Practical 1:

Data wranling1:
Here we have to process data
Here dataset used the excel file
Execute the normal commands : shape , head , tail , describe , info , dtypes

Check the if entries are null isnull and isnnull().sum()

Df.columns
Create a list of the imp columns take the desired columns from the original df as string
Df[imp_colums]

The use the StandardScaler on this imp columns
From sklearn.preprocessing import StandardScaler
Scaler = StandardScaler()
df[imp_columns]=scaler.fit_transform(df[imp_columns])

Use the labelencoder
from sklearn.preprocessing import LabelEncoder
encoding_list = ['Branch','Gender','Board[10th]','Board[12th]','Category']
df[encoding_list] = df[encoding_list].apply(LabelEncoder().fit_transform)

Or
For col in encoding_list :
    Le = LabelEncoder()
    Df[col] = Le.fit_transform(df[col])

StandardScaler : transforma the values suuch that the mean becomes 0 and the standard deviation becomes 1

`LabelEncoder` is a tool used to **convert categorical labels** (strings or text values) into **numeric values**, usually for use in machine learning models that only accept numbers.

Practical 2 : data wrangling 2:

Here we have to create a manual dataset:

import pandas as pd

import numpy as np

df=pd.DataFrame()

df['Rollo']=[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

df['Maths']=[66, 85, 78, 60, 45, 56, 70, np.nan, 80, 110]

df['Science']=[90, 83, 46, 78, 84, 57, 68, 43, 67, 58]

df['English']=[79, 83, 57, 66, 49, 87, 73, 69, 52, 68]

df['Attendance']=[90, 80, 74, 86, '93%', 88, 69, 77, 95, 96]

Use normal commands : df.info() df.describe() df.isnull().sum()

Replace the null values with the mean of that columns
df['Maths'].fillna(df['Maths'].mean(),inplace=True)

Converting the string data from attendance into number:
df['Attendance']=pd.to_numeric(df['Attendance'], errors='coerce')

invalid parsing (string value) will be NaN

df['Attendance'].fillna(df['Attendance'].mean(), inplace=True)

To get outliers:

import seaborn as sns

sns.boxplot(y=df['Maths'])

sns.boxplot(y=df['English'])

Values out of the range 0-100 will marked as the outliers

Q1 = df['Maths'].quantile(0.25)

Q3 = df['Maths'].quantile(0.75)

IQR = Q3 - Q1

lower_bound = Q1 - 1.1 * IQR

upper_bound = Q3 + 1.1 * IQR

Replace the outliers with the upper bound and lower bound as needed

df['Maths'] = df['Maths'].clip(lower=lower_bound, upper=upper_bound)

In the end apply the minmaxscaler to attendance column
xscaled= x−xmin / xmax−xmin

After scaling the xmin becomes zerro and the xmax becomes one

Practical 3 A:

import pandas as pd

df=pd.DataFrame()

df['Age Group']=['Young Adult', 'Young Adult', 'Young Adult', 'Young Adult', 'Mid Age Adult', 'Mid Age Adult','Senior Adult', 'Senior Adu

df['Income']=[30000,35000,32000,33000, 50000,55000,75000,80000,78000,82000]

Normal commands : info() , describe

One of the goal of this practical is to group the numerical variables with respect to categorical variable like below income vs age-group

import seaborn as sns

import matplotlib.pyplot as plt

sns.boxplot(x='Age Group', y='Income', data=df)

plt.xlabel('Age Group')

plt.ylabel('Income')

plt.title('Age Group wise Box Plot of Income ')

plt.title('Age Group wise Box Plot of Income ')

Now we have to create a summery statistics wrt to age group

print(df.groupby('Age Group')['Income'].mean())

print(df.groupby('Age Group')['Income'].median())

Same for min , max also can use describe on this instead of these functions

Practical 3 B:

Display the basic statistical information for iris dataset

```
import pandas as pd

from sklearn import datasets

iris=datasets.load_iris()

df=pd.DataFrame(iris['data'])

df[4]=iris['target']

df.rename(columns={0:'SepalLengthcm', 1:'SepalWidthcm', 2:'PetalLengthcm', 3:'PetalWidthcm', 4:'Species'}, inplace=True)
```

Info and describe

Get four box plots of each sepallength petallength spealwidth and petallength wrt species

```
import seaborn as sns

import matplotlib.pyplot as plt

# Create the boxplot

sns.boxplot(x='Species', y='SepalLengthcm', data=df)

plt.xlabel("Species")

plt.ylabel("Sepal Length (cm)")

plt.title("Species-wise Boxplot of Sepal Length")

df.mean()

df.groupby(['Species']).mean()

df.median()

df.groupby(['Species']).median()

df.groupby(['Species']).count()
```

df.SepalLengthcm.std()

0.8280661279778629

df.SepalWidthcm.std()

0.435866284936698

df.PetalLengthcm.std()

1.7652982332594667

df.PetalWidthcm.std()

0.7622376689603465

df.quantile(0.5) do this for 0.25 and 0.75

Practical 4 : linear regression

Dataset provided separately

There are some nul values replace them with the mean values

df.fillna(df.mean()) all the columns wise null values are replaced

Rename colum ‘medv’ to ‘price’

df.rename(columns={'MEDV':'PRICE'}, inplace=True)

Y = df[‘price’]

x=df.drop('PRICE',axis=1)

from sklearn.model_selection import train_test_split

xtrain, xtest, ytrain, ytest=train_test_split(x,y,test_size=0.2, random_state=0)

From sklearn.linear_model import LinearReression

Regressor = LinearRegression()

regressor.fit(xtrain , train)

Ypred = regressor.predict(xtest)

From sklearn.metrics import mean_squeared_error , mean_absolute_error

Mse = mean_squared_error(ytest , ypred)

Mae = mean_absolute_error(ytest , ypred)

From sklearn.metrics import r2_score

R2 = r2_score(ytest , ypred)

Practical 5 : logistic regression

Data separately given called Social_media_ads.csv

Perform the normal data preprocessing steps

Drop the user id column it is not relevant

df.drop([‘user id’] , axis = 1 , inplace=True)

Labelencode the gender column

from sklearn.preprocessing import LabelEncoder

label_encoder = LabelEncoder()

df['Gender'] = label_encoder.fit_transform(df['Gender'])

Separate the x and y

x=df[['Gender','Age','EstimatedSalary']]

y=df['Purchased']

Train test split

from sklearn.model_selection import train_test_split

xtrain,xtest, ytrain, ytest=train_test_split(x,y,test_size=0.2,random_state=42)

Train the model

from sklearn.linear_model import LogisticRegression

model=LogisticRegression()

model.fit(xtrain,ytrain)

ypred=model.predict(xtest)

Accuracy score

from sklearn.metrics import accuracy_score

accuracy=accuracy_score(ytest, ypred)

Accuracy

Confusion matrix

from sklearn.metrics import confusion_matrix

cm=confusion_matrix(ytest,ypred)

TN, FP, FN, TP = cm.ravel()

from sklearn.metrics import classification_report

report=classification_report(ytest, ypred)

print(report)

1. Precision – *"How precise are your positive predictions?"*

Precision=TPTP+FP\text{Precision} = \frac{TP}{TP + FP}Precision=TP+FPTP

🧠 Intuition:

- Of all the times you **predicted positive**, how many were **actually correct**?
- Measures **trustworthiness** of your positive predictions.

🎨 Analogy:

Imagine a **spam filter** that flags emails as spam.

- Precision = "Of all the emails marked as spam, how many were really spam?"

A low precision means many legit emails were marked as spam (false positives).

🎯 2. Recall – *"How well do you catch all the actual positives?"*

Recall=TPTP+FN\text{Recall} = \frac{TP}{TP + FN}Recall=TP+FNTP

🧠 Intuition:

- Of all the **actual positives**, how many did you **correctly catch**?
- Measures **completeness** of your positive detection.

🎨 Analogy:

Now imagine a **medical test** for detecting cancer.

- Recall = "Of all the people who actually had cancer, how many did the test catch?"

A low recall means many real cases were missed (false negatives).

🎯 3. F1-Score – *"Balance between precision and recall"*

F1=2⋅Precision⋅RecallPrecision+Recall\text{F1} = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}F1=2⋅Precision+RecallPrecision⋅Recall

🧠 Intuition:

- F1 is the **harmonic mean** of precision and recall.
- It balances the **trade-off** between catching positives and avoiding false alarms.

🎨 Analogy:

In fraud detection:

- High precision = few false fraud alerts
- High recall = catching most actual frauds
- F1 = good if your model avoids false alarms **and** doesn't miss real frauds

Practical 8 : data visualization 1

No explicite dataset sdd using the seaborn
import seaborn as sns

df=sns.load_dataset("titanic")

Perform normal operations

Take only those columns having numerical data
And find the correlation between them

numeric_df = df.select_dtypes(include=["number"])

corr_matrix = numeric_df.corr()

corr_matrix

Show countplot for all the classes

sns.countplot(x='pclass', hue='survived', data=df)

plt.title('Survival Count by Passenger Class')

plt.xlabel('Passenger Class')

plt.ylabel('Survival Count')

plt.show()

Plot a barplot of each survival of each gender

sns.barplot(x="sex", y="survived", data=df)

plt.title("Survival Rate by Gender")

plt.xlabel("Gender")

plt.ylabel("Survival Rate")

plt.show()

Draw a histogram for age

```
sns.histplot(x='age', hue='survived', data=df, multiple="stack")


plt.title('Survival Count by Age')

plt.xlabel('Age')

plt.ylabel('Survival Count')

plt.show()


plt.figure(figsize=(8, 5))

sns.barplot(x="embarked", y="survived", data=df, ci=None, palette="coolwarm")

plt.title("Survival Rate by Embarkation Port")

plt.xlabel("Port of Embarkation")

plt.ylabel("Survival Rate")

plt.show()

sns.boxplot(x="survived", y="fare", data=df, palette="coolwarm")

plt.title("Fare Distribution for Survivors and Non-Survivors")

plt.xlabel("Survival Status (0 = No, 1 = Yes)")

plt.ylabel("Fare Price")

plt.show()

sns.barplot(x="sibsp", y="survived", data=df, ci=None, palette="coolwarm")

plt.title("Survival Rate by Number of Siblings/Spouses Aboard")

plt.xlabel("Number of Siblings/Spouses Aboard")

plt.ylabel("Survival Rate")

plt.show()
```

```
sns.barplot(x="deck", y="survived", data=df, ci=None, palette="coolwarm",
order=df["deck"].value_counts().index)

plt.title("Survival Rate by Deck")

plt.xlabel("Deck")

plt.ylabel("Survival Rate")

plt.show()
```

```
# Write a code to check how the price of the ticket (column name: 'fare') for each passenger

# is distributed by plotting a histogram.

plt.figure(figsize=(8, 5))

sns.histplot(df, x="fare", hue="pclass", bins=30, kde=True, palette="coolwarm")

plt.title("Distribution of Fare by Passenger Class")

plt.xlabel("Fare Price")

plt.ylabel("Number of Passengers")

plt.xlim(0, 300) # Excluding extreme outliers for better visualization

plt.show()
```

Practical A9 :

Plot a box plot for distribution of age with respect to each gender along

with the information about whether they survived or not. (Column names : 'sex' and 'age')

```
plt.figure(figsize=(8, 6))

sns.boxplot(x="sex", y="age", hue="survived", data=df, palette="coolwarm")

# Labels and title

plt.xlabel("Gender")

plt.ylabel("Age")

plt.title("Age Distribution by Gender and Survival Status")

plt.legend(title="Survived", labels=["No (0)", "Yes (1)"])
```

Practical 10 : here the iris.csv is need separately

After getting the dataframe
Start by normal commands

Draw the histogram o each feature

```
plt.hist(df['SepalLengthCm'], bins=20) # Adjust the number of bins as needed

plt.title(f'Histogram of SepalLengthCm')

plt.xlabel('SepalLengthCm')

plt.ylabel('Frequency')
```

Repeat this for all four feature and get the four histograms

Draw boxplot of each feature

```
plt.figure(figsize=(10, 6)) # Adjust figure size as needed


df.boxplot(column=['SepalLengthCm', 'SepalWidthCm', 'PetalLengthCm', 'PetalWidthCm'])

plt.title('Boxplots of Iris Features')

plt.ylabel('Cm')

plt.show()
```

Identify the outliers

```
# Calculate Q1, Q3, and IQR

Q1 = df['SepalWidthCm'].quantile(0.25)

Q3 = df['SepalWidthCm'].quantile(0.75)
```

```
IQR = Q3 - Q1

# Define bounds for outliers

lower_bound = Q1 - 1.5 * IQR

upper_bound = Q3 + 1.5 * IQR


outliers = df[(df['SepalWidthCm'] < lower_bound) | (df['SepalWidthCm'] > upper_bound)]



# Lets do a comparitive analysis of all species on PetalWidthCm

import seaborn as sns

sns.boxplot(x='Species', y='PetalWidthCm', data=df)

plt.title('Species-wise Boxplot of PetalWidthCm')

# Draw Specieswise Boxplot for PetalWidthCm

plt.show()


sns.histplot(data=df, x='PetalWidthCm', hue='Species', bins=10, kde=False)

plt.title('Histogram of Petal Width by Species')

plt.xlabel('Petal Width (cm)')

plt.ylabel('Frequency')

plt.show()
```