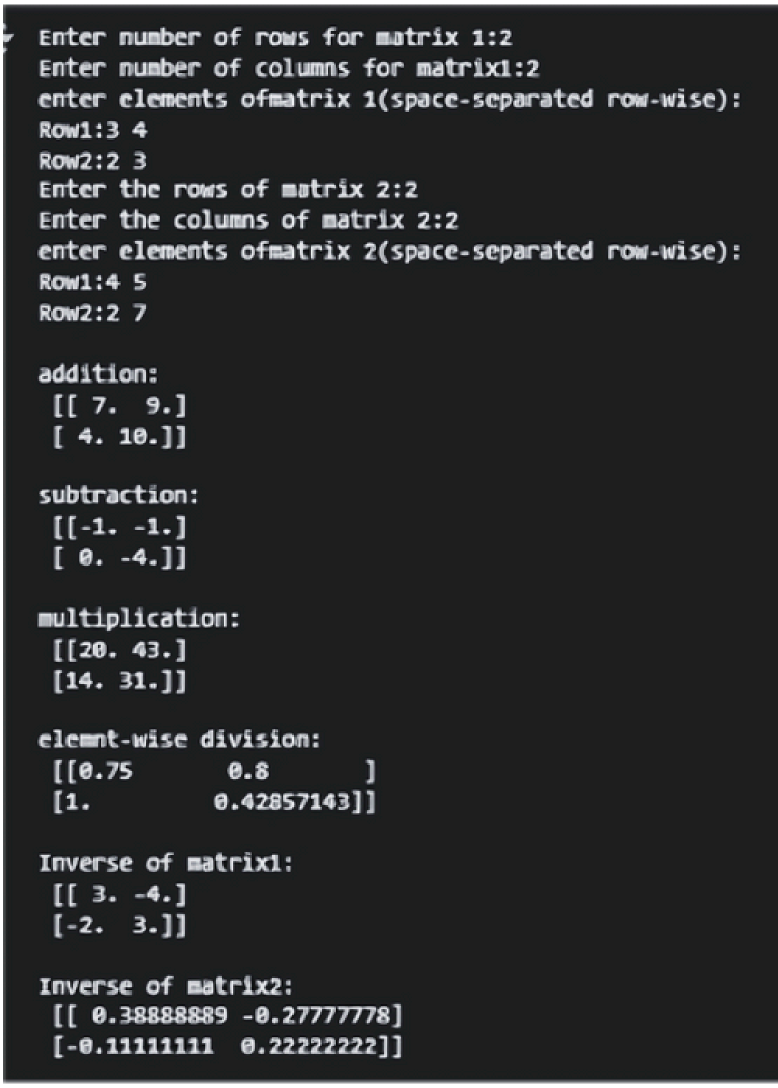
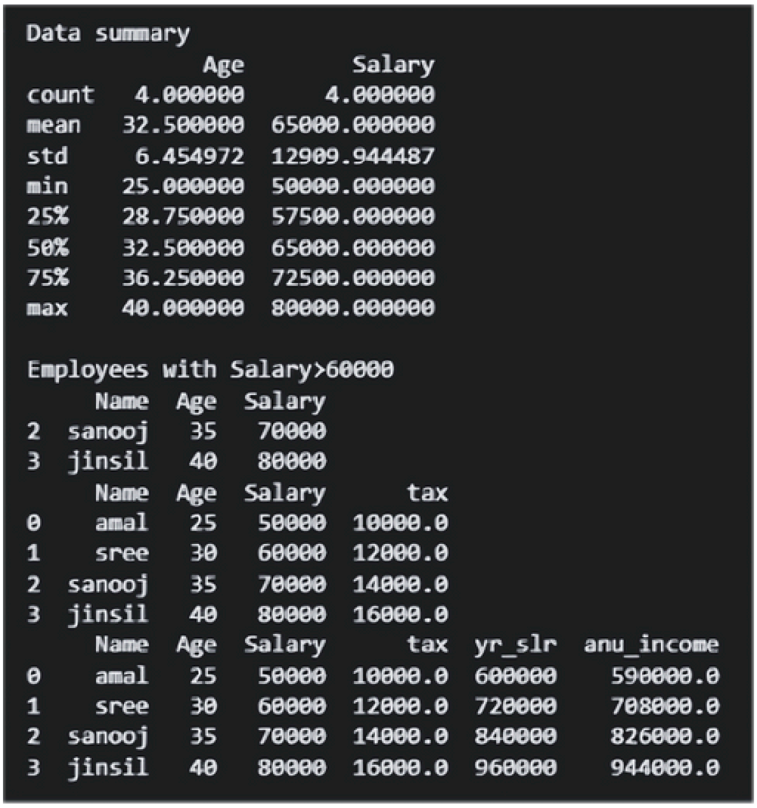
```
import numpy as np
def get_matrix(rows,cols,name):
  print(f"enter elements of{name}(space-separated row-wise):")
  matrix=[]
  for i in range(rows):
    row = list(map(float,input(f"Row{i+1}:").split()))
    matrix.append(row)
  return np.array(matrix)

#input for matrix1
r1=int(input("Enter number of rows for matrix 1:"))
c1=int(input("Enter number of columns for matrix1:"))
matrix1=get_matrix(r1,c1,"matrix 1")

# input for matrix2
r2 = int(input("Enter the rows of matrix 2:"))
c2 = int(input("Enter the columns of matrix 2:"))
matrix2 = get_matrix(r2, c2, "matrix 2")

#Addition
if matrix1.shape == matrix2.shape:
  print("\naddition:\n",matrix1+matrix2)
else:
  print("Addition not possible")

#subtraction
if matrix1.shape == matrix2.shape:
  print("\nsubtraction:\n",matrix1-matrix2)
else:
  print("Subtraction not possible")

#multiplication
if  c1==c2:
    print("\nmultiplication:\n",np.dot(matrix1,matrix2))
else:
    print("\multiplication not possible")

#division(element wise)
if matrix1.shape == matrix2.shape:
    try:
      print("\nelemnt-wise division:\n",matrix1 /matrix2)
    except ZeroDivisionError:
      print("\nDivision by zero detected")
else:
    print("\nDivision not possible")

#inverse
def matrix_inverse(matrix,name):
  if matrix.shape[0]!=matrix.shape[1]:
    print(f"\n{name} is not square,Inverse not possible.")
    return
  try:
    inv=np.linalg.inv(matrix)
    print(f"\nInverse of {name}:\n",inv)
  except np.linalg.LinAlgError:
    print(f"\n{name} is not invertible.")
matrix_inverse(matrix1,"matrix1")
matrix_inverse(matrix2,"matrix2")
```

```
import pandas as pd
data = {'Name':['amal', 'sree','sanooj','jinsil'],
    'Age':[25, 30, 35, 40],
    'Salary':[50000,60000,70000,80000]}
df=pd.DataFrame(data)
print("Data summary")
print(df.describe())
print("\nEmployees with Salary>60000")
print(df[df['Salary']>60000])
df['tax']=df['Salary'] * 0.2
print(df)
df['yr_slr']=df['Salary']*12
df['anu_income']=df['yr_slr']-df['tax']
print(df)
#print("tax",df["tax"])
```



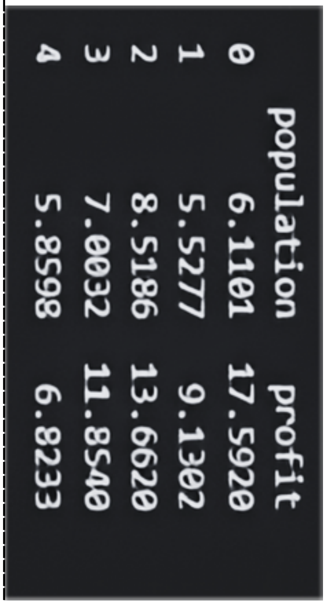
```
Enter number of rows for matrix 1:2
Enter number of columns for matrix1:2
enter elements ofmatrix 1(space-separated row-wise):
Row1:3 4
Row2:2 3
Enter the rows of matrix 2:2
Enter the columns of matrix 2:2
enter elements ofmatrix 2(space-separated row-wise):
Row1:4 5
Row2:2 7

addition:
 [[ 7.  9.]
 [ 4. 10.]]

subtraction:
 [[-1. -1.]
 [ 0. -4.]]

multiplication:
 [[20. 43.]
 [14. 31.]]

elemnt-wise division:
 [[0.75       0.8       ]
 [1.         0.42857143]]

Inverse of matrix1:
 [[ 3. -4.]
 [-2.  3.]]

Inverse of matrix2:
 [[ 0.38888889 -0.27777778]
 [-0.11111111  0.22222222]]
```

```
import plotly.express as px
import pandas as pd
data={'product':['A','B','C','D'],
   'Sales':[120,340,290,410]}
df=pd.DataFrame(data)
fig = px.bar(df,x='product', y='Sales',color='product',title='Product sales')
fig.show()
```



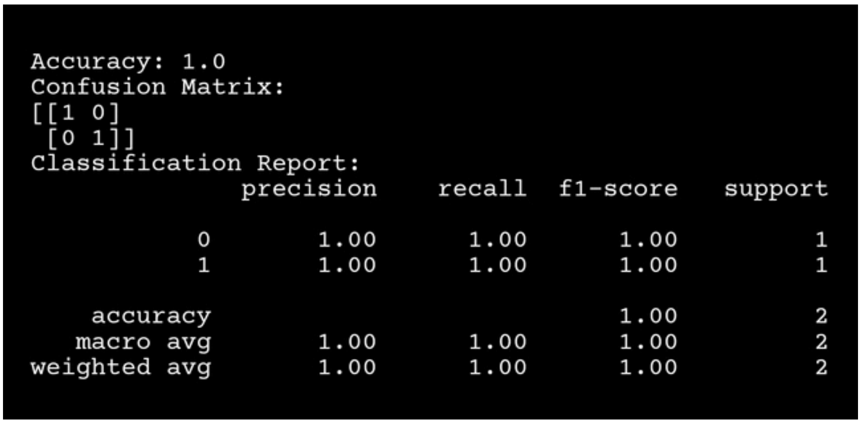
```
Data summary
             Age        Salary
count   4.000000      4.000000
mean   32.500000  65000.000000
std     6.454972  12909.944487
min    25.000000  50000.000000
25%    28.750000  57500.000000
50%    32.500000  65000.000000
75%    36.250000  72500.000000
max    40.000000  80000.000000

Employees with Salary>60000
     Name  Age  Salary
2  sanooj   35   70000
3  jinsil   40   80000
     Name  Age  Salary      tax
0    amal   25   50000  10000.0
1    sree   30   60000  12000.0
2  sanooj   35   70000  14000.0
3  jinsil   40   80000  16000.0
     Name  Age  Salary      tax  yr_slr  anu_income
0    amal   25   50000  10000.0  600000    590000.0
1    sree   30   60000  12000.0  720000    708000.0
2  sanooj   35   70000  14000.0  840000    826000.0
3  jinsil   40   80000  16000.0  960000    944000.0
```

```
import numpy as np
import pandas as pd
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
import matplotlib.pyplot as plt
data = pd.read_csv('Week1_Ex1.txt',header=None,names=['population','profit'])
print(data.head())
```



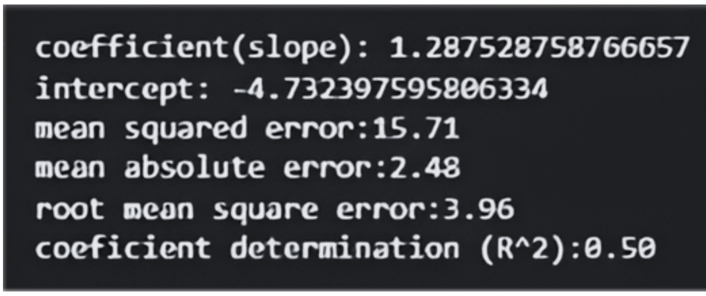
```
   population   profit
0      6.1101  17.5920
1      5.5277   9.1302
2      8.5186  13.6620
3      7.0032  11.8540
4      5.8598   6.8233
```

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix

data=pd.read_csv("heart-ds.csv")
data['target']=np.where(data['target']>0,1,0)

X=data.drop('target', axis=1)
y=data['target']

scaler=StandardScaler()
X_scaled=scaler.fit_transform(X)

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

svm_model=SVC(kernel='linear')
svm_model.fit(X_train, y_train)

y_pred=svm_model.predict(X_test)

accuracy=accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy_score(y_test, y_pred))
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))
print("Classification Report:\n", classification_report(y_test, y_pred))
```



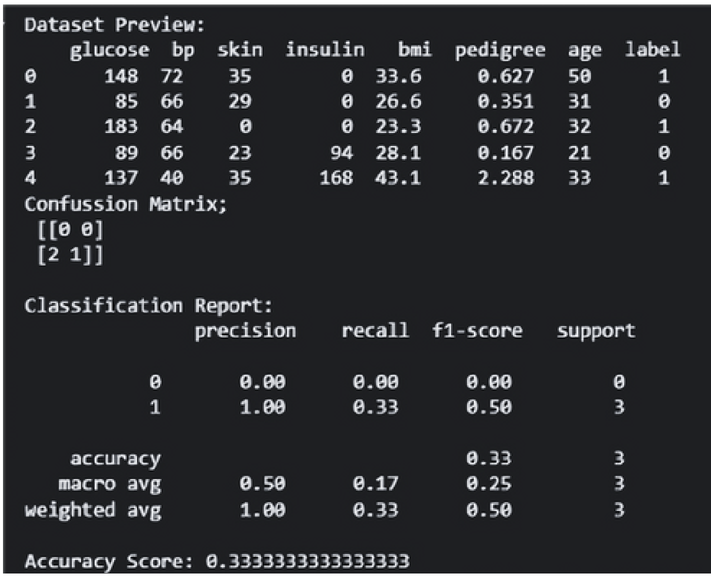
```
Accuracy: 1.0
Confusion Matrix:
[[1 0]
 [0 1]]
Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00         1
           1       1.00      1.00      1.00         1

    accuracy                           1.00         2
   macro avg       1.00      1.00      1.00         2
weighted avg       1.00      1.00      1.00         2
```

```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error,mean_absolute_error,r2_score
data = pd.read_csv('Week1_Ex1.txt',header=None,names=['population','profit'])
X = data['population'].values.reshape(-1,1)
y = data['profit'].values
x_train,x_test,y_train,y_test=train_test_split(X,y,test_size=0.20,random_state=42)
model = LinearRegression()
model.fit(x_train,y_train)
y_pred = model.predict(x_test)
print('coefficient(slope):',model.coef_[0])
print('intercept:',model.intercept_)
print('mean squared error:%.2f' % mean_squared_error(y_test,y_pred))
print('mean absolute error:%.2f' % mean_absolute_error(y_test,y_pred))
print('root mean square error:%.2f' % np.sqrt(mean_squared_error(y_test,y_pred)))
print('coeficient determination (R^2):%.2f' % r2_score(y_test,y_pred))
```



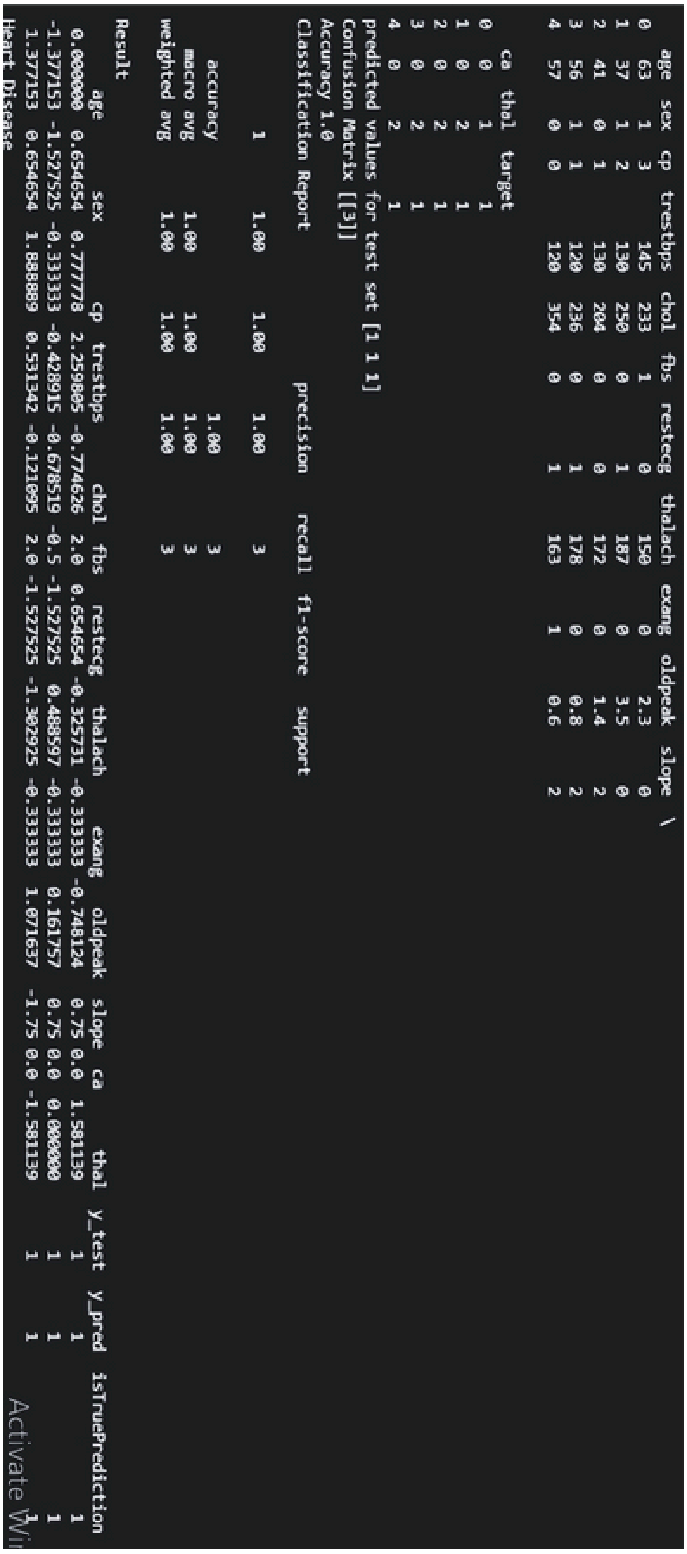
```
coefficient(slope): 1.287528758766657
intercept: -4.732397595806334
mean squared error:15.71
mean absolute error:2.48
root mean square error:3.96
coeficient determination (R^2):0.50
```

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn. metrics import classification_report,confusion_matrix, accuracy_score

pima = pd.read_csv("/content/pima_indians_diabetes_sample.csv")
print("Dataset Preview:\n",pima.head())

feature_cols =['insulin','bmi','age','glucose','bp','pedigree']
X = pima[feature_cols]
y = pima['label']

X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.3,random_state=1)

model = LogisticRegression(max_iter=200)
model.fit(X_train, y_train)

y_pred = model.predict(X_test)

print("Confussion Matrix;\n",confusion_matrix(y_test,y_pred))
print("\nClassification Report:\n", classification_report(y_test, y_pred))
print("Accuracy Score:",accuracy_score(y_test,y_pred))
```

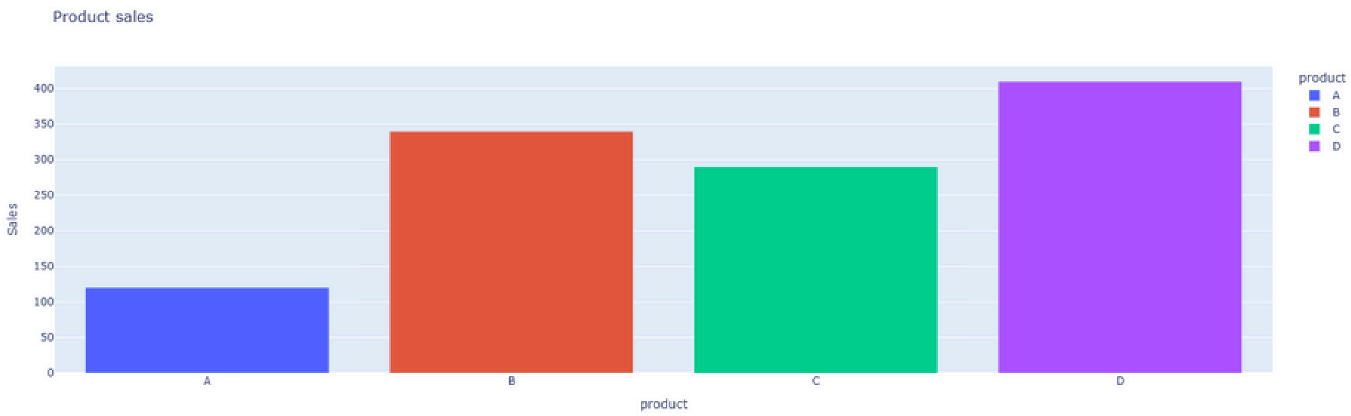


```
Dataset Preview:
    glucose  bp  skin  insulin   bmi  pedigree  age  label
0      148  72    35        0  33.6     0.627   50      1
1       85  66    29        0  26.6     0.351   31      0
2      183  64     0        0  23.3     0.672   32      1
3       89  66    23       94  28.1     0.167   21      0
4      137  40    35      168  43.1     2.288   33      1
Confussion Matrix;
 [[0 0]
 [2 1]]

Classification Report:
               precision    recall  f1-score   support

           0       0.00      0.00      0.00         0
           1       1.00      0.33      0.50         3

    accuracy                           0.33         3
   macro avg       0.50      0.17      0.25         3
weighted avg       1.00      0.33      0.50         3

Accuracy Score: 0.3333333333333333
```

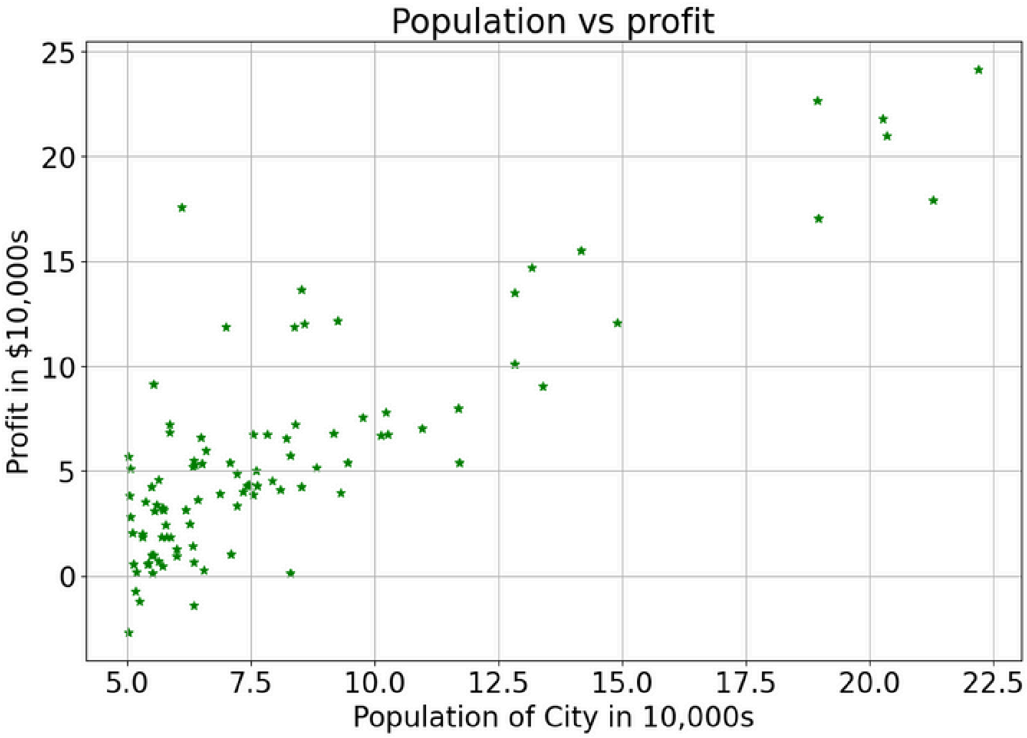
```
import pandas as pd
data = pd.read_csv('heart.csv')
print(data.head())

from sklearn.preprocessing import StandardScaler
std = StandardScaler()
fit = std.fit(data.drop('target',axis=1))
scaled_features = std.transform(data.drop('target',axis=1))
scaled_features

feat = pd.DataFrame(scaled_features,columns = data.columns[:-1])
data.columns[:-1]

from sklearn.model_selection import train_test_split
x = feat
y = data.target
x_train,x_test,y_train,y_test = train_test_split(x,y,test_size =0.3,random_state =101)

from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors=3)
knn.fit(x_train,y_train)
y_pred = knn.predict(x_test)
print('predicted values for test set',y_pred)

from sklearn.metrics import confusion_matrix,accuracy_score,classification_report
confusion = confusion_matrix(y_test,y_pred)
print('Confusion Matrix',confusion)
accuracy = accuracy_score(y_test,y_pred)
print('Accuracy',accuracy)
classification = classification_report(y_test,y_pred)
print('Classification Report',classification)

import numpy as np
result = x_test.copy()
result["y_test"] = y_test
result["y_pred"] = y_pred
result["isTruePrediction"] = np.where(result["y_test"] == result["y_pred"],1,0)
print('Result\n',result.head().to_string(index=False))

new_point = [[63,1,3,145,233,1,0,150,0,2.3,0,0,1]]

new_scaled_point = std.transform(new_point)
new_prediction = knn.predict(new_scaled_point)
if(new_prediction[0]==1):
   print('Heart Disease')
else:
   print('No Disease')
```



```
   age  sex  cp  trestbps  chol  fbs  restecg  thalach  exang  oldpeak  slope  \
0   63    1   3       145   233    1        0      150      0      2.3      0
1   37    1   2       130   250    0        1      187      0      3.5      0
2   41    0   1       130   204    0        0      172      0      1.4      2
3   56    1   1       120   236    0        1      178      0      0.8      2
4   57    0   0       120   354    0        1      163      1      0.6      2

   ca  thal  target
0   0     1       1
1   0     2       1
2   0     2       1
3   0     2       1
4   0     2       1
predicted values for test set [1 1 1]
Confusion Matrix [[3]]
Accuracy 1.0
Classification Report               precision    recall  f1-score   support


           1       1.00      1.00      1.00         3


    accuracy                           1.00         3
   macro avg       1.00      1.00      1.00         3
weighted avg       1.00      1.00      1.00         3

Result
      age       sex        cp  trestbps      chol  fbs   restecg   thalach     exang   oldpeak  slope  ca      thal  y_test  y_pred  isTruePrediction
 0.000000  0.654654  0.777778  2.259805 -0.774626  2.0  0.654654 -0.325731 -0.333333 -0.748124   0.75 0.0  1.581139       1       1                 1
-1.377153 -1.527525 -0.333333 -0.428915 -0.678519 -0.5 -1.527525  0.488597 -0.333333  0.161757   0.75 0.0  0.000000       1       1                 1
 1.377153  0.654654  1.888889  0.531342 -0.121095  2.0 -1.527525 -1.302925 -0.333333  1.071637  -1.75 0.0 -1.581139       1       1                 1
Heart Disease
```

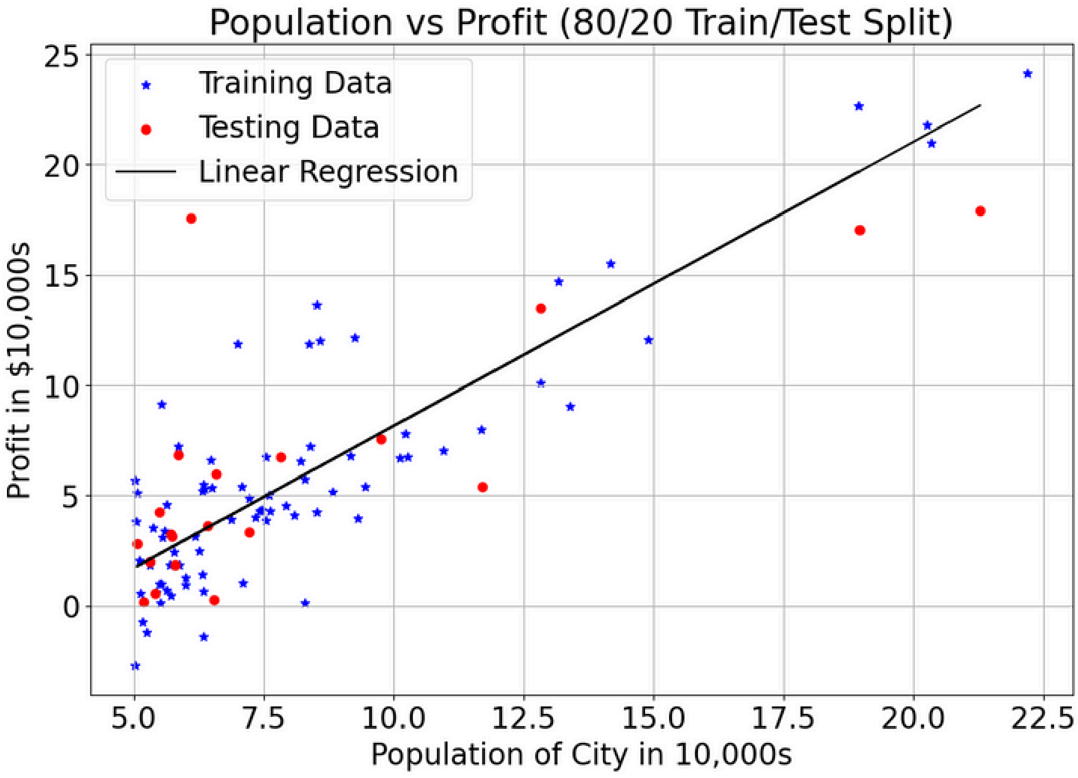


---



```
plt.figure(figsize=(12,8))
common_font_size=20
plt.rcParams['font.size']=common_font_size
plt.scatter(data['population'],data['profit'],color='green',marker='*',s=40)
plt.title('Population vs profit',fontsize=common_font_size + 4)
plt.xlabel('Population of City in 10,000s')
plt.ylabel('Profit in $10,000s')
plt.grid(True)
plt.savefig('Populationvsprofit.jpeg',bbox_inches='tight',dpi=600)
plt.show()
```

---



```
plt.figure(figsize=(12,8))
plt.scatter(x_train,y_train,color='blue',marker='*',label='Training Data')
plt.scatter(x_test,y_test,color='red',marker='o',label='Testing Data')
plt.plot(x_test,y_pred,color='black',label='Linear Regression')
plt.title('Population vs Profit (80/20 Train/Test Split)',fontsize=common_font_size + 4)
plt.xlabel('Population of City in 10,000s')
plt.ylabel('Profit in $10,000s')
plt.legend()
plt.grid(True)
plt.savefig('LR Model.jpeg',bbox_inches='tight',dpi=600)
plt.show()
```