Here is a compact explanation of your DSBDA code block — line by line and keyword by keyword — with library use, key terms, and function meanings for screenshot purposes:

📦 Libraries Used

python

```python
import pandas as pd      # For handling dataframes (loading, analyzing, cleaning data)
import numpy as np       # For numerical operations (used internally or when needed)
from sklearn.preprocessing import MinMaxScaler  # For feature scaling (not used here but imported
```

📄 Data Loading

python

```python
df = pd.read_csv('train.csv')
```

- `read_csv` : loads CSV into a DataFrame
- `df` : variable to hold the dataset

👁 Basic Exploration

python

```python
df                        # Displays the whole dataframe in console
df.head()                 # Shows first 5 rows
df.tail()                 # Shows last 5 rows
df.isnull().sum()         # Shows total missing values per column
df.describe()             # Stats summary: mean, std, min, max, etc.
df.info()                 # Structure: column names, non-null counts, datatypes
df.dtypes                 # Lists data types of all columns
df.shape                  # (rows, columns) count
```

python

```python
def impute_age(cols):             # Custom function to fill missing Age
    Age = cols[0]                 # First column is Age
    Pclass = cols[1]              # Second column is Passenger Class

    if pd.isnull(Age):            # Check if Age is missing
        if Pclass == 1: return 37  # Assumes avg age for Class 1
        elif Pclass == 2: return 29
        else: return 24
    else:
        return Age                # If not missing, keep it
```

Copy Edit

🛠 Apply the Function

python Copy Edit

```python
df['Age'] = df[['Age', 'Pclass']].apply(impute_age, axis=1)
```

- `apply()` : applies function row-wise (`axis=1`)
- `[['Age', 'Pclass']]` : selects both columns to pass as input

🧹 Clean Data

python Copy Edit

```python
df.drop('Cabin', axis=1, inplace=True)  # Drops column 'Cabin'
df.dropna(inplace=True)                 # Drops all rows with any missing value
df.head()                               # Show updated data
```

- `drop()` : removes column/row
- `axis=1` : column-wise
- `inplace=True` : modifies original DataFrame

❗ Note

- `MinMaxScaler` was imported but not used.
- There's a typo: `return Agedf['Age'] =` — should be `return Age` (fix before running).

📦 Library Used

python Copy Edit

```python
import seaborn as sns
```

- `seaborn`: high-level data visualization library built on Matplotlib
- Ideal for statistical plots and analyzing patterns visually

📄 Dataset Loading

python Copy Edit

```python
df = sns.load_dataset('titanic')
```

- Loads the **Titanic** dataset from Seaborn's built-in datasets
- `df`: DataFrame with passenger info like age, sex, class, survived, etc.

👁️ Initial Preview

python Copy Edit

```python
df.head()
```

- Displays first 5 rows of the DataFrame to check structure

📊 Boxplot Visualization

python Copy Edit

```python
sns.boxplot(x='sex', y='age', data=df, hue='survived')
```

- Shows **distribution of age** grouped by `sex`
- `hue='survived'`: different colors for survived (1, d not (0)

- `hue='survived'` : different colors for survived (1) and not (0)
- Helps spot age trends among male/female survivors

📈 Categorical Count Plot

python

Copy Edit

```python
sns.catplot(x='survived', data=df, kind='count', hue='sex')
```

- Bar plot showing how many survived (1) vs not (0)
- Split by `sex` to show gender-based survival distribution
- `kind='count'` : count of occurrences for each category

🖼️ Figure Size Setting

python

Copy Edit

```python
sns.set(rc={'figure.figsize':(5,5)})
```

- `rc` : runtime configuration
- Sets default plot size to **5x5 inches**

python

Copy Edit

```python
import pandas as pd       # Pandas library for data manipulation (reading, cleaning, analyzing)
import seaborn as sns     # Seaborn for statistical plotting, built on matplotlib
import matplotlib.pyplot as plt  # Matplotlib for general plotting, controlling graph features
import numpy as np         # NumPy for numerical operations, handling arrays and matrices

df = pd.read_csv("iris.csv")  # Loads the Iris dataset from CSV into a DataFrame
print(df.head())              # Shows the first 5 rows of the dataset to inspect initial data
print("\nFeature Types:")      # Print a label for the feature types (data types of each column)
print(df.dtypes)              # Displays the data type of each column (e.g., int64, float64, obje

# Visualizing the distribution of numerical features using histograms
sns.histplot(df['sepal_length'])  # Plots histogram for sepal length, shows distribution
sns.histplot(df['petal_length'])  # Plots histogram for petal length
sns.histplot(df['sepal_width'])   # Plots histogram for sepal width
sns.histplot(df['petal_width'])   # Plots histogram for petal width
sns.histplot(df['species'])       # Plots histogram for species distribution (categorical)

# Counts the occurrences of each species in the dataset
df['species'].value_counts()      # Returns counts of each species (Iris-setosa, Iris-versicolor, 

# Boxplots show the spread and outliers of numerical features
sns.boxplot(df['sepal_length'])  # Creates a boxplot to visualize the distribution of sepal lengt
sns.boxplot(df['petal_length'])  # Creates a boxplot for petal length distribution
sns.boxplot(df['sepal_width'])   # Boxplot for sepal width to check the distribution
sns.boxplot(df['petal_width'])   # Boxplot for petal width distribution

# Create a list of multiple features to plot in one boxplot
data_to_plot = [df['sepal_length'], df['sepal_width'], df['petal_length'], df['petal_width']]  # 
fig = plt.figure(1, figsize=(12,8))     # Creates a new figure of size 12x8 inches
ax = fig.add_subplot(111)                # Adds a subplot to the figure (1x1 grid, first plot)
bp = ax.boxplot(data_to_plot)           # Creates a boxplot for all the selected features (sepal a
```

Detailed Explanation:

- `pandas` : Powerful library for data manipulation tasks such as reading, cleaning, and analyzing structured data.
  - `read_csv()` : Reads data from a CSV file into a Pandas DataFrame, which is a table-like structure.
  - `dtypes` : Shows the data types (e.g., integers, floats) for each column in the dataset.
- `seaborn` : High-level statistical plotting library built on Matplotlib that simplifies the creation of complex

- `seaborn`: High-level statistical plotting library built on Matplotlib that simplifies the creation of complex plots.
  - `histplot()`: Used to plot the distribution of numerical variables, showing how data points are spread.
  - `boxplot()`: A graphical representation of the distribution of data (min, max, median, quartiles) and outliers. Useful for spotting variations and detecting outliers.
- `matplotlib`: Low-level plotting library that helps to customize graphs and figures.
  - `figure()`: Creates a new figure for plotting with custom size.
  - `add_subplot()`: Adds a subplot to the figure to organize multiple plots in one figure.
- **Histograms** are used to understand the **distribution of data** for features like sepal length, petal length, etc., and to visually assess if the data follows a normal distribution.
- **Boxplots** show the **spread** of the data, **central tendency (median)**, and **outliers** for each feature (sepal length, petal length, etc.).
- `value_counts()`: A method to count how many times each unique value appears in a categorical column (e.g., species names).

python Copy Edit

```python
import numpy as np          # NumPy for numerical operations and handling arrays
import pandas as pd         # Pandas for data manipulation (loading, cleaning, analyzing)
import matplotlib.pyplot as plt  # Matplotlib for creating plots
%matplotlib inline           # Ensures plots are displayed inline in Jupyter notebooks
import seaborn as sns        # Seaborn for high-level data visualization
import warnings              # For suppressing warnings during execution
warnings.filterwarnings('ignore')  # Ignore warnings that might clutter output

df = pd.read_csv('tiatanic.csv')  # Loads Titanic dataset into a Pandas DataFrame
df.info()                    # Displays basic info about the dataset (e.g., column names, non-
df.describe()                # Shows summary statistics for numerical columns (mean, std, min,
df.shape                     # Returns the number of rows and columns in the dataset (shape of

# Histograms showing distribution of the 'Fare' feature
sns.histplot(x='Fare', data=df)     # Plots the distribution of the 'Fare' column

# Sets default figure size for subsequent plots
sns.set(rc={'figure.figsize':(3,3)})   # Smaller figure size for better layout

# Distribution of 'Age' with 70 bins
sns.displot(x='Age', data=df, bins=70)  # Creates a histogram of 'Age' with custom bins

# Sets larger figure size for the next plot
sns.set(rc={'figure.figsize':(5,5)})   # Sets the figure size to 5x5 inches for the next plots

# Counts the number of survivors, split by passenger class
sns.catplot(x='Survived', data=df, kind='count', hue='Pclass')  # Shows survival count by passeng

# Counts the number of survivors, split by sex
sns.catplot(x='Survived', data=df, kind='count', hue='Sex')  # Shows survival count by sex (male/
```

Detailed Explanation:

- `pandas` : A powerful library for data manipulation; it handles tasks like reading CSV files, cleaning data, and performing statistical analysis.
  - `read_csv()` : Reads data from a CSV file into a DataFrame for easier analysis.
  - `info()` : Provides an overview of the dataset, such as column names, data types, and non-null counts.
  - `describe()` : Provides summary statistics for numerical columns like mean, standard deviation, minimum, maximum, etc.
  - `shape` : Displays the number of rows and columns in the dataset.
- `seaborn` : Used for high-level visualization, making it easier to create attractive plots.

python

```
; np         # NumPy for numerical operations and handling arrays
is pd        # Pandas for data manipulation (loading, cleaning, analyzing)
.ib.pyplot as plt  # Matplotlib for creating plots
.ine          # Ensures plots are displayed inline in Jupyter notebooks
 as sns      # Seaborn for high-level data visualization
:             # For suppressing warnings during execution
warnings('ignore')  # Ignore warnings that might clutter output

v('tiatanic.csv')  # Loads Titanic dataset into a Pandas DataFrame
                    # Displays basic info about the dataset (e.g., column names, non-null counts)
                    # Shows summary statistics for numerical columns (mean, std, min, max)
                    # Returns the number of rows and columns in the dataset (shape of the DataFrame)

iowing distribution of the 'Fare' feature
'Fare', data=df)     # Plots the distribution of the 'Fare' column

 figure size for subsequent plots
gure.figsize':(3,3)})   # Smaller figure size for better layout

 of 'Age' with 70 bins
Age', data=df, bins=70)  # Creates a histogram of 'Age' with custom bins

igure size for the next plot
gure.figsize':(5,5)})   # Sets the figure size to 5x5 inches for the next plots

mber of survivors, split by passenger class
Survived', data=df, kind='count', hue='Pclass')  # Shows survival count by passenger class

mber of survivors, split by sex
Survived', data=df, kind='count', hue='Sex')  # Shows survival count by sex (male/female)
```

Detailed Explanation:

- `pandas` : A powerful library for data manipulation; it handles tasks like reading CSV files, cleaning data, and performing statistical analysis.
  - `read_csv()` : Reads data from a CSV file into a DataFrame for easier analysis.
  - `info()` : Provides an overview of the dataset, such as column names, data types, and non-null counts.
  - `describe()` : Provides summary statistics for numerical columns like mean, standard deviation, minimum, maximum, etc.
  - `shape` : Displays the number of rows and columns in the dataset.
- `seaborn` : Used for high-level visualization, making it easier to create attractive plots.

- `seaborn`: Used for high-level visualization, making it easier to create attractive plots.
    - `histplot()`: Creates a histogram to visualize the distribution of a numerical feature (e.g., 'Fare').
    - `displot()`: Similar to `histplot()`, but offers more customization options like bin sizes.
    - `catplot()`: Used for categorical plots. Here, it's used to visualize survival counts split by `Pclass` (passenger class) and `Sex` (gender).
- `matplotlib`: Provides control over plot appearance, including figure size.
    - `set(rc={'figure.figsize':(x,y)})`: Changes the default figure size to make plots more readable or fit better on screen.
- Histograms help in understanding the distribution of continuous variables (like `Fare` and `Age`).
- `catplot()` is useful to visualize the count of different categories in categorical variables (like survival rate by class or sex).

python Copy Edit

```python
import pandas as pd        # Import pandas for data manipulation (loading, cleaning, analyzing)
import numpy as np         # Import numpy for numerical operations
import seaborn as sns      # Import seaborn for statistical visualizations (e.g., violin plot)

# Loading the loan dataset into a DataFrame
data = pd.read_csv("loan_data.csv")
data                        # Display the DataFrame for inspection

# Basic Data Information
data.info()                 # Displays basic info about the dataset (column names, non-null counts
data.describe()             # Generates summary statistics for numerical columns (mean, std, min,
data.isnull().sum()         # Checks the number of missing values in each column

# Mean calculations (average values for numerical columns)
mean = data.mean(numeric_only=True)  # Calculate mean for numerical columns
mean                                 # Display the mean values
data['LoanAmount'].mean()            # Mean of the 'LoanAmount' column
data['Loan_Amount_Term'].mean()      # Mean of the 'Loan_Amount_Term' column

# Median calculations (central tendency of numerical columns)
median = data.median(numeric_only=True)  # Calculate the median for numerical columns
median                                   # Display the median values
data['Age'].median()                     # Median of the 'Age' column

# Minimum and Maximum Values
minimum = data.min(numeric_only=True)  # Find the minimum values for each column
minimum                                # Display the minimum values
maximum = data.max(numeric_only=True)  # Find the maximum values for each column
maximum                                # Display the maximum values

# Standard Deviation (spread of data)
std = data.std(numeric_only=True)     # Calculate the standard deviation for numerical columns
std                                    # Display standard deviations
data['Age'].std()                      # Standard deviation of the 'Age' column

# Grouping data by a categorical feature ('Age') and counting occurrences
data.groupby('Age').count()            # Group by 'Age' and count occurrences for each group

# Loading another dataset (Iris dataset) for analysis
data = pd.read_csv("iris.csv")  # Read the Iris dataset into a DataFrame
data                             # Display the DataFrame for inspection

# Grouping data in the Iris dataset by 'Species' and calculating statistics
data.groupby('Species').count()  # Count the number of entries for each species
data.groupby('Species').mean()   # Calculate the mean for each species' numerical columns
data.Species.mode()               # Find the mode (most frequent value) of the 'Species' column

# Standard Deviation for Iris dataset features
data.SepalWidthCm.std()           # Standard deviation of SepalWidthCm
data.SepalLengthCm.std()          # Standard deviation of SepalLengthCm
```

```python
# Standard Deviation for Iris dataset features
data.SepalWidthCm.std()            # Standard deviation of SepalWidthCm
data.SepalLengthCm.std()           # Standard deviation of SepalLengthCm

# Skewness of the data (measuring asymmetry of distribution)
data.skew()                        # Calculate skewness for numerical features

# Visualizing data using a violin plot (distribution of SepalWidthCm across Species)
sns.violinplot(x="SepalWidthCm", y="Species", data=data)  # Violin plot for 'SepalWidthCm' by spe
```

Detailed Explanation:

- `pandas`: Used for handling and manipulating tabular data (like reading and cleaning datasets).
  - `read_csv()`: Reads data from a CSV file into a Pandas DataFrame.
  - `info()`: Shows data types, non-null counts, and general info about the DataFrame.
  - `describe()`: Summarizes numerical columns (mean, std deviation, min, max, etc.).
  - `isnull().sum()`: Checks for missing data by column.
  - `mean()`, `median()`, `min()`, `max()`: Calculate the mean, median, minimum, and maximum values for the data.
  - `std()`: Calculates the standard deviation, which indicates the spread of the data.
  - `groupby()`: Groups data by a categorical column (like 'Age' or 'Species') and aggregates it (count or mean).
- `numpy`: Provides support for large, multi-dimensional arrays and matrices, along with a collection of mathematical functions to operate on these arrays.
- `seaborn`: Built on Matplotlib, it is used to make attractive and informative statistical graphics.
  - `violinplot()`: Combines aspects of boxplots and kernel density plots, showing the distribution of a numerical variable across different categories (in this case, SepalWidthCm across Species).
- `skew()`: Calculates the skewness of a dataset, which measures the asymmetry of the distribution. Positive skew means the data is skewed to the right, and negative skew means it's skewed to the left.

Here's a compact line-by-line explanation with unique/technical terms and no line spacing, ideal for screenshotting:

python

```
import numpy as np#imports NumPy for numerical operations like arrays and matrices
import pandas as pd#imports pandas for data handling using DataFrame structure
import matplotlib.pyplot as plt#imports matplotlib for data visualization
from sklearn.model_selection import train_test_split#imports function to split data into train/te
from sklearn.linear_model import LogisticRegression#imports logistic regression algorithm
from sklearn.preprocessing import StandardScaler#imports standardization tool for feature scaling
from sklearn.metrics import confusion_matrix,ConfusionMatrixDisplay,classification_report,accurac

data = pd.read_csv('Social_Network_Ads.csv')#loads CSV dataset into a DataFrame called data
data.head(5)#displays first 5 rows to understand basic structure
data.info()#shows data types, non-null counts, and memory usage
data.describe()#returns statistical summary (mean, std, etc.) of numeric columns
data.isnull().sum()#checks for missing (null) values in each column
data.shape#shows dataset dimensions (rows, columns)
x = data.iloc[:,2:4]#selects columns at index 2 and 3 (Age, EstimatedSalary) as features
y = data.iloc[:,4]#selects column at index 4 (Purchased) as target label
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.25, random_state=42)#splits
scale = StandardScaler()#creates scaler object to normalize features
x_train = scale.fit_transform(x_train)#fits scaler on training data and transforms it
x_test = scale.transform(x_test)#transforms test data using the same scaling parameters
lr = LogisticRegression(random_state = 0,solver = 'lbfgs')#initializes logistic regression model
lr.fit(x_train,y_train)#trains the model using scaled training data
pred = lr.predict(x_test)#predicts class labels for test features

print(x_test[:10])#prints first 10 rows of scaled test features
print('-'*15)#prints 15 dashes as separator
print(pred[:10])#prints first 10 predicted values
print('Expected Output:',pred[:10])#shows predicted output (same as above)
print('-'*15)
print('Predicted Output:\n',y_test[:10])#shows actual labels (ground truth) for comparison
matrix = confusion_matrix(y_test,pred,labels = lr.classes_)#computes confusion matrix using true
print(matrix)#prints confusion matrix

tp, fn, fp, tn = confusion_matrix(y_test,pred,labels=[1,0]).reshape(-1)#extracts TP, FN, FP, TN v
conf_matrix = ConfusionMatrixDisplay(confusion_matrix=matrix,display_labels=lr.classes_)#creates
conf_matrix.plot(cmap=plt.cm.Blues)#plots matrix with blue color gradient
plt.show()#displays the confusion matrix plot
print('\nAccuracy: {:.2f}'.format(accuracy_score(y_test,pred)))#prints accuracy score (correct pr
print('Error Rate: ',(fp+fn)/(tp+tn+fn+fp))#calculates and prints error rate
print('Sensitivity (Recall or True positive rate) :',tp/(tp+fn))#prints recall: ability to find a
print('Specificity (True negative rate) :',tn/(fp+tn))#prints specificity: ability to find all ne
print('Precision (Positive predictive value) :',tp/(tp+fp))#prints precision: how many predicted
print('Precision (False Positive Rate):',fp/(tn+fp))#prints false positive rate: proportion of ne
```

Let me know if you want an explanation in Hindi/Marathi or need a diagram or chart for presentation.

python

Copy Edit

```
trices
re

it data into train/test sets
ion algorithm
l for feature scaling
cation_report,accuracy_score, precision_score, recall_score, f1_score#imports various evaluation me

ame called data


umns


 as features

ndom_state=42)#splits dataset into 75% train and 25% test with fixed seed for reproducibility

sforms it
 parameters
tic regression model using 'lbfgs' optimizer and fixed random seed









for comparison
on matrix using true vs predicted labels


acts TP, FN, FP, TN values from matrix for class 1
lr.classes_)#creates visual display object for confusion matrix


acy score (correct predictions / total)

ll: ability to find all positive samples
bility to find all negative samples
: how many predicted positives are correct
ate: proportion of negatives incorrectly classified as positive
```

```
import numpy as np#imports NumPy for numerical computations, especially arrays
import pandas as pd#imports pandas for DataFrame creation and data analysis
import matplotlib.pyplot as plt#imports matplotlib for plotting graphs
import seaborn as sns#imports seaborn for enhanced data visualization

from sklearn.datasets import fetch_california_housing#imports built-in California housing dataset
from sklearn.model_selection import train_test_split#imports function to split data into training
from sklearn.linear_model import LinearRegression#imports linear regression algorithm
from sklearn.metrics import mean_squared_error#imports MSE metric for regression error calculatio

california = fetch_california_housing()#loads the California housing dataset
data = pd.DataFrame(california.data, columns=california.feature_names)#converts dataset into pand
data['MEDV'] = california.target#adds target column 'MEDV' (Median House Value)

print("Missing Values:\n", data.isnull().sum())#checks for null/missing values in dataset
sns.set(rc={'figure.figsize': (11.7, 8.27)})#sets default figure size for plots using seaborn
sns.histplot(data['MEDV'], bins=30, kde=True)#plots histogram with KDE for target variable distri
plt.title("Distribution of Target Variable")#sets title for histogram
plt.show()#displays the histogram plot

correlation_matrix = data.corr().round(2)#computes correlation matrix rounded to 2 decimal places
plt.figure(figsize=(10, 8))#sets figure size for heatmap
sns.heatmap(data=correlation_matrix, annot=True, cmap='coolwarm')#plots heatmap showing correlati
plt.title("Feature Correlation Heatmap")#sets title for heatmap
plt.show()#displays the heatmap

features = ["AveRooms", "AveOccup"]#selects two features: Average rooms and Average occupancy
target = data["MEDV"]#sets target variable as 'MEDV'

plt.figure(figsize=(10, 5))#sets figure size for scatter plots
for i, col in enumerate(features):#loops over selected features
    plt.subplot(1, len(features), i+1)#creates subplots for each feature
    plt.scatter(data[col], target, marker='o')#scatter plot of feature vs target
    plt.title(f"{col} vs MEDV")#sets plot title dynamically
    plt.xlabel(col)#x-axis label
    plt.ylabel("MEDV")#y-axis label

X = data[["AveRooms", "AveOccup"]]#sets independent variables (features)
Y = data["MEDV"]#sets dependent variable (target)

X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2, random_state=42)#splits

model = LinearRegression()#creates LinearRegression model object
model.fit(X_train, Y_train)#fits the model on training data

Y_pred = model.predict(X_test)#predicts target values for test features

mse = mean_squared_error(Y_test, Y_pred)#calculates Mean Squared Error
print(f"Mean Squared Error: {mse:.2f}")#prints MSE rounded to 2 decimals
```

Copy Edit

```
erical computations, especially arrays
ataFrame creation and data analysis
 matplotlib for plotting graphs
r enhanced data visualization

ornia_housing#imports built-in California housing dataset
n_test_split#imports function to split data into training and testing sets
egression#imports linear regression algorithm
_error#imports MSE metric for regression error calculation

loads the California housing dataset
umns=california.feature_names)#converts dataset into pandas DataFrame with column names
rget column 'MEDV' (Median House Value)

.sum())#checks for null/missing values in dataset
7)})#sets default figure size for plots using seaborn
True)#plots histogram with KDE for target variable distribution
le")#sets title for histogram


)#computes correlation matrix rounded to 2 decimal places
 size for heatmap
ot=True, cmap='coolwarm')#plots heatmap showing correlation between features and target
#sets title for heatmap


cts two features: Average rooms and Average occupancy
le as 'MEDV'

 size for scatter plots
 over selected features
reates subplots for each feature
r='o')#scatter plot of feature vs target
t title dynamically



 independent variables (features)
 (target)

_test_split(X, Y, test_size=0.2, random_state=42)#splits dataset into 80% train and 20% test sets

rRegression model object
el on training data

 target values for test features

#calculates Mean Squared Error
prints MSE rounded to 2 decimals
```

Copy Edit

```
import numpy as np#imports NumPy for numerical operations
import pandas as pd#imports pandas for structured data manipulation
from sklearn.model_selection import train_test_split#imports data split function for training/tes
from sklearn.naive_bayes import GaussianNB#imports Gaussian Naive Bayes classifier
import matplotlib.pyplot as plt#imports matplotlib for plotting
import seaborn as sns#imports seaborn for enhanced visualizations
from sklearn.metrics import confusion_matrix,ConfusionMatrixDisplay,classification_report,accurac
from sklearn.preprocessing import LabelEncoder#imports label encoder to convert categorical label

data = pd.read_csv('Iris.csv')#loads Iris dataset from CSV file
data.head(5)#displays first 5 rows of the dataset
data.describe(include='all')#shows statistical summary for all columns including object type
data.info()#displays data types, nulls, and memory usage
print(data.shape)#prints number of rows and columns
data['Species'].unique()#displays unique class labels in Species column
data.isnull().sum()#checks for missing/null values

x = data.iloc[:,1:5]#selects feature columns (SepalLength, SepalWidth, PetalLength, PetalWidth)
y = data.iloc[:,5:]#selects target column 'Species'
encode = LabelEncoder()#creates label encoder object to convert string labels to integers
y = encode.fit_transform(y)#applies encoding and converts y to 1D array

x_train,x_test,y_train,y_test = train_test_split(x,y,test_size=0.3,random_state=0)#splits data in

naive_bayes = GaussianNB()#initializes Gaussian Naive Bayes model
naive_bayes.fit(x_train,y_train)#fits/trains the model on training data
pred = naive_bayes.predict(x_test)#predicts class labels on test data

pred#shows predicted output values
y_test#shows actual target values
matrix = confusion_matrix(y_test,pred,labels=naive_bayes.classes_)#creates confusion matrix compa
print(matrix)#prints confusion matrix

tp, fn, fp, tn = confusion_matrix(y_test,pred,labels=[1,0]).reshape(-1)#extracts TP, FN, FP, TN f

conf_matrix = ConfusionMatrixDisplay(confusion_matrix=matrix,display_labels=naive_bayes.classes_)
conf_matrix.plot(cmap=plt.cm.YlGn)#plots the matrix with Yellow-Green colormap
plt.show()#displays the confusion matrix plot

print(classification_report(y_test,pred))#prints precision, recall, f1-score for each class
print('\nAccuracy: {:.2f}'.format(accuracy_score(y_test,pred)))#prints overall accuracy (correct
print('Error Rate: ',(fp+fn)/(tp+tn+fn+fp))#prints error rate
print('Sensitivity (Recall or True positive rate) :',tp/(tp+fn))#recall for class 1 (TPR)
print('Specificity (True negative rate) :',tn/(fp+tn))#TNR: correctly identified negatives
print('Precision (Positive predictive value) :',tp/(tp+fp))#how many predicted positives are actu
print('False Positive Rate :',fp/(tn+fp))#proportion of actual negatives incorrectly labeled as p
```

⚠️ Note: `tp, fn, fp, tn` line is only valid for binary classification. Iris is multi-class, so this line may throw an error unless the model or labels are filtered. Let me know if you want to fix or adjust that for multi-class metrics.

python

Copy Edit

```
cal operations
ctured data manipulation
est_split#imports data split function for training/testing
imports Gaussian Naive Bayes classifier
plotlib for plotting
nhanced visualizations
x,ConfusionMatrixDisplay,classification_report,accuracy_score, precision_score, recall_score, f1_sco
der#imports label encoder to convert categorical labels to numbers

ataset from CSV file
ataset
cal summary for all columns including object type
 memory usage
 columns
ass labels in Species column
 values

s (SepalLength, SepalWidth, PetalLength, PetalWidth)
Species'
er object to convert string labels to integers
g and converts y to 1D array

split(x,y,test_size=0.3,random_state=0)#splits data into 70% training and 30% testing

sian Naive Bayes model
s the model on training data
 class labels on test data



s=naive_bayes.classes_)#creates confusion matrix comparing true vs predicted labels


ed,labels=[1,0]).reshape(-1)#extracts TP, FN, FP, TN from confusion matrix for binary comparison

ion_matrix=matrix,display_labels=naive_bayes.classes_)#sets up confusion matrix display
e matrix with Yellow-Green colormap
ot

rints precision, recall, f1-score for each class
score(y_test,pred)))#prints overall accuracy (correct predictions / total)
#prints error rate
 rate) :',tp/(tp+fn))#recall for class 1 (TPR)
tn/(fp+tn))#TNR: correctly identified negatives
) :',tp/(tp+fp))#how many predicted positives are actually correct
roportion of actual negatives incorrectly labeled as positive
```

⚠ Note: `tp, fn, fp, tn` line is only valid for binary classification. Iris is multi-class, so this line may throw an error unless the model or labels are filtered. Let me know if you want to fix or adjust that for multi-class metrics.