FIND-S:

```
import csv
data = list(csv.reader(open('csv1.csv')))
hypothesis = data[1][:6]
for i, row in enumerate(data):
    if row[6] == 'yes':
        hypothesis = ['?' if row[j] != hypothesis[j] else hypothesis[j] for j in range(6)]
    print(f"Instance {i}: {hypothesis}")
print("\nMaximally specific hypothesis:", hypothesis)
```

OUTPUT:

```
Instance 0: ['sunny', 'warm', 'normal', 'strong', 'warm', 'same']
Instance 1: ['sunny', 'warm', 'normal', 'strong', 'warm', 'same']
Instance 2: ['sunny', 'warm', '?', 'strong', 'warm', 'same']
Instance 3: ['sunny', 'warm', '?', 'strong', 'warm', 'same']
Instance 4: ['sunny', 'warm', '?', 'strong', '?', '?']

Maximally specific hypothesis: ['sunny', 'warm', '?', 'strong', '?', '?']
```

CANDIDATE ELIMINATION:

```
import pandas as pd
data = pd.read_csv('csv1.csv').values
s = data[0][:-1].copy()
g = [["?"] * len(s) for _ in range(len(s))]
```

```
for i, row in enumerate(data):
    if row[-1] == "yes":
        s = ['?' if s[j] != row[j] else s[j] for j in range(len(s))]
    else:
        for j in range(len(s)):
            if s[j] != row[j]: g[j][j] = s[j]
g = [hyp for hyp in g if '?' in hyp][:2]
print("\nFinal Specific Hypothesis (s):", s)
print("\nFinal General Hypotheses (g):", g)
```

OUTPUT:

Final Specific Hypothesis (s): ['sunny', 'warm', '?', 'strong', '?', '?']

Final General Hypotheses (g): [['sunny', '?', '?', '?', '?', '?'], ['?', 'warm', '?', '?', '?', '?']]

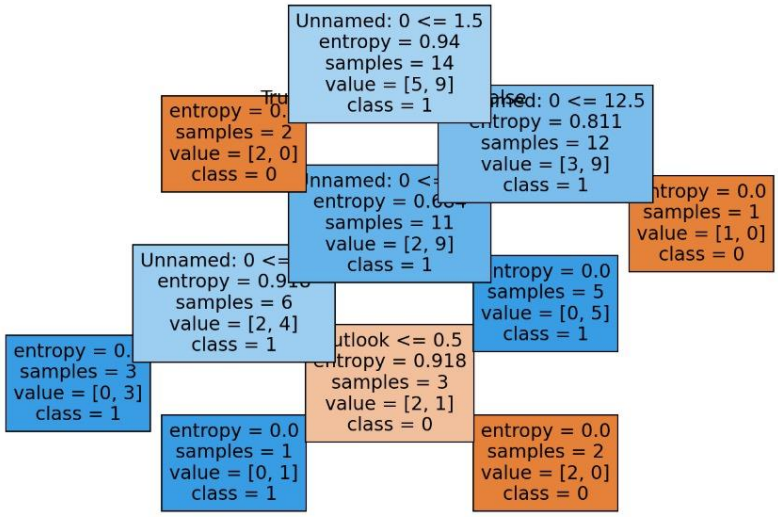
DECISION TREE:

```
import pandas as pd
from sklearn import tree
import matplotlib.pyplot as plt
from sklearn.preprocessing import LabelEncoder
data = pd.read_csv('tennis.csv')
print(data.head())
label_encoders = {}
for column in data.columns:
    if data[column].dtype == 'object':
        le = LabelEncoder()
```

```
        data[column] = le.fit_transform(data[column])
        label_encoders[column] = le
print(data.head())
X = data.iloc[:, :-1]  # Features
y = data.iloc[:, -1]   # Target
clf = tree.DecisionTreeClassifier(criterion='entropy')  # Use entropy to mimic ID3
clf = clf.fit(X, y)
plt.figure(figsize=(20, 10))  # Increase the size for better readability
tree.plot_tree(clf, filled=True, feature_names=X.columns, class_names=[str(cls) for cls in
clf.classes_])
plt.show()
```

OUTPUT:

```
Unnamed: 0   Outlook Temperature Humidity    Wind PlayTennis
0           0    Sunny         Hot     High    Weak        No
1           1    Sunny         Hot     High  Strong        No
2           2  Overcast        Hot     High    Weak       Yes
3           3     Rain        Mild     High    Weak       Yes
4           4     Rain        Cool   Normal    Weak       Yes
  Unnamed: 0  Outlook  Temperature  Humidity  Wind  PlayTennis
0           0        2           1         0     1           0
1           1        2           1         0     0           0
2           2        0           1         0     1           1
3           3        1           2         0     1           1
4           4        1           0         1     1           1
```



NAÏVE BAYES:

```
import pandas as pd

from sklearn.model_selection import train_test_split

from sklearn.naive_bayes import GaussianNB

from sklearn.metrics import accuracy_score, precision_score, recall_score

data = pd.read_csv('Titanic.csv').drop(['PassengerId', 'Name', 'Ticket', 'Cabin'], axis=1).dropna()

data['Sex'] = data['Sex'].map({'male': 0, 'female': 1})

data['Embarked'] = data['Embarked'].map({'C': 0, 'Q': 1, 'S': 2})

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

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

classifier = GaussianNB().fit(X_train, y_train)

y_pred = classifier.predict(X_test)
```

```
print(f'Accuracy: {accuracy_score(y_test, y_pred):.2f}')
print(f'Precision: {precision_score(y_test, y_pred):.2f}')
print(f'Recall: {recall_score(y_test, y_pred):.2f}')
```

OUTPUT:

```
Accuracy: 0.76
Precision: 0.73
Recall: 0.73
```

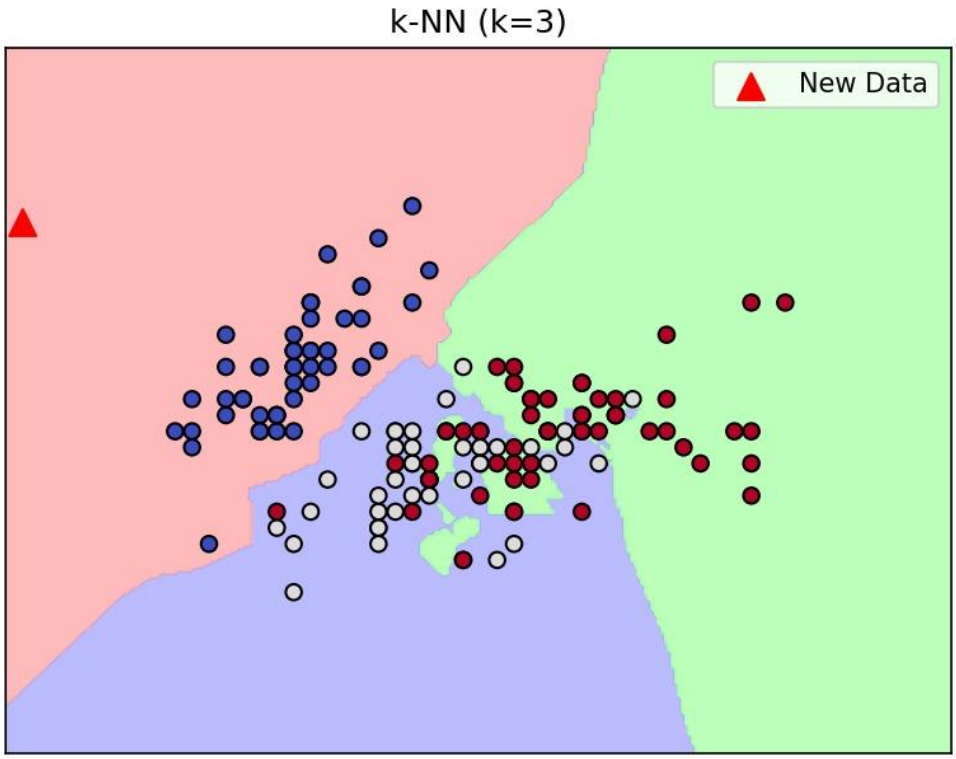
KNN:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from matplotlib.colors import ListedColormap
df = pd.read_csv('Iris.csv')
X = df[['SepalLengthCm', 'SepalWidthCm']].values
label_encoder = LabelEncoder()
y = label_encoder.fit_transform(df['Species'])
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
k = int(input("Enter k: "))
clf = KNeighborsClassifier(n_neighbors=k).fit(X_train, y_train)
try:
```

```
    new_data = np.array([list(map(float, input("Enter new data (sepal_length,sepal_width): ").split(',')))])
    pred = clf.predict(new_data)
    print("Prediction:", label_encoder.inverse_transform(pred))
    h = .02
    xx, yy = np.meshgrid(np.arange(X[:, 0].min() - 1, X[:, 0].max() + 1, h),
                         np.arange(X[:, 1].min() - 1, X[:, 1].max() + 1, h))
    Z = clf.predict(np.c_[xx.ravel(), yy.ravel()]).reshape(xx.shape)
    plt.contourf(xx, yy, Z, cmap=ListedColormap(['#FFAAAA', '#AAAAFF', '#AAFFAA']), alpha=0.8)
    plt.scatter(X[:, 0], X[:, 1], c=y, edgecolor='k', marker='o', cmap='coolwarm')
    plt.scatter(new_data[:, 0], new_data[:, 1], c='red', marker='^', s=100, label='New Data')
    plt.xlim(xx.min(), xx.max())
    plt.ylim(yy.min(), yy.max())
    plt.xticks([]), plt.yticks([])
    plt.title(f'k-NN (k={k})')
    plt.legend()
    plt.show()
except ValueError:
    print("Invalid input. Use format: sepal_length,sepal_width.")
```

OUTPUT:

```
Enter k: 3
Enter new data (sepal_length,sepal_width): 3.4,4.3
Prediction: ['Iris-setosa']
```

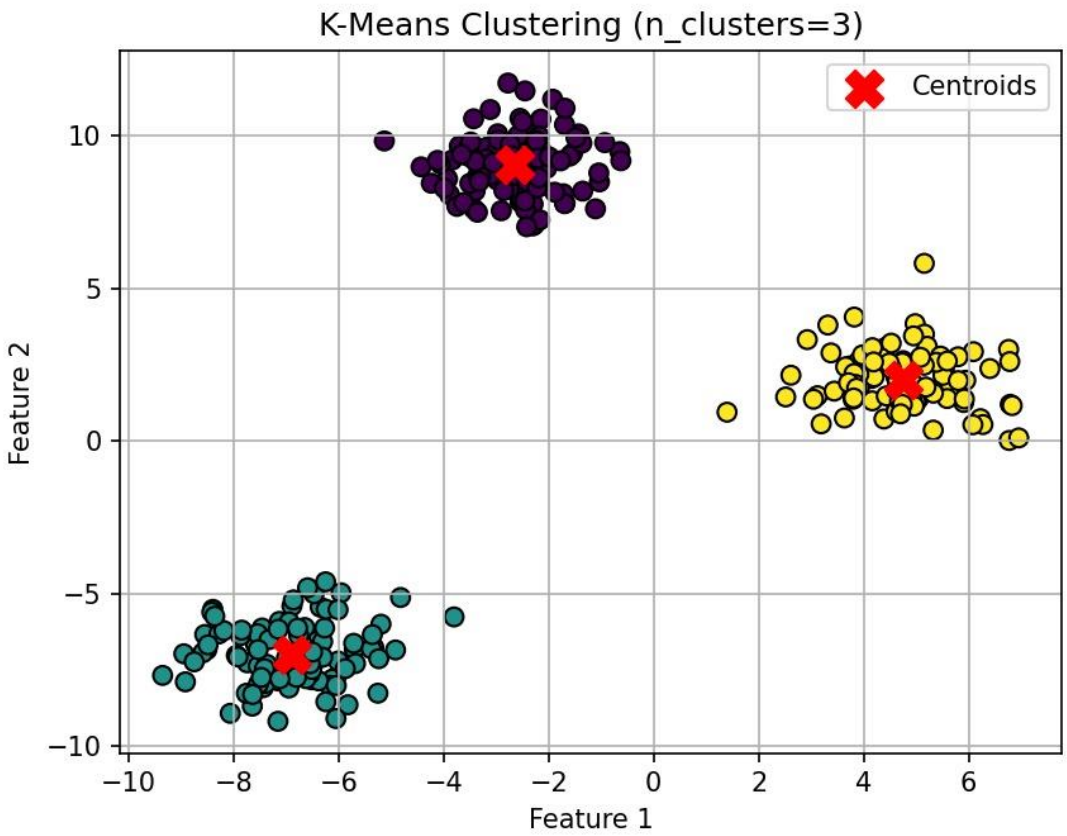


KMeans:

```
import numpy as np

import matplotlib.pyplot as plt

from sklearn.datasets import make_blobs

from sklearn.cluster import KMeans

n_clusters = int(input("Enter the number of clusters: "))

X, _ = make_blobs(n_samples=300, centers=n_clusters, random_state=42)

kmeans = KMeans(n_clusters=n_clusters, random_state=42).fit(X)

plt.scatter(X[:, 0], X[:, 1], c=kmeans.labels_, cmap='viridis', marker='o', edgecolor='k', s=50)

plt.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:, 1], c='red', marker='X',
s=200, label='Centroids')
```

```
plt.title(f'K-Means Clustering (n_clusters={n_clusters})')
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
plt.legend()
plt.grid()
plt.show()
```

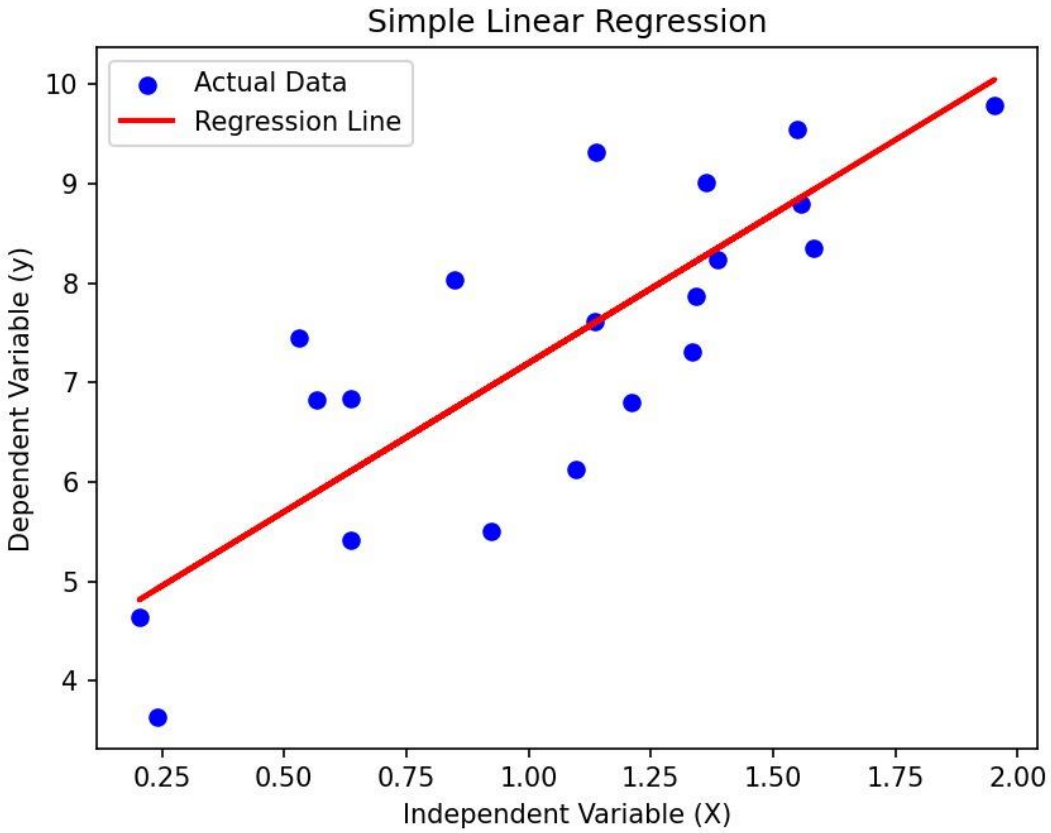
OUTPUT:



LINEAR REGRESSION

```
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 LinearRegression
np.random.seed(0)
X = 2 * np.random.rand(100, 1)  # Independent variable
y = 4 + 3 * X + np.random.randn(100, 1)  # Dependent variable with some noise
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
model = LinearRegression().fit(X_train, y_train)
y_pred = model.predict(X_test)
plt.scatter(X_test, y_test, color='blue', label='Actual Data')
plt.plot(X_test, y_pred, color='red', linewidth=2, label='Regression Line')
plt.title('Simple Linear Regression')
plt.xlabel('Independent Variable (X)')
plt.ylabel('Dependent Variable (y)')
plt.legend()
plt.show()
print(f'Intercept: {model.intercept_[0]:.2f}')
print(f'Slope: {model.coef_[0][0]:.2f}')
```

OUTPUT:


Simple Linear Regression
Actual Data
Regression Line
Dependent Variable (y)
Independent Variable (X)
10
9
8
7
6
5
4
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00