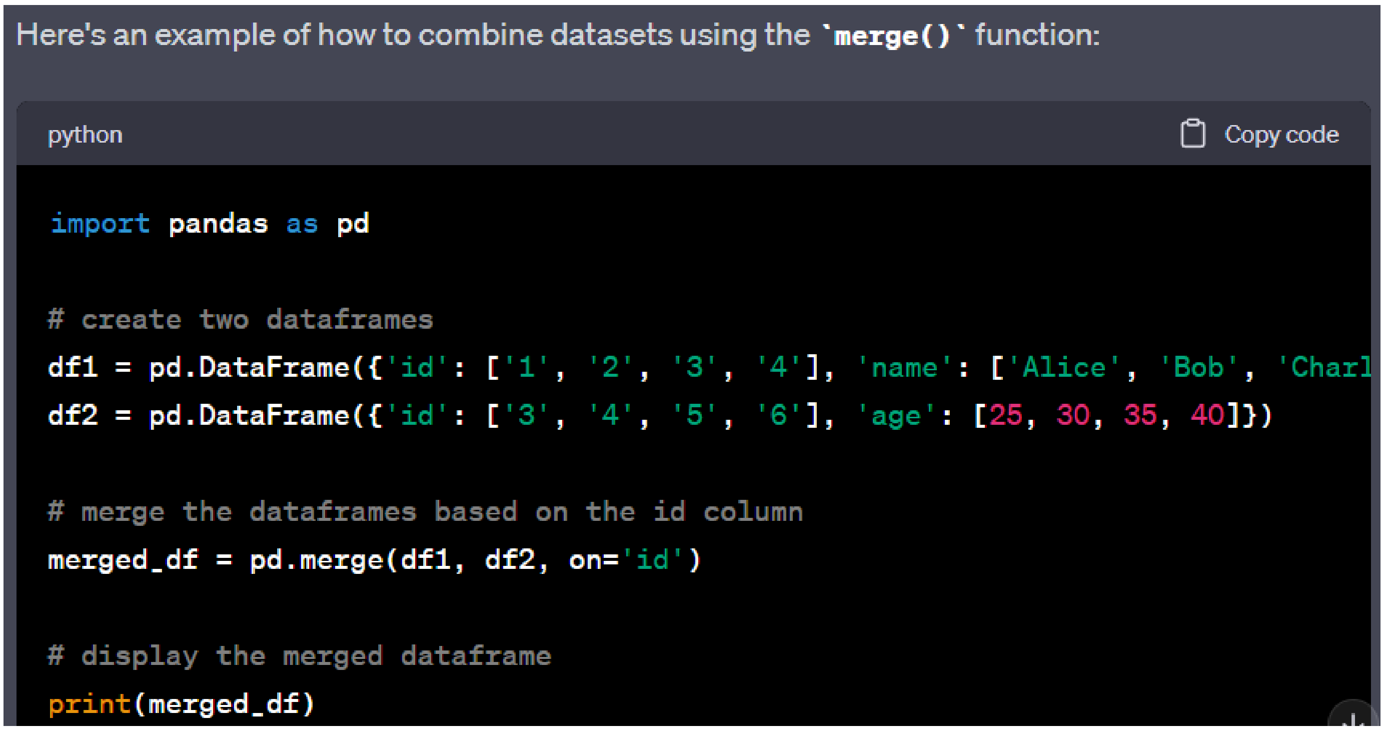
Combining datasets is a common task in data science and involves merging two or more datasets based on common columns or indices. Pandas provides a variety of functions for combining datasets, including `merge()`, `concat()`, and `join()`, each of which can be used for different types of data.

Here are some common scenarios where combining datasets is useful:

Combining data from different sources: You may have data from different sources that needs to be combined into a single dataset. For example, you might have data from different departments within a company that needs to be combined for analysis.

Adding new columns or rows to a dataset: You may need to add new columns or rows to a dataset based on data from another source. For example, you might have a dataset of sales data that needs to be augmented with customer demographic data.

Splitting a dataset into multiple parts: You may need to split a dataset into multiple parts for analysis or visualization. For example, you might have a large dataset that needs to be split into smaller subsets based on a certain criteria.



Here's an example of how to combine datasets using the `merge()` function:

```python
import pandas as pd

# create two dataframes
df1 = pd.DataFrame({'id': ['1', '2', '3', '4'], 'name': ['Alice', 'Bob', 'Charl
df2 = pd.DataFrame({'id': ['3', '4', '5', '6'], 'age': [25, 30, 35, 40]})

# merge the dataframes based on the id column
merged_df = pd.merge(df1, df2, on='id')

# display the merged dataframe
print(merged_df)
```

In this example, we create two dataframes `df1` and `df2` with different columns. We then merge the dataframes based on the `id` column using the `merge()` function. This creates a new dataframe `merged_df` that contains all columns from both dataframes, but only rows where the `id` column matches.

The `merge()` function provides many options for specifying the type of join, the columns to use for the join, and how to handle missing values.

In addition to `merge()`, pandas also provides the `concat()` and `join()` functions for combining datasets. `concat()` is used to concatenate two or more dataframes along a particular axis, while `join()` is used to join two dataframes based on their indices.

Unit-IV

1. Explain visualization with matplotlib

Data Visualization is the process of presenting data in the form of graphs or charts. It helps to understand large and complex amounts of data very easily. It allows the decision-makers to make decisions very efficiently and also allows them in identifying new trends and patterns very easily. It is also used in high-level data analysis for Machine Learning and Exploratory Data Analysis (EDA). Data visualization can be done with various tools like Tableau, Power BI, Python.

In this article, we will discuss how to visualize data with the help of the Matplotlib library of Python.

Matplotlib

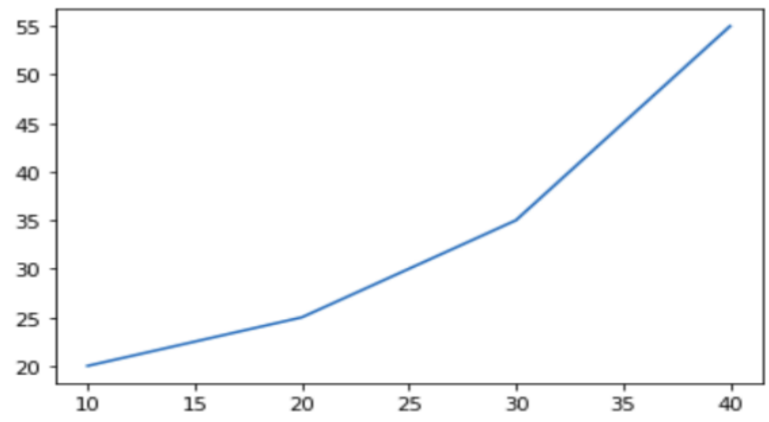
Matplotlib is a low-level library of Python which is used for data visualization. It is easy to use and emulates MATLAB like graphs and visualization. This library is built on the top of NumPy arrays and consist of several plots like line chart, bar chart, histogram, etc. It provides a lot of flexibility but at the cost of writing more code.

Pyplot

Pyplot is a Matplotlib module that provides a MATLAB-like interface. Matplotlib is designed to be as usable as MATLAB, with the ability to use Python and the advantage of being free and open-source. Each pyplot function makes some change to a figure: e.g., creates a figure, creates a plotting area in a figure, plots some lines in a plotting area, decorates the plot with labels, etc. The various plots we can utilize using Pyplot are Line Plot, Histogram, Scatter, 3D Plot, Image, Contour, and Polar.

After knowing a brief about Matplotlib and pyplot let's see how to create a simple plot.

```
import matplotlib.pyplot as plt


# initializing the data

x = [10, 20, 30, 40]

y = [20, 25, 35, 55]

# plotting the data

plt.plot(x, y)

plt.show()
```



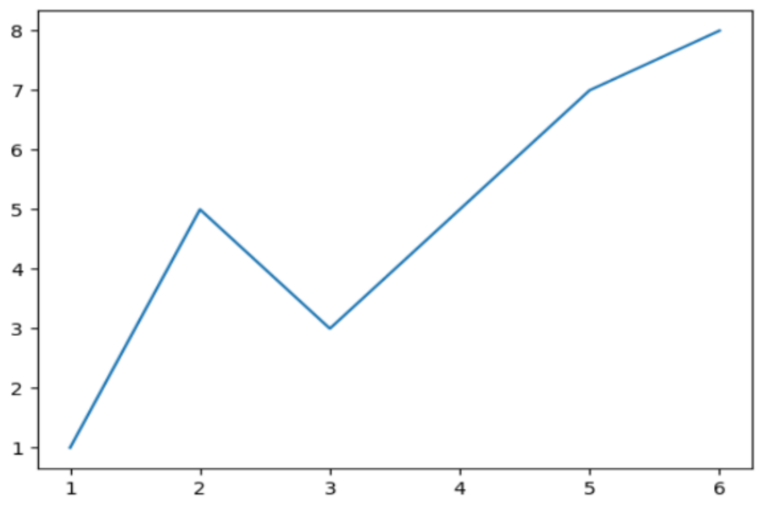
Plot a Line Plot in Matplotlib

To plot a line plot in Matplotlib, you use the generic `plot()` function from the PyPlot instance. There's no specific `lineplot()` function - the generic one automatically plots using lines or markers.

Let's make our own small dataset to work with:

```
import matplotlib.pyplot as plt

x = [1, 2, 3, 4, 5, 6]
y = [1, 5, 3, 5, 7, 8]

plt.plot(x, y)
plt.show()
```

This results in a simple line plot:



matplotlib.pyplot.scatter() in Python

Matplotlib is a comprehensive library for creating static, animated, and interactive visualizations in Python. It is used for plotting various plots in Python like scatter plot, bar charts, pie charts, line plots, histograms, 3-D plots and many more.

matplotlib.pyplot.scatter()

Scatter plots are used to observe relationship between variables and uses dots to represent the relationship between them. The **scatter()** method in the matplotlib library is used to draw a scatter plot. Scatter plots are widely used to represent relation among variables and how change in one affects the other.

Syntax

The syntax for scatter() method is given below:

matplotlib.pyplot.scatter(x_axis_data, y_axis_data, s=None, c=None, marker=None, cmap=None, vmin=None, vmax=None, alpha=None, linewidths=None, edgecolors=None)

The scatter() method takes in the following parameters:

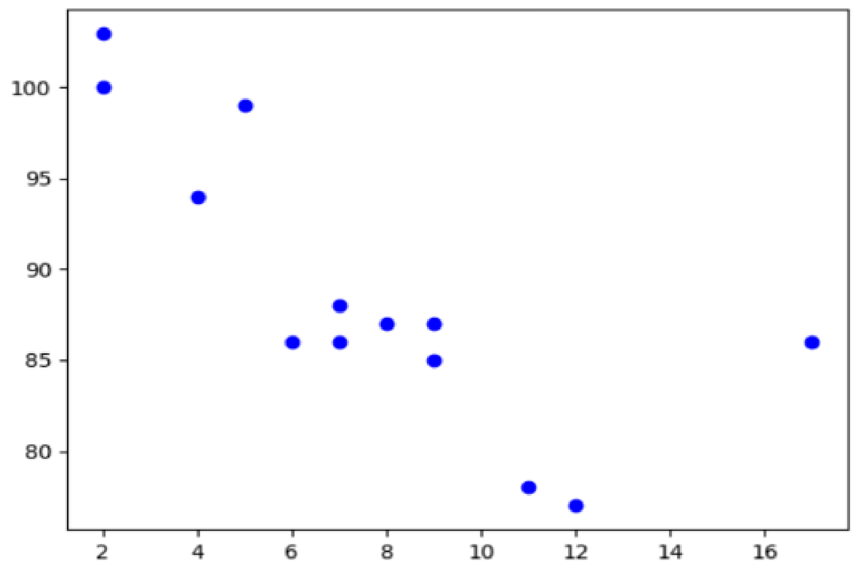
- **x_axis_data-** An array containing x-axis data
- **y_axis_data-** An array containing y-axis data
- **s-** marker size (can be scalar or array of size equal to size of x or y)
- **c-** color of sequence of colors for markers
- marker- marker style
- **cmap-** cmap name

- **linewidths-** width of marker border
- **edgecolor-** marker border color
- **alpha-** blending value, between 0 (transparent) and 1 (opaque)

Except x_axis_data and y_axis_data all other parameters are optional and their default value is None. Below are the scatter plot examples with various parameters.

Example 1: This is the most basic example of a scatter plot.

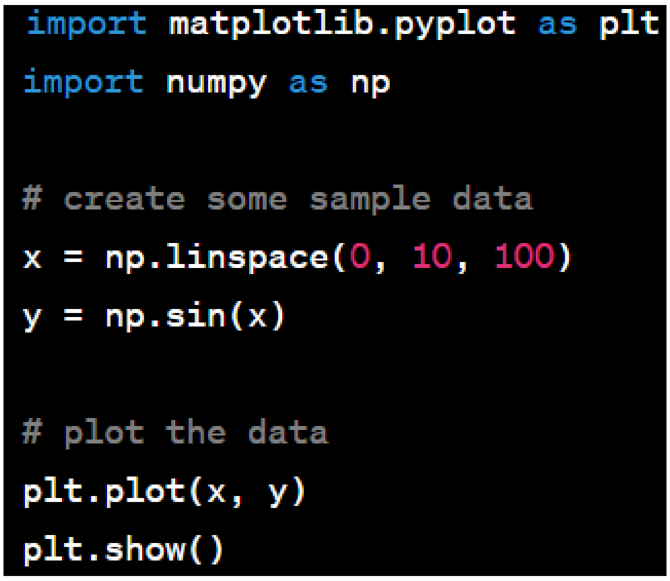
```
import matplotlib.pyplot as plt

x =[5, 7, 8, 7, 2, 17, 2, 9,

    4, 11, 12, 9, 6]

y =[99, 86, 87, 88, 100, 86,

    103, 87, 94, 78, 77, 85, 86]

plt.scatter(x, y, c ="blue")

# To show the plot

plt.show()
```



OR

Matplotlib is a popular data visualization library in Python that allows you to create a wide range of static, animated, and interactive visualizations. It provides a variety of functions for creating plots, charts, histograms, scatterplots, and more.

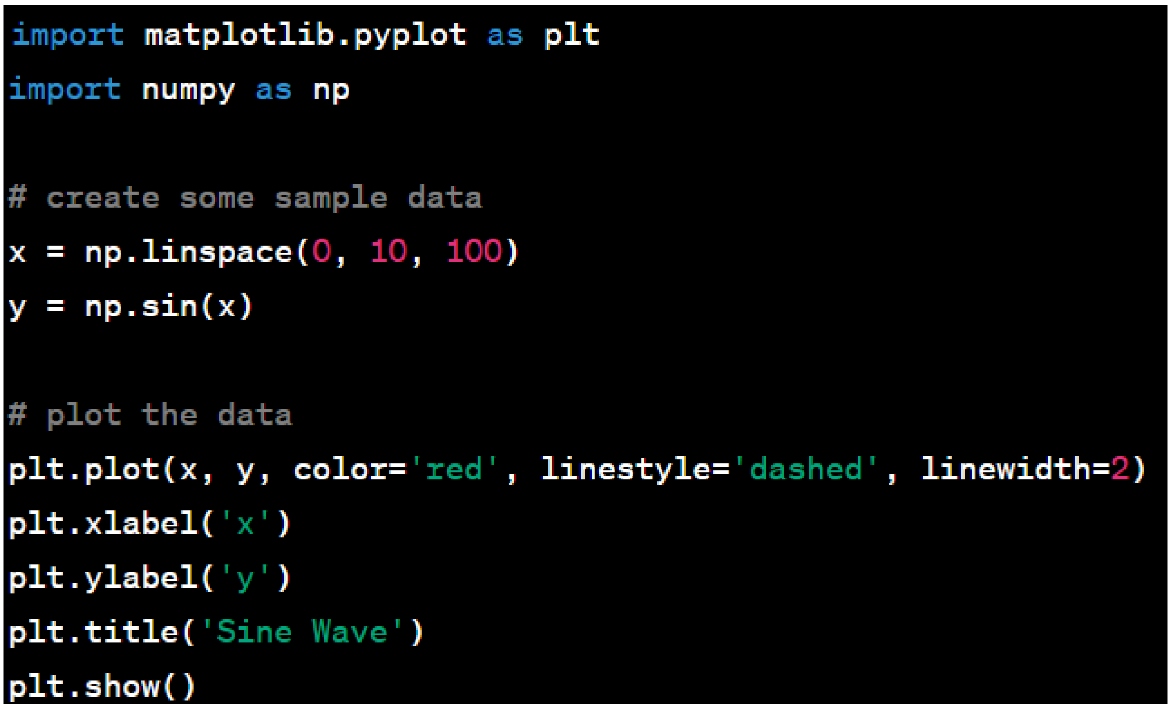
To create a basic plot with matplotlib, you first need to import the library and specify the data you want to plot:



```
import matplotlib.pyplot as plt
import numpy as np

# create some sample data
x = np.linspace(0, 10, 100)
y = np.sin(x)

# plot the data
plt.plot(x, y)
plt.show()
```

In this example, we create an array `x` with 100 evenly spaced points between 0 and 10, and an array `y` with the sine of each point in `x`. We then plot the data using the `plot()` function and display the plot using the `show()` function.

Matplotlib provides many options for customizing the plot, such as changing the color, line style, axis labels, title, and more. Here's an example of how to customize the plot:



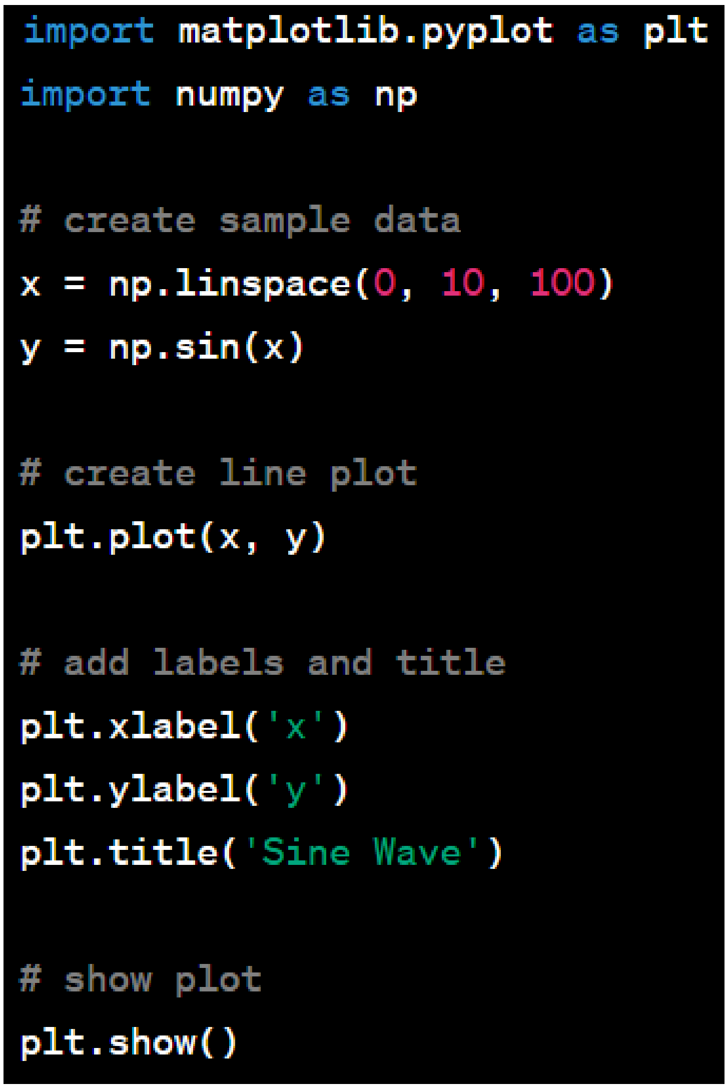
```python
import matplotlib.pyplot as plt
import numpy as np

# create some sample data
x = np.linspace(0, 10, 100)
y = np.sin(x)

# plot the data
plt.plot(x, y, color='red', linestyle='dashed', linewidth=2)
plt.xlabel('x')
plt.ylabel('y')
plt.title('Sine Wave')
plt.show()
```

In this example, we add options to the `plot()` function to change the color, line style, and linewidth of the plot. We also add labels to the x-axis, y-axis, and title using the `xlabel()`, `ylabel()`, and `title()` functions.

Matplotlib also provides many other types of plots, such as scatterplots, histograms, bar charts, and more. You can find more examples and documentation on the Matplotlib website.

Overall, Matplotlib is a powerful and flexible library for creating visualizations in Python. It is widely used in data science and provides many options for customizing and creating high-quality plots.

2. Illustrate line plots



```python
import matplotlib.pyplot as plt
import numpy as np

# create sample data
x = np.linspace(0, 10, 100)
y = np.sin(x)

# create line plot
plt.plot(x, y)

# add labels and title
plt.xlabel('x')
plt.ylabel('y')
plt.title('Sine Wave')

# show plot
plt.show()
```

In this example, we first create some sample data using NumPy to generate 100 evenly spaced points between 0 and 10, and the sine of each point in **`x`**.

We then create a line plot using the **`plot()`** function from Matplotlib, passing in **`x`** and **`y`** as the x and y data, respectively. Matplotlib automatically draws a line connecting the points in the **`x`** and **`y`** arrays.

Next, we add labels to the x and y axes using the **`xlabel()`** and **`ylabel()`** functions, and a title to the plot using the **`title()`** function.

Finally, we display the plot using the **`show()`** function.

Line plots are useful for visualizing trends in data over time or across different values of a variable. They can also be used to compare multiple datasets by overlaying multiple lines on the same plot.

OR

Line plot:

Lineplot Is the most popular plot to draw a relationship between x and y with the possibility of several semantic groupings.

Syntax : *sns.lineplot(x=None, y=None)*

Parameters:

x, y: *Input data variables; must be numeric. Can pass data directly or reference columns in data.*

Let's visualize the data with a line plot and pandas:

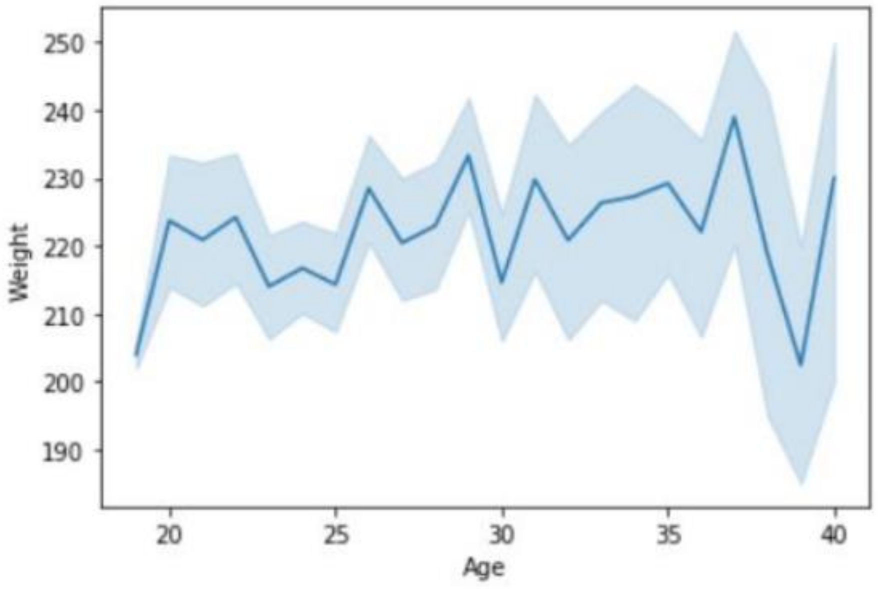
Example 1:

Python3

```
# import module

import seaborn as sns

import pandas



# loading csv

data = pandas.read_csv("nba.csv")
```

```
# plotting lineplot

sns.lineplot( data['Age'], data['Weight'])
```

Output:

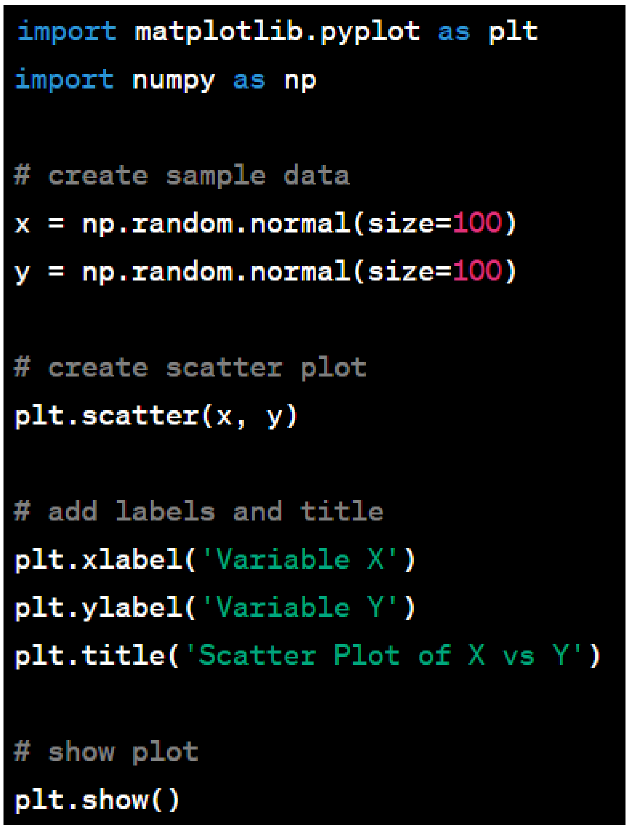


3. Describe scatter plots

A scatter plot is a type of data visualization that displays the relationship between two numerical variables by plotting them as points on a two-dimensional coordinate system. Each point represents a single observation in the dataset, with its position on the x and y axes determined by the values of the two variables being compared.

Scatter plots are commonly used in data analysis and machine learning to explore and visualize the relationship between two variables, and to identify any patterns or trends that may exist. They are particularly useful for identifying correlations between variables, and for detecting outliers or unusual observations in the data.

Here's an example of creating a scatter plot using Matplotlib:



```python
import matplotlib.pyplot as plt
import numpy as np

# create sample data
x = np.random.normal(size=100)
y = np.random.normal(size=100)

# create scatter plot
plt.scatter(x, y)

# add labels and title
plt.xlabel('Variable X')
plt.ylabel('Variable Y')
plt.title('Scatter Plot of X vs Y')

# show plot
plt.show()
```

In this example, we create two arrays of 100 random numbers drawn from a normal distribution using NumPy. We then create a scatter plot using the `scatter()` function from Matplotlib, passing in `x` and `y` as the x and y data, respectively. Each point in the plot represents a single observation in the dataset, with its position on the x and y axes determined by the values of the two variables being compared.

Next, we add labels to the x and y axes using the `xlabel()` and `ylabel()` functions, and a title to the plot using the `title()` function.

Finally, we display the plot using the `show()` function.

Scatter plots can also be customized with various options such as point size, color, and shape, and can be used to display additional information such as groupings or trends by using different colors or markers for different subsets of the data.

OR

Scatterplot Can be used with several semantic groupings which can help to understand well in a graph against continuous/categorical data. It can draw a two-dimensional graph.

Syntax: *seaborn.scatterplot(x=None, y=None)*

Parameters:

x, y: *Input data variables that should be numeric.*

Returns: *This method returns the Axes object with the plot drawn onto it.*

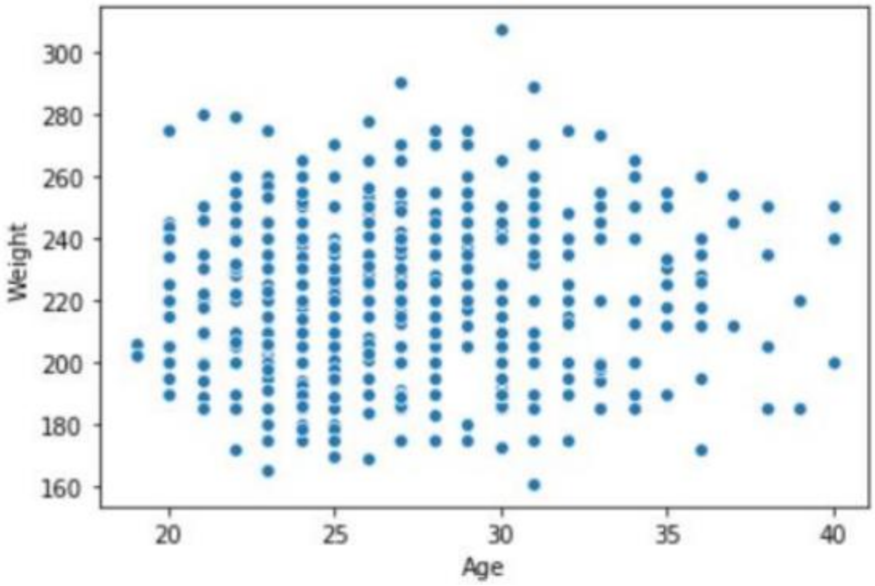
Let's visualize the data with a scatter plot and pandas:

Example 1:

Python3

```
# import module

import seaborn

import pandas

# load csv

data = pandas.read_csv("nba.csv")



# plotting

seaborn.scatterplot(data['Age'],data['Weight'])
```

Output:



4. Summarize density and contour plots.

Density and contour plots are two types of data visualizations that are commonly used in data analysis and machine learning to display the distribution of data in two-dimensional space.

A density plot is a type of data visualization that displays the distribution of data values as a smooth curve that represents an estimate of the underlying probability density function. In a density plot, the x and y axes represent the values of the two variables being compared, and the color of the plot represents the density of data points in that region of the plot.

A contour plot is a type of data visualization that displays the distribution of data values as a set of contour lines that connect points of equal value. In a contour plot, the x and y axes represent the values of the two variables being compared, and the contour lines represent areas of the plot where the data values are of equal density or value.

Density and contour plots can be useful for identifying patterns and trends in data, and for identifying areas of high or low density in the data distribution. They can also be customized with various options such as color maps, smoothing functions, and line widths to improve their readability and visual impact.

5. Describe three dimensional plotting

Matplotlib was introduced keeping in mind, only two-dimensional plotting. But at the time when the release of 1.0 occurred, the 3d utilities were developed upon the 2d and thus, we have 3d implementation of data available today! The 3d plots are enabled by importing the mplot3d toolkit. In this article, we will deal with the 3d plots using matplotlib.

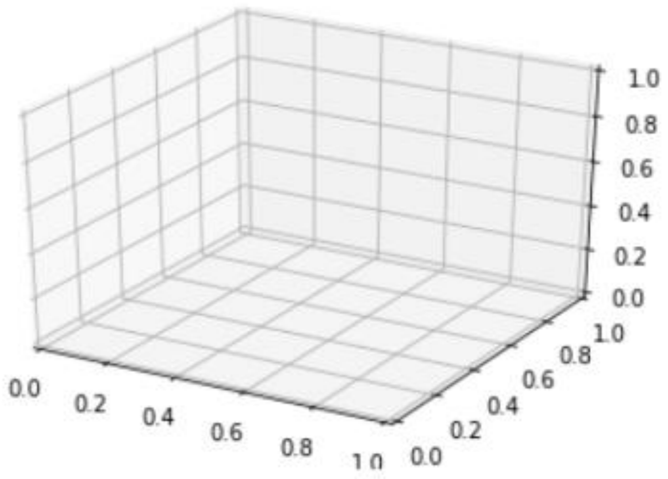
Example:

Python3

```
importnumpy as np
```

```
importmatplotlib.pyplot as plt


fig =plt.figure()
ax =plt.axes(projection ='3d')
```

Output:



With the above syntax three -dimensional axes are enabled and data can be plotted in 3 dimensions. 3 dimension graph gives a dynamic approach and makes data more interactive. Like 2-D graphs, we can use different ways to represent 3-D graph. We can make a scatter plot, contour plot, surface plot, etc. Let’s have a look at different 3-D plots.

6. Illustrate data analysis using stasmodel

7. Write short note on plotly.

Plotly is a Python library for creating interactive, web-based visualizations. It provides a variety of chart types, including scatter plots, line charts, bar charts, and more, as well as 3D and geographic visualizations. Plotly's visualizations are highly customizable, and can be easily embedded in web pages or notebooks.

One of the key features of Plotly is its ability to create interactive visualizations. Users can hover over data points to view additional information, zoom in and out of charts, and even manipulate data on the fly. Plotly's interactivity can be especially useful for exploring complex data sets and identifying patterns and trends.

Plotly can be used in a variety of contexts, including data exploration, data analysis, and data presentation. It can also be integrated with other Python libraries, such as Pandas and NumPy, to streamline the data visualization process.

Here's an example of using Plotly to create a scatter plot:

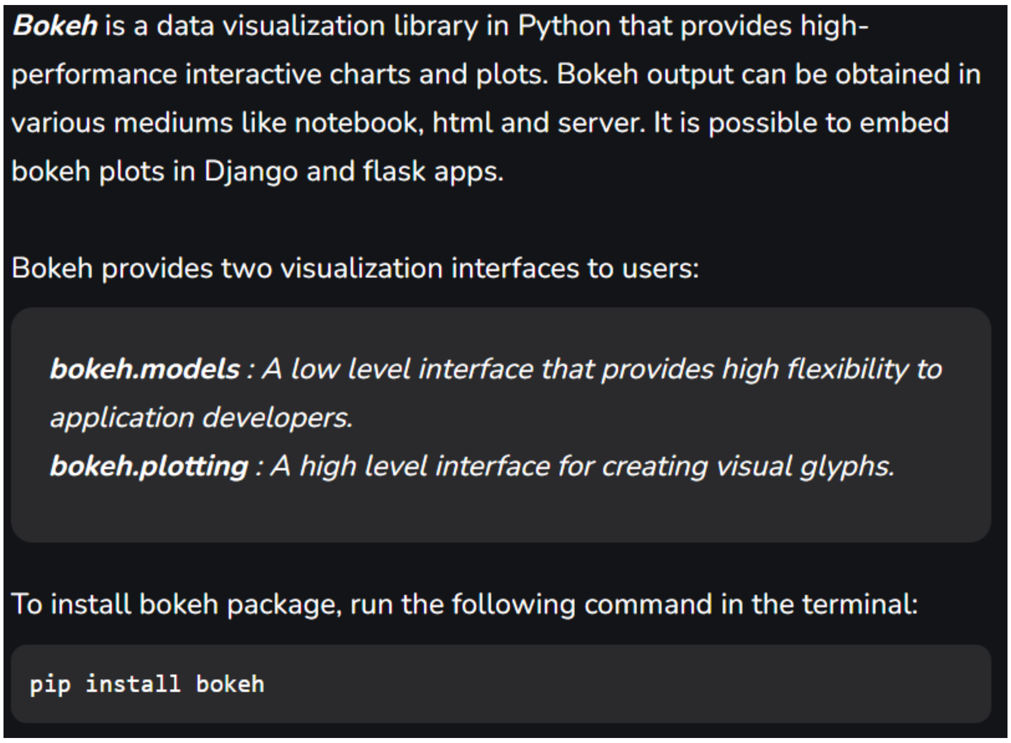
```python
import plotly.express as px
import pandas as pd

# load sample data
data = pd.read_csv('sample_data.csv')

# create scatter plot
fig = px.scatter(data, x='x', y='y', color='group')

# show plot
fig.show()
```

In this example, we first load a sample dataset into a Pandas DataFrame. We then create a scatter plot using the `scatter()` function from Plotly Express, passing in the data, the `x` and `y` variables, and the `color` variable to group data by color.

We then use the `show()` method to display the plot in the output. The resulting plot is interactive, allowing users to hover over data points and view additional information.

Overall, Plotly is a powerful tool for creating interactive, web-based visualizations in Python, and is a valuable library for any data scientist or analyst.

8. Discuss interactive data visualization using bokeh



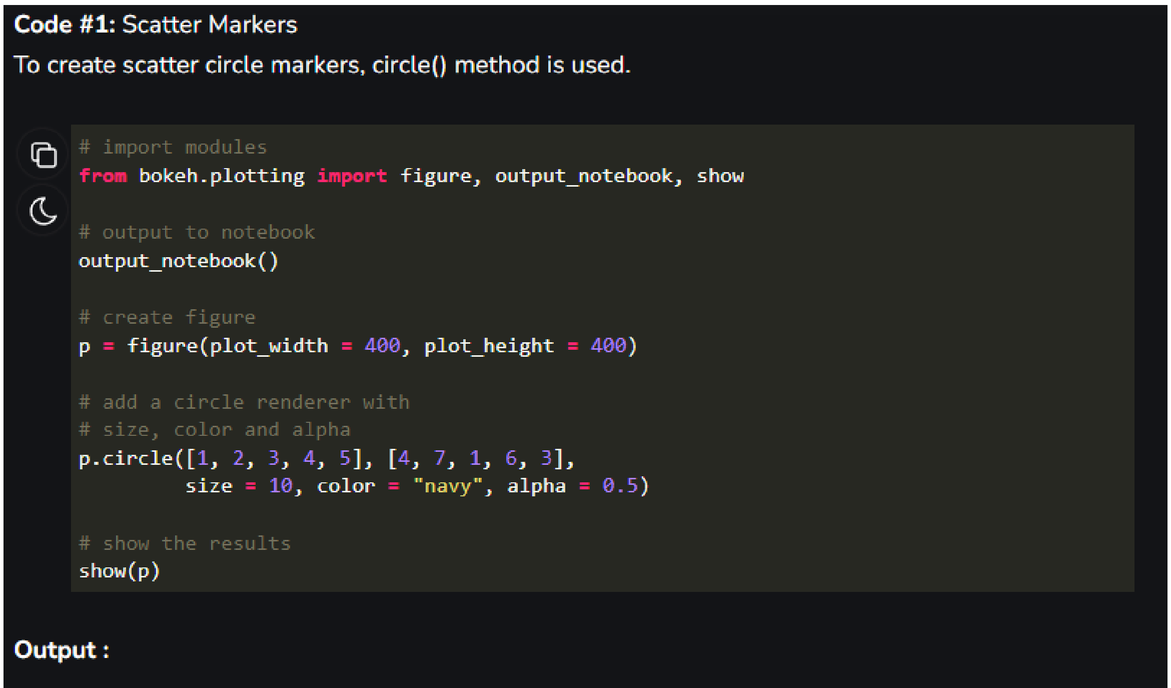
Bokeh is a data visualization library in Python that provides high-performance interactive charts and plots. Bokeh output can be obtained in various mediums like notebook, html and server. It is possible to embed bokeh plots in Django and flask apps.

Bokeh provides two visualization interfaces to users:

> ***bokeh.models*** *: A low level interface that provides high flexibility to application developers.*
> ***bokeh.plotting*** *: A high level interface for creating visual glyphs.*

To install bokeh package, run the following command in the terminal:

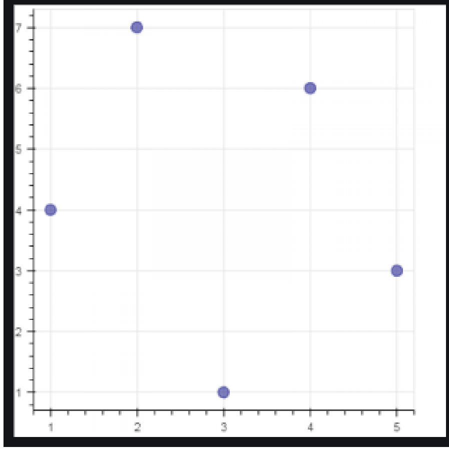
```
pip install bokeh
```



Code #1: Scatter Markers
To create scatter circle markers, circle() method is used.

```python
# import modules
from bokeh.plotting import figure, output_notebook, show

# output to notebook
output_notebook()

# create figure
p = figure(plot_width = 400, plot_height = 400)

# add a circle renderer with
# size, color and alpha
p.circle([1, 2, 3, 4, 5], [4, 7, 1, 6, 3],
         size = 10, color = "navy", alpha = 0.5)

# show the results
show(p)
```

Output :

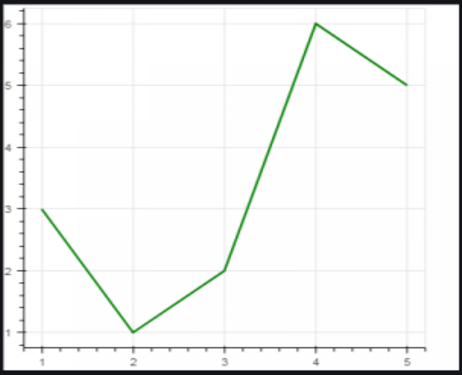


Code #2: Single line

To create a single line, line() method is used.

```
# import modules
from bokeh.plotting import figure, output_notebook, show

# output to notebook
output_notebook()

# create figure
p = figure(plot_width = 400, plot_height = 400)

# add a line renderer
p.line([1, 2, 3, 4, 5], [3, 1, 2, 6, 5],
        line_width = 2, color = "green")

# show the results
show(p)
```

Output :

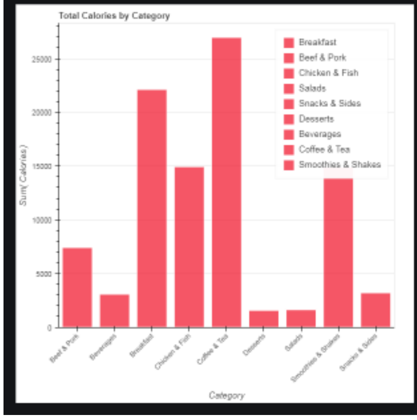


Code #3: Bar Chart

Bar chart presents categorical data with rectangular bars. The length of the bar is proportional to the values that are represented.

```
# import necessary modules
import pandas as pd
from bokeh.charts import Bar, output_notebook, show

# output to notebook
output_notebook()

# read data in dataframe
df = pd.read_csv(r"D:/kaggle/mcdonald/menu.csv")

# create bar
p = Bar(df, "Category", values = "Calories",
        title = "Total Calories by Category",
                        legend = "top_right")

# show the results
show(p)
```

Output :

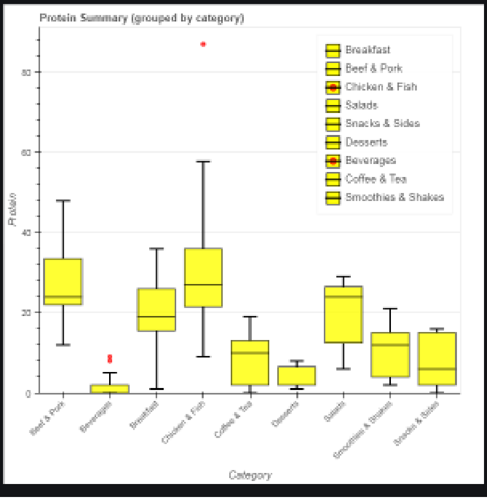


Code #4: Box Plot

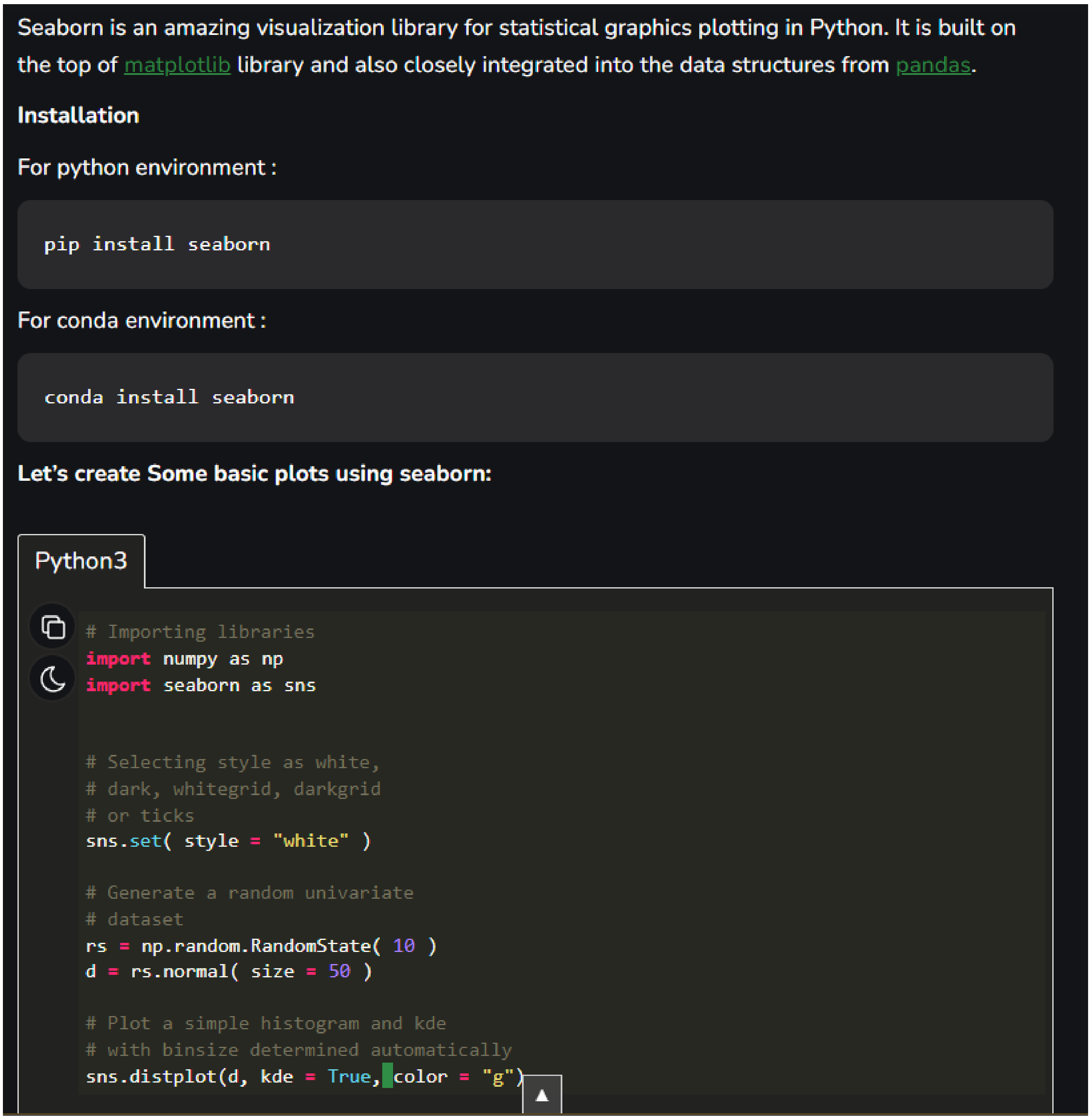
Box plot is used to represent statistical data on a plot. It helps to summarize statistical properties of various data groups present in the data.

```python
# import necessary modules
from bokeh.charts import BoxPlot, output_notebook, show
import pandas as pd

# output to notebook
output_notebook()

# read data in dataframe
df = pd.read_csv(r"D:/kaggle / mcdonald / menu.csv")

# create bar
p = BoxPlot(df, values = "Protein", label = "Category",
            color = "yellow", title = "Protein Summary (grouped by category)",
             legend = "top_right")

# show the results
show(p)
```

Output :



9. Discuss data visualization with python seaborn



Seaborn is an amazing visualization library for statistical graphics plotting in Python. It is built on the top of matplotlib library and also closely integrated into the data structures from pandas.

Installation

For python environment :

```
pip install seaborn
```

For conda environment :

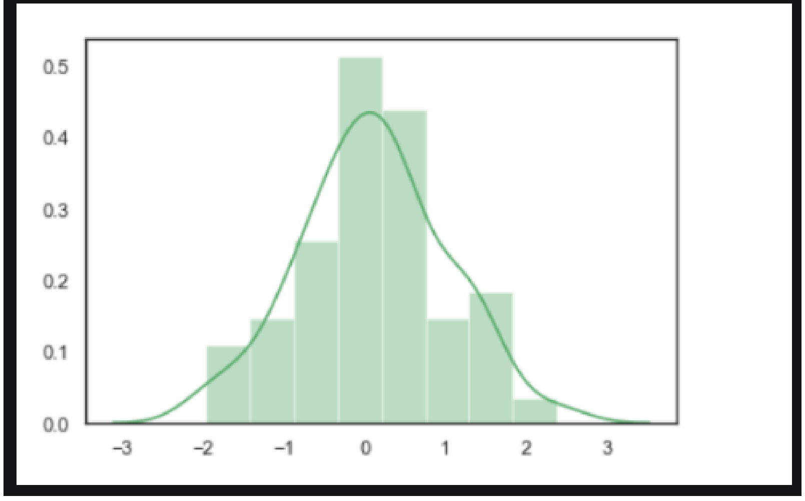
```
conda install seaborn
```

Let's create Some basic plots using seaborn:

Python3

```python
# Importing libraries
import numpy as np
import seaborn as sns


# Selecting style as white,
# dark, whitegrid, darkgrid
# or ticks
sns.set( style = "white" )

# Generate a random univariate
# dataset
rs = np.random.RandomState( 10 )
d = rs.normal( size = 50 )

# Plot a simple histogram and kde
# with binsize determined automatically
sns.distplot(d, kde = True, color = "g")
```

Output:



Seaborn: statistical data visualization

Seaborn helps to visualize the statistical relationships, To understand how variables in a dataset are related to one another and how that relationship is dependent on other variables, we perform statistical analysis. This Statistical analysis helps to visualize the trends and identify various patterns in the dataset.

These are the plot will help to visualize:

- Line Plot
- Scatter Plot
- Box plot
- Point plot
- Count plot
- Violin plot
- Swarm plot
- Bar plot
- KDE Plot

10. Summarize data bining in python.

Unit –V

1. Describe Decision tree in detail

Decision Tree is the most powerful and popular tool for classification and prediction. A Decision tree is a flowchart-like tree structure, where each internal node denotes a test on an attribute, each branch represents an outcome of the test, and each leaf node (terminal node) holds a class label.

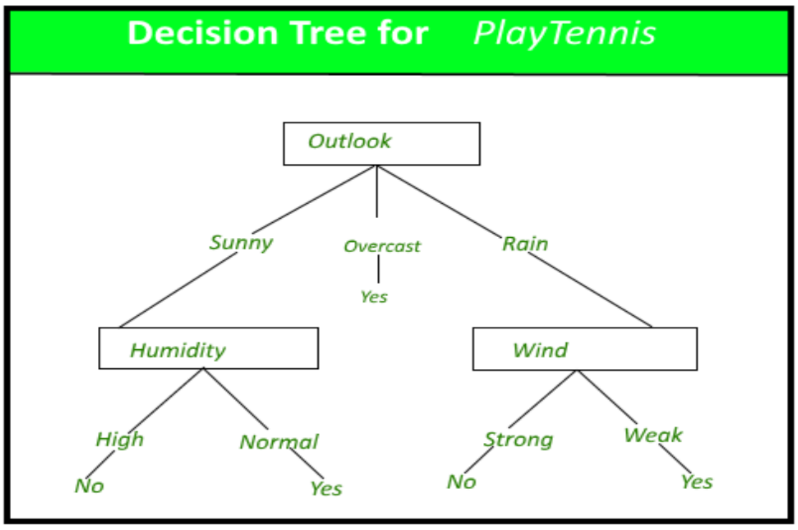
A decision tree is a type of supervised learning algorithm that is commonly used in machine learning for classification and regression analysis. The decision tree algorithm constructs a tree-like model of decisions and their possible consequences.

In a decision tree, each internal node represents a decision or test on an input variable, and each leaf node represents the outcome of the decision or the final classification. The input data is recursively split based on the values of the input features until a stopping criterion is met. The decision tree algorithm searches for the best split at each internal node using some impurity measure, such as Gini impurity or entropy, to determine the best split.

The decision tree algorithm can be applied to both categorical and numerical input data. For categorical data, the decision tree algorithm can use measures such as information gain or gain ratio to evaluate the best split at each internal node. For numerical data, the algorithm can use measures such as mean squared error or mean absolute error to evaluate the best split.

- Decision trees have several advantages, including:
  They are easy to interpret and understand, and their decision-making process can be easily explained to non-experts.
  They can handle both categorical and numerical data.
  They can be used for classification and regression analysis.
  They can handle missing values and noisy data.
  They can be used for feature selection.

- However, decision trees also have some limitations, including:
  They can easily overfit the training data if the tree is too complex.
  They can be sensitive to small variations in the data.
  They may not be suitable for problems where the decision boundary is nonlinear.
  They may not perform well when the classes are highly imbalanced.



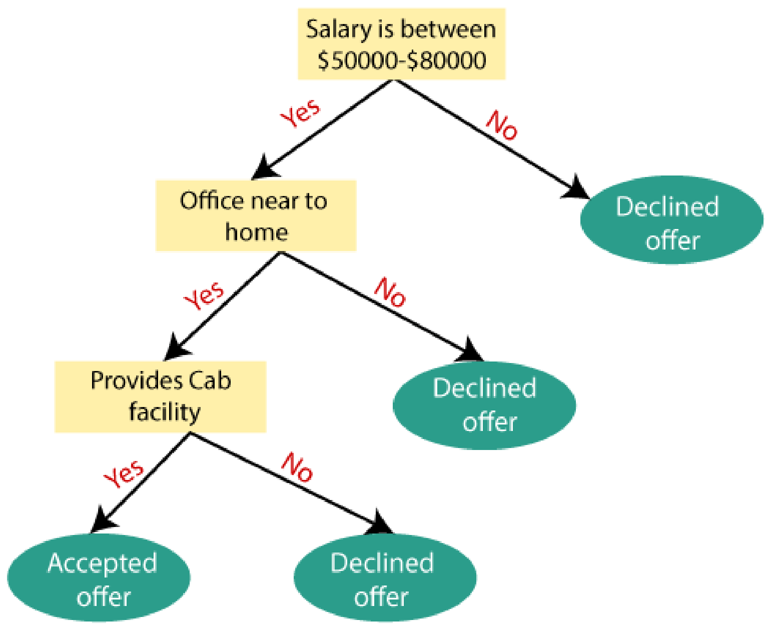
2 Illustrate how does the decision tree work.

In a decision tree, for predicting the class of the given dataset, the algorithm starts from the root node of the tree. This algorithm compares the values of root attribute with the record (real dataset) attribute and, based on the comparison, follows the branch and jumps to the next node.

For the next node, the algorithm again compares the attribute value with the other sub-nodes and move further. It continues the process until it reaches the leaf node of the tree. The complete process can be better understood using the below algorithm:

- **Step-1:** Begin the tree with the root node, says S, which contains the complete dataset.
- **Step-2:** Find the best attribute in the dataset using **Attribute Selection Measure (ASM).**
- **Step-3:** Divide the S into subsets that contains possible values for the best attributes.
- **Step-4:** Generate the decision tree node, which contains the best attribute.
- **Step-5:** Recursively make new decision trees using the subsets of the dataset created in step -3. Continue this process until a stage is reached where you cannot further classify the nodes and called the final node as a leaf node.

Example: Suppose there is a candidate who has a job offer and wants to decide whether he should accept the offer or Not. So, to solve this problem, the decision tree starts with the root node (Salary attribute by ASM). The root node splits further into the next decision node (distance from the office) and one leaf node based on the corresponding labels. The next decision node further gets split into one decision node (Cab facility) and one leaf node. Finally, the decision node splits into two leaf nodes (Accepted offers and Declined offer). Consider the below diagram:



Attribute Selection Measures

While implementing a Decision tree, the main issue arises that how to select the best attribute for the root node and for sub-nodes. So, to solve such problems there is a technique which is called as **Attribute selection measure or ASM.** By this measurement, we can easily select the best attribute for the nodes of the tree. There are two popular techniques for ASM, which are:

- **Information Gain**
- **Gini Index**

1. Information Gain:

- Information gain is the measurement of changes in entropy after the segmentation of a dataset based on an attribute.
- It calculates how much information a feature provides us about a class.
- According to the value of information gain, we split the node and build the decision tree.
- A decision tree algorithm always tries to maximize the value of information gain, and a node/attribute having the highest information gain is split first. It can be calculated using the below formula:

1. Information Gain= Entropy(S)- [(Weighted Avg) *Entropy(each feature)

Entropy: Entropy is a metric to measure the impurity in a given attribute. It specifies randomness in data. Entropy can be calculated as:

```
Entropy(s)= -P(yes)log2 P(yes)- P(no) log2 P(no)
```

Where,

- **S= Total number of samples**
- **P(yes)= probability of yes**
- **P(no)= probability of no**

2. Gini Index:

- Gini index is a measure of impurity or purity used while creating a decision tree in the CART(Classification and Regression Tree) algorithm.
- An attribute with the low Gini index should be preferred as compared to the high Gini index.
- It only creates binary splits, and the CART algorithm uses the Gini index to create binary splits.
- Gini index can be calculated using the below formula:

Gini Index= 1- $\sum_j P_j^2$

3. Illustrate steps for making decision tree

1. Start with your idea

Begin your diagram with one main idea or decision. You'll start your tree with a decision node before adding single branches to the various decisions you're deciding between.

For example, if you want to create an app but can't decide whether to build a new one or upgrade an existing one, use a decision tree to assess the possible outcomes of each.

In this case, the initial decision node is:

- Create an app

The three options—or branches—you're deciding between are:

- Building a new scheduling app
- Upgrading an existing scheduling app
- Building a team productivity app

2. Add chance and decision nodes

After adding your main idea to the tree, continue adding chance or decision nodes after each decision to expand your tree further. A chance node may need an alternative branch after it because there could be more than one potential outcome for choosing that decision.

For example, if you decide to build a new scheduling app, there's a chance that your revenue from the app will be large if it's successful with customers. There's also a chance the app will be unsuccessful, which could result in a small revenue. Mapping both potential outcomes in your decision tree is key.

3. Expand until you reach end points

Keep adding chance and decision nodes to your decision tree until you can't expand the tree further. At this point, add end nodes to your tree to signify the completion of the tree creation process.

Once you've completed your tree, you can begin analyzing each of the decisions.

4. Calculate tree values

Ideally, your decision tree will have quantitative data associated with it. The most common data used in decision trees is monetary value.

For example, it'll cost your company a specific amount of money to build or upgrade an app. It'll also cost more or less money to create one app over another. Writing these values in your tree under each decision can help you in the decision-making process.

You can also try to estimate expected value you'll create, whether large or small, for each decision. Once you know the cost of each outcome and the probability it will occur, you can calculate the expected value of each outcome using the following formula:

- Expected value (EV) = (First possible outcome x Likelihood of outcome) + (Second possible outcome x Likelihood of outcome) - Cost

Calculate the expected value by multiplying both possible outcomes by the likelihood that each outcome will occur and then adding those values. You'll also need to subtract any initial costs from your total.

5. Evaluate outcomes

Once you have your expected outcomes for each decision, determine which decision is best for you based on the amount of risk you're willing to take. The highest expected value may not always be the one you want to go for. That's because, even though it could result in a high reward, it also means taking on the highest level of project risk.

Keep in mind that the expected value in decision tree analysis comes from a probability algorithm. It's up to you and your team to determine how to best evaluate the outcomes of the tree.

4 Summarize Entropy

Entropy is defined as the randomness or measuring the disorder of the information being processed in Machine Learning. Further, in other words, we can say that **entropy is the machine learning metric that measures the unpredictability or impurity in the system**

Entropy is a useful tool in machine learning to understand various concepts such as feature selection, building decision trees, and fitting classification models, etc. Being a machine learning engineer and professional data scientist, you must have in-depth knowledge of entropy in machine learning.

Consider a data set having a total number of N classes, then the entropy (E) can be determined with the formula below:

$$E = -\sum_{i=1}^{N} P_i \log_2 P_i$$

Where;

P_i = Probability of randomly selecting an example in class I;

Entropy always lies between 0 and 1, however depending on the number of classes in the dataset, it can be greater than 1.

The intuition behind entropy is that it measures the uncertainty or randomness in a dataset. If a dataset is completely pure, meaning all the samples belong to the same class, then the entropy is 0 and there is no uncertainty. On the other hand, if a dataset is completely impure, meaning the samples are evenly distributed across all classes, then the entropy is maximum and there is high uncertainty.

In decision tree algorithms, the goal is to split the dataset in a way that reduces the entropy the most, thus making the resulting subsets as pure as possible. The feature that achieves the greatest reduction in entropy is chosen as the split feature. By repeatedly splitting the dataset based on the selected features, a decision tree is built that can be used for classification or regression tasks.

5 Explain neural networks

A neural network is a type of machine learning model that is inspired by the structure and function of the human brain. It consists of multiple layers of interconnected nodes, or artificial neurons, that can learn to recognize patterns and make predictions based on input data.

The basic building block of a neural network is an artificial neuron, which receives input from other neurons or from the outside world, processes the input using a mathematical function, and produces an output that is sent to other neurons. The input to a neuron is weighted, meaning that some inputs are given more importance than others, and the weights are learned through a process called backpropagation.

A neural network typically consists of three types of layers: input layer, hidden layers, and output layer. The input layer receives the raw data, such as images or text, and passes it on to the hidden layers. The hidden layers perform computations on the input and pass the results on to the output layer, which produces the final output, such as a classification label or a numerical value.

The learning process of a neural network involves adjusting the weights of the connections between the neurons in response to the error or loss between the predicted output and the actual output. This is done using an optimization algorithm, such as stochastic gradient descent, that updates the weights in the direction of the steepest descent of the loss function.

One of the key advantages of neural networks is their ability to learn complex and non-linear relationships between the input and output data. They can be used for a variety of tasks, such as image recognition, natural language processing, and time series prediction. However, neural networks can also be computationally expensive and require large amounts of data to train effectively.

Neural networks have several use cases across many industries, such as the following:

- Medical diagnosis by medical image classification
- Targeted marketing by social network filtering and behavioral data analysis
- Financial predictions by processing historical data of financial instruments
- Electrical load and energy demand forecasting
- Process and quality control
- Chemical compound identification

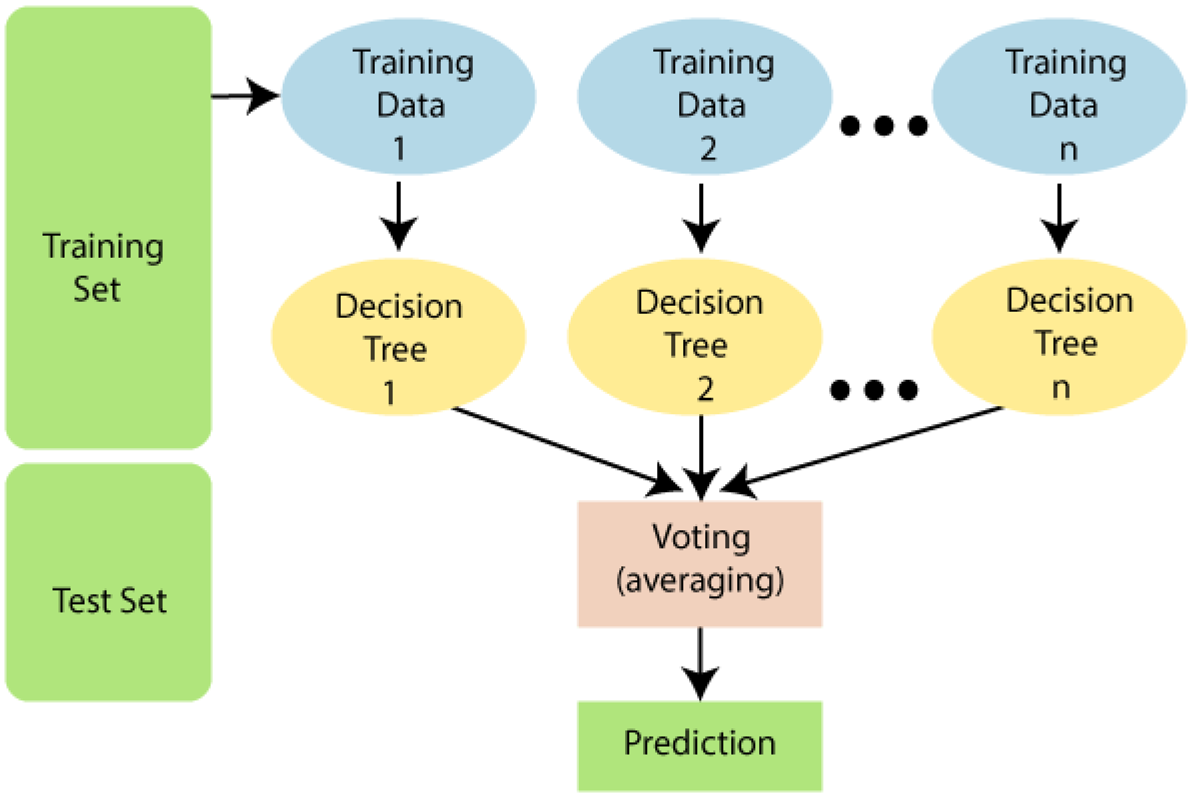
6 | Dicuss Random forest.

Random Forest is a popular machine learning algorithm that belongs to the supervised learning technique. It can be used for both Classification and Regression problems in ML. It is based on the concept of **ensemble learning,** which is a process of *combining multiple classifiers to solve a complex problem and to improve the performance of the model.*

As the name suggests, ***"Random Forest is a classifier that contains a number of decision trees on various subsets of the given dataset and takes the average to improve the predictive accuracy of that dataset."*** Instead of relying on one decision tree, the random forest takes the prediction from each tree and based on the majority votes of predictions, and it predicts the final output.

The greater number of trees in the forest leads to higher accuracy and prevents the problem of overfitting.

The below diagram explains the working of the Random Forest algorithm:



Below are some points that explain why we should use the Random Forest algorithm:

- It takes less training time as compared to other algorithms.
- It predicts output with high accuracy, even for the large dataset it runs efficiently.
- It can also maintain accuracy when a large proportion of data is missing.

There are mainly four sectors where Random forest mostly used:

1. **Banking:** Banking sector mostly uses this algorithm for the identification of loan risk.

2. **Medicine:** With the help of this algorithm, disease trends and risks of the disease can be identified.
3. **Land Use:** We can identify the areas of similar land use by this algorithm.
4. **Marketing:** Marketing trends can be identified using this algorithm.

Advantages of Random Forest

- Random Forest is capable of performing both Classification and Regression tasks.
- It is capable of handling large datasets with high dimensionality.
- It enhances the accuracy of the model and prevents the overfitting issue.

Disadvantages of Random Forest

- Although random forest can be used for both classification and regression tasks, it is not more suitable for Regression tasks.

7 Summarize backpropogation

Backpropagation is an algorithm used for training artificial neural networks in data science. It is a supervised learning technique that uses a mathematical method called gradient descent to adjust the weights of the connections between neurons in a neural network.

The goal of backpropagation is to minimize the error or loss between the predicted output of a neural network and the actual output. It works by propagating the error backwards through the network, from the output layer to the input layer, and adjusting the weights of the connections along the way.

The backpropagation algorithm consists of the following steps:

Forward propagation: The input data is fed forward through the network, and the predicted output is calculated using the current weights.

Calculate error: The difference between the predicted output and the actual output is calculated as the error or loss.

Backward propagation: The error is propagated backward through the network, and the gradient of the error with respect to each weight is calculated.

Update weights: The weights are updated in the direction of the negative gradient of the error, using a learning rate that determines the step size of the update.

Repeat: The process is repeated for a fixed number of iterations or until the error converges to a minimum.

The backpropagation algorithm allows neural networks to learn from data by adjusting the weights of the connections between neurons. By minimizing the error or loss function, the network can learn to make accurate predictions on new data.

Backpropagation is a powerful and widely used algorithm in deep learning and neural network training. However, it can be computationally expensive and may suffer from issues such as vanishing gradients or overfitting if not properly tuned.

Types of Backpropagation

There are two types of backpropagation networks.

- **Static backpropagation:** Static backpropagation is a network designed to map static inputs for static outputs. These types of networks are capable of solving static classification problems such as OCR (Optical Character Recognition).
- **Recurrent backpropagation:** Recursive backpropagation is another network used for fixed-point learning. Activation in recurrent backpropagation is feed-forward until a fixed value is reached. Static backpropagation provides an instant mapping, while recurrent backpropagation does not provide an instant mapping.

Advantages:

- It is simple, fast, and easy to program.
- Only numbers of the input are tuned, not any other parameter.
- It is Flexible and efficient.
- No need for users to learn any special functions.

Disadvantages:

- It is sensitive to noisy data and irregularities. Noisy data can lead to inaccurate results.
- Performance is highly dependent on input data.
- Spending too much time training.
- The matrix-based approach is preferred over a mini-batch.

Perceptron is a building block of an Artificial Neural Network.

Perceptron is Machine Learning algorithm for supervised learning of various binary classification tasks. Further, ***Perceptron is also understood as an Artificial Neuron or neural network unit that helps to detect certain input data computations in business intelligence***.

Perceptron model is also treated as one of the best and simplest types of Artificial Neural networks. However, it is a supervised learning algorithm of binary classifiers. Hence, we can consider it as a single-layer neural network with four main parameters, i.e., **input values, weights and Bias, net sum, and an activation function.**

Advantages of Multi-Layer Perceptron:

- A multi-layered perceptron model can be used to solve complex non-linear problems.
- It works well with both small and large input data.
- It helps us to obtain quick predictions after the training.
- It helps to obtain the same accuracy ratio with large as well as small data.

Disadvantages of Multi-Layer Perceptron:

- In Multi-layer perceptron, computations are difficult and time-consuming.
- In multi-layer Perceptron, it is difficult to predict how much the dependent variable affects each independent variable.
- The model functioning depends on the quality of the training.

Characteristics of Perceptron

The perceptron model has the following characteristics.

1. Perceptron is a machine learning algorithm for supervised learning of binary classifiers.
2. In Perceptron, the weight coefficient is automatically learned.
3. Initially, weights are multiplied with input features, and the decision is made whether the neuron is fired or not.
4. The activation function applies a step rule to check whether the weight function is greater than zero.
5. The linear decision boundary is drawn, enabling the distinction between the two linearly separable classes +1 and -1.
6. If the added sum of all input values is more than the threshold value, it must have an output signal; otherwise, no output will be shown.

Limitations of Perceptron Model

A perceptron model has limitations as follows:

- The output of a perceptron can only be a binary number (0 or 1) due to the hard limit transfer function.
- Perceptron can only be used to classify the linearly separable sets of input vectors. If input vectors are non-linear, it is not easy to classify them properly.

9 Explain Key features of mapreduce

The key features of MapReduce are:

Scalability: MapReduce is highly scalable and can process petabytes of data on a cluster of commodity hardware.

Fault-tolerance: MapReduce is designed to handle failures of individual nodes in a cluster, and automatically retries failed tasks on other nodes.

Data locality: MapReduce tries to process data on the same node where it is stored, which reduces network traffic and improves performance.

Simplified programming model: MapReduce provides a simple and high-level programming model that abstracts away the details of distributed computing, making it easier for developers to write parallel and distributed programs.

Parallelism: MapReduce breaks down large tasks into smaller sub-tasks that can be processed in parallel, which reduces processing time and improves efficiency.

Support for various data formats: MapReduce supports various data formats such as text, binary, and key-value pairs, making it flexible and versatile for processing different types of data.

Customizable: MapReduce is customizable and can be extended with user-defined functions or libraries for specialized processing tasks.

10 Explain feedforward neural network

A feedforward neural network, also known as a multilayer perceptron, is a type of artificial neural network that consists of an input layer, one or more hidden layers, and an output layer. In a feedforward neural network, information flows in a forward direction, from the input layer through the hidden layers to the output layer, without any feedback loops.

Each layer in a feedforward neural network consists of a set of neurons, or nodes, that are connected to the neurons in the adjacent layers. The connections between neurons are weighted, which allows the network to learn and generalize from data. The weights are adjusted during the training process using an optimization algorithm, such as gradient descent, to minimize the error or loss between the predicted output and the actual output.

The input layer of a feedforward neural network receives the input data, which is typically represented as a vector of features. The input data is then multiplied by the weights of the connections between the input layer and the first hidden layer, and passed through an activation function, which produces the output of the first hidden layer. This process is repeated for each subsequent hidden layer, until the output layer is reached.

The output layer of a feedforward neural network produces the final output, which can be a scalar value, a vector of probabilities, or a vector of class labels. The activation function used in the output layer depends on the type of task being performed. For example, in a binary classification task, the output layer may use a sigmoid activation function to produce a probability value between 0 and 1.

Feedforward neural networks are widely used for supervised learning tasks, such as classification and regression, and can be trained on a variety of data types, such as text, images, and time series data. They can also be used as a building block for more complex neural network architectures, such as convolutional neural networks and recurrent neural networks.