Program 1

Write a program to Simple linear regression on one variable using Placement .csv dataset

```
import warnings
warnings.filterwarnings('ignore')
```

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

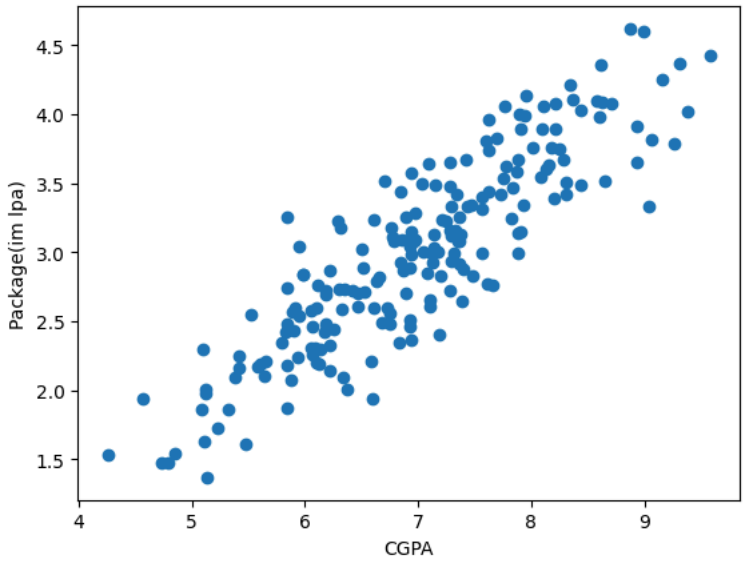
```
df = pd.read_csv('placement.csv')
df.head()
```

	cgpa	package
0	6.89	3.26
1	5.12	1.98
2	7.82	3.25
3	7.42	3.67
4	6.94	3.57

Next steps: Generate code with df | New interactive sheet

```
plt.scatter(df['cgpa'], df['package'])
plt.xlabel('CGPA')
plt.ylabel('Package(im lpa)')
```

```
Text(0, 0.5, 'Package(im lpa)')
```




```
X = df.iloc[:,0:1]
y = df.iloc[:,-1]
y
```

	package
0	3.26
1	1.98
2	3.25
3	3.67
4	3.57
...	...
195	2.46
196	2.57
197	3.24
198	3.96
199	2.33

200 rows × 1 columns

dtype: float64

```
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 = 2)
```

```
from sklearn.linear_model import LinearRegression
lr = LinearRegression()
lr.fit(X_train,y_train)
```

▾ LinearRegression

```
LinearRegression()
```

```
X_test
```

	cgpa
112	8.58
29	7.15
182	5.88
199	6.22
193	4.57
85	4.79
10	5.32
54	6.86
115	8.35
35	6.87
12	8.94
92	7.90
13	6.93
126	5.91
174	7.32
2	7.82
44	5.09
3	7.42
113	6.94
14	7.73
23	6.19
25	7.28
6	6.73
134	7.20
165	8.21
173	6.75
45	7.87
65	7.60
48	8.63
122	5.12
178	8.15
64	7.36
9	8.31
57	6.60
78	6.59
71	7.47
128	7.93
176	6.29
131	6.37
53	6.47

Next steps: Generate code with `X_test` New interactive sheet

```
y_test
```

	package
112	4.10
29	3.49
182	2.08
199	2.33
193	1.94
85	1.48
10	1.86
54	3.09
115	4.21
35	2.87
12	3.65
92	4.00
13	2.89
126	2.60
174	2.99
2	3.25
44	1.86
3	3.67
113	2.37
14	3.42
23	2.48
25	3.65
6	2.60
134	2.83
165	4.08
173	2.56
45	3.58
65	3.81
48	4.09
122	2.01
178	3.63
64	2.92
9	3.51
57	1.94
78	2.21
71	3.34
131	2.01

```
lr.predict(X_test.iloc[2].values.reshape(1,1))
```

```
array([2.38464568])
```

```
plt.scatter(df['cgpa'], df['package'])
plt.plot(X_test,lr.predict(X_test),color='red')
plt.xlabel('CGPA')
plt.ylabel('Package(in lpa)')
```

Program 2

WAP to implement Linear regression on no. of hours studied and score obtained dataset

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score
import warnings
warnings.filterwarnings('ignore')
```

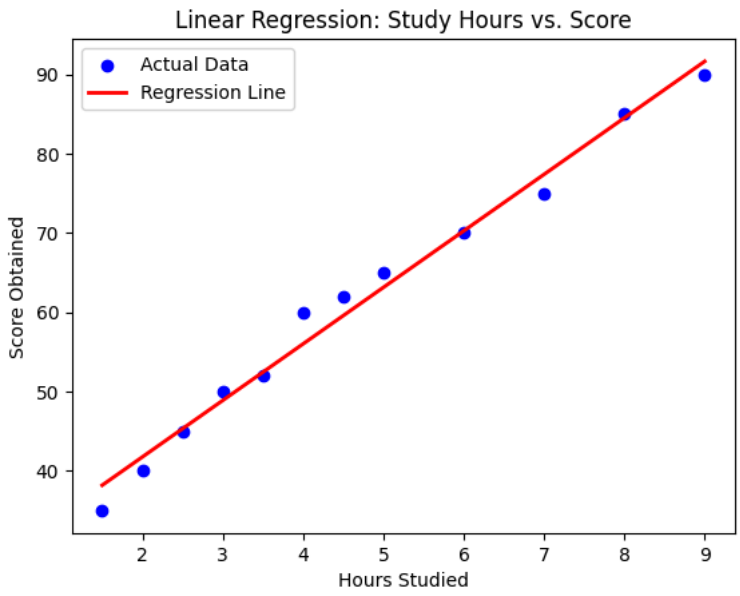
```
data = {
    "Hours_Studied": [1.5, 2.0, 2.5, 3.0, 3.5, 4.0, 4.5, 5.0, 6.0, 7.0, 8.0, 9.0],
    "Score_Obtained": [35, 40, 45, 50, 52, 60, 62, 65, 70, 75, 85, 90]
}
df = pd.DataFrame(data)
```

```
X = df[["Hours_Studied"]]
y = df["Score_Obtained"]

model = LinearRegression()
model.fit(X, y)

y_pred = model.predict(X)
```

```
# Results
print("Intercept:", model.intercept_)
print("Coefficient:", model.coef_[0])
print("R2 Score:", r2_score(y, y_pred))
```

```
Intercept: 27.472513089005233
Coefficient: 7.130890052356022
R2 Score: 0.9851423609797287
```

```
plt.scatter(X, y, color="blue", label="Actual Data")
plt.plot(X, y_pred, color="red", linewidth=2, label="Regression Line")
plt.xlabel("Hours Studied")
plt.ylabel("Score Obtained")
plt.title("Linear Regression: Study Hours vs. Score")
plt.legend()
plt.show()
```




```
hours = np.array([[5.5]])
predicted_score = model.predict(hours)
print(f"Predicted score for 5.5 hours of study: {predicted_score[0]:.2f}")
```

```
Predicted score for 5.5 hours of study: 66.69
```

Program 3

WAP to implement Multiple Linear Regression to predict prices of new homes based on area, bed rooms and age by using homeprices.csv.

```
import warnings
warnings.filterwarnings('ignore')
```

```
import pandas as pd
import numpy as np
from sklearn import linear_model
```

```
df = pd.read_csv('homeprices.csv')
```

```
df.bedrooms.median()
```

```
3.5
```

```
df.bedrooms = df.bedrooms.fillna(df.bedrooms.median())
df
```

	area	bedrooms	age	price
0	2600	3	20	550000
1	3000	4	15	565000
2	3200	3	18	610000
3	3600	3	30	595000
4	4000	5	8	760000
5	4100	6	8	810000

Next steps: Generate code with df | New interactive sheet

```
reg = linear_model.LinearRegression()
reg.fit(df.drop('price',axis = 'columns'),df.price)
```

```
LinearRegression
LinearRegression()
```

```
reg.coef_
```

```
array([  119.67905405, 13097.24903475, -4207.28764479])
```

```
reg.intercept_
```

```
np.float64(256461.14864864858)
```

```
reg.predict(pd.DataFrame([[3000, 3, 25]], columns=['area','bedrooms','age']))
```

```
array([549607.86679537])
```

```
print("Coefficients:", reg.coef_)
print("Intercept:", reg.intercept_)
print("Prediction:", reg.predict([[3000,3,25]]))
```

```
Coefficients: [  119.67905405 13097.24903475 -4207.28764479]
Intercept: 256461.14864864858
Prediction: [549607.86679537]
```

Program 4

WAP to implement Linear regression on Temperature data.

```
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score
import warnings
warnings.filterwarnings("ignore")
```

```
data = pd.read_csv("temperatures.csv")
data.sample(5)
```

	YEAR	JAN	FEB	MAR	APR	MAY	JUN	JUL	AUG	SEP	OCT	NOV	DEC	ANNUAL	JAN-FEB	MAR-MAY	JUN-SEP	OCT-DEC
11	1912	23.70	26.07	28.70	31.29	33.30	33.18	31.05	30.18	30.19	29.64	26.37	23.70	28.95	24.88	31.10	31.15	26.57
107	2008	23.97	25.48	30.34	32.13	33.86	32.15	31.24	30.69	30.92	30.81	28.15	25.91	29.64	24.72	32.11	31.25	28.29
10	1911	23.22	24.58	27.04	31.27	33.78	32.23	31.44	30.80	30.10	29.43	25.70	23.71	28.62	23.90	30.70	31.14	26.31
116	2017	26.45	29.46	31.60	34.95	35.84	33.82	31.88	31.72	32.22	32.29	29.60	27.18	31.42	27.95	34.13	32.41	29.69
115	2016	26.94	29.72	32.62	35.38	35.72	34.03	31.64	31.79	31.66	31.98	30.11	28.01	31.63	28.33	34.57	32.28	30.03

```
X = data[['YEAR']]
y = data['ANNUAL']
```

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

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

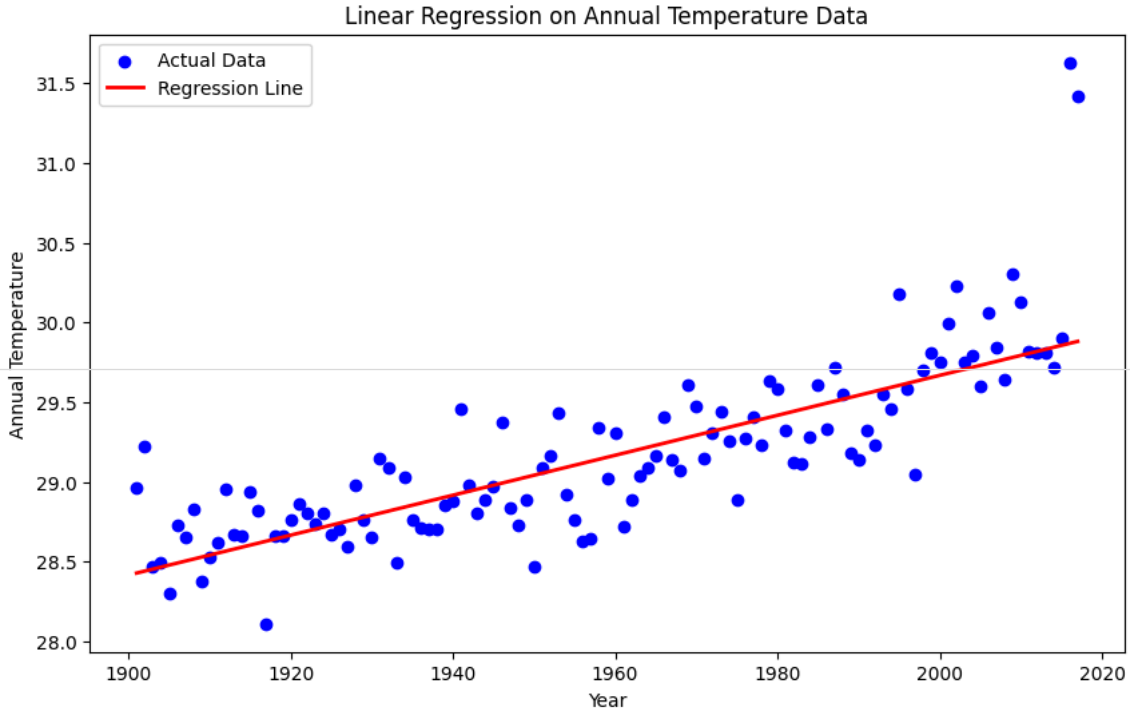
```
▾ LinearRegression
LinearRegression()
```

```
y_pred = model.predict(X_test)
```

```
print("Mean Squared Error:", mean_squared_error(y_test, y_pred))
print("R² Score:", r2_score(y_test, y_pred))
print("Slope (Coefficient):", model.coef_[0])
print("Intercept:", model.intercept_)
```

```
Mean Squared Error: 0.2062573233109893
R² Score: 0.5733931056341353
Slope (Coefficient): 0.012525296221545391
Intercept: 4.617051349163596
```

```
plt.figure(figsize=(10,6))
plt.scatter(X, y, color='blue', label="Actual Data")
plt.plot(X, model.predict(X), color='red', linewidth=2, label="Regression Line")
plt.xlabel("Year")
plt.ylabel("Annual Temperature")
plt.title("Linear Regression on Annual Temperature Data")
plt.legend()
plt.show()
```


Linear Regression on Annual Temperature Data
Actual Data
Regression Line
Annual Temperature
Year
31.5
31.0
30.5
30.0
29.5
29.0
28.5
28.0
1900
1920
1940
1960
1980
2000
2020

Program 5

Write a program to implement Binary logistic regression to predict if a person will buy life insurance based on his age using file insurance_data.csv.

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
import warnings
warnings.filterwarnings("ignore")
```

```
data = pd.read_csv("insurance_data.csv")
data.sample(5)
```

	age	bought_insurance
5	56	1
10	18	0
17	58	1
21	26	0
4	46	1

```
X = data[['age']]
y = data['bought_insurance']
```

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

```
model = LogisticRegression()
model.fit(X_train, y_train)
```

```
▾ LogisticRegression
LogisticRegression()
```

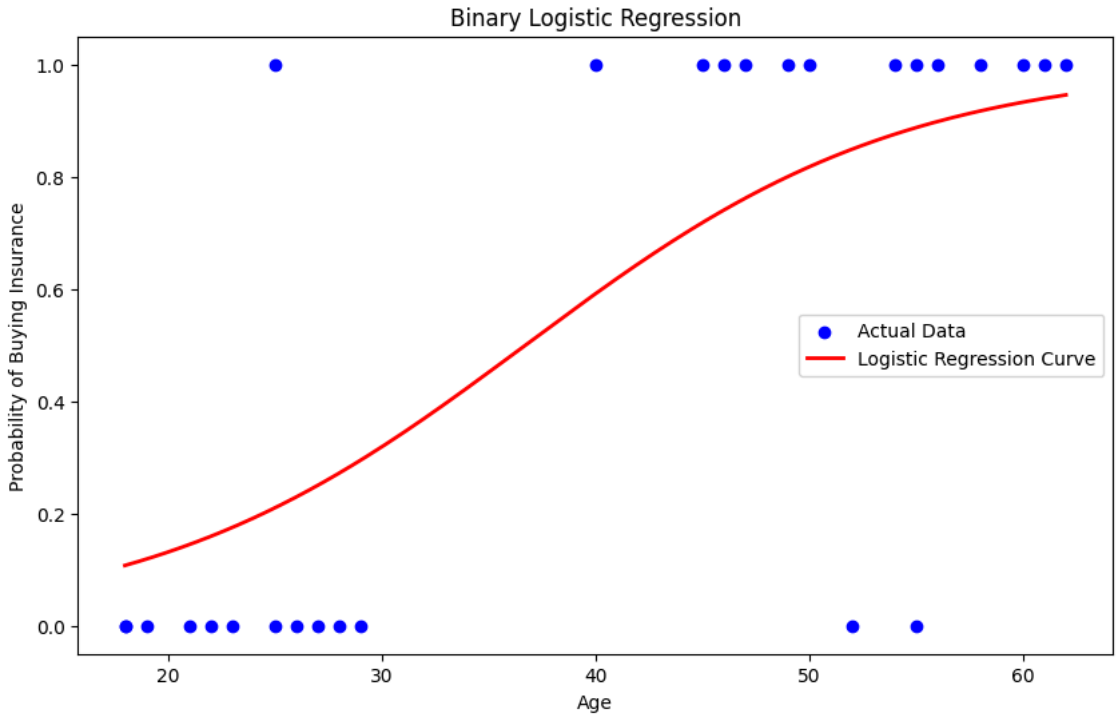
```
y_pred = model.predict(X_test)
```

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

```
Accuracy: 1.0

Confusion Matrix:
 [[4 0]
 [0 2]]

Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00         4
           1       1.00      1.00      1.00         2

    accuracy                           1.00         6
   macro avg       1.00      1.00      1.00         6
weighted avg       1.00      1.00      1.00         6
```

```
X_plot = np.linspace(data['age'].min(), data['age'].max(), 200).reshape(-1,1)
y_prob = model.predict_proba(X_plot)[:,1]

plt.figure(figsize=(10,6))
plt.scatter(data['age'], data['bought_insurance'], color='blue', label="Actual Data")
plt.plot(X_plot, y_prob, color='red', linewidth=2, label="Logistic Regression Curve")
plt.xlabel("Age")
plt.ylabel("Probability of Buying Insurance")
plt.title("Binary Logistic Regression")
plt.legend()
```

```
plt.show()
```


Binary Logistic Regression
Probability of Buying Insurance
Age
Actual Data
Logistic Regression Curve
0.0
0.2
0.4
0.6
0.8
1.0
20
30
40
50
60

Program 6

WAP to Perform Logistic Regression on employee retention dataset available on kaggle.

```
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, OneHotEncoder
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import (
    accuracy_score,
    confusion_matrix,
    classification_report,
    roc_auc_score,
    roc_curve
)
import warnings
warnings.filterwarnings('ignore')
```

```
df = pd.read_csv("WA_Fn-UseC_-HR-Employee-Attrition.csv")
```

```
print(df.head())
print(df["Attrition"].value_counts())
```

```
   Age Attrition     BusinessTravel  DailyRate              Department  \
0   41       Yes      Travel_Rarely       1102                   Sales
1   49        No  Travel_Frequently        279  Research & Development
2   37       Yes      Travel_Rarely       1373  Research & Development
3   33        No  Travel_Frequently       1392  Research & Development
4   27        No      Travel_Rarely        591  Research & Development

   DistanceFromHome  Education EducationField  EmployeeCount  EmployeeNumber  \
0                 1          2  Life Sciences              1               1
1                 8          1  Life Sciences              1               2
2                 2          2          Other              1               4
3                 3          4  Life Sciences              1               5
4                 2          1        Medical              1               7

   ...  RelationshipSatisfaction StandardHours  StockOptionLevel  \
0  ...                         1            80                 0
1  ...                         4            80                 1
2  ...                         2            80                 0
3  ...                         3            80                 0
4  ...                         4            80                 1

   TotalWorkingYears  TrainingTimesLastYear WorkLifeBalance  YearsAtCompany  \
0                  8                      0               1               6
1                 10                      3               3              10
2                  7                      3               3               0
3                  8                      3               3               8
4                  6                      3               3               2

  YearsInCurrentRole  YearsSinceLastPromotion  YearsWithCurrManager
0                  4                        0                     5
1                  7                        1                     7
2                  0                        0                     0
3                  7                        3                     0
4                  2                        2                     2

[5 rows x 35 columns]
Attrition
No     1233
Yes     237
Name: count, dtype: int64
```

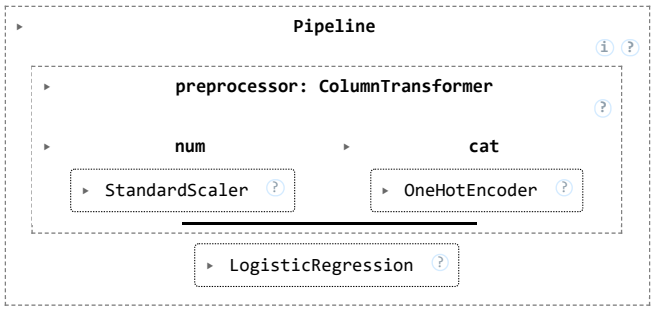
```
df["Attrition_flag"] = df["Attrition"].map({"Yes": 1, "No": 0})

# Features
num_features = [
    "Age", "MonthlyIncome", "DistanceFromHome","YearsAtCompany", "YearsInCurrentRole", "PercentSalaryHike"
]

cat_features = [
    "BusinessTravel", "Department", "EducationField", "JobRole", "MaritalStatus", "OverTime"
]
```

```
X = df[num_features + cat_features]
y = df["Attrition_flag"]
```

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

```
numeric_transformer = Pipeline(steps=[
    ("scaler", StandardScaler())
])

categorical_transformer = Pipeline(steps=[
    ("onehot", OneHotEncoder(drop="first", handle_unknown="ignore"))
])

preprocessor = ColumnTransformer(
    transformers=[
        ("num", numeric_transformer, num_features),
        ("cat", categorical_transformer, cat_features)
    ]
)
```

```
#pipelines
clf = Pipeline(steps=[
    ("preprocessor", preprocessor),
    ("classifier", LogisticRegression(
        solver="liblinear", random_state=42, class_weight="balanced"))
])
# model training
clf.fit(X_train, y_train)
```

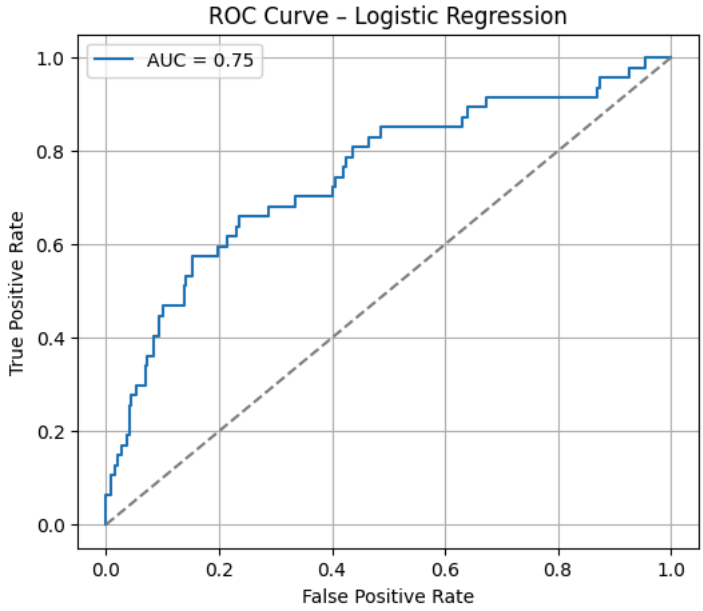



```
y_pred = clf.predict(X_test)
y_proba = clf.predict_proba(X_test)[:, 1]

print("Accuracy:", accuracy_score(y_test, y_pred))
print("\nConfusion Matrix:\n", confusion_matrix(y_test, y_pred))
print("\nClassification Report:\n", classification_report(y_test, y_pred))
print("ROC AUC Score:", roc_auc_score(y_test, y_proba))
```

```
Accuracy: 0.7448979591836735

Confusion Matrix:
 [[190  57]
 [ 18  29]]

Classification Report:
              precision    recall  f1-score   support

           0       0.91      0.77      0.84       247
           1       0.34      0.62      0.44        47

    accuracy                           0.74       294
   macro avg       0.63      0.69      0.64       294
weighted avg       0.82      0.74      0.77       294

ROC AUC Score: 0.7480403135498319
```

```
fpr, tpr, thresholds = roc_curve(y_test, y_proba)

plt.figure(figsize=(6, 5))
plt.plot(fpr, tpr, label=f"AUC = {roc_auc_score(y_test, y_proba):.2f}")
plt.plot([0, 1], [0, 1], linestyle="--", color="gray")
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("ROC Curve – Logistic Regression")
```

```
plt.legend()
plt.grid(True)
plt.show()
```



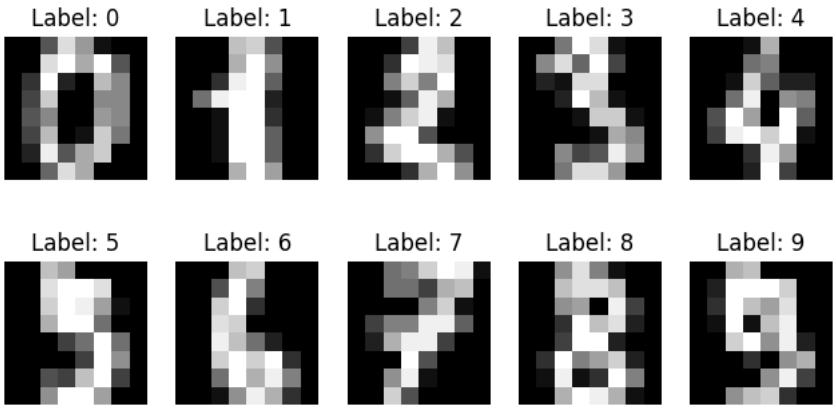
Program 7

WAP to implement Multiclass Logistic Regression on digits dataset

```
import matplotlib.pyplot as plt
from sklearn.datasets import load_digits
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
import seaborn as sns
import warnings
warnings.filterwarnings("ignore")
```

```
digits = load_digits()

X = digits.data
y = digits.target

print("Shape of X:", X.shape)
print("Shape of y:", y.shape)

plt.figure(figsize=(8,4))
for i in range(10):
    plt.subplot(2,5,i+1)
    plt.imshow(digits.images[i], cmap="gray")
    plt.title(f"Label: {digits.target[i]}")
    plt.axis("off")
plt.show()
```

```
Shape of X: (1797, 64)
Shape of y: (1797,)
```




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

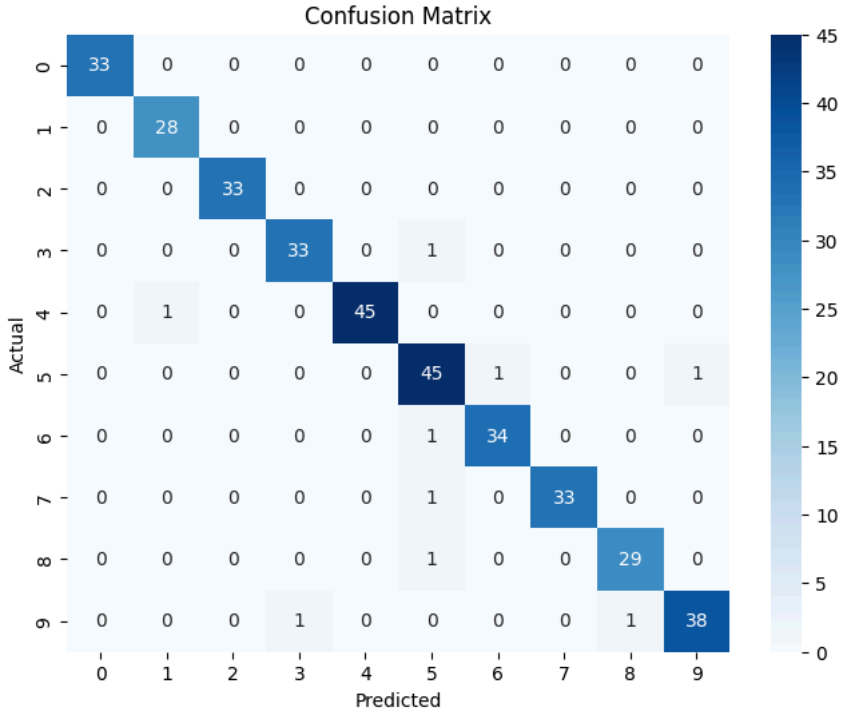
```
model = LogisticRegression(max_iter=10000)
model.fit(X_train, y_train)
```

```
LogisticRegression
LogisticRegression(max_iter=10000)
```

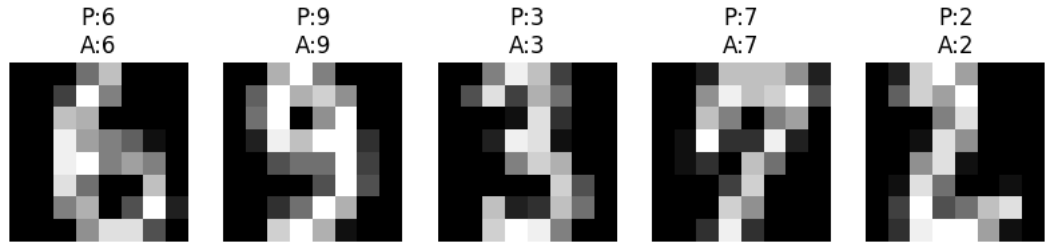
```
y_pred = model.predict(X_test)
```

```
print("Accuracy:", accuracy_score(y_test, y_pred))
print("\nClassification Report:\n", classification_report(y_test, y_pred))

cm = confusion_matrix(y_test, y_pred)

plt.figure(figsize=(8,6))
sns.heatmap(cm, annot=True, cmap="Blues", fmt="d",
            xticklabels=digits.target_names,
            yticklabels=digits.target_names)
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.title("Confusion Matrix")
plt.show()
```

```
Accuracy: 0.975

Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        33
           1       0.97      1.00      0.98        28
           2       1.00      1.00      1.00        33
           3       0.97      0.97      0.97        34
           4       1.00      0.98      0.99        46
           5       0.92      0.96      0.94        47
           6       0.97      0.97      0.97        35
           7       1.00      0.97      0.99        34
           8       0.97      0.97      0.97        30
           9       0.97      0.95      0.96        40

    accuracy                           0.97       360
   macro avg       0.98      0.98      0.98       360
weighted avg       0.98      0.97      0.98       360
```




```
plt.figure(figsize=(10,4))
for i in range(5):
    plt.subplot(1,5,i+1)
    plt.imshow(X_test[i].reshape(8,8), cmap="gray")
    plt.title(f"P:{y_pred[i]}\nA:{y_test[i]}")
    plt.axis("off")
plt.show()
```



Program 8

WAP to implement Logistic Regression on Iris Dataset.

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

from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
import warnings
warnings.filterwarnings('ignore')
```

```
iris = load_iris()
df = pd.DataFrame(iris.data, columns=iris.feature_names)
df["target"] = iris.target
df["species"] = df["target"].map({0: "setosa", 1: "versicolor", 2: "virginica"})
df.head()
```

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	target	species
0	5.1	3.5	1.4	0.2	0	setosa
1	4.9	3.0	1.4	0.2	0	setosa
2	4.7	3.2	1.3	0.2	0	setosa
3	4.6	3.1	1.5	0.2	0	setosa
4	5.0	3.6	1.4	0.2	0	setosa

Next steps: Generate code with df | New interactive sheet

```
X = df[iris.feature_names]
y = df["target"]

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

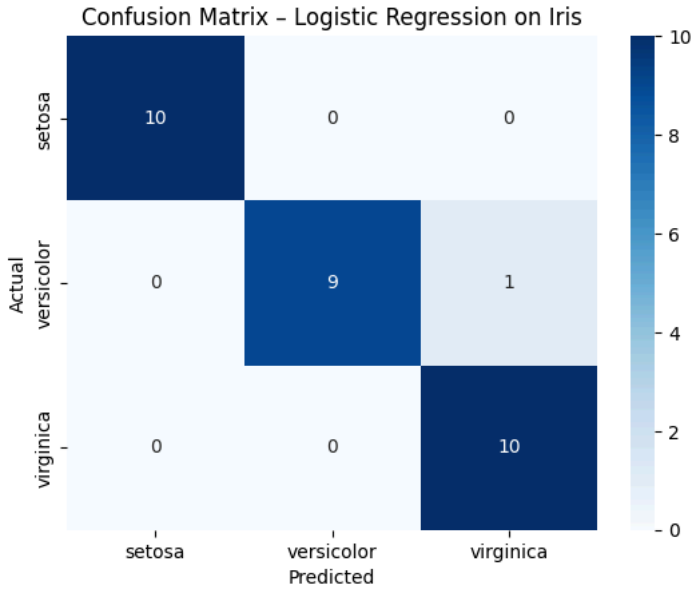
```
# Logistic regression model
model = LogisticRegression(max_iter=200, random_state=42)
# model training
model.fit(X_train, y_train)
```

```
LogisticRegression
LogisticRegression(max_iter=200, random_state=42)
```

```
# Predictions
y_pred = model.predict(X_test)

print("Accuracy:", accuracy_score(y_test, y_pred))
print("\nConfusion Matrix:\n", confusion_matrix(y_test, y_pred))
print("\nClassification Report:\n", classification_report(y_test, y_pred, target_names=iris.target_names))
```

```
Accuracy: 0.9666666666666667

Confusion Matrix:
 [[10  0  0]
 [ 0  9  1]
 [ 0  0 10]]

Classification Report:
              precision    recall  f1-score   support

      setosa       1.00      1.00      1.00        10
  versicolor       1.00      0.90      0.95        10
   virginica       0.91      1.00      0.95        10

    accuracy                           0.97        30
   macro avg       0.97      0.97      0.97        30
weighted avg       0.97      0.97      0.97        30
```

```
cm = confusion_matrix(y_test, y_pred)
sns.heatmap(cm, annot=True, fmt="d", cmap="Blues",
            xticklabels=iris.target_names,
            yticklabels=iris.target_names)
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.title("Confusion Matrix – Logistic Regression on Iris")
plt.show()
```



Program 9

WAP to implement Decision Tree Classifier on drug.csv dataset

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier, plot_tree
from sklearn.preprocessing import LabelEncoder
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings("ignore")
```

```
df = pd.read_csv("drug.csv")
df.sample(5)
```

	Age	Sex	BP	Cholesterol	Na_to_K	Drug
59	34	M	HIGH	HIGH	18.703	DrugY
30	18	F	NORMAL	NORMAL	8.750	drugX
89	50	F	NORMAL	NORMAL	17.211	DrugY
13	74	F	LOW	HIGH	20.942	DrugY
141	64	F	LOW	NORMAL	25.741	DrugY

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 200 entries, 0 to 199
Data columns (total 6 columns):
 #   Column       Non-Null Count  Dtype  
---  ------       --------------  -----  
 0   Age          200 non-null    int64  
 1   Sex          200 non-null    object 
 2   BP           200 non-null    object 
 3   Cholesterol  200 non-null    object 
 4   Na_to_K      200 non-null    float64
 5   Drug         200 non-null    object 
dtypes: float64(1), int64(1), object(4)
memory usage: 9.5+ KB
```

```
# Encode categorical columns
le_sex = LabelEncoder()
df["Sex"] = le_sex.fit_transform(df["Sex"])

le_bp = LabelEncoder()
df["BP"] = le_bp.fit_transform(df["BP"])

le_chol = LabelEncoder()
df["Cholesterol"] = le_chol.fit_transform(df["Cholesterol"])

# Encode target variable
le_drug = LabelEncoder()
df["Drug"] = le_drug.fit_transform(df["Drug"])

df.head()
```

	Age	Sex	BP	Cholesterol	Na_to_K	Drug
0	23	0	0	0	25.355	0
1	47	1	1	0	13.093	3
2	47	1	1	0	10.114	3
3	28	0	2	0	7.798	4
4	61	0	1	0	18.043	0

Next steps: Generate code with df | New interactive sheet

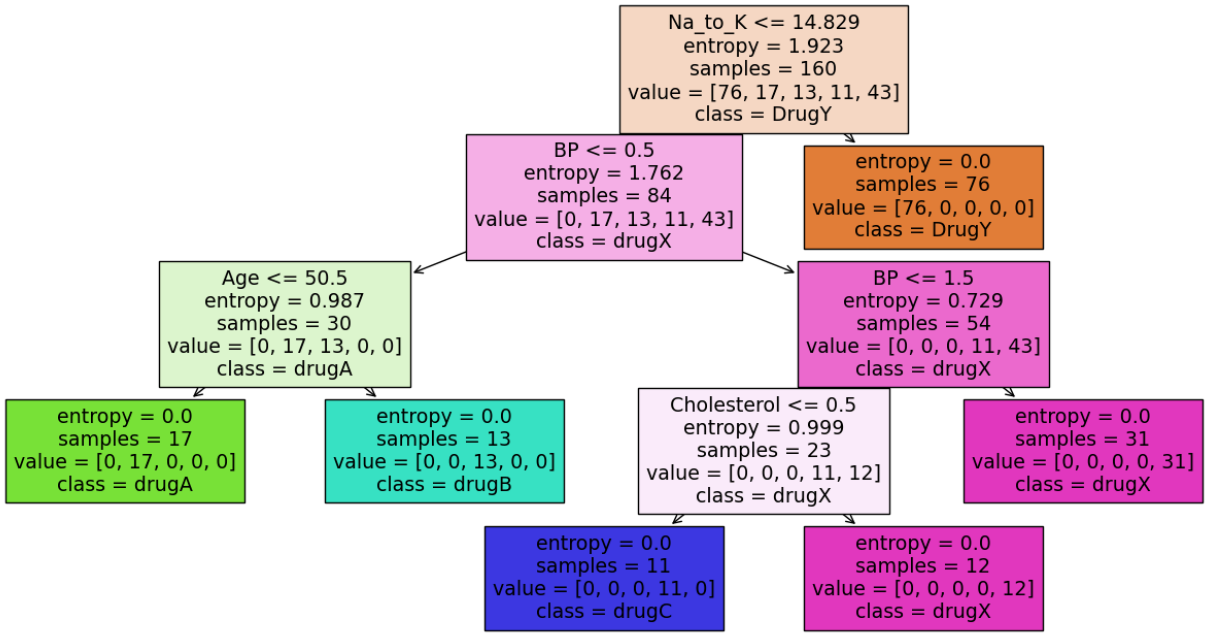
```
X = df.drop("Drug", axis=1)
y = df["Drug"]
```

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

```
clf = DecisionTreeClassifier(criterion="entropy", max_depth=4, random_state=42)
clf.fit(X_train, y_train)
```

```
DecisionTreeClassifier(criterion='entropy', max_depth=4, random_state=42)
```

```
plt.figure(figsize=(16,8))
plot_tree(clf, feature_names=X.columns, class_names=le_drug.classes_, filled=True)
plt.show()
```



Program 10

Write a program to implement the naive Bayesian classifier on titanic .CSV file.

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import accuracy_score, classification_report
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import confusion_matrix
import warnings
warnings.filterwarnings("ignore")
```

```
titanic = pd.read_csv("titanic.csv")
titanic.sample(5)
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
329	330	1	1	Hippach, Miss. Jean Gertrude	female	16.0	0	1	111361	57.9792	B18	C
201	202	0	3	Sage, Mr. Frederick	male	NaN	8	2	CA. 2343	69.5500	NaN	S
493	494	0	1	Artagaveytia, Mr. Ramon	male	71.0	0	0	PC 17609	49.5042	NaN	C
83	84	0	1	Carrau, Mr. Francisco M	male	28.0	0	0	113059	47.1000	NaN	S

```
titanic = titanic.drop(["PassengerId", "Name", "Ticket", "Cabin"], axis=1)
```

```
# Fill missing Age with median
titanic["Age"].fillna(titanic["Age"].median(), inplace=True)

# Fill missing Embarked with most frequent value
titanic["Embarked"].fillna(titanic["Embarked"].mode()[0], inplace=True)
```

```
le = LabelEncoder()

titanic["Sex"] = le.fit_transform(titanic["Sex"])
titanic["Embarked"] = le.fit_transform(titanic["Embarked"])
```

```
X = titanic.drop("Survived", axis=1)
y = titanic["Survived"]
```

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

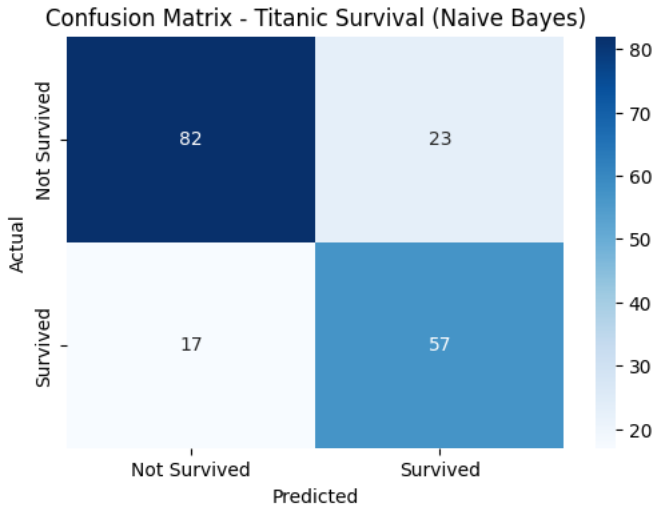
```
nb = GaussianNB()
nb.fit(X_train, y_train)
```

```
▾ GaussianNB
GaussianNB()
```

```
y_pred = nb.predict(X_test)

print("Accuracy:", accuracy_score(y_test, y_pred))
print("\nClassification Report:\n", classification_report(y_test, y_pred))
```

```
Accuracy: 0.776536312849162

Classification Report:
              precision    recall  f1-score   support

           0       0.83      0.78      0.80       105
           1       0.71      0.77      0.74        74

    accuracy                           0.78       179
   macro avg       0.77      0.78      0.77       179
weighted avg       0.78      0.78      0.78       179
```

```
# Compute confusion matrix
cm = confusion_matrix(y_test, y_pred)

# Plot heatmap
plt.figure(figsize=(6,4))
sns.heatmap(cm, annot=True, fmt="d", cmap="Blues",
            xticklabels=["Not Survived", "Survived"],
            yticklabels=["Not Survived", "Survived"])
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.title("Confusion Matrix - Titanic Survival (Naive Bayes)")
plt.show()
```



Program 11

Write a program to implement the naive Bayesian classifier on email dataset spam.csv file.

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings("ignore")
```

```
spam = pd.read_csv("spam.csv", encoding="latin-1")
spam.sample(5)
```

	v1	v2	Unnamed: 2	Unnamed: 3	Unnamed: 4
4587	ham	I wanted to wish you a Happy New Year and I wa...	NaN	NaN	NaN
3402	ham	Good night my dear.. Sleepwell&Take care	NaN	NaN	NaN
2830	ham	Thanx 4 sending me home...	NaN	NaN	NaN
3520	ham	Hey... are you going to quit soon? Xuhui and i...	NaN	NaN	NaN
2746	ham	K da:)how many page you want?	NaN	NaN	NaN

```
# Keep only necessary columns
spam = spam.rename(columns={"v1": "label", "v2": "message"})
spam = spam[["label", "message"]]

# Encode labels: ham=0, spam=1
spam["label"] = spam["label"].map({"ham": 0, "spam": 1})
```

```
X = spam["message"]
y = spam["label"]

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

```
nb = MultinomialNB()
nb.fit(X_train_tfidf.toarray(), y_train)
```

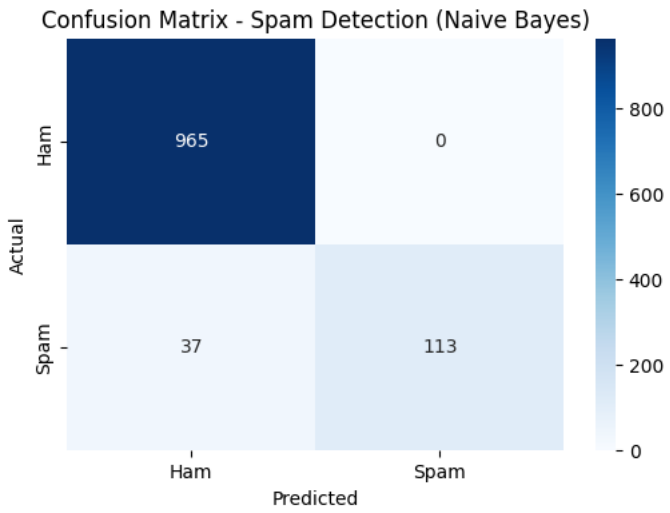
```
▾ MultinomialNB
MultinomialNB()
```

```
y_pred = nb.predict(X_test_tfidf.toarray())

print("Accuracy:", accuracy_score(y_test, y_pred))
print("\nClassification Report:\n", classification_report(y_test, y_pred))
```

```
Accuracy: 0.9668161434977578

Classification Report:
              precision    recall  f1-score   support

           0       0.96      1.00      0.98       965
           1       1.00      0.75      0.86       150

    accuracy                           0.97      1115
   macro avg       0.98      0.88      0.92      1115
weighted avg       0.97      0.97      0.96      1115
```

```
cm = confusion_matrix(y_test, y_pred)

plt.figure(figsize=(6,4))
sns.heatmap(cm, annot=True, fmt="d", cmap="Blues",
            xticklabels=["Ham", "Spam"],
            yticklabels=["Ham", "Spam"])
plt.xlabel("Predicted")
plt.ylabel("Actual")
```

```
plt.title("Confusion Matrix - Spam Detection (Naive Bayes)")
plt.show()
```




```
# Example test messages
custom_messages = [
    "Congratulations! You won $1000. Click here to claim your prize now!", "Hey, are we still meeting for lunch today?", "Lowes
    "Dear friend, I hope you are doing well. Let's catch up soon."
]
# TF-IDF
custom_tfidf = vectorizer.transform(custom_messages)
predictions = nb.predict(custom_tfidf.toarray())

for msg, label in zip(custom_messages, predictions):
    print(f"Message: {msg}\nPrediction: {'Spam' if label == 1 else 'Ham'}\n")
```

```
Message: Congratulations! You won $1000. Click here to claim your prize now!
Prediction: Spam

Message: Hey, are we still meeting for lunch today?
Prediction: Ham

Message: Lowest price guaranteed! Buy cheap meds online now!
Prediction: Ham

Message: Dear friend, I hope you are doing well. Let's catch up soon.
Prediction: Ham
```

Program 12

Write a program to implement k-nearest neighbours (KNN) on iris.csv dataset.

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

from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
import warnings
warnings.filterwarnings('ignore')
```

```
iris = load_iris()

df = pd.DataFrame(iris.data, columns=iris.feature_names)
df["species"] = iris.target

# Mapping
df["species"] = df["species"].map({0: "setosa", 1: "versicolor", 2: "virginica"})
df.head()
```

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	species
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5.0	3.6	1.4	0.2	setosa

Next steps: Generate code with df | New interactive sheet

```
X = df[iris.feature_names]
y = df["species"]
```

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

```
# KNN Model
knn = KNeighborsClassifier(n_neighbors=5)

# training
knn.fit(X_train, y_train)
```

```
KNeighborsClassifier
KNeighborsClassifier()
```

```
y_pred = knn.predict(X_test)

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

```
Accuracy: 1.0

Confusion Matrix:
 [[10  0  0]
 [ 0 10  0]
 [ 0  0 10]]

Classification Report:
              precision    recall  f1-score   support

      setosa       1.00      1.00      1.00        10
  versicolor       1.00      1.00      1.00        10
   virginica       1.00      1.00      1.00        10
```

```
    accuracy                           1.00        30
   macro avg       1.00      1.00      1.00        30
weighted avg       1.00      1.00      1.00        30
```

```
cm = confusion_matrix(y_test, y_pred)
sns.heatmap(cm, annot=True, fmt="d", cmap="Blues",
            xticklabels=knn.classes_,
            yticklabels=knn.classes_)
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.title("Confusion Matrix – KNN on Iris")
plt.show()
```