DATA SCIENCE AND APPLICATIONS (Lab Manual)

Experiment 1:- Demonstrate all the basic plots using Matplotlib package and python programming.

Problem statement:

Generate and display the following basic types of plots using Matplotlib

- Line Plot: Represent data points with lines connecting them.
- Scatter Plot: Show relationships between two variables using individual points.
- Bar Plot: Display data with rectangular bars, typically used for comparisons.
- Histogram: Visualize the distribution of a dataset.
- Pie Chart: Show proportions of categories as slices of a pie.

Program:

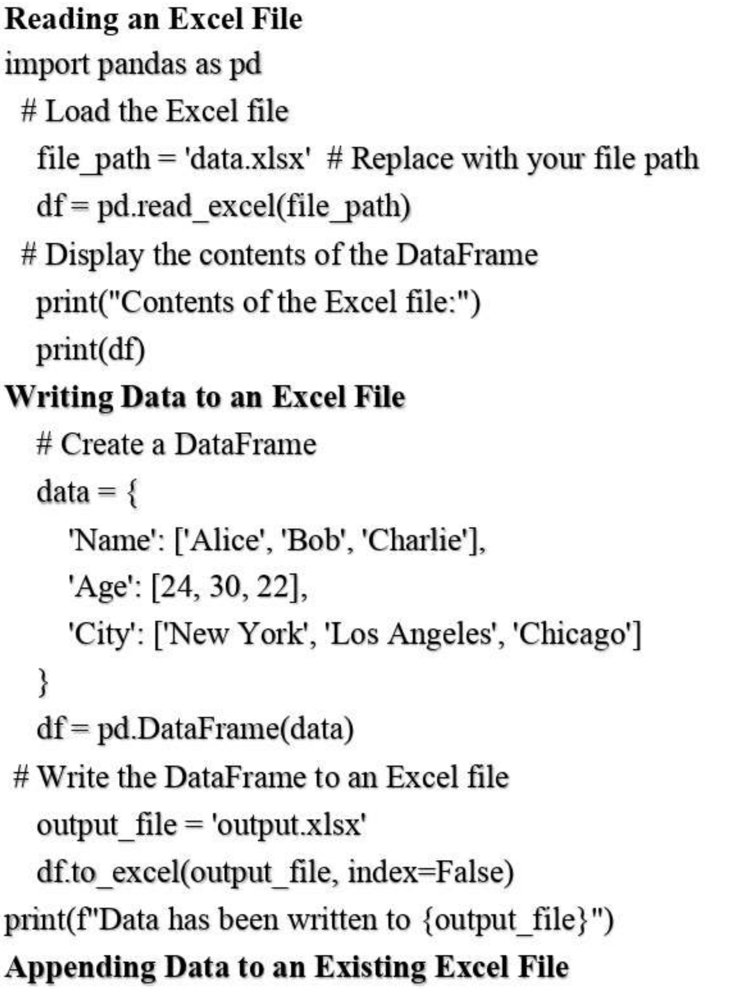
```
import matplotlib.pyplot as plt

import numpy as np
# Generate some data for plotting
x = np.linspace(0, 10, 100)
y = np.sin(x)
plt.figure()
plt.plot(x,y)
plt.title("Line Chart")
categories=['A','B','C','D']
values=[20,35,30,25]
 plt.figure()
plt.bar(categories,values)
plt.title("Bar Chart")


x=np.random.randn(100)
y=np.random.randn(100)
colors=np.random.rand(100)
sizes=100*np.random.rand(100)
 plt.figure()
plt.scatter(x,y,c=colors, s=sizes, alpha=0.5)
plt.title("Scatter Plot")

sizes = [30, 20, 25, 15, 10]
labels = ['A', 'B', 'C', 'D', 'E']
plt.figure()


plt.pie(sizes, labels=labels, autopct="%1.1f%%") plt.title("Pie Chart")
plt.show()
```

Experiment 2:- Implement a python program to perform File Operations on Excel Dataset.

Problem statement:

The objective of this program is to enable efficient reading, writing, and modification of Excel files, which is essential for data manipulation tasks such as data analysis, cleaning, and exporting data to Excel format.

Program:



Reading an Excel File

```
import pandas as pd
 # Load the Excel file
  file_path = 'data.xlsx'  # Replace with your file path
  df = pd.read_excel(file_path)
 # Display the contents of the DataFrame
  print("Contents of the Excel file:")
  print(df)
```

Writing Data to an Excel File

```
  # Create a DataFrame
  data = {
    'Name': ['Alice', 'Bob', 'Charlie'],
    'Age': [24, 30, 22],
    'City': ['New York', 'Los Angeles', 'Chicago']
  }
  df = pd.DataFrame(data)
 # Write the DataFrame to an Excel file
  output_file = 'output.xlsx'
  df.to_excel(output_file, index=False)
print(f"Data has been written to {output_file}")
```

Appending Data to an Existing Excel File

```
# Load the existing Excel file
  file_path = 'output.xlsx'  # Replace with your file path
  existing_df = pd.read_excel(file_path)
    new_data = {
    'Name': ['David', 'Eva'],
    'Age': [28, 26],
    'City': ['Houston', 'Phoenix']

  }

  new_df = pd.DataFrame(new_data)
  # Append new data to the existing DataFrame
```

```
    combined_df = pd.concat([existing_df, new_df], ignore_index=True)
    # Write the updated DataFrame back to the Excel file
    combined_df.to_excel(file_path, index=False)

    print(f"New data has been appended to {file_path}")
```

Filtering Data from an Excel File

```
    # Load the Excel file
    file_path = 'data.xlsx'  # Replace with your file path
    df = pd.read_excel(file_path)
    # Filter rows where Age is greater than 25
    filtered_df = df[df['Age'] > 25]

    # Display the filtered DataFrame
    print("Filtered data (Age > 25):")
    print(filtered_df)
```

Creating Charts from Excel Data

```
import matplotlib.pyplot as plt
    # Load the Excel file
    file_path = 'data.xlsx'  # Replace with your file path
    df = pd.read_excel(file_path)
    # Create a bar chart
    df.plot(kind='bar', x='Name', y='Age', title='Age of Individuals', legend=False)
    plt.ylabel('Age')
    plt.show()
```

Experiment3:- Write a python program to perform Array operations using the Numpy package.

Problem Statement:

To implement a Python program that demonstrates various array operations using the NumPy package. NumPy is a powerful library that enables efficient operations on arrays and matrices, making it essential for numerical computations and data manipulation tasks.

Program:

```
import numpy as np #
Create arrays
a = np.array([1, 2, 3, 4, 5])
b = np.array([6, 7, 8, 9, 10])

print("Array a", a)
print("Array b", b)
print("Sum of array a and b", np.add(a,b))
print("Difference of array a and b", np.subtract(a,b))
print("Product of arrays a and b", np.multiply(a,b))
print("Division of arrays a and b", np.divide(a,b))
print("Square root of array a:",np.sqrt(a))
print("Exponential of array a:",np.exp(a))
print("Minimum value of array a:",np.min(a))
print("Maximum value of array b:",np.max(b))
print("Mean of array a:",np.mean(a)) print("Standard
deviation of array b:",np.std(b)) print("Sum of
elements in array a:",np.sum(a))
c=np.array([[1,2],[3,4],[5,6]])
print("Array c:")
print(c)
print("Reshaped array c:")
print(np.reshape(c,(2,3)))

d=np.array([[1,2,3],[4,5,6]])
print("Array d:")
print(d)


print("Transposed array d:")
print(np.transpose(d))
```

Experiment 4:- Build an Artificial Neural Network by implementing the Backpropagation algorithm and test the same using appropriate data sets.

Problem Statement:

1. Implement an Artificial Neural Network (ANN) using a simple feedforward architecture.
2. Implement the Backpropagation algorithm to update the weights and biases based on the error between the predicted output and the true output.

Program:

```
import numpy as np
x=np.array(([2,9],[1,9],[3,6]),dtype=float)
y=np.array(([92],[86],[89]),dtype=float)
x=x/np.amax(x,axis=0)
y=y/100
def sigmoid(x):
return 1/(1+np.exp(-x))
def derivation_sigmoid(x): return
x*(1-x)
epoch=5000 lr=0.1
inputlayer_neurons=2
hiddenlayer_neurons=3
outputlayer_neurons=1
wb=np.random.uniform(size=(inputlayer_neurons,hiddenlayer_neurons))
bb=np.random.uniform(size=(1,hiddenlayer_neurons))
wout=np.random.uniform(size=(hiddenlayer_neurons,outputlayer_neurons))
bout=np.random.uniform(size=(1,outputlayer_neurons))
for i in range(epoch):
hinp1=np.dot(x,wb)
hinp=hinp1+bb hlayer_act=sigmoid(hinp)
outinp1=np.dot(hlayer_act,wout)
outinp=outinp1+bout
output=sigmoid(outinp)
EO=y-output outgrad=derivation_sigmoid(output)
d_output=EO*outgrad EH=d_output.dot(wout.T)
hiddengrad=derivation_sigmoid(hlayer_act)
d_hiddenlayer=EH*hiddengrad
wout+=hlayer_act.T.dot(d_output)*lr
wb+=x.T.dot(d_output)*lr
print("Inpput:\n" +str(x))
print("Actual:\n"+str(y))
print("Predicted:\n",output)
```

Experiment 5:- Demonstrate Linear Regression operation using python programming.

Problem Statement:

1. Implement Simple Linear Regression using pre-built machine learning libraries like scikit-learn).
2. Train the model using a dataset and make predictions based on input features.
3. Evaluate the model's performance using metrics such as Mean Squared Error (MSE) or R-squared (R^2).

Program:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt import
seaborn as sns
dataset = pd.read_csv('advertising.csv')
dataset.head(10)
dataset.shape
dataset.isna().sum()
dataset.duplicated().any()
fig, axs = plt.subplots(3, figsize = (5,5))
plt1 = sns.boxplot(dataset['TV'], ax = axs[0])
plt2 = sns.boxplot(dataset['Newspaper'], ax = axs[1])
plt3 = sns.boxplot(dataset['Radio'], ax = axs[2])
plt.tight_layout()
sns.distplot(dataset['Sales']);
sns.pairplot(dataset, x_vars=['TV', 'Radio', 'Newspaper'], y_vars='Sales', height=4, aspect=1,
kind='scatter')
plt.show()
sns.heatmap(dataset.corr(), annot = True)
plt.show()
from sklearn.model_selection
import train_test_split from sklearn.linear_model
import LinearRegression from sklearn
import metrics
x = dataset[['TV']]
y = dataset['Sales']
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size = 0.3, random_state = 100) slr=
LinearRegression()
slr.fit(x_train, y_train) print('Intercept: ',
slr.intercept_) print('Coefficient:',
slr.coef_)
print('Regression Equation: Sales = 6.948 + 0.054 * TV')
```

```
plt.scatter(x_train, y_train)
plt.plot(x_train, 6.948 + 0.054*x_train, 'r') plt.show()
#Prediction of Test and Training set result

y_pred_slr= slr.predict(x_test) x_pred_slr=
slr.predict(x_train)
print("Prediction for test set: {}".format(y_pred_slr))
slr_diff = pd.DataFrame({'Actual value': y_test, 'Predicted value': y_pred_slr})

slr_diff
#Predict for any value slr.predict([[56]])
# print the R-squared value for the model from
sklearn.metrics import accuracy_score
print('R squared value of the model: {:.2f}'.format(slr.score(x,y)*100))
```

Experiment 6:- Train a regularized logistic regression classifier on the in-build iris dataset using scikit- learn. Train the model and report the best classification accuracy.

Problem Statement:

To train a regularized logistic regression classifier on the Iris dataset using the Python scikit-learn library. The objective is to build a logistic regression model, apply regularization to improve its generalization ability, and then evaluate its performance by reporting the best classification accuracy.

Program:

```
# Importing the necessary libraries
 import numpy as np
import matplotlib.pyplot as plt import
pandas as pd
# Importing the dataset
dataset = pd.read_csv('iris.csv')
dataset.describe()
dataset.info()
# Splitting the dataset into the Training set and Test set
X = dataset.iloc[:, [0,1,2, 3]].values
y = dataset.iloc[:, 4].values
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.25, random_state = 0)
 # Feature Scaling
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
# Fitting Logistic Regression to the Training set
from sklearn.linear_model import LogisticRegression
classifier = LogisticRegression(random_state = 0, solver='lbfgs', multi_class='auto')
  classifier.fit(X_train, y_train)
  # Predicting the Test set results
  y_pred = classifier.predict(X_test)
  # Predict probabilities
  probs_y=classifier.predict_proba(X_test)
  probs_y = np.round(probs_y, 2)
  res = "{:<10} | {:<10} | {:<10} | {:<13} | {:<5}".format("y_test", "y_pred", "Setosa(%)",
  "versicolor(%)", "virginica(%)\n")
  res += "-"*65+"\n"
  res += "\n".join("{:<10} | {:<10} | {:<10} | {:<13} | {:<10}".format(x, y, a, b, c) for x, y, a, b,
  c in zip(y_test, y_pred, probs_y[:,0], probs_y[:,1], probs_y[:,2]))
  res += "\n"+"-"*65+"\n"
  print(res)
  # Making the Confusion Matrix
  from sklearn.metrics import confusion_matrix

  cm = confusion_matrix(y_test, y_pred)
  print(cm)

  # Plot confusion matrix
  import seaborn as sns
  import pandas as pd
  # confusion matrix sns heatmap


  ## https://www.kaggle.com/agungor2/various-confusion-matrix-plots
  ax = plt.axes()
  df_cm = cm
  sns.heatmap(df_cm, annot=True, annot_kws={"size": 30}, fmt='d',cmap="Blues", ax = ax )
  ax.set_title('Confusion Matrix')
plt.show()
```

Experiment 7:- Write a python program to perform Data Manipulation operations using Pandas package.

Problem Statement:

To perform various data manipulation operations using the Pandas library in Python. The operations should cover a range of typical tasks that are performed when working with structured data, such as:

1. Loading data from different file formats (CSV, Excel, etc.).
2. Cleaning and transforming the data.
3. Filtering, sorting, and aggregating the data.
4. Handling missing values.
5. Merging and joining datasets.

Program:

```
import pandas as pd
data={

        'Name':['John','Emma','Sant','Lisa','Tom'],
        'Age':[25,30,28,32,27],
         'Country':['USA','Canada','India','UK','Australia'],
         'Salary':[50000,60000,70000,80000,65000]
      }
df=pd.DataFrame(data)
print("Original DataFrame")
print(df)
name_age=df[['Name','Age']]
print("Original DataFrame") print(df)
name_age=df[['Name','Age']]
print("Name and Age columns")
print(name_age)
sorted_df=df.sort_values("Salary",ascending=False) print("\nsorted
DataFrame(by ssalary in descending order)") print(sorted_df)
average_Salary=df['Salary'].mean()
print("\nAverage salary",average_Salary)
df['Experience']=[3,6,4,8,5]
print("\nDataFrame with added experience")
print(df)
df.loc[df['Name']=='Emma','Salary']=65000
print("\nDataFrame with updating emma salary")
print(df)
df.drop('Experience',axis=1)
print("\nDataFrame after deleting the column ")
print(df)
```

Experiment 8. Develop a MapReduce program to find the grades of students in python.

Problem statement:

To develop a MapReduce program that computes grades for students based on their average marks in 4 subjects, and assigns grades in the following categories:

- A+ : Average >= 90
- A : 80 <= Average < 90
- B+ : 70 <= Average < 80
- B : 60 <= Average < 70
- C+ : 50 <= Average < 60
- C : 40 <= Average < 50
- F : Average < 40

Program:

```
import csv
from mrjob.job import MRJob
class MRStudentGrades(MRJob):
    # Mapper function: Process input data and compute average marks
    def mapper(self, _, line):
        # Using csv.reader to parse the CSV line
        reader = csv.reader([line])
        for data in reader:
            # Extract student name and marks (assuming marks are integers)
            name = data[0].strip()
            marks = list(map(int, data[1:]))  # Convert marks to integers
            # Calculate the average of the marks
            average = sum(marks) / len(marks)
            # Emit student name as key, and average marks as value
            yield name, average

    # Reducer function: Assign grades based on average marks
    def reducer(self, key, values):
     # There will be only one average value per student
        average = list(values)[0]
     # Determine the grade based on average marks
        if average >= 90:
            grade = 'A+'
        elif average >= 80:
            grade = 'A'
        elif average >= 70:
            grade = 'B+'
        elif average >= 60:
          grade = 'B'
      elif average >= 50:
          grade = 'C+'
      elif average >= 40:
          grade = 'C'
      else:
          grade = 'F'
      # Yield student name and their grade
      yield key, grade
if __name__ == '__main__':
   MRStudentGrades.run()
```