**Exp 6**

User

**You**

**import numpy as np # linear algebra**

**import pandas as pd # data processing, CSV file I/O (e.g. pd.read\_csv)**

**import matplotlib.pyplot as plt**

**import seaborn as sns**

ChatGPT

**ChatGPT**

Sure! Here are the meanings of each line:

1. **import numpy as np**: This line imports the NumPy library, a popular Python library for numerical operations, and assigns it the alias **np** for easier referencing in the code.
2. **import pandas as pd**: This line imports the Pandas library, another popular Python library used for data manipulation and analysis, and assigns it the alias **pd** for easier referencing in the code.
3. **import matplotlib.pyplot as plt**: This line imports the pyplot module from the Matplotlib library, which is a widely used plotting library in Python. It assigns the alias **plt** for easier referencing in the code.
4. **import seaborn as sns**: This line imports the Seaborn library, which is built on top of Matplotlib and provides additional functionality for statistical data visualization. It assigns the alias **sns** for easier referencing in the code.

User

**You**

**iris=pd.read\_csv('E:\DS&BDA\PR\iris.csv')**

ChatGPT

**ChatGPT**

This line reads a CSV file named "iris.csv" located at the path 'E:\DS&BDA\PR\iris.csv' into a Pandas DataFrame named **iris**.

User

**You**

**iris.head()**

ChatGPT

**ChatGPT**

This line displays the first few rows of the DataFrame **iris**, allowing you to quickly inspect the structure and contents of the data.

User

**You**

**iris['Species'].unique()**

ChatGPT

**ChatGPT**

This line retrieves the unique values present in the "Species" column of the **iris** DataFrame.

User

**You**

**iris.drop(columns="Id",inplace=True)**

ChatGPT

**ChatGPT**

This line removes the column named "Id" from the **iris** DataFrame in place, meaning it modifies the DataFrame directly without creating a new copy.

User

**You**

**g=sns.relplot(x='SepalLengthCm',y='SepalWidthCm',data=iris,hue='Species',style='Species') g.fig.set\_size\_inches(10,5) plt.show()**

ChatGPT

**ChatGPT**

This code generates a relational plot (scatter plot) using Seaborn's **relplot** function. It visualizes the relationship between "SepalLengthCm" and "SepalWidthCm" columns from the **iris** DataFrame. The points are colored by the "Species" column, and each species is represented with a different marker style.

Additionally, it adjusts the size of the plot's figure to 10 inches in width and 5 inches in height using **g.fig.set\_size\_inches(10,5)**. Finally, it displays the plot using **plt.show()**.

User

**You**

**plt.figure(figsize=(15,10))**

**plt.subplot(2,2,1)**

**sns.violinplot(x='Species',y='PetalLengthCm',data=iris)**

**plt.subplot(2,2,2)**

**sns.violinplot(x='Species',y='PetalWidthCm',data=iris)**

**plt.subplot(2,2,3)**

**sns.violinplot(x='Species',y='SepalLengthCm',data=iris)**

**plt.subplot(2,2,4)**

**sns.violinplot(x='Species',y='SepalWidthCm',data=iris)**

**plt.show()**

ChatGPT

**ChatGPT**

This code creates a figure with a size of 15 inches in width and 10 inches in height using **plt.figure(figsize=(15,10))**.

Then, it sets up a grid of subplots with 2 rows and 2 columns using **plt.subplot(2,2,1)**, **plt.subplot(2,2,2)**, **plt.subplot(2,2,3)**, and **plt.subplot(2,2,4)**.

For each subplot:

* It creates a violin plot using Seaborn's **violinplot** function, showing the distribution of data for each species across the specified feature (PetalLengthCm, PetalWidthCm, SepalLengthCm, SepalWidthCm).
* The x-axis represents the species, and the y-axis represents the corresponding feature.
* Each subplot is dedicated to one of the four features.

Finally, it displays the entire plot grid using **plt.show()**.

User

**You**

**plt.subplots(figsize=(10,7))**

**sns.violinplot(data=iris)**

**sns.swarmplot( data=iris)**

**plt.show()**

ChatGPT

**ChatGPT**

This code creates a single plot with two types of plots overlaid: a violin plot and a swarm plot.

* **plt.subplots(figsize=(10,7))**: This creates a figure with a size of 10 inches in width and 7 inches in height.
* **sns.violinplot(data=iris)**: This plots violin plots for each numerical column in the **iris** DataFrame. A violin plot shows the distribution of the data across different levels of a categorical variable.
* **sns.swarmplot(data=iris)**: This overlays a swarm plot on top of the violin plot. A swarm plot shows each data point individually, which helps to visualize the distribution more clearly.
* **plt.show()**: This displays the combined plot.

User

**You**

**iris.plot.area(y=['SepalLengthCm','SepalWidthCm','PetalLengthCm','PetalWidthCm'],alpha=0.4,figsize=(12, 6));**

ChatGPT

**ChatGPT**

This code creates an area plot using the **plot.area** function on the specified columns ('SepalLengthCm', 'SepalWidthCm', 'PetalLengthCm', 'PetalWidthCm') of the **iris** DataFrame.

* **y=['SepalLengthCm','SepalWidthCm','PetalLengthCm','PetalWidthCm']**: This specifies the columns to be plotted on the y-axis.
* **alpha=0.4**: This sets the transparency level of the filled areas in the plot to 0.4, making them partially transparent.
* **figsize=(12, 6)**: This sets the size of the figure to be 12 inches in width and 6 inches in height.

Finally, it displays the area plot.

User

**You**

**iris.corr()**

ChatGPT

**ChatGPT**

This code computes the correlation matrix for the numeric columns of the **iris** DataFrame using the **.corr()** method. The correlation matrix shows the pairwise correlations between all pairs of numeric columns in the DataFrame.

User

**You**

**plt.subplots(figsize = (8,8))**

**sns.heatmap(iris.corr(),annot=True,fmt="f").set\_title("Corelation of attributes (petal length,width and sepal length,width) among Iris species")**

**plt.show()**

ChatGPT

**ChatGPT**

This code creates a heatmap using Seaborn's **heatmap** function to visualize the correlation matrix of the numeric columns in the **iris** DataFrame.

* **plt.subplots(figsize = (8,8))**: This creates a figure with a size of 8 inches in width and 8 inches in height.
* **sns.heatmap(iris.corr(), annot=True, fmt="f")**: This generates the heatmap. It uses the correlation matrix computed from **iris.corr()**. The **annot=True** parameter adds numeric annotations to each cell of the heatmap, and **fmt="f"** formats the annotations as floating-point numbers.
* **.set\_title("Correlation of attributes (petal length, width, and sepal length, width) among Iris species")**: This sets the title for the heatmap.

Finally, it displays the heatmap plot.

User

**You**

**X=iris.iloc[:,0:4].values**

**y=iris.iloc[:,4].values**

ChatGPT

**ChatGPT**

This code extracts the feature variables (**X**) and the target variable (**y**) from the **iris** DataFrame.

* **X = iris.iloc[:, 0:4].values**: This extracts the values of the first four columns (features) of the **iris** DataFrame and assigns them to the variable **X**. The **iloc** function is used for integer-location based indexing.
* **y = iris.iloc[:, 4].values**: This extracts the values of the fifth column (target) of the **iris** DataFrame and assigns them to the variable **y**.

User

**You**

**from sklearn.preprocessing import LabelEncoder**

**le = LabelEncoder()**

**y = le.fit\_transform(y)**

ChatGPT

**ChatGPT**

This code imports the **LabelEncoder** class from the **sklearn.preprocessing** module and creates an instance of it called **le**.

* **le = LabelEncoder()**: This initializes a LabelEncoder object.
* **y = le.fit\_transform(y)**: This applies the label encoding transformation to the target variable **y**. It converts the categorical labels in **y** (representing different species of iris) into numerical labels.

User

**You**

**#Metrics**

**from sklearn.metrics import make\_scorer, accuracy\_score,precision\_score**

**from sklearn.metrics import classification\_report**

**from sklearn.metrics import confusion\_matrix**

**from sklearn.metrics import accuracy\_score ,precision\_score,recall\_score,f1\_score #Model Select**

**from sklearn.naive\_bayes import GaussianNB**

ChatGPT

**ChatGPT**

This code snippet imports various metrics and modules related to model evaluation and selection from the scikit-learn library.

* **Metrics**:
  + **make\_scorer**: This function creates a scoring function from a given metric or a callable.
  + **accuracy\_score**, **precision\_score**, **recall\_score**, **f1\_score**: These functions compute different classification performance metrics such as accuracy, precision, recall, and F1 score.
  + **classification\_report**: This function generates a text report showing the main classification metrics.
  + **confusion\_matrix**: This function computes the confusion matrix to evaluate the accuracy of a classification.
* **Model Selection**:
  + **GaussianNB**: This is a class representing the Gaussian Naive Bayes classifier, which is a probabilistic classifier based on Bayes' theorem with the assumption of independence between features.

User

**You**

**#Train and Test split**

**X\_train,X\_test,y\_train,y\_test=train\_test\_split(X,y,test\_size=0.3,random\_state=0)**

ChatGPT

**ChatGPT**

It seems like you're using a train-test split function to divide your dataset into training and testing sets. However, it seems like you haven't imported the necessary function from scikit-learn. Here's the missing import statement:

python

Copy code

from sklearn.model\_selection import train\_test\_split

Once you add this import statement, you can use the **train\_test\_split** function to split your data into training and testing sets. The parameters are:

* **X**: The feature variables.
* **y**: The target variable.
* **test\_size**: The proportion of the dataset to include in the test split. Here, it's set to 0.3, meaning 30% of the data will be used for testing.
* **random\_state**: Controls the shuffling applied to the data before applying the split. Setting a fixed value ensures reproducibility of the results.

User

**You**

**gaussian = GaussianNB()**

**gaussian.fit(X\_train, y\_train)**

**Y\_pred = gaussian.predict(X\_test)**

**accuracy\_nb=round(accuracy\_score(y\_test,Y\_pred)\* 100, 2)**

**acc\_gaussian = round(gaussian.score(X\_train, y\_train) \* 100, 2)**

**cm = confusion\_matrix(y\_test, Y\_pred)**

**accuracy = accuracy\_score(y\_test,Y\_pred)**

**precision =precision\_score(y\_test, Y\_pred,average='micro')**

**recall = recall\_score(y\_test, Y\_pred,average='micro')**

**f1 = f1\_score(y\_test,Y\_pred,average='micro')**

**print('Confusion matrix for Naive Bayes\n',cm)**

**print('accuracy\_Naive Bayes: %.3f' %accuracy)**

**print('precision\_Naive Bayes: %.3f' %precision)**

**print('recall\_Naive Bayes: %.3f' %recall)**

**print('f1-score\_Naive Bayes : %.3f' %f1)**

ChatGPT

**ChatGPT**

This code snippet performs the following tasks:

1. Instantiates a Gaussian Naive Bayes classifier: **gaussian = GaussianNB()**.
2. Fits the classifier to the training data: **gaussian.fit(X\_train, y\_train)**.
3. Predicts the target values for the test data: **Y\_pred = gaussian.predict(X\_test)**.
4. Computes and prints various evaluation metrics such as accuracy, precision, recall, and F1-score:
   * Computes accuracy using **accuracy\_score**.
   * Computes precision using **precision\_score**.
   * Computes recall using **recall\_score**.
   * Computes F1-score using **f1\_score**.
5. Prints the confusion matrix using **confusion\_matrix**.

These metrics are then printed for evaluation purposes.

It's a comprehensive approach to evaluating the performance of the Gaussian Naive Bayes classifier on your dataset.

Top of Form

**Gaussian Naive Bayes (GaussianNB) is a classification algorithm based on Bayes' theorem** with the assumption of independence between features. It's a probabilistic classifier that assigns the most probable class label to a given input based on the feature values.

The "Gaussian" in its name indicates that it assumes that the likelihood of the features follows a Gaussian (normal) distribution. This means that it works well with continuous data.

Despite its simplicity and the strong independence assumptions it makes, Gaussian Naive Bayes often performs surprisingly well in practice, especially in text classification and other domains with high-dimensional feature spaces. It's computationally efficient and can handle large datasets with ease.