**NUMBER 1**

**1a:** Write a program in python (using matplotlib.pyplot) to create a line plot.

|  |
| --- |
| import matplotlib.pyplot as plt  x = [1,2,3,4,5,6,7,8,9,10]  y = [3,9,6,12,5,1,10,5,4,9]  plt.plot(x,y, label='lineplots', color='b')  plt.xlabel('X-axis')  plt.ylabel('Y-axis')  plt.title('Line Plot')  plt.legend()  plt.show() |
|  |

**1b**: Write a program in python (using matplotlib.pyplot) to create a scatter plot.

Use of scatterplot: Scatter plots are usually used to compare two variables (three if you are plotting in 3 dimensions), looking for correlation or groups.

|  |
| --- |
| import matplotlib.pyplot as plt  x = [1,2,3,4,5,6,7,8,9,10]  y = [3,9,6,12,5,1,10,5,4,9]  plt.scatter(x,y, label='scplots', color='r', s=60, marker="X")  plt.xlabel('X-axis')  plt.ylabel('Y-axis')  plt.title('Scatter Plot')  plt.legend()  plt.show() |
|  |

**NUMBER 2**

A bar chart or bar graph is a chart or graph that presents categorical data with rectangular bars.

**2a**:A simple bar graph:

|  |
| --- |
| import matplotlib.pyplot as plt  x = [2,4,6,8,10]  y=[3,9,11,2,6]  plt.bar(x,y,label ='Bars')  plt.xlabel('X-axis')  plt.ylabel('Y-axis')  plt.title('Bar Graph1')  plt.legend()  plt.show() |
|  |

**2b**: A bar graph with two values for comparison, using different colors.

|  |
| --- |
| import matplotlib.pyplot as plt  x1 = [2,4,6,8,10]  y1=[3,9,11,2,6]  x2=[1,3,5,7,9]  y2=[6,4,7,8,3]  plt.bar(x1,y1,label ='Bars1', color='g')  plt.bar(x2,y2,label = 'Bars2', color='r')  plt.xlabel('X-axis')  plt.ylabel('Y-axis')  plt.title('Bar Graph2')  plt.legend()  plt.show() |
|  |

Some other important attributes for bar graph:

**width** : scalar or array-like, optional

The width(s) of the bars (default: 0.8)

**color** : scalar or array-like, optional

The colors of the bar faces.

Note: barh creates a horizontal bar graph. The last example using barh

|  |
| --- |
| import matplotlib.pyplot as plt  x1 = [2,4,6,8,10]  y1=[3,9,11,2,6]  x2=[1,3,5,7,9]  y2=[6,4,7,8,3]  plt.barh(x1,y1,label ='Bars1', color='g')  plt.barh(x2,y2,label = 'Bar2', color='r')  plt.xlabel('X-axis')  plt.ylabel('Y-axis')  plt.title('Bar Graph2')  plt.legend()  plt.show() |
|  |

**2c:** A stacked bar graph with two values for comparison, using different colors. We use the same data as above and stack the values of y2 above y1. For this we use the parameter ‘bottom’ which indicates which value will be at the bottom. Also we use the ‘arange’ method of numpy to evenly space the bars.

|  |
| --- |
| import matplotlib.pyplot as plt  import numpy as np  N=6  y1=[3,9,11,2,6,4]  y2=[6,4,7,8,3,4]  xvalues = np.arange(N)  plt.bar(xvalues,y1,color='b', label ='Team1')  plt.bar(xvalues,y2, color='r', bottom =y1, label = 'Team2')  plt.xticks(xvalues, ('V1', 'V2', 'V3', 'V4', 'V5'))  plt.xlabel('Teams')  plt.ylabel('Scores')  plt.title('Stacked Bar Graphs')  plt.legend() |
|  |

**NUMBER 3**

Create histograms in python (using matplotlib.pyplot)

**3a:** A simple histogram

|  |
| --- |
| import matplotlib.pyplot as plt  x = [21,22,23,4,5,6,77,8,9,10,31,32,33,34,35,36,37,18,49,50,100,78,71,73,79,74]  num\_bins = 5  plt.hist(x, num\_bins, color='r', rwidth = 0.8)  plt.show() |
|  |

**3b:** Same histogram with an alpha value

|  |
| --- |
| import matplotlib.pyplot as plt  x=[21,22,23,4,5,6,77,8,9,10,31,32,33,34,35,36,37,18,49,50,100,78,71,73,79,74]  num\_bins = 5  plt.hist(x, num\_bins, color='r', alpha = 0.5, rwidth = 0.8)  plt.show() |
|  |

**3c:** Same histogram with histtype attribute

|  |
| --- |
| import matplotlib.pyplot as plt  x=[21,22,23,4,5,6,77,8,9,10,31,32,33,34,35,36,37,18,49,50,100,78,71,73,79,74]  num\_bins = 5  plt.hist(x, num\_bins, color='r', alpha = 0.5, rwidth = 0.8, histtype='stepfilled')  plt.show() |
|  |
| **histtype** : {'bar', 'barstacked', 'step', 'stepfilled'}, optional  The type of histogram to draw.   * 'bar' is a traditional bar-type histogram. If multiple data are given the bars are arranged side by side. * 'barstacked' is a bar-type histogram where multiple data are stacked on top of each other. * 'step' generates a lineplot that is by default unfilled. * 'stepfilled' generates a lineplot that is by default filled.   Default is 'bar' |

**NUMBER 4**

**Experiments:** Create stackplots and pie-charts in python (using matplotlib.pyplot)

**4a:** A stackplot

|  |
| --- |
| import matplotlib.pyplot as plt  days=[1,2,3,4,5]  sleeping = [7,8,6,11,7]  eating = [2,3,4,3,2]  working = [7,8,7,2,2]  playing = [8,5,7,8,13]  plt.plot([],[],color ='m', label = 'Sleeping', linewidth = 5)  plt.plot([],[],color ='c', label = 'Eating', linewidth = 5)  plt.plot([],[],color ='r', label = 'Working', linewidth = 5)  plt.plot([],[],color ='y', label = 'Playing', linewidth = 5)  plt.stackplot(days, sleeping, eating, working, playing, colors = ['m','c','r','y'])  plt.xlabel('Days')  plt.ylabel('Hours')  plt.title('Stack Plot1')  plt.xticks(days, ('Mon', 'Tue', 'Wed', 'Thur', 'Fri'))  plt.legend()  plt.show() |
|  |

**4b:** Same stackplot using a different way for legend

|  |
| --- |
| import matplotlib.pyplot as plt    days = [1, 2, 3, 4, 5]  sleeping = [7,8,6,11,7]  eating = [2,3,4,3,2]  working = [7,8,7,2,2]  playing = [8,5,7,8,13]  my\_labels = ["Sleeping ", "Eating", "Working", "Playing"]  fig, ax = plt.subplots()  ax.stackplot(days, sleeping, eating, working, playing, labels=my\_labels)  ax.legend(loc=1)  plt.xticks(days, ('Mon', 'Tue', 'Wed', 'Thur', 'Fri'))  plt.show() |
|  |

**4c:** Pie-Charts (with one slice exploded)

|  |
| --- |
| import matplotlib.pyplot as plt  days = [1, 2, 3, 4, 5]  slices = [7,2,2,13]  cols = ['r','y','g','b']  my\_labels = ["Sleeping ", "Eating", "Working", "Playing"]  plt.pie(slices,  labels=my\_labels,  colors = cols,  startangle=45,  explode =(0,0.2,0,0),  shadow = True,  autopct = '%1.1f%%')  plt.axis('equal')  plt.legend(loc=3)  plt.show() |
|  |

**NUMBER 5**

**Experiment:** Data-Preprocessing

|  |
| --- |
| *#Data Preprocessing*  *#Importing the Libraries*  import matplotlib.pyplot as plt  import pandas as pd  import numpy as np  *#Importing the Dataset*  dataset = pd.read\_csv('Data.csv')  X=dataset.iloc[:,:-1].values  Y=dataset.iloc[:,3].values  *#Handle missing data*  from sklearn.preprocessing import Imputer  imputer = Imputer(missing\_values="NaN", strategy="mean", axis=0)  imputer.fit(X[:,1:3])  X[:,1:3]=imputer.transform(X[:,1:3])  *#Encoding Categorical Data*  from sklearn.preprocessing import LabelEncoder, OneHotEncoder  labelencoder = LabelEncoder()  X[:,0]=labelencoder.fit\_transform(X[:,0])  onehotencoder = OneHotEncoder(categorical\_features =[0])  X = onehotencoder.fit\_transform(X).toarray()  labelencoder\_Y = LabelEncoder()  Y=labelencoder.fit\_transform(Y)  *#Splitting the dataset into Training set and Test set*  from sklearn.model\_selection import train\_test\_split  X\_train, X\_test, Y\_train, Y\_test = train\_test\_split(X,Y,test\_size=0.2, random\_state=0)  *#Feature Scaling*  from sklearn.preprocessing import StandardScaler  sc\_X = StandardScaler()  X\_train = sc\_X.fit\_transform(X\_train) |

**NUMBER 6**

**Experiment:** Simple Linear Regression

**Dataset:** The dataset Salary\_Data.csv is provided for the experiment. Download the dataset from

|  |
| --- |
| # Simple Linear Regression  # Importing the libraries  import numpy as np  import matplotlib.pyplot as plt  import pandas as pd  # Importing the dataset  dataset = pd.read\_csv('Salary\_Data.csv')  X = dataset.iloc[:, :-1].values  y = dataset.iloc[:, 1].values  # Splitting the dataset into the Training set and Test set  from sklearn.cross\_validation import train\_test\_split  X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size = 1/3, random\_state = 0)  # Fitting Simple Linear Regression to the Training set  from sklearn.linear\_model import LinearRegression  regressor = LinearRegression()  regressor.fit(X\_train, y\_train)  # Predicting the Test set results  y\_pred = regressor.predict(X\_test)  # Visualising the Training set results  plt.scatter(X\_train, y\_train, color = 'red')  plt.plot(X\_train, regressor.predict(X\_train), color = 'blue')  plt.title('Salary vs Experience (Training set)')  plt.xlabel('Years of Experience')  plt.ylabel('Salary')  plt.show()  # Visualising the Test set results  plt.scatter(X\_test, y\_test, color = 'red')  plt.plot(X\_train, regressor.predict(X\_train), color = 'blue')  plt.title('Salary vs Experience (Test set)')  plt.xlabel('Years of Experience')  plt.ylabel('Salary')  plt.show() |

**NUMBER 7**

**Experiment:** Logistic Regression

**Dataset:** The dataset Social\_Network\_Ads.csv is provided for the experiment. Download the dataset from

|  |
| --- |
| # Logistic Regression  # Importing the libraries  import numpy as np  import matplotlib.pyplot as plt  import pandas as pd  # Importing the dataset  dataset = pd.read\_csv('Social\_Network\_Ads.csv')  X = dataset.iloc[:, [2, 3]].values  y = dataset.iloc[:, 4].values  # Splitting the dataset into the Training set and Test set  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\_X = StandardScaler()  X\_train = sc\_X.fit\_transform(X\_train)  X\_test = sc\_X.transform(X\_test)  # Fitting Logistic Regression to the Training set  from sklearn.linear\_model import LogisticRegression  classifier = LogisticRegression(random\_state = 0)  classifier.fit(X\_train, y\_train)  # Predicting the Test set results  y\_pred = classifier.predict(X\_test)  # Making the Confusion Matrix  from sklearn.metrics import confusion\_matrix  cm = confusion\_matrix(y\_test, y\_pred)  # Visualising the Training set results  from matplotlib.colors import ListedColormap  X\_set, y\_set = X\_train, y\_train  X1, X2 = np.meshgrid(np.arange(start = X\_set[:, 0].min() - 1, stop = X\_set[:, 0].max() + 1, step = 0.01),  np.arange(start = X\_set[:, 1].min() - 1, stop = X\_set[:, 1].max() + 1, step = 0.01))  plt.contourf(X1, X2, classifier.predict(np.array([X1.ravel(), X2.ravel()]).T).reshape(X1.shape),  alpha = 0.75, cmap = ListedColormap(('red', 'green')))  plt.xlim(X1.min(), X1.max())  plt.ylim(X2.min(), X2.max())  for i, j in enumerate(np.unique(y\_set)):  plt.scatter(X\_set[y\_set == j, 0], X\_set[y\_set == j, 1],  c = ListedColormap(('red', 'green'))(i), label = j)  plt.title('Logistic Regression (Training set)')  plt.xlabel('Age')  plt.ylabel('Estimated Salary')  plt.legend()  plt.show()  # Visualising the Test set results  from matplotlib.colors import ListedColormap  X\_set, y\_set = X\_test, y\_test  X1, X2 = np.meshgrid(np.arange(start = X\_set[:, 0].min() - 1, stop = X\_set[:, 0].max() + 1, step = 0.01),  np.arange(start = X\_set[:, 1].min() - 1, stop = X\_set[:, 1].max() + 1, step = 0.01))  plt.contourf(X1, X2, classifier.predict(np.array([X1.ravel(), X2.ravel()]).T).reshape(X1.shape),  alpha = 0.75, cmap = ListedColormap(('red', 'green')))  plt.xlim(X1.min(), X1.max())  plt.ylim(X2.min(), X2.max())  for i, j in enumerate(np.unique(y\_set)):  plt.scatter(X\_set[y\_set == j, 0], X\_set[y\_set == j, 1],  c = ListedColormap(('red', 'green'))(i), label = j)  plt.title('Logistic Regression (Test set)')  plt.xlabel('Age')  plt.ylabel('Estimated Salary')  plt.legend()  plt.show() |
|  |
|  |

**NUMBER 8**

**Experiment:** Natural Language Processing

**Model:** Bag of Words

**Note:** Requires internet connection at runtime for downloading latest stopwords of nltk library

|  |
| --- |
| # Natural Language Processing  # Importing the libraries  import numpy as np  import matplotlib.pyplot as plt  import pandas as pd  # Importing the dataset  dataset = pd.read\_csv('Restaurant\_Reviews.tsv', delimiter = '\t', quoting = 3)  # Cleaning the texts  import re  import nltk  nltk.download('stopwords')  from nltk.corpus import stopwords  from nltk.stem.porter import PorterStemmer  corpus = []  for i in range(0, 1000):  review = re.sub('[^a-zA-Z]', ' ', dataset['Review'][i])  review = review.lower()  review = review.split()  ps = PorterStemmer()  review = [ps.stem(word) for word in review if not word in set(stopwords.words('english'))]  review = ' '.join(review)  corpus.append(review)  # Creating the Bag of Words model  from sklearn.feature\_extraction.text import CountVectorizer  cv = CountVectorizer(max\_features = 1500)  X = cv.fit\_transform(corpus).toarray()  y = dataset.iloc[:, 1].values  # Splitting the dataset into the Training set and Test set  from sklearn.cross\_validation import train\_test\_split  X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size = 0.20, random\_state = 0)  # Fitting Naive Bayes to the Training set  from sklearn.naive\_bayes import GaussianNB  classifier = GaussianNB()  classifier.fit(X\_train, y\_train)  # Predicting the Test set results  y\_pred = classifier.predict(X\_test)  # Making the Confusion Matrix  from sklearn.metrics import confusion\_matrix  cm = confusion\_matrix(y\_test, y\_pred) |

**NUMBER 9**

**Experiment:** KNN

**Dataset:** This experiment uses the popular IRIS dataset in csv format.

|  |
| --- |
| # KNN  # Importing the libraries  import numpy as np  import matplotlib.pyplot as plt  import pandas as pd  # Importing the dataset  dataset = pd.read\_csv('IRIS.csv')  X = dataset.iloc[:, :-1].values  y = dataset.iloc[:, 4].values  # Splitting the dataset into the Training set and Test set  from sklearn.model\_selection import train\_test\_split  X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size = 0.20, random\_state = 0)  # Feature Scaling  from sklearn.preprocessing import StandardScaler  sc\_X = StandardScaler()  X\_train = sc\_X.fit\_transform(X\_train)  X\_test = sc\_X.transform(X\_test)  # Fitting KNN to the Training set  from sklearn.neighbors import KNeighborsClassifier  classifier = KNeighborsClassifier(n\_neighbors=5)  classifier.fit(X\_train, y\_train)  # Predicting the Test set results  y\_pred = classifier.predict(X\_test)  # Making the Confusion Matrix  from sklearn.metrics import classification\_report, confusion\_matrix  print(confusion\_matrix(y\_test, y\_pred))  print(classification\_report(y\_test, y\_pred)) |
|  |

**NUMBER 10**

**Experiment:** Use of Seaborn

**Dataset:** This experiment uses the popular IRIS dataset in csv format.

|  |
| --- |
| # Visualizations using Seaborn Library  # Importing the libraries  import matplotlib.pyplot as plt  import pandas as pd  import seaborn as sb  # Importing the dataset  dataset = pd.read\_csv('IRIS.csv')  #Box Plot  sb.boxplot(x='species',y='petal\_length',data=dataset)  #Pair-Plot  sb.pairplot(dataset, kind="scatter");  sb.pairplot(dataset, hue="species");  #Heatmap  plt.figure(figsize=(7,4))  sb.heatmap(dataset.corr(),annot=True,cmap='summer') |
| Boxplot: |
| Pairplot (kind="scatter"): |
| Pairplot (hue="species"); |
| HeatMap: |