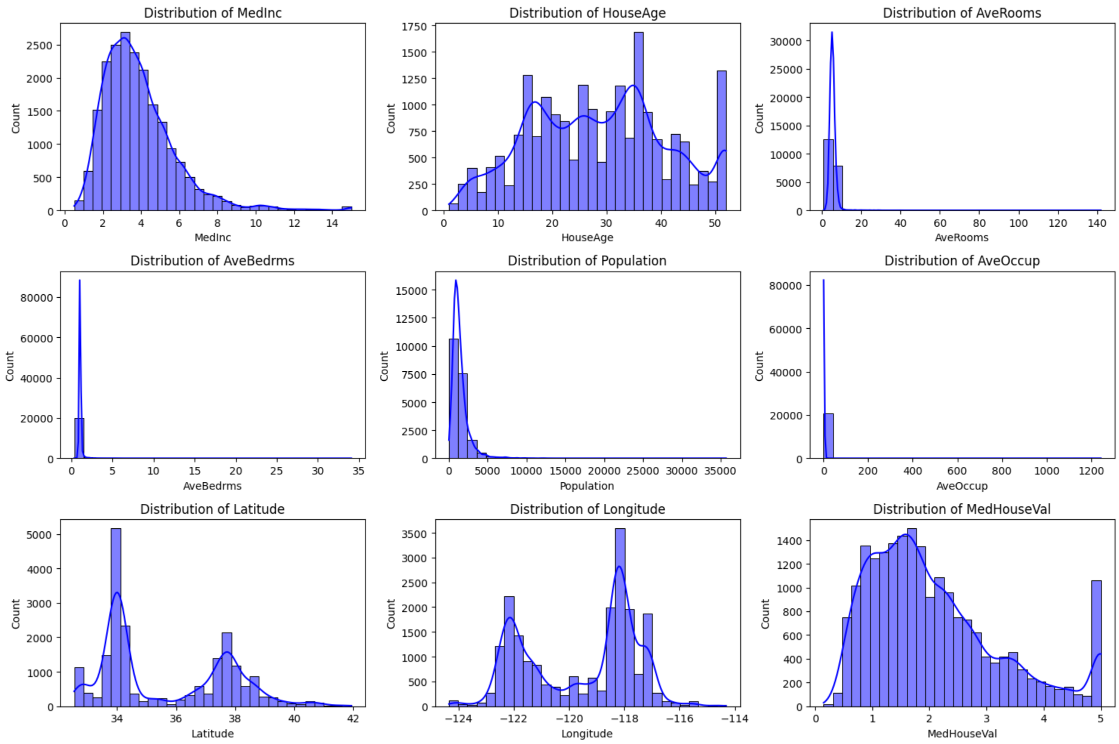
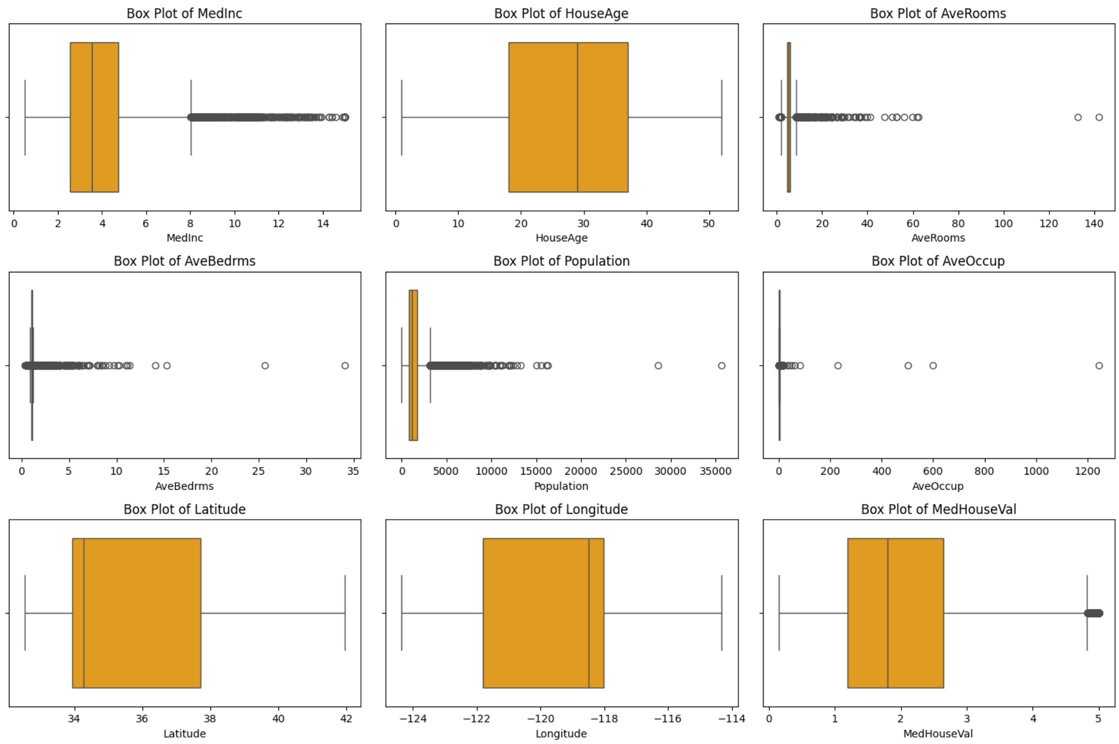
Program_1

June 8, 2025

[1]:
```python
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.datasets import fetch_california_housing

cali = fetch_california_housing(as_frame=True)
housing_df = cali.frame

numerical_features = housing_df.select_dtypes(include=[np.number]).columns

plt.figure(figsize=(15, 10))
for i, feature in enumerate(numerical_features):
    plt.subplot(3, 3, i + 1)
    sns.histplot(housing_df[feature], kde=True, bins=30, color='blue')
    plt.title(f'Distribution of {feature}')
plt.tight_layout()
plt.show()

plt.figure(figsize=(15, 10))
for i, feature in enumerate(numerical_features):
    plt.subplot(3, 3, i + 1)
    sns.boxplot(x=housing_df[feature], color='orange')
    plt.title(f'Box Plot of {feature}')
plt.tight_layout()
plt.show()

print("Outliers Detection:")
outliers_summary = {}
for feature in numerical_features:
    Q1 = housing_df[feature].quantile(0.25)
    Q3 = housing_df[feature].quantile(0.75)
    IQR = Q3 - Q1
    lower_bound = Q1 - 1.5 * IQR
    upper_bound = Q3 + 1.5 * IQR
    outliers = housing_df[(housing_df[feature] < lower_bound) |␣
 ↪(housing_df[feature] > upper_bound)]
```

```
    outliers_summary[feature] = len(outliers)
    print(f"{feature}: {len(outliers)} outliers")

print("\nDataset Summary:")
print(housing_df.describe())
```





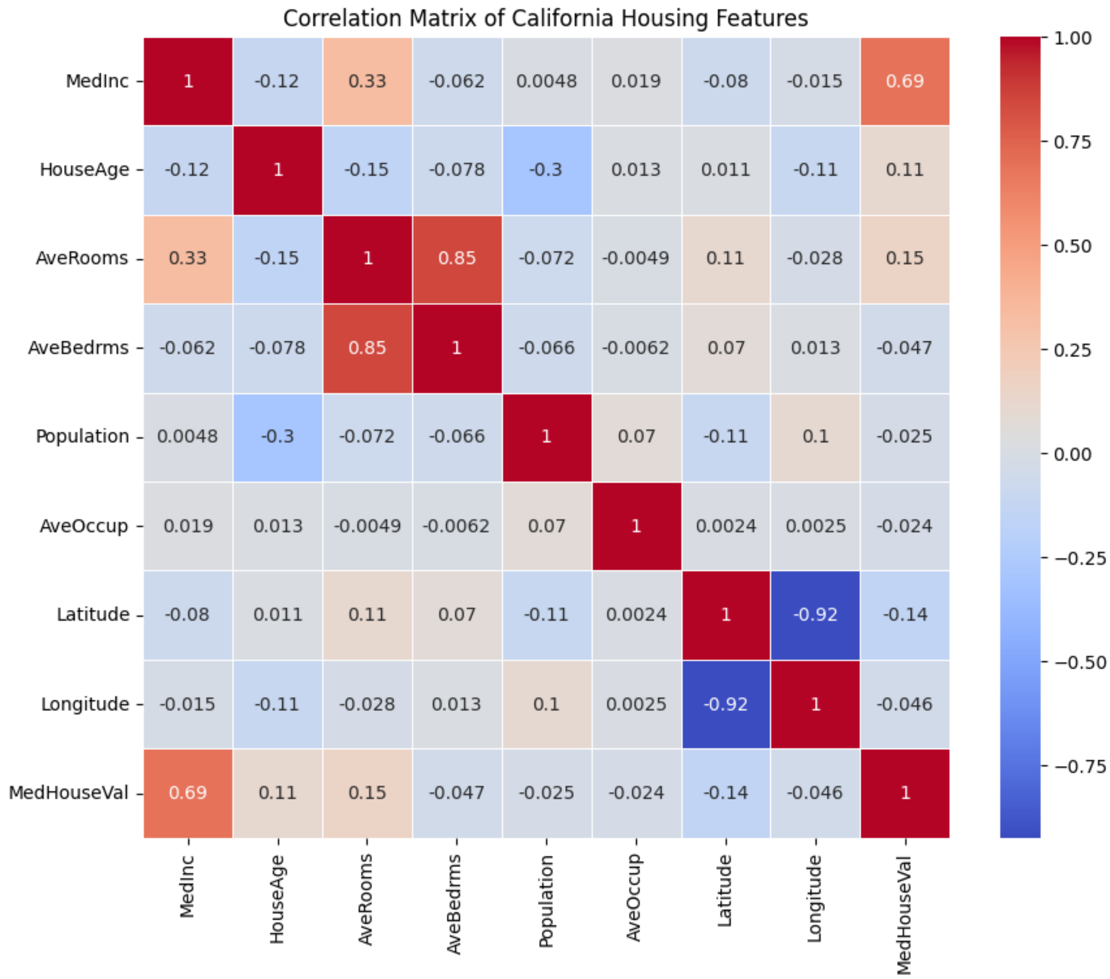

```
Outliers Detection:
MedInc: 681 outliers
HouseAge: 0 outliers
AveRooms: 511 outliers
AveBedrms: 1424 outliers
Population: 1196 outliers
AveOccup: 711 outliers
Latitude: 0 outliers
Longitude: 0 outliers
MedHouseVal: 1071 outliers

Dataset Summary:
             MedInc      HouseAge      AveRooms     AveBedrms    Population  \
count  20640.000000  20640.000000  20640.000000  20640.000000  20640.000000
mean       3.870671     28.639486      5.429000      1.096675   1425.476744
std        1.899822     12.585558      2.474173      0.473911   1132.462122
min        0.499900      1.000000      0.846154      0.333333      3.000000
25%        2.563400     18.000000      4.440716      1.006079    787.000000
50%        3.534800     29.000000      5.229129      1.048780   1166.000000
75%        4.743250     37.000000      6.052381      1.099526   1725.000000
max       15.000100     52.000000    141.909091     34.066667  35682.000000

           AveOccup      Latitude     Longitude   MedHouseVal
count  20640.000000  20640.000000  20640.000000  20640.000000
```

mean	3.070655	35.631861	-119.569704	2.068558
std	10.386050	2.135952	2.003532	1.153956
min	0.692308	32.540000	-124.350000	0.149990
25%	2.429741	33.930000	-121.800000	1.196000
50%	2.818116	34.260000	-118.490000	1.797000
75%	3.282261	37.710000	-118.010000	2.647250
max	1243.333333	41.950000	-114.310000	5.000010

Program_2

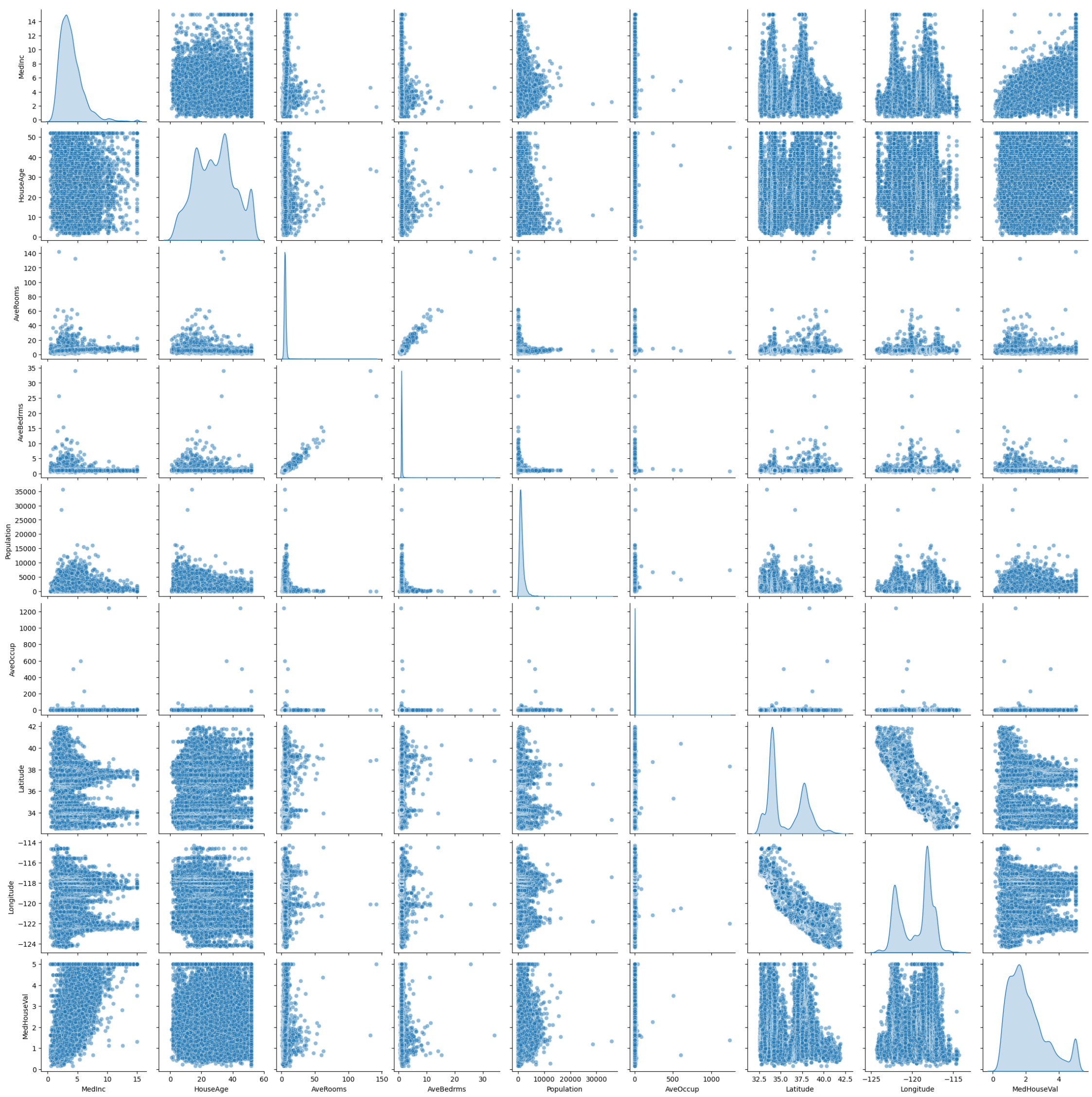
June 8, 2025

[1]:
```python
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.datasets import fetch_california_housing

cali = fetch_california_housing(as_frame=True)
data = cali.frame

numeric_data = data.select_dtypes(include=[float, int])
correlation_matrix = numeric_data.corr()

plt.figure(figsize=(10, 8))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', linewidths=0.5)
plt.title('Correlation Matrix of California Housing Features')
plt.show()

sns.pairplot(data, diag_kind='kde', plot_kws={'alpha': 0.5})
plt.suptitle('Pair Plot of California Housing Features', y=1.02)
plt.show()
```



Correlation Matrix of California Housing Features
MedInc
HouseAge
AveRooms
AveBedrms
Population
AveOccup
Latitude
Longitude
MedHouseVal
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75

Pair Plot of California Housing Features



Program_3

June 8, 2025

[1]:
```python
from sklearn.datasets import load_iris
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler
import pandas as pd
import matplotlib.pyplot as plt
```

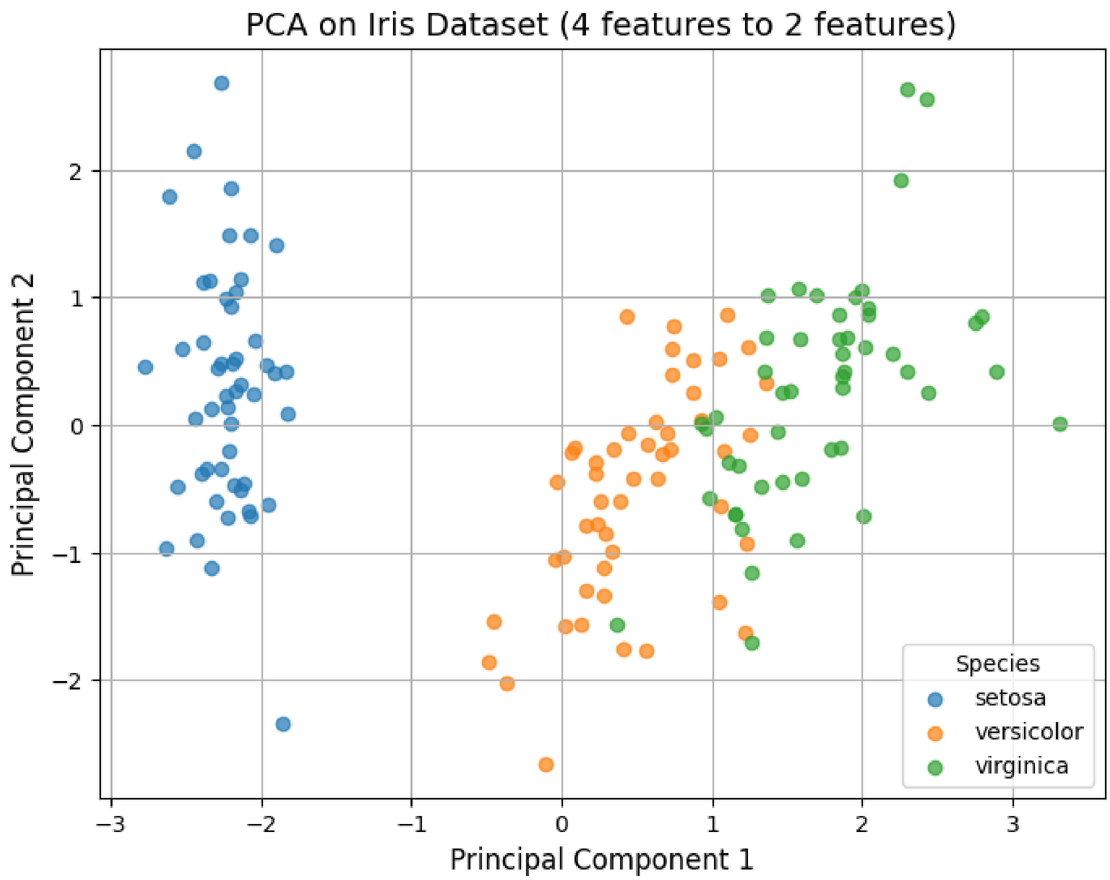
[2]:
```python
iris = load_iris()
features = iris.data
target = iris.target
```

[3]:
```python
scaler = StandardScaler()
features_standardized = scaler.fit_transform(features)
```

[4]:
```python
pca = PCA(n_components=2)
features_pca = pca.fit_transform(features_standardized)
```

[5]:
```python
pca_df = pd.DataFrame(data=features_pca, columns=["Principal Component 1",␣
↪"Principal Component 2"])
pca_df["Target"] = target
```

[6]:
```python
plt.figure(figsize=(8, 6))
for label, color in zip(iris.target_names, ["red", "green", "blue"]):
    plt.scatter(
        pca_df.loc[pca_df["Target"] == list(iris.target_names).index(label),␣
↪"Principal Component 1"],
        pca_df.loc[pca_df["Target"] == list(iris.target_names).index(label),␣
↪"Principal Component 2"],
        label=label,
        alpha=0.7
    )

plt.title("PCA on Iris Dataset (4 features to 2 features)", fontsize=14)
plt.xlabel("Principal Component 1", fontsize=12)
plt.ylabel("Principal Component 2", fontsize=12)
plt.legend(title="Species")
plt.grid()
plt.show()
```



[7]:
```python
explained_variance = pca.explained_variance_ratio_
print("Explained Variance by each Principal Component:")
print("Principal Component 1: ",explained_variance[0])
print("Principal Component 2: ",explained_variance[1])
print("Total Variance Retained: ",sum(explained_variance))
```

```
Explained Variance by each Principal Component:
Principal Component 1:  0.7296244541329989
Principal Component 2:  0.22850761786701768
Total Variance Retained:  0.9581320720000166
```

Program_4

June 8, 2025

[ ]:
```python
import pandas as pd
data = pd.read_csv("Dataset.csv")
```

[ ]:
```python
print(data)
```

[ ]:
```python
def find_s_algorithm(data):
 """Implements the Find-S algorithm to find the most specific hypothesis."""
 attributes = data.iloc[:, :-1].values
 target = data.iloc[:, -1].values

 for i in range(len(target)):
    if target[i] == "Yes":
        hypothesis = attributes[i].copy()
    break

 for i in range(len(target)):
    if target[i] == "Yes":
        for j in range(len(hypothesis)):
            if hypothesis[j] != attributes[i][j]:
                hypothesis[j] = '?'

 return hypothesis
final_hypothesis = find_s_algorithm(data)
print("Most Specific Hypothesis:", final_hypothesis)
```

Program_5

June 8, 2025

[1]:
```
import numpy as np
import pandas as pd
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score

np.random.seed(42)
values = np.random.rand(100)
labels = np.where(values[:50] <= 0.5, 'Class1', 'Class2')
labels = np.concatenate([labels, [None]*50])

df = pd.DataFrame({
    "Value": values,
    "Label": labels
})

X_train = df.loc[:49, ["Value"]]
y_train = df.loc[:49, "Label"]
X_test = df.loc[50:, ["Value"]]
true_labels = np.where(values[50:] <= 0.5, 'Class1', 'Class2')

k_values = [1, 2, 3, 4, 5, 20, 30]
for k in k_values:
    knn = KNeighborsClassifier(n_neighbors=k)
    knn.fit(X_train, y_train)
    preds = knn.predict(X_test)
    acc = accuracy_score(true_labels, preds) * 100
    print(f"Accuracy for k={k}: {acc:.2f}%")

    print(preds)
```

```
Accuracy for k=1: 100.00%
['Class2' 'Class2' 'Class2' 'Class2' 'Class2' 'Class2' 'Class1' 'Class1'
 'Class1' 'Class1' 'Class1' 'Class1' 'Class2' 'Class1' 'Class1' 'Class2'
 'Class1' 'Class2' 'Class1' 'Class2' 'Class2' 'Class1' 'Class1' 'Class2'
 'Class2' 'Class2' 'Class2' 'Class1' 'Class1' 'Class1' 'Class2' 'Class2'
 'Class1' 'Class1' 'Class1' 'Class1' 'Class2' 'Class2' 'Class2' 'Class1'
 'Class1' 'Class2' 'Class2' 'Class2' 'Class2' 'Class1' 'Class2' 'Class1'
 'Class1' 'Class1']
```

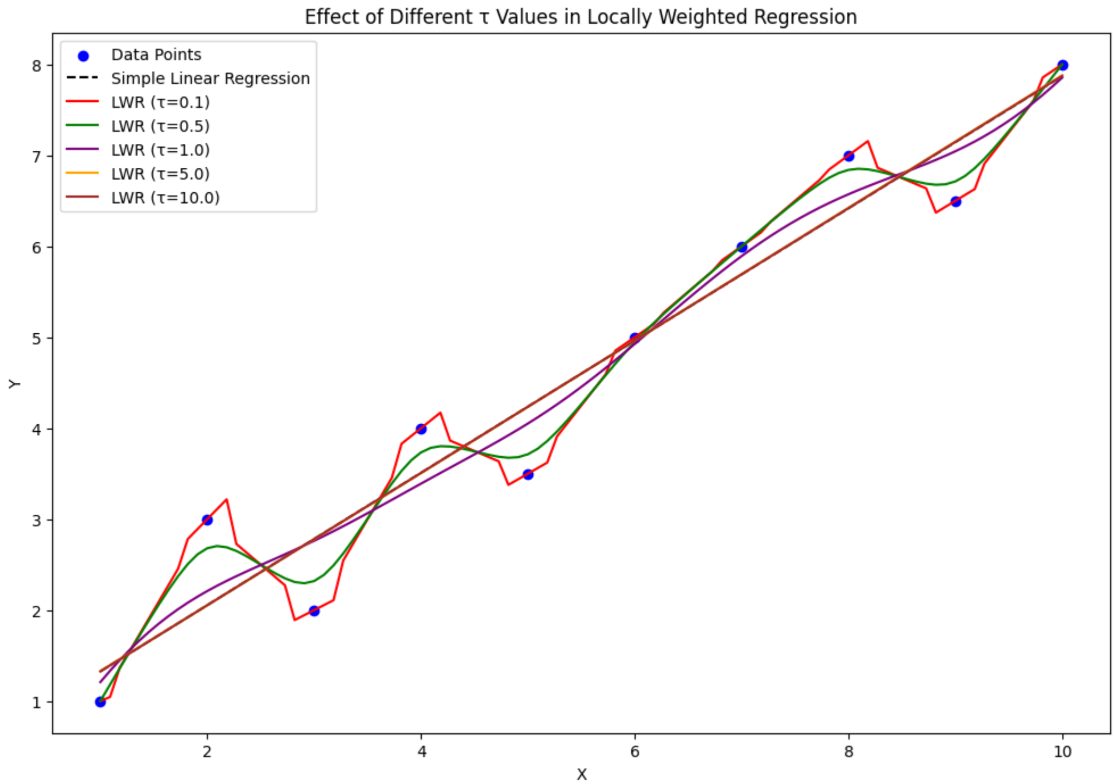
```
Accuracy for k=2: 100.00%
['Class2' 'Class2' 'Class2' 'Class2' 'Class2' 'Class2' 'Class1' 'Class1'
 'Class1' 'Class1' 'Class1' 'Class1' 'Class2' 'Class1' 'Class1' 'Class2'
 'Class1' 'Class2' 'Class1' 'Class2' 'Class2' 'Class1' 'Class1' 'Class2'
 'Class2' 'Class2' 'Class2' 'Class1' 'Class1' 'Class1' 'Class2' 'Class2'
 'Class1' 'Class1' 'Class1' 'Class1' 'Class2' 'Class2' 'Class2' 'Class1'
 'Class1' 'Class2' 'Class2' 'Class2' 'Class2' 'Class1' 'Class2' 'Class1'
 'Class1' 'Class1']
Accuracy for k=3: 98.00%
['Class2' 'Class2' 'Class2' 'Class2' 'Class2' 'Class2' 'Class1' 'Class1'
 'Class1' 'Class1' 'Class1' 'Class1' 'Class2' 'Class1' 'Class1' 'Class2'
 'Class1' 'Class2' 'Class1' 'Class2' 'Class2' 'Class1' 'Class1' 'Class2'
 'Class2' 'Class2' 'Class2' 'Class1' 'Class1' 'Class1' 'Class2' 'Class2'
 'Class1' 'Class1' 'Class1' 'Class1' 'Class2' 'Class2' 'Class2' 'Class1'
 'Class1' 'Class2' 'Class2' 'Class2' 'Class2' 'Class2' 'Class2' 'Class1'
 'Class1' 'Class1']
Accuracy for k=4: 98.00%
['Class2' 'Class2' 'Class2' 'Class2' 'Class2' 'Class2' 'Class1' 'Class1'
 'Class1' 'Class1' 'Class1' 'Class1' 'Class2' 'Class1' 'Class1' 'Class2'
 'Class1' 'Class2' 'Class1' 'Class2' 'Class2' 'Class1' 'Class1' 'Class2'
 'Class2' 'Class2' 'Class2' 'Class1' 'Class1' 'Class1' 'Class2' 'Class2'
 'Class1' 'Class1' 'Class1' 'Class1' 'Class2' 'Class2' 'Class2' 'Class1'
 'Class1' 'Class2' 'Class2' 'Class2' 'Class2' 'Class2' 'Class2' 'Class1'
 'Class1' 'Class1']
Accuracy for k=5: 98.00%
['Class2' 'Class2' 'Class2' 'Class2' 'Class2' 'Class2' 'Class1' 'Class1'
 'Class1' 'Class1' 'Class1' 'Class1' 'Class2' 'Class1' 'Class1' 'Class2'
 'Class1' 'Class2' 'Class1' 'Class2' 'Class2' 'Class1' 'Class1' 'Class2'
 'Class2' 'Class2' 'Class2' 'Class1' 'Class1' 'Class1' 'Class2' 'Class2'
 'Class1' 'Class1' 'Class1' 'Class1' 'Class2' 'Class2' 'Class2' 'Class1'
 'Class1' 'Class2' 'Class2' 'Class2' 'Class2' 'Class2' 'Class2' 'Class1'
 'Class1' 'Class1']
Accuracy for k=20: 98.00%
['Class2' 'Class2' 'Class2' 'Class2' 'Class2' 'Class2' 'Class1' 'Class1'
 'Class1' 'Class1' 'Class1' 'Class1' 'Class2' 'Class1' 'Class1' 'Class2'
 'Class1' 'Class2' 'Class1' 'Class2' 'Class2' 'Class1' 'Class1' 'Class2'
 'Class2' 'Class2' 'Class2' 'Class1' 'Class1' 'Class1' 'Class2' 'Class2'
 'Class1' 'Class1' 'Class1' 'Class1' 'Class2' 'Class2' 'Class2' 'Class1'
 'Class1' 'Class2' 'Class2' 'Class2' 'Class2' 'Class2' 'Class2' 'Class1'
 'Class1' 'Class1']
Accuracy for k=30: 100.00%
['Class2' 'Class2' 'Class2' 'Class2' 'Class2' 'Class2' 'Class1' 'Class1'
 'Class1' 'Class1' 'Class1' 'Class1' 'Class2' 'Class1' 'Class1' 'Class2'
 'Class1' 'Class2' 'Class1' 'Class2' 'Class2' 'Class1' 'Class1' 'Class2'
 'Class2' 'Class2' 'Class2' 'Class1' 'Class1' 'Class1' 'Class2' 'Class2'
 'Class1' 'Class1' 'Class1' 'Class1' 'Class2' 'Class2' 'Class2' 'Class1'
 'Class1' 'Class2' 'Class2' 'Class2' 'Class2' 'Class1' 'Class2' 'Class1'
 'Class1' 'Class1']
```

Program_6

June 8, 2025

[1]:
```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression

def gaussian_kernel(x, x_query, tau):
    return np.exp(- (x - x_query) ** 2 / (2 * tau ** 2))

def locally_weighted_regression(X, y, x_query, tau):
    X_b = np.c_[np.ones(len(X)), X]
    x_query_b = np.array([1, x_query])
    W = np.diag(gaussian_kernel(X, x_query, tau))
    # Use pseudo-inverse for stability
    theta = np.linalg.pinv(X_b.T @ W @ X_b) @ X_b.T @ W @ y
    return x_query_b @ theta

X = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9, 10])
y = np.array([1, 3, 2, 4, 3.5, 5, 6, 7, 6.5, 8])

X_query = np.linspace(1, 10, 100)
tau_values = [0.1, 0.5, 1.0, 5.0, 10.0]

lin_reg = LinearRegression()
lin_reg.fit(X.reshape(-1, 1), y)
y_lin = lin_reg.predict(X_query.reshape(-1, 1))

plt.figure(figsize=(12, 8))
plt.scatter(X, y, color='blue', label='Data Points')
plt.plot(X_query, y_lin, color='black', linestyle='dashed', label='Simple␣
 ↪Linear Regression')
colors = ['red', 'green', 'purple', 'orange', 'brown']
for tau, color in zip(tau_values, colors):
    y_lwr = np.array([locally_weighted_regression(X, y, x_q, tau) for x_q in␣
 ↪X_query])
    plt.plot(X_query, y_lwr, color=color, label=f'LWR ( ={tau})')
plt.title("Effect of Different  Values in Locally Weighted Regression")
plt.xlabel("X")
plt.ylabel("Y")
```

```
plt.legend()
plt.show()
```



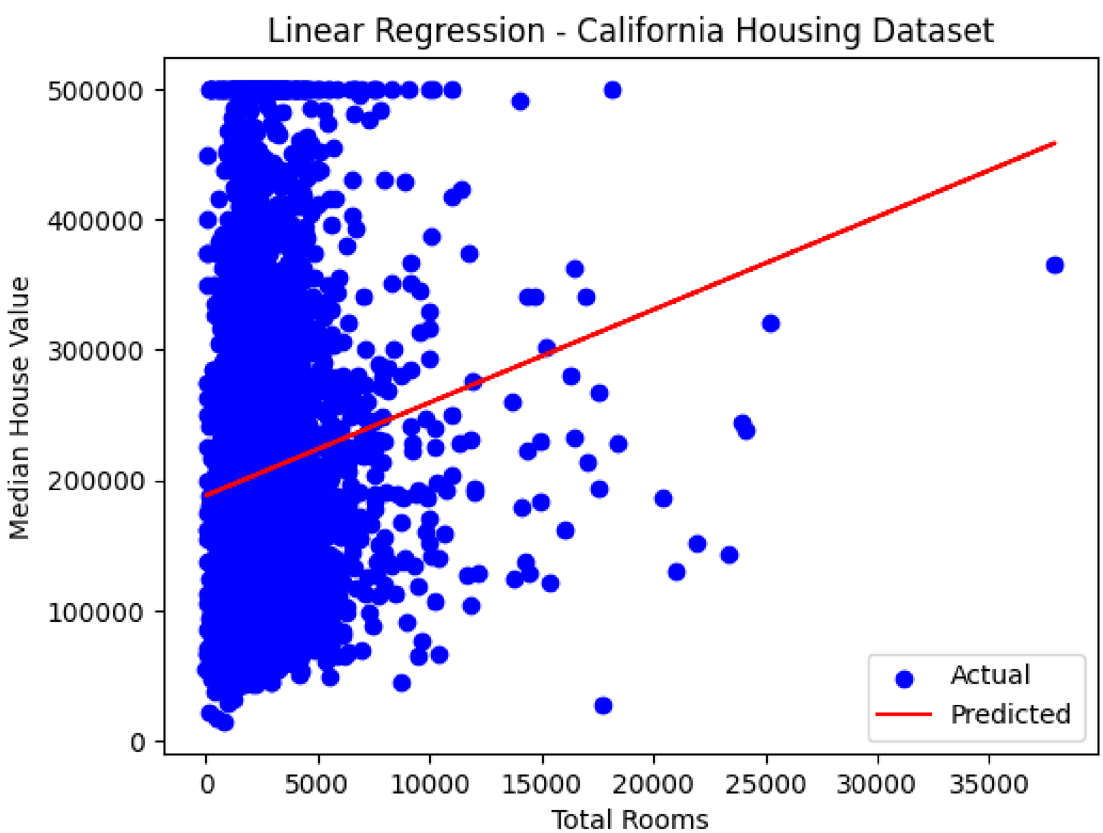
Program_7

June 8, 2025

[4]:

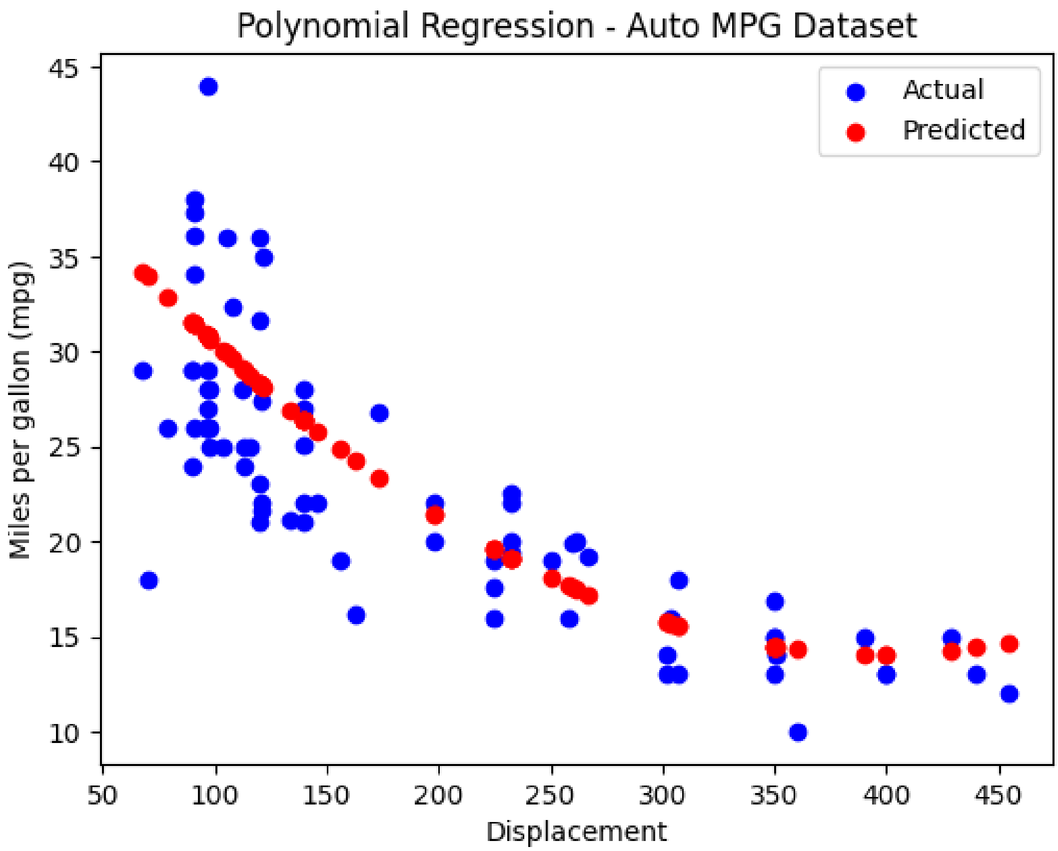
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import PolynomialFeatures, StandardScaler
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.pipeline import make_pipeline

def linear_regression_boston_housing():
    housing = pd.read_csv('Datasets/housing.csv')
    X = housing["total_rooms"].values.reshape(-1, 1)
    y = housing["median_house_value"].values
    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)
    y_pred = model.predict(X_test)
    plt.scatter(X_test, y_test, color="blue", label="Actual")
    plt.plot(X_test, y_pred, color="red", label="Predicted")
    plt.xlabel("Total Rooms")
    plt.ylabel("Median House Value")
    plt.title("Linear Regression - California Housing Dataset")
    plt.legend()
    plt.show()
    print("Linear Regression - California Housing Dataset")
    print("Mean Squared Error:", mean_squared_error(y_test, y_pred))
    print("R^2 Score:", r2_score(y_test, y_pred))

def polynomial_regression_auto_mpg():
    data = sns.load_dataset('mpg')
    data = data.dropna()
    X = data["displacement"].values.reshape(-1, 1)
    y = data["mpg"].values
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
↪random_state=42)
```

```
    poly_model = make_pipeline(PolynomialFeatures(degree=2), StandardScaler(),
 ↪LinearRegression())
    poly_model.fit(X_train, y_train)
    y_pred = poly_model.predict(X_test)
    plt.scatter(X_test, y_test, color="blue", label="Actual")
    plt.scatter(X_test, y_pred, color="red", label="Predicted")
    plt.xlabel("Displacement")
    plt.ylabel("Miles per gallon (mpg)")
    plt.title("Polynomial Regression - Auto MPG Dataset")
    plt.legend()
    plt.show()
    print("Polynomial Regression - Auto MPG Dataset")
    print("Mean Squared Error:", mean_squared_error(y_test, y_pred))
    print("R^2 Score:", r2_score(y_test, y_pred))

if __name__ == "__main__":
    linear_regression_boston_housing()
    polynomial_regression_auto_mpg()
```



Linear Regression - California Housing Dataset

```
Mean Squared Error: 12868608472.627417
R^2 Score: 0.017970062300526446
```




```
Polynomial Regression - Auto MPG Dataset
Mean Squared Error: 20.649054718308783
R^2 Score: 0.5954385038809514
```

Program_8

June 8, 2025

[ ]:
```
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.tree import DecisionTreeClassifier, plot_tree
from sklearn.metrics import accuracy_score, classification_report,
    confusion_matrix
from sklearn.tree import export_graphviz
from IPython.display import Image
import pydotplus
import warnings
warnings.filterwarnings('ignore')
```

[ ]:
```
!pip install graphviz
!apt-get install graphviz -y # To install graphviz system-wide
```

```
Requirement already satisfied: graphviz in /usr/local/lib/python3.11/dist-
packages (0.20.3)
Reading package lists… Done
Building dependency tree… Done
Reading state information… Done
graphviz is already the newest version (2.42.2-6ubuntu0.1).
0 upgraded, 0 newly installed, 0 to remove and 34 not upgraded.
```

[ ]:
```
import graphviz
```

[ ]:
```
data = pd.read_csv(r'/content/Breast Cancer Dataset.csv')
```

[ ]:
```
pd.set_option('display.max_columns', None)
```

[ ]:
```
data.diagnosis.unique()
```

[ ]:
```
array(['M', 'B'], dtype=object)
```

[ ]:
```
df = data.drop(['id'], axis=1)
df['diagnosis'] = df['diagnosis'].map({'M':1, 'B':0}) # Malignant:1, Benign:0
```

[ ]:
```python
X = df.drop('diagnosis', axis=1)  # Drop the 'diagnosis' column (target)
y = df['diagnosis']
```

[ ]:
```python
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)
```

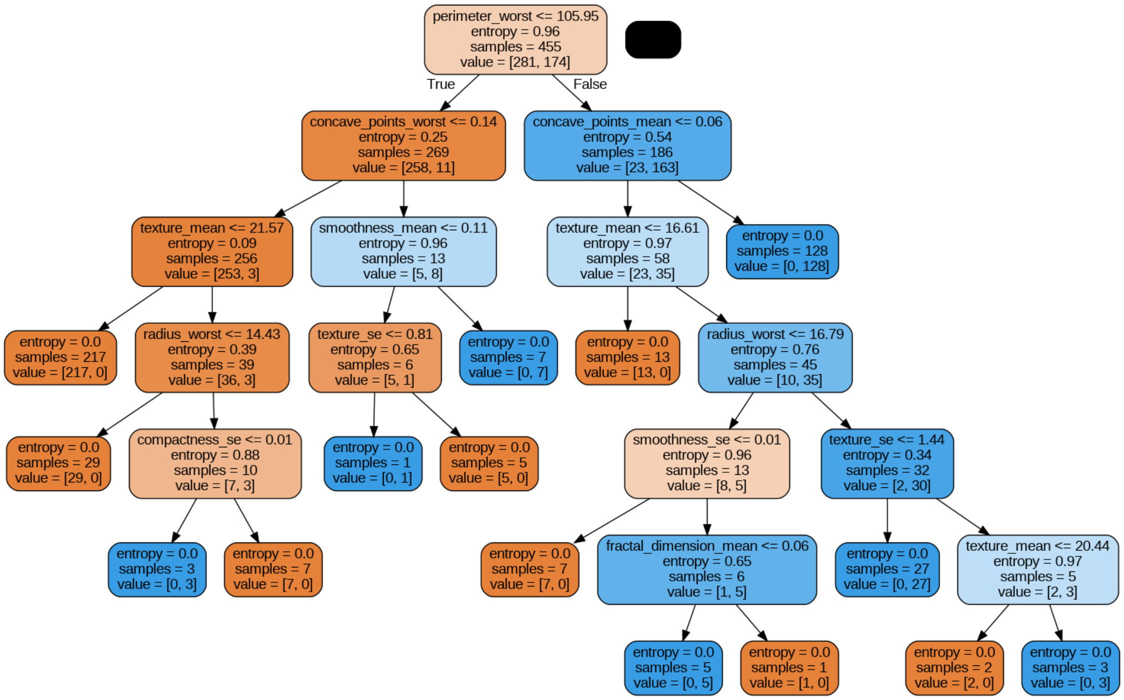
[ ]:
```python
model = DecisionTreeClassifier(criterion='entropy') #criteria = gini, entropy
model.fit(X_train, y_train)
model
```

[ ]:
```
DecisionTreeClassifier(criterion='entropy')
```

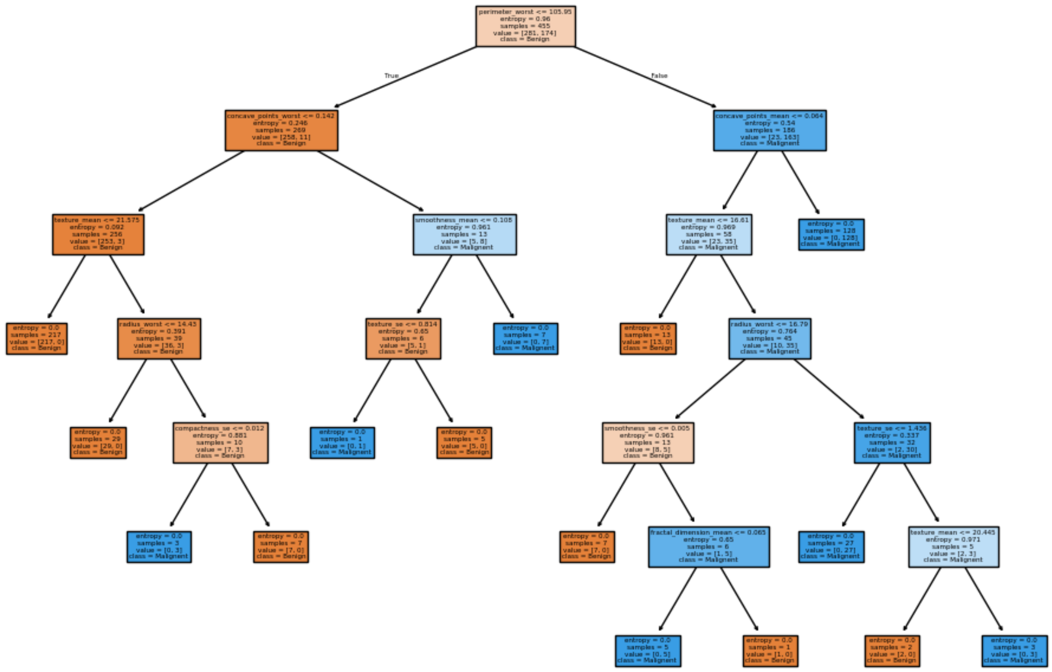
[1]:
```python
import math
def entropy(column):
  counts = column.value_counts()
  probabilities = counts / len(column)
  return -sum(probabilities * probabilities.apply(math.log2))

def conditional_entropy(data, X, target):
  feature_values = data[X].unique()  # Corrected: use .unique() on the series
  weighted_entropy = 0
  for value in feature_values:
    subset = data[data[feature] == value]
    weighted_entropy += (len(subset) / len(data)) * entropy(subset[target])
    return weighted_entropy

def information_gain(data, X, target):
  total_entropy = entropy(data[target])
  feature_conditional_entropy = conditional_entropy(data, X, target)
  return total_entropy - feature_conditional_entropy
  for feature in X:
    ig = information_gain(df,feature,'diagnosis')
    print(f"Information Gain for {feature}: {ig}")
```

[ ]:
```python
dot_data = export_graphviz(model, out_file=None,feature_names=X_train.
 ↪columns,rounded=True, proportion=False,precision=2, filled=True)
graph = pydotplus.graph_from_dot_data(dot_data)
Image(graph.create_png())
```

[ ]:



[ ]:
```
plt.figure(figsize=(12, 8))
plot_tree(model, filled=True, feature_names=X.columns, class_names=['Benign',␣
↪'Malignent'])
plt.show()
```



```
[ ]: y_pred = model.predict(X_test)
     y_pred
```

```
[ ]: array([1, 1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1,
            1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 1, 0, 0,
            0, 0, 0, 0, 0, 0, 1, 0, 1, 1, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 1, 0,
            0, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1,
            0, 0, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 1, 1, 1, 1,
            1, 0, 0, 0])
```

```
[ ]: accuracy = accuracy_score(y_test, y_pred) * 100
     classification_rep = classification_report(y_test, y_pred)
      # Print the results
     print("Accuracy:", accuracy)
     print("Classification Report:\n", classification_rep)
```

```
Accuracy: 90.35087719298247
Classification Report:
               precision    recall  f1-score   support

           0       0.92      0.93      0.93        76
           1       0.86      0.84      0.85        38

    accuracy                           0.90       114
```

```
   macro avg       0.89      0.89      0.89       114
weighted avg       0.90      0.90      0.90       114
```

[ ]:
```
df.head(1)
```

[ ]:
```
   diagnosis  radius_mean  texture_mean  perimeter_mean  area_mean  \
0          1        17.99         10.38           122.8     1001.0

   smoothness_mean  compactness_mean  concavity_mean  concave_points_mean  \
0           0.1184            0.2776          0.3001               0.1471

   symmetry_mean  fractal_dimension_mean  radius_se  texture_se  perimeter_se  \
0         0.2419                 0.07871      1.095      0.9053         8.589

   area_se  smoothness_se  compactness_se  concavity_se  concave_points_se  \
0    153.4       0.006399         0.04904       0.05373            0.01587

   symmetry_se  fractal_dimension_se  radius_worst  texture_worst  \
0      0.03003              0.006193         25.38          17.33

   perimeter_worst  area_worst  smoothness_worst  compactness_worst  \
0            184.6      2019.0            0.1622             0.6656

   concavity_worst  concave_points_worst  symmetry_worst  \
0           0.7119                0.2654          0.4601

   fractal_dimension_worst
0                   0.1189
```

[ ]:
```
new = [[12.5, 19.2, 80.0, 500.0, 0.085, 0.1, 0.05, 0.02, 0.17, 0.06,0.4, 1.0, 2.
 ↪5, 40.0, 0.006, 0.02, 0.03, 0.01, 0.02, 0.003,16.0, 25.0, 105.0, 900.0, 0.
 ↪13, 0.25, 0.28, 0.12, 0.29, 0.08]]
y_pred = model.predict(new)
if y_pred[0] == 0:
  print("Prediction: Benign")
else:
  print("Prediction: Malignant")
```

```
Prediction: Benign
```

Program_9

June 8, 2025

[1]:
```python
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import fetch_olivetti_faces
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import confusion_matrix, accuracy_score, roc_auc_score
from sklearn.preprocessing import label_binarize

data = fetch_olivetti_faces()
X, y = data.data, data.target

x_train, x_test, y_train, y_test = train_test_split(X, y, test_size=0.3,␣
↪random_state=42)

nb = GaussianNB()
nb.fit(x_train, y_train)
y_pred = nb.predict(x_test)

accuracy = round(accuracy_score(y_test, y_pred) * 100, 2)
print(f"Naive Bayes Accuracy: {accuracy}%")
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))

mis_idx = np.where(y_pred != y_test)[0]
print(f"Number of misclassified images: {len(mis_idx)}")
plt.figure(figsize=(10, 2))
for i, idx in enumerate(mis_idx[:5]):
    plt.subplot(1, 5, i+1)
    plt.imshow(x_test[idx].reshape(64, 64), cmap='gray')
    plt.title(f"T:{y_test[idx]},P:{y_pred[idx]}")
    plt.axis('off')
plt.show()

y_test_bin = label_binarize(y_test, classes=np.unique(y_test))
y_pred_prob = nb.predict_proba(x_test)
for i in range(y_test_bin.shape[1]):
    auc = roc_auc_score(y_test_bin[:, i], y_pred_prob[:, i])
    print(f"Class {i} AUC: {auc:.2f}")
```

```
Naive Bayes Accuracy: 74.17%
Confusion Matrix:
 [[3 0 0 … 0 0 0]
 [0 3 0 … 0 0 0]
 [0 0 2 … 0 0 0]
 …
 [0 0 0 … 3 0 0]
 [0 0 0 … 0 0 0]
 [0 0 0 … 0 0 4]]
Number of misclassified images: 31
```

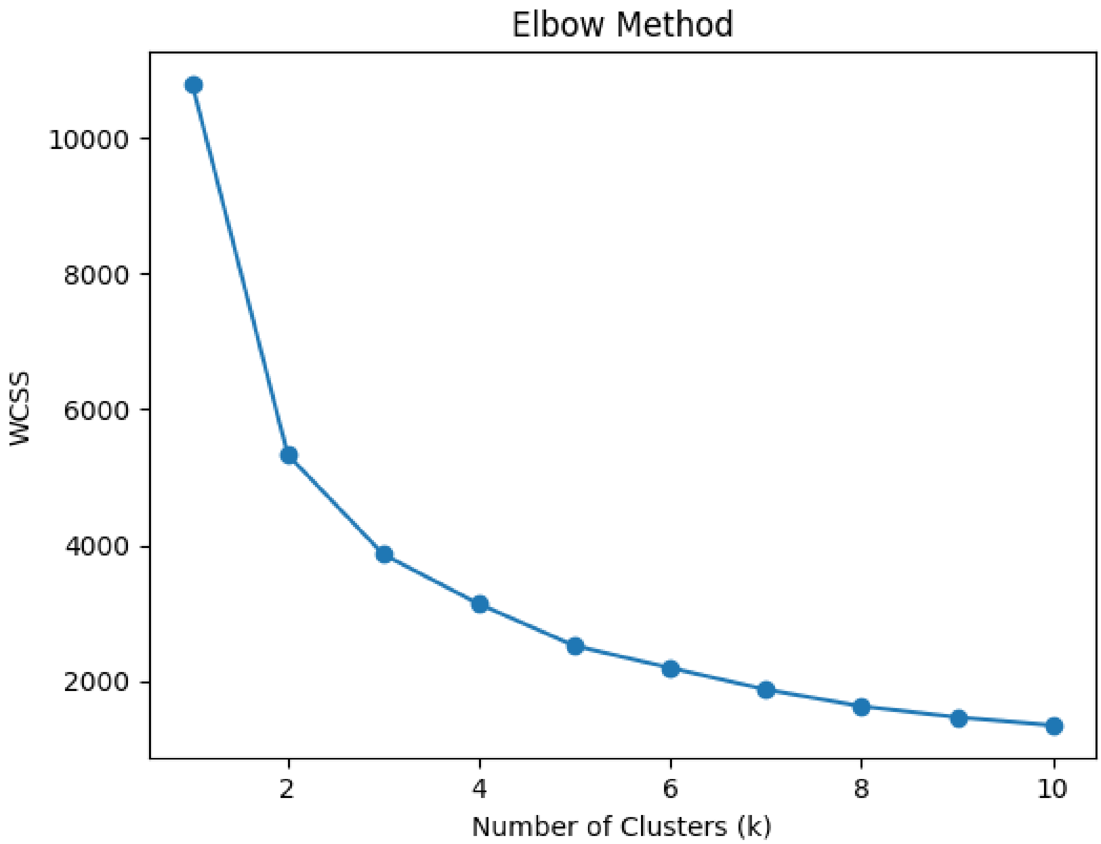

```
Class 0 AUC: 0.80
Class 1 AUC: 0.88
Class 2 AUC: 1.00
Class 3 AUC: 1.00
Class 4 AUC: 1.00
Class 5 AUC: 1.00
Class 6 AUC: 0.67
Class 7 AUC: 0.50
Class 8 AUC: 1.00
Class 9 AUC: 0.97
Class 10 AUC: 0.83
Class 11 AUC: 1.00
Class 12 AUC: 0.98
Class 13 AUC: 0.83
Class 14 AUC: 1.00
Class 15 AUC: 0.99
Class 16 AUC: 0.48
Class 17 AUC: 0.49
Class 18 AUC: 0.50
Class 19 AUC: 0.49
Class 20 AUC: 0.49
Class 21 AUC: 0.50
Class 22 AUC: 0.48
Class 23 AUC: 0.48
Class 24 AUC: 0.49
Class 25 AUC: 0.48
Class 26 AUC: 0.49
```

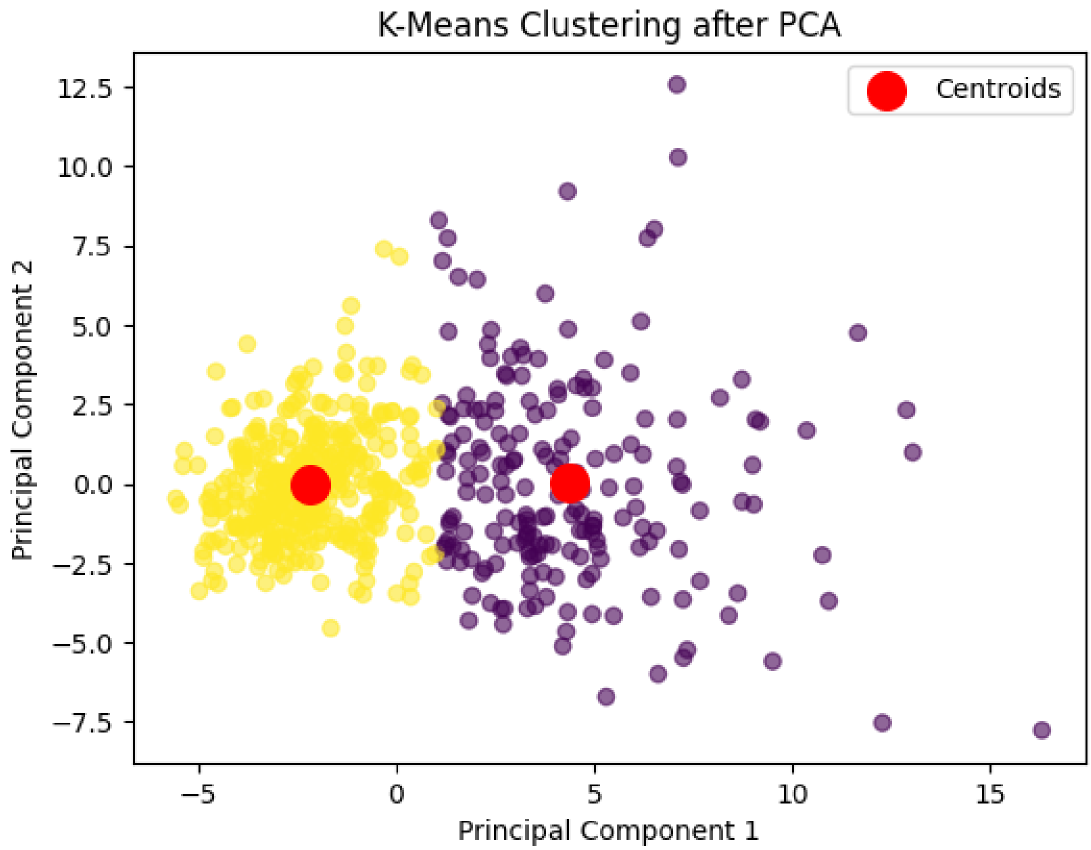
```
Class 27 AUC: 0.49
Class 28 AUC: 0.45
Class 29 AUC: 0.49
Class 30 AUC: 0.49
Class 31 AUC: 0.50
Class 32 AUC: 0.49
Class 33 AUC: 0.49
Class 34 AUC: 0.49
Class 35 AUC: 0.48
Class 36 AUC: 0.49
Class 37 AUC: 0.49
Class 38 AUC: 0.50
```

Program_10

June 8, 2025

[1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA

data = pd.read_csv("Datasets/Wisconsin Breast Cancer dataset.csv")
df = data.drop(['id', 'Unnamed: 32'], axis=1)
df['diagnosis'] = df['diagnosis'].map({'M':1, 'B':0})
X = df.drop(columns=["diagnosis"])

X_scaled = StandardScaler().fit_transform(X)
X_pca = PCA(n_components=2).fit_transform(X_scaled)

wcss = []
for k in range(1, 11):
    kmeans = KMeans(n_clusters=k, random_state=42, n_init=10)
    kmeans.fit(X_pca)
    wcss.append(kmeans.inertia_)
plt.plot(range(1, 11), wcss, marker="o")
plt.xlabel("Number of Clusters (k)")
plt.ylabel("WCSS")
plt.title("Elbow Method")
plt.show()

optimal_k = 2
kmeans = KMeans(n_clusters=optimal_k, random_state=42, n_init=10)
clusters = kmeans.fit_predict(X_pca)
plt.scatter(X_pca[:, 0], X_pca[:, 1], c=clusters, cmap="viridis", alpha=0.6)
plt.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:, 1],␣
  ↪s=200, c="red", label="Centroids")
plt.xlabel("Principal Component 1")
plt.ylabel("Principal Component 2")
plt.title("K-Means Clustering after PCA")
plt.legend()
plt.show()
```


Elbow Method
WCSS
Number of Clusters (k)
10000
8000
6000
4000
2000
2
4
6
8
10


K-Means Clustering after PCA
Centroids
12.5
10.0
7.5
5.0
2.5
0.0
−2.5
−5.0
−7.5
−5
0
5
10
15
Principal Component 1
Principal Component 2