PR1 DW1 Perform the following operations using Python on any opensource dataset

```
import pandas as pd
import numpy as np
~data={'NAME':['jay','prince','nisha','neha','raj','riya'],
     'Age':[17,20,21,18,19,25],
     'Gender':['M','M','F','F','M','M'],
     'Marks':[90,75,80,84,71,86]    }
~df = pd.DataFrame(data)
~print("Original Data:")
  print(df)
~data1 = {
    'NAME': ['jay', 'prince', 'nisha', 'neha', 'raj', 'riya'],
    'Age': [17, 20, 21, 18, 19, 25],
    'Gender': ['M', 'M', 'F', 'F', 'M', 'M'],
    'Marks': [90, 75, np.nan, 84, np.nan, 86]  # Proper missing values }
~df2 = pd.DataFrame(data1)
  print("\nData with Missing Marks:")
  print(df2)
~df['Gender'] = df['Gender'].map({'M': 0, 'F': 1}).astype(float)
  df = df[df['Marks'] > 75]
  df = df.drop(['Age'], axis=1)
  print("\nCleaned Data (Marks > 75, Gender M/F mapped to 0/1):")
  print(df)
```

or

```
df = pd.read_csv("iris.csv")
print(df.head())
print("\nMissing Values in Each Column:")
print(df.isnull().sum())
print("\nData Description:")
print(df.describe())
df['species'] = df['species'].map({'setosa': 0, 'versicolor': 1, 'virginica': 2})
df = df[df['petal_length'] > 1.5]
df = df.drop(['sepal_width'], axis=1)
print("\nCleaned Dataset:")
print(df.head())
```

PR2 DW2 Create an “Academic performance” dataset of students

```
~import pandas as pd
  import numpy as np
~data_frame=pd.read_csv("Academic_performace.csv")
  print(data_frame)
~print(data_frame.head())
~print(data_frame.tail())
~print(data_frame.describe())
~print(data_frame.info())
~print(data_frame.shape)
~print(data_frame.isnull().any().any())
~print(data_frame.isnull().sum())

~import seaborn as sns
 import matplotlib.pyplot as plt
 from scipy import stats
 x=data_frame['Sno']
 y=data_frame['AnnouncementsView']
 sns.regplot(x='Sno', y='AnnouncementsView',data=data_frame)
 sns.boxplot(x=data_frame['AnnouncementsView'])
 z=np.abs(stats.zscore(data_frame['AnnouncementsView']))
 print(z)
 threshold=3
 print(np.where(z>3))

~df=pd.DataFrame({
    'Income':[15000,1000,120000,10000],
    'Age':[25,18,42,51],
    'Department':['HR','Legal','Marketing','Management']})
print(df)
df_scaled=df.copy()
col_names=['Income','Age']
features=df_scaled[col_names]
```

PR3 Descriptive Statistics - Measures of Central Tendency and variability

```
~import pandas as pd
 import numpy as np
 import statistics as st
~data_frame = pd.read_csv("/home/ubuntu/DSBDA/test_AV3.csv")
~print(data_frame)
~print(data_frame.info())
~print(data_frame.mean)
~print(data_frame['ApplicantIncome'].mean())
~print(data_frame['ApplicantIncome'].median())
~print(data_frame.mode())
~print(data_frame['ApplicantIncome'].mode())
~print(data_frame.median)
~print(data_frame['ApplicantIncome'].std())
~print(data_frame['LoanAmount'].std())
~print(data_frame['LoanAmount'].var())

~data_frame = pd.read_csv("iris.csv")
~print(data_frame)
~print(data_frame.info())
~setosa = data_frame['species'] == 'setosa'
print(data_frame[setosa].describe())

~virginica = data_frame['species'] == 'virginica'
  print(data_frame[virginica].describe())
~versicolor = data_frame['species'] == 'versicolor'
  print(data_frame[versicolor].describe())

~print('\nIris-versicolor')
  ver = data_frame['species'] == 'versicolor'
  print(data_frame[ver].describe())
~print('\nIris-virginica')
  virg = data_frame['species'] == 'virginica'
  print(data_frame[virg].describe())
grouped = data_frame.groupby('species')
for name, group in grouped:
    print(group.select_dtypes(include=np.number).values.tolist())
```

PR4 Linear RegressionModel

```
import pandas as pd
import numpy as np
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error

df = pd.read_csv("Boston.csv")

print(df.head())
print(df.describe())

X = df.drop("MEDV", axis=1)
y = df["MEDV"]

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33,
random_state=42)

model = LinearRegression()
model.fit(X_train, y_train)

y_pred = model.predict(X_test)
print("Model Coefficients:", model.coef_)

print("\nFirst 10 Predicted vs Actual Prices:")
for i in range(10):
    print(f"Predicted: {y_pred[i]}, Actual: {y_test.iloc[i]}")

mse = mean_squared_error(y_test, y_pred)
rmse = np.sqrt(mse)
print(f"\nMean Squared Error (MSE): {mse}")
print(f"Root Mean Squared Error (RMSE): {rmse}")
```

PR5 logistic regression

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import (
   confusion_matrix,
   accuracy_score,
   precision_score,
   recall_score,
   f1_score
)
dataset = pd.read_csv('Social_Network_Ads.csv')

print(dataset.head())
print(dataset.tail())

X = dataset[['Age', 'EstimatedSalary']]
Y = dataset['Purchased']

X_train, X_test, Y_train, Y_test = train_test_split(
   X, Y, test_size=0.25, random_state=0
)

scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

classifier = LogisticRegression(random_state=0)
classifier.fit(X_train, Y_train)

Y_pred = classifier.predict(X_test)
cm = confusion_matrix(Y_test, Y_pred)
```

```
tp = cm[1, 1]
fp = cm[0, 1]
tn = cm[0, 0]
fn = cm[1, 0]

accuracy = accuracy_score(Y_test, Y_pred)
error_rate = 1 - accuracy
precision = precision_score(Y_test, Y_pred)
recall = recall_score(Y_test, Y_pred)
f1 = f1_score(Y_test, Y_pred)

print("Confusion Matrix:")
print(cm)
print(f"True Positive: {tp}")
print(f"False Positive: {fp}")
print(f"True Negative: {tn}")
print(f"False Negative: {fn}")
print(f"Accuracy: {accuracy:.4f}")
print(f"Error Rate: {error_rate:.4f}")
print(f"Precision: {precision:.4f}")
print(f"Recall: {recall:.4f}")
print(f"F1 Score: {f1:.4f}")

plt.figure(figsize=(6, 6))
sns.heatmap(
    cm,
    annot=True,
    fmt="d",
    cmap="Blues",
    xticklabels=["Predict No", "Predict Yes"],
    yticklabels=["Actual No", "Actual Yes"]
)
plt.title("Confusion Matrix")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.show()
```

PR6 Naïve Bayes classificationalgorithm

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import confusion_matrix, accuracy_score, precision_score,
recall_score,f1_score

data = pd.read_csv('iris.csv')
X = data.drop('species', axis=1)
y = data['species']

print("Value of X=",X)
print( "Value of Y=",y)

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

print("X_train shape:", X_train.shape)
print("y_train shape:", y_train.shape)

model= GaussianNB()
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
cm = confusion_matrix(y_test, y_pred)
print("Confusion Matrix:\n", cm)

TP = cm[0, 0]  # True positives (class 0)
FP = cm[0, 1]  # False positives (class 1)
TN = cm[1, 1]  # True negatives (class 1)
FN = cm[1, 0]

accuracy = accuracy_score(y_test, y_pred)
error_rate = 1 - accuracy
precision = precision_score(y_test, y_pred, average='macro')
recall = recall_score(y_test, y_pred, average='macro')
f1 = f1_score(y_test, y_pred, average='macro')

print(f"Accuracy: {accuracy:.4f}")
print(f"Error Rate: {error_rate:.4f}")
print(f"Precision: {precision:.4f}")
print(f"Recall: {recall:.4f}")
print(f"F1 Score: {f1:.4f}")
```

PR7 Tokenization ,POSTagging, stopwords removal

```
import nltk
from nltk.tokenize import word_tokenize
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer, WordNetLemmatizer
from nltk import pos_tag
from sklearn.feature_extraction.text import TfidfVectorizer
import string
~nltk.download('punkt_tab')
  nltk.download('stopwords')
  nltk.download('wordnet')
 nltk.download('averaged_perceptron_tagger_eng')

document =""" Natural Langauage processing is a field of AI It aims to enable to
understand ,interpret and generate human langaueg """

tokens= word_tokenize(document)
pos_tags = pos_tag(tokens)
stop_words = set(stopwords.words('english'))
filtered_tokens=[word for word in tokens if word.lower() not in stop_words and word
not in
string.punctuation]
stemmer =PorterStemmer()
stemmed_tokens=[stemmer.stem(word) for word in filtered_tokens]
lemmatizer = WordNetLemmatizer()
lemmatized_tokens =[lemmatizer.lemmatize(word)for word in filtered_tokens]

print("\n original doc\n",document)
print("\n tokens\n", tokens)
print("\n pos tag\n",pos_tag)
print("\n filtered token after removed stop word removal\n", filtered_tokens)
print("\n stemmed tokens\n",stemmed_tokens)
print("\n Lemmetaized Tokens\n",lemmatized_tokens)

~corpus = [document]
  vectorizer = TfidfVectorizer()
  x = vectorizer.fit_transform(corpus)
  tfidf_matrix = x.toarray()
  terms = vectorizer.get_feature_names_out()
~print("\nTerm Frequency and Inverse Document Frequency (TF-IDF):")
 for i, term in enumerate(terms):
      print(f"Term: {term}, TF-IDF: {tfidf_matrix[0][i]:.4f}")
```

PR8 DV1 titanic 891rows

```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

titanic = sns.load_dataset('titanic')
print(titanic.head())

plt.title('Distribution of Fare Prices of Titanic Passengers')
plt.xlabel('Fare')
plt.ylabel('Density')
plt.show()

plt.figure(figsize=(10, 6))
sns.histplot(titanic['fare'], bins=30, kde=True, color='blue', stat='density')
plt.title('Distribution of Fare Prices of Titanic Passengers')
plt.xlabel('Fare')
plt.ylabel('Density')
plt.grid(True)
plt.show()
```

PR9 DV2 titanic Plotaboxplot

```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

titanic = sns.load_dataset('titanic')
titanic.head(10)

titanic.info()
titanic.describe()
titanic.loc[:,["survived","alive"]]

sns.boxplot(x='sex', y='age', data=titanic)
plt.show()

sns.boxplot(x='sex', y='age', hue='survived', data=titanic, palette="Set2")
```

PR10 Data Visualization III:- Download the Iris flower dataset

```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

df = pd.read_csv('iris.csv')

print("Features And Types in the iris dataset")
print (df.dtypes)

df.drop(columns='Species').hist(bins=20, figsize=(10,8),
color='skyblue',edgecolor='black')
plt.suptitle('Histogram of Each Feature',fontsize=16)
plt.tight_layout()
plt.show()

plt.figure(figsize=(10,8))
sns.boxplot(x='Species',y='Spetal_length',data=df)
sns.boxplot(x='Species',y='Spetal_length',data=df)
sns.boxplot(x='Species',y='Spetal_length',data=df)
sns.boxplot(x='Species',y='Spetal_length',data=df)
plt.tight_layout()
plt.show()

sns.pairplot(df,hue='Species')
plt.suptitle('pairplot of iris dataset',fontsize=16)
plt.tight_layout()
plt.show()
```

PR13 Scala

```
//checks if a number is positive, negative, or zero
object Number{
def main (args: Array[String]) {
var number = (-100);
if(number==0) {
println("number is zero");              }
else if(number>0) {
println("number is positive");                }
else { println("number is negative");               } }  }
```

```
//Print name
object Name {
def main (args: Array[String]) {
println("my name is name");
}  }
```

```
// concanate string
object String {
def main (args: Array[String]) {
var text: String = "Scala in programming langauge";
println("string is "+text);
} }
```

```
// compares two numbers
object Find_Number{
def main (args: Array[String]) {
var number1 = 20;
var number2 = 40;
if(number1>number2)  {
println("large number is "+number1);    }
else
{ println("large number is "+number2);          }    }   }
```

```
// addition of no
object Addition {
  def main(args: Array[String]) {
    val num1 = 10
    val num2 = 20
    val sum = num1 + num2
    println("The sum is: " + sum)  }     }
```