INDEX

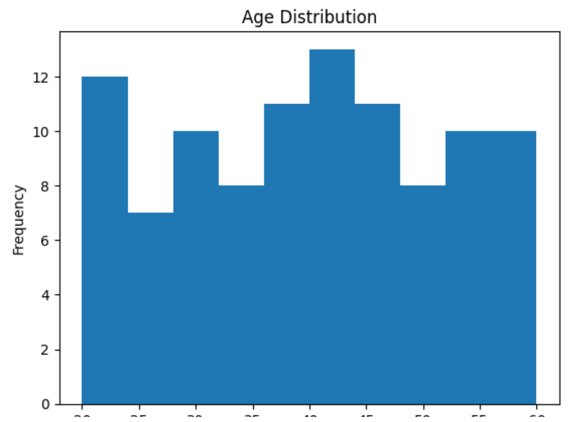
<u>**LAB 1**</u>

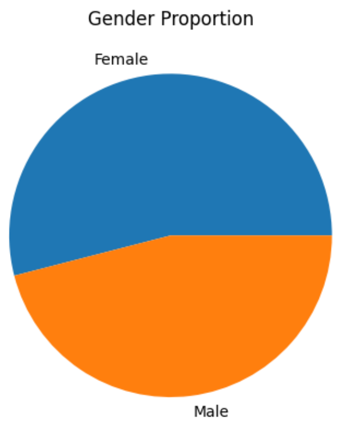
Aim: Construct various type of plots/charts like histogram, bar chart, pie-chart and scatter plot by importing data from a CSV format file. Further label different axes and data in a plot.

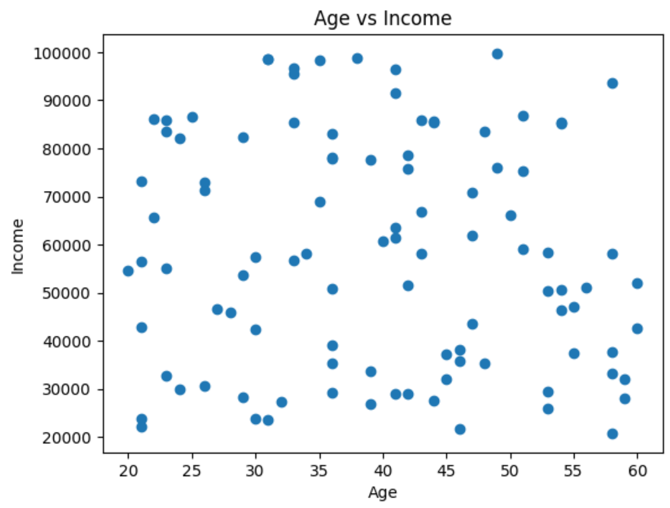
Program:

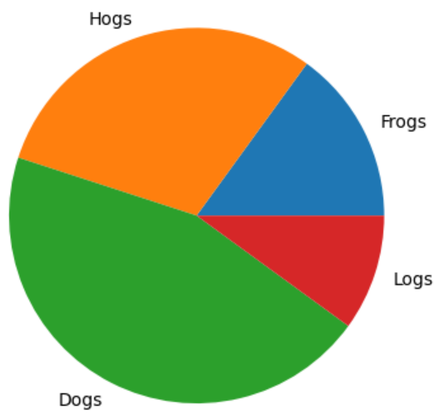
```
import random
import pandas as pd
import matplotlib.pyplot as plt
data={'age':[random.randint(20,60) for _ in
range(100)],'gender':[random.choice(["Male","Female"]) for _ in
range(100)],'income':[random.randint(20000,100000) for _ in range(100)]}
df=pd.DataFrame(data)
plt.hist(df['age'])
plt.xlabel('Age')
plt.ylabel('Frequency')
plt.title('Age Distribution')
plt.show()
plt.bar(df['gender'].unique(), df['gender'].value_counts())
plt.xlabel('Gender')
plt.ylabel('Count')
plt.title('Gender Comparison')
plt.show()
print(type(df['gender'].value_counts()))
print(df['gender'].value_counts())
print(type(df['gender'].unique()))
print(df['gender'].unique())
import matplotlib.pyplot as plt
labels = 'Frogs', 'Hogs', 'Dogs', 'Logs'
sizes = [15, 30, 45, 10]

fig, ax = plt.subplots()
ax.pie(sizes, labels=labels)
plt.pie(df['gender'].value_counts(), labels=df['gender'].unique())
plt.title('Gender Proportion')
plt.show()

plt.scatter(df['age'], df['income'])
plt.xlabel('Age')
plt.ylabel('Income')
plt.title('Age vs Income')
plt.show()
```

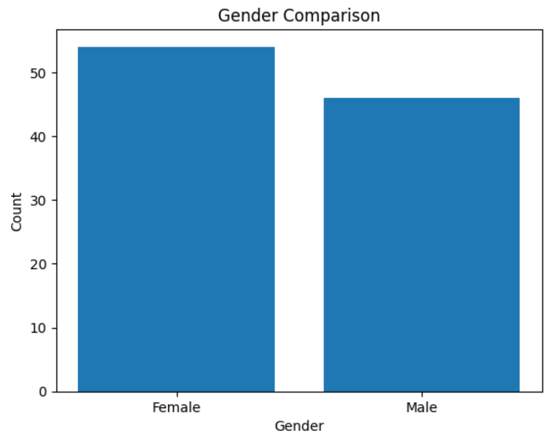
Output:


Age Distribution
Frequency
0
2
4
6
8
10
12


Gender Proportion
Female
Male


Age vs Income
Income
100000
90000
80000
70000
60000
50000
40000
30000
20000
20
25
30
35
40
45
50
55
60
Age


Hogs
Frogs
Logs
Dogs


Gender Comparison
Count
0
10
20
30
40
50
Female
Male
Gender

LAB 2

Aim: Perform data cleaning and data preprocessing tasks.

Program:

```
import pandas as pd
import numpy as np
df = pd.read_csv('/content/data.csv')
print(df.columns)
print(df.isnull().sum())
df['income'].fillna(df['income'].mean(), inplace=True)
df['date'].fillna(method='ffill', inplace=True)
df.dropna(inplace=True)
df = df.drop('gender', axis=1)# Remove irrelevant column
df = df.assign(age_squared=lambda x: x['age']**2)# Insert new column with
age squared
df = df.rename(columns={'income': 'annual_income'})# Rename target variable
column
print(df.columns)
from sklearn.preprocessing import MinMaxScaler, StandardScaler, RobustScaler
scaler = MinMaxScaler()
df['age'] = scaler.fit_transform(df[['age']])
scaler = StandardScaler()
df[['annual_income', 'age_squared']] =
scaler.fit_transform(df[['annual_income', 'age_squared']])
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
df['marital_status'] = le.fit_transform(df['marital_status'])
print(df)
```

Output:

```
Index(['age', 'income', 'date', 'marital_status', 'gender'], dtype='object')
age               0
income            1
date              1
marital_status    0
gender            0
dtype: int64

Index(['age', 'annual_income', 'date', 'marital_status', 'age_squared'],
dtype='object')

       age  annual_income        date  marital_status  age_squared
0  0.545455      -0.313112  2020-01-01               0    -0.128298
1  0.818182       0.939336  2020-01-02               1     0.827650
2  0.000000      -1.565561  2020-01-03               0    -1.587378
3  0.409091      -0.939336  2020-01-04               1    -0.549671
4  0.227273       0.000000  2020-01-05               0    -1.052802
5  0.590909       0.000000  2020-01-06               1     0.020545
6  1.000000       1.565561  2020-01-07               0     1.548805
7  0.727273       0.313112  2020-01-07               1     0.492230
8  0.954545       1.252449  2020-01-09               0     1.362227
9  0.272727      -1.252449  2020-01-10               1    -0.933308
```

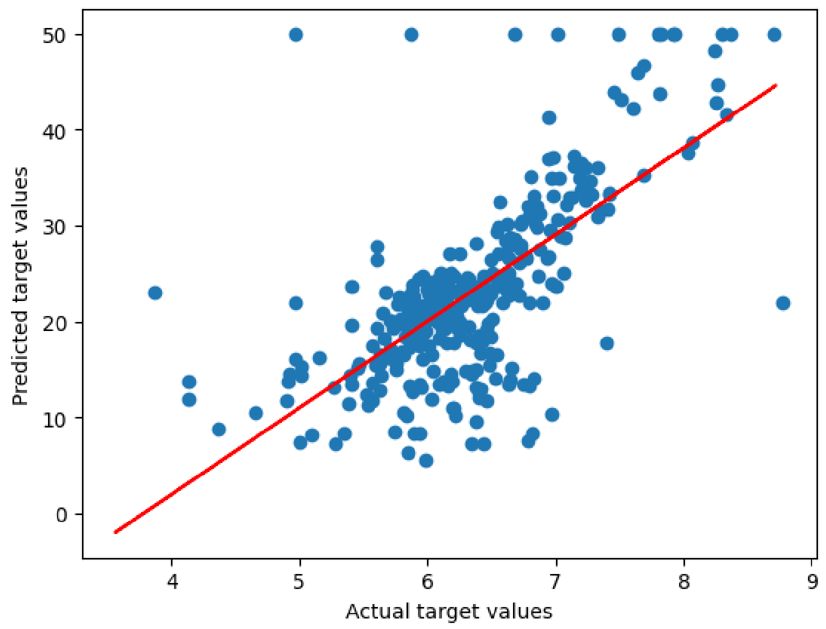
LAB 3

Aim: Perform linear regression on boston datasets.

Program:

```
import pandas as pd
import numpy as np
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
df=pd.read_csv("/content/drive/MyDrive/Class/My_lab/Lab4/Boston.csv")
X=df[['rm']]
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)
import matplotlib.pyplot as plt
plt.scatter(X_train, y_train)
plt.plot(X_test,y_pred,color='red')
plt.xlabel('Actual target values')
plt.ylabel('Predicted target values')
plt.show()
```

Output:



LAB 4

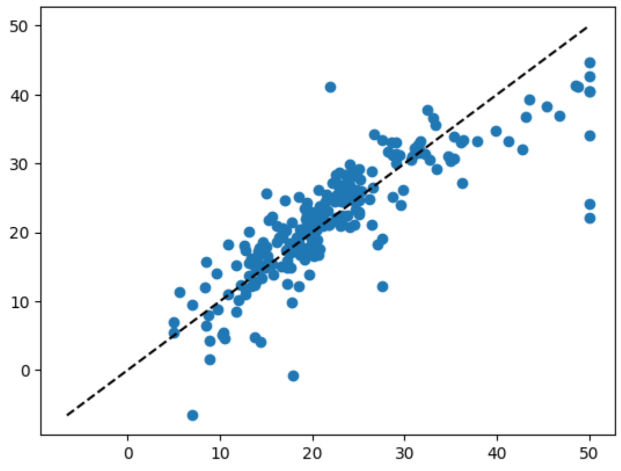
Aim: Perform multiple linear regression on boston datasets.

Program:

```
import pandas as pd
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
df=pd.read_csv("/content/drive/MyDrive/Class/My_lab/Lab5/Boston.csv")
X=df[['crim','zn','indus','chas','nox','rm','age','dis','rad','tax','ptratio
','black','lstat']]
y=df['medv']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.5,
random_state=42)
model=LinearRegression()
model.fit(X_train,y_train)
y_pred=model.predict(X_test)
import matplotlib.pyplot as plt
plt.scatter(y_test,y_pred)
lims = [min(min(y_test), min(y_pred)), max(max(y_test), max(y_pred))]
plt.plot(lims, lims, 'k--')
from sklearn.metrics import r2_score
print(r2_score(y_pred,y_test))
```

Output:

```
0.6253969844551934
```



LAB 5

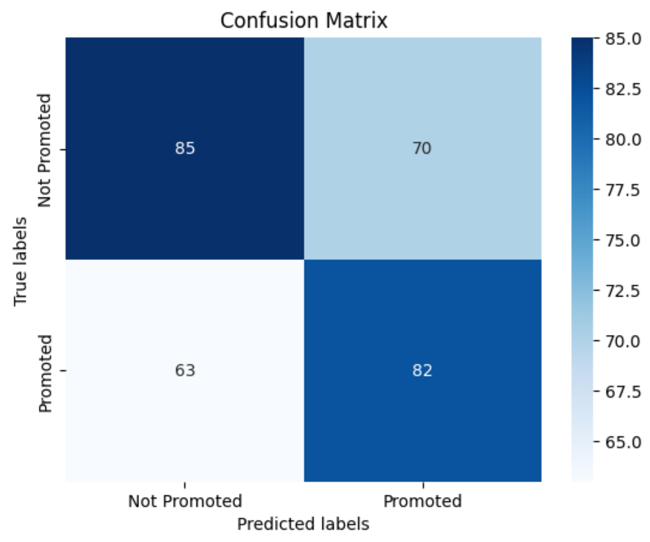
Aim: Perform logistics regression on generated datasets.

Program:

```
import numpy as np
import pandas as pd
age=np.random.normal(40,10,n)
gender=np.random.choice(["Male","Female"],n)
education = np.random.choice(['high school', 'college', 'graduate'], n)
job_level = np.random.choice(['junior', 'senior'], n)
last_evaluation = np.random.uniform(0.4, 1, n)
average_monthly_hours = np.random.randint(100, 300, n)
time_spend_company = np.random.randint(1, 10, n)
number_of_projects = np.random.randint(1, 7, n)
work_accident = np.random.choice([0, 1], n)
promotion = np.random.choice([0, 1], n)
salary = np.random.choice(['low', 'medium', 'high'], n)
dictionary={'age':age,'gender':gender,'education':education,'job_level':job_
level,'last_evaluation':last_evaluation,'average_monthly_hours':average_mont
hly_hours,'time_spend_company':time_spend_company,'number_of_projects':numbe
r_of_projects,'work_accident':work_accident,'promotion':promotion,'salary':s
alary}
df=pd.DataFrame(dictionary)
data=pd.get_dummies(df,columns=['gender','education','job_level','salary'])
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import confusion_matrix,classification_report
from sklearn.model_selection import train_test_split
model=LogisticRegression()
X=data.drop('promotion',axis=1)
Y=data['promotion']
train_x,test_x,train_y,test_y=train_test_split(X,Y,test_size=0.3)
model.fit(train_x,train_y)
y_pred=model.predict(test_x)
print(classification_report(test_y, y_pred))
print(confusion_matrix(test_y, y_pred))
import matplotlib.pyplot as plt
import seaborn as sns
cm = confusion_matrix(test_y, y_pred)
ax = plt.subplot()
sns.heatmap(cm, annot=True, ax=ax, cmap='Blues', fmt='g')
ax.set_xlabel('Predicted labels')
ax.set_ylabel('True labels')
ax.set_title('Confusion Matrix')
ax.xaxis.set_ticklabels(['Not Promoted', 'Promoted'])
ax.yaxis.set_ticklabels(['Not Promoted', 'Promoted'])
plt.show()
```

Output:

```
precision    recall  f1-score   support

           0       0.57      0.55      0.56       155
           1       0.54      0.57      0.55       145

    accuracy                           0.56       300
   macro avg       0.56      0.56      0.56       300
weighted avg       0.56      0.56      0.56       300
[[85 70]
 [63 82]]
```



LAB 6

Aim: Write a program to train the Naïve Bayes classifier for a sample training data set stored as a .csv file. Compute the accuracy of the classifier, considering the test dataset.

Program:

```
import pandas as pd
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import accuracy_score, confusion_matrix
spam_df = pd.read_csv('/content/drive/MyDrive/Class/My_lab/Lab 7 Naive
Bayes/spam.csv', encoding='latin-1')
spam_df = spam_df.drop(['Unnamed: 2', 'Unnamed: 3', 'Unnamed: 4'], axis=1)
spam_df = spam_df.rename(columns={'v1': 'label', 'v2': 'text'})
train_data = spam_df[:4400]
test_data = spam_df[4400:]
vectorizer = CountVectorizer()
train_vectors = vectorizer.fit_transform(train_data["text
test_vectors = vectorizer.transform(test_data["text"])
nb_classifier = MultinomialNB()
nb_classifier.fit(train_vectors, train_data["label"])
predictions = nb_classifier.predict(test_vectors)
accuracy = accuracy_score(test_data["label"], predictions)
print("Accuracy:", accuracy)
print("Confusion Matrix:")
print(confusion_matrix(test_data["label"], predictions))
```

Output:

```
Accuracy: 0.9863481228668942
Confusion Matrix:
[[1015    8]
 [   8  141]]
```

LAB 7

Aim: Write a program to implement K-Nearest Neighbour algorithm to classify the sample data set into different classes. Print both correct and wrong predictions.

Program:

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.datasets import load_iris
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import r2_score
data=load_iris()
x_train,x_test,y_train,y_test=train_test_split(data.data,data.target,test_si
ze=0.2)
model=KNeighborsClassifier(n_neighbors=3)
model.fit(x_train,y_train)
y_pred=model.predict(x_test)
print(r2_score(y_test,y_pred))
for i in range(len(y_pred)):
    if y_pred[i] != y_test[i]:
       print(f"Wrong prediction: Actual label = {y_test[i]}, Predicted label
= {y_pred[i]}")
```

Output:

```
0.8861138861138861
Wrong prediction: Actual label = 1, Predicted label = 2
Wrong prediction: Actual label = 2, Predicted label = 1
Wrong prediction: Actual label = 2, Predicted label = 1
```

LAB 8

Aim: Write a program to train a Decision Tree Classifier to classify the sample dataset into different categories.

Program:

```
import numpy as np
import pandas as pd
from sklearn.datasets import load_iris
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
X=load_iris().data
y=load_iris().target
x_train,x_test,y_train,y_test=train_test_split(X,y,test_size=0.2,random_stat
e=2)
dt=DecisionTreeClassifier()
dt.fit(x_train,y_train)
y_pred=dt.predict(x_test)
print("Accuracy: ",accuracy_score(y_test,y_pred))
from sklearn.model_selection import GridSearchCV

params = {'max_depth': [1, 2, 3, 4, 5]}
clf = DecisionTreeClassifier()
grid_search = GridSearchCV(clf, param_grid=params, cv=5)
grid_search.fit(x_train, y_train)
print("Best parameters:", grid_search.best_params_)
```

Output:

```
Accuracy:  0.9333333333333333

Best parameters: {'max_depth': 5}
```

LAB 9

Aim: Write a program to implement K-means algorithm to form clusters in a sample of datasets on wine dataset.

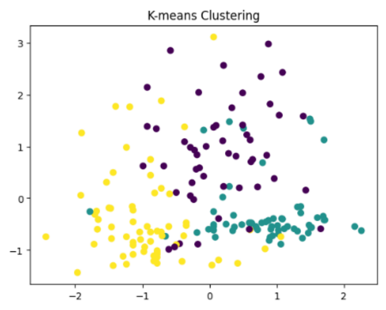
Dataset description: It contains 178 observations of wine grown in the same region in Italy. Each observation is from one of three cultivars (the 'Class' feature), with 13 constituent features that are the result of a chemical analysis.

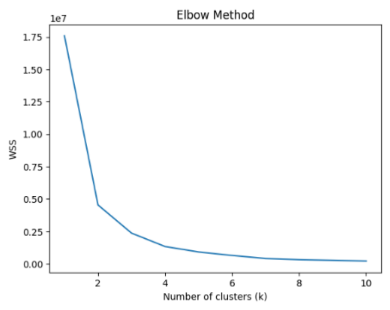
Program:

```python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
df=pd.read_csv("/content/wine.data",header=None)
df=df.drop(0,axis=1)
print(df.shape)
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
wine_data_scaled = scaler.fit_transform(df)
kmeans = KMeans(n_clusters=3, random_state=42)
kmeans.fit(wine_data_scaled)
plt.scatter(wine_data_scaled[:, 0], wine_data_scaled[:, 1],
c=kmeans.labels_)
plt.title('K-means Clustering')
plt.show()
wss_values = []
for k in range(1, 11):
    kmeans = KMeans(n_clusters=k, random_state=42)
    kmeans.fit(df)
    wss_values.append(kmeans.inertia_)
# Plot the WSS values against different values of k
plt.plot(range(1, 11), wss_values)
plt.title('Elbow Method')
plt.xlabel('Number of clusters (k)')
plt.ylabel('WSS')
plt.show()
```

Output:

```
(178, 13)
```





LAB 10

Aim: Write a program to implement Principal Component Analysis to reduce the number of dimensions and thereby the size of the raw dataset.

Program:

```
import pandas as pd
import numpy as np
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
# set seed for reproducibility
np.random.seed(42)
data = np.random.randn(1000, 10) # generate random data
# create DataFrame with column names
df = pd.DataFrame(data, columns=['feature_1', 'feature_2', 'feature_3',
'feature_4', 'feature_5', 'feature_6', 'feature_7', 'feature_8',
'feature_9', 'feature_10'])
df['label'] = np.random.randint(0, 2, size=1000) # add labels to DataFrame

print(df.shape)
X = df.drop('label', axis=1)
y = df['label']
scaler = StandardScaler()
X = scaler.fit_transform(X)
pca = PCA()
X_pca = pca.fit_transform(X)
explained_variance = pca.explained_variance_ratio_ #Determine the number of
principal components to retain
n_components = np.argmax(np.cumsum(explained_variance) >= 0.90) + 1
pca = PCA(n_components=n_components)#Transform the features using the
retained principal components
X_pca = pca.fit_transform(X)
new_df = pd.concat([pd.DataFrame(X_pca), y], axis=1)
new_df.to_csv('new_data.csv', index=False)
print(new_df.shape)
```

Output:

```
(1000, 11)

(1000, 10)
```