RAJALAKSHMI ENGINEERING COLLEGE

Approved by AICTE | Affiliated to Anna University | Accredited by NAAC

Department of Computer Science and Engineering

CS23334 Fundamentals of Data Science Lab

III semester II Year (2023R)

Name of the Student : Keerthana C

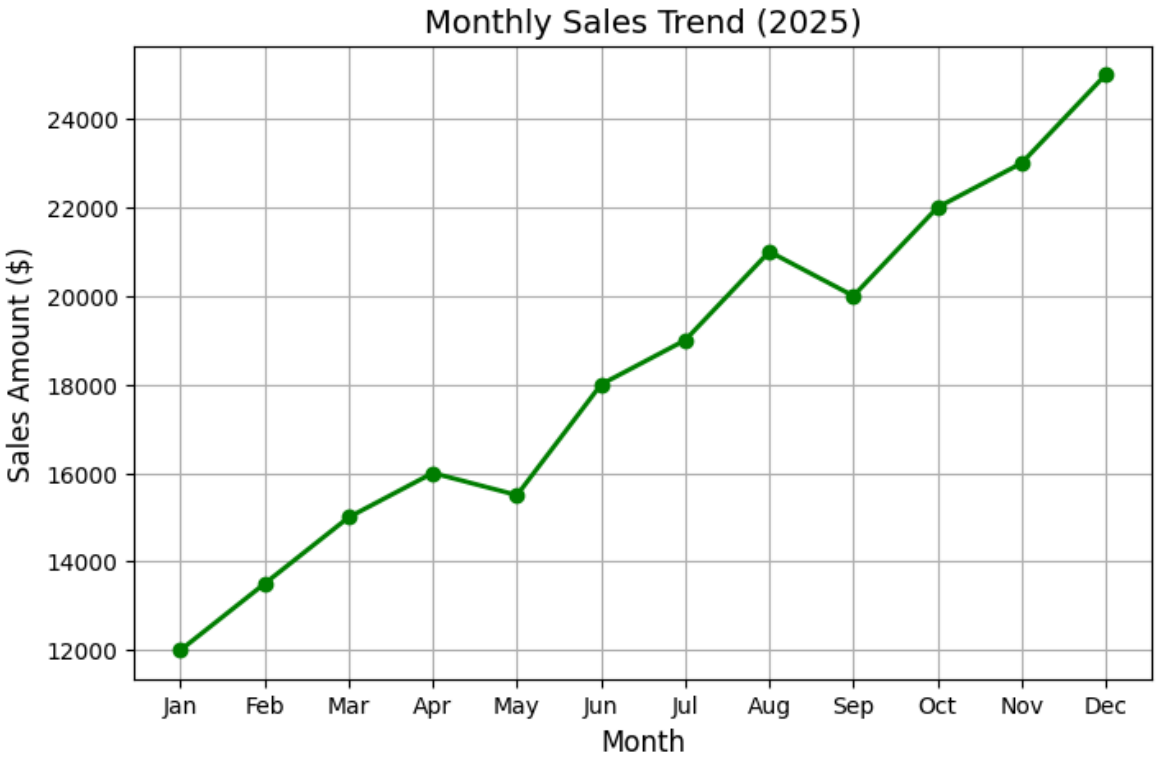
Register Number : 240701253

LINE PLOT

In [1]:
```
import matplotlib.pyplot as plt
```

In [2]:
```
months = ["Jan", "Feb", "Mar", "Apr", "May", "Jun", "Jul", "Aug", "Sep", "Oct",
sales = [12000, 13500, 15000, 16000, 15500, 18000, 19000, 21000, 20000, 22000, 2
```

In [3]:
```
plt.figure(figsize=(8,5))
plt.plot(months, sales, color='green', marker='o', linestyle='-', linewidth=2)

plt.title("Monthly Sales Trend (2025)", fontsize=14)
plt.xlabel("Month", fontsize=12)
plt.ylabel("Sales Amount ($)", fontsize=12)
plt.grid(True)

plt.show()
```

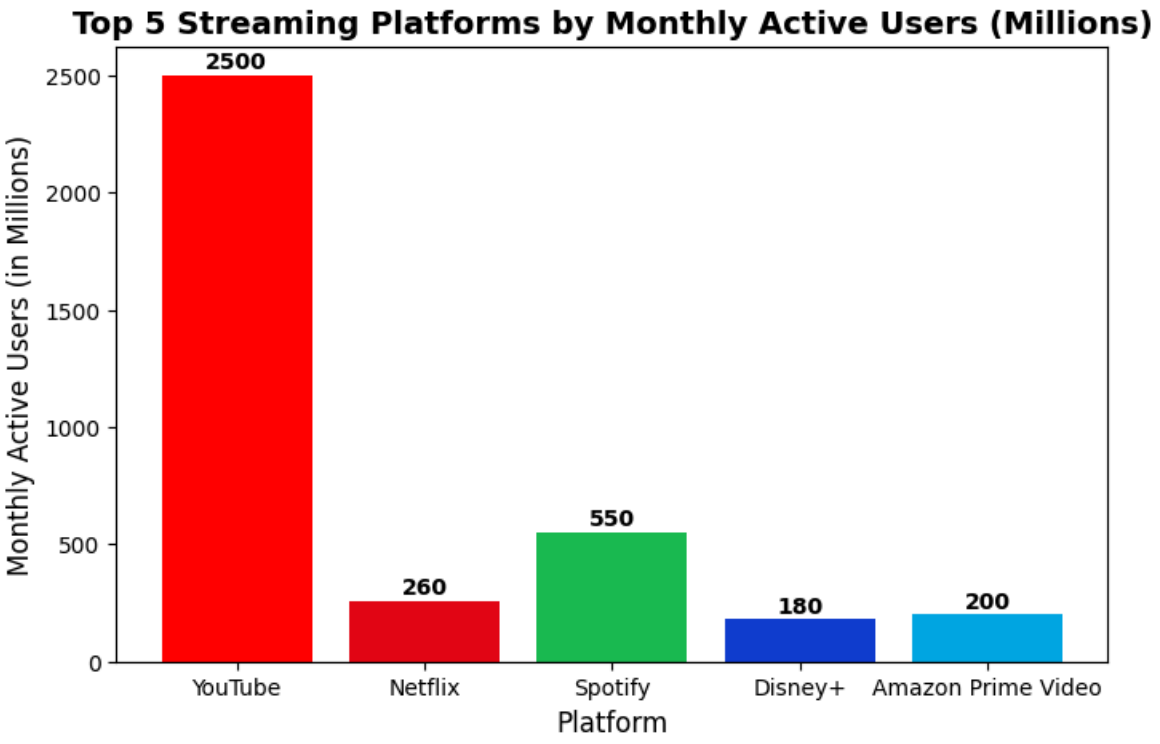


In [ ]:

In [1]:
```
import matplotlib.pyplot as plt
```

In [2]:
```
platforms = ['YouTube', 'Netflix', 'Spotify', 'Disney+', 'Amazon Prime Video']
users = [2500, 260, 550, 180, 200]
```

In [3]:
```
plt.figure(figsize=(8,5))
bars = plt.bar(platforms, users, color=['#FF0000', '#E50914', '#1DB954', '#113CC
plt.title('Top 5 Streaming Platforms by Monthly Active Users (Millions)', fontsi
plt.xlabel('Platform', fontsize=12)
plt.ylabel('Monthly Active Users (in Millions)', fontsize=12)
for bar in bars:
    plt.text(bar.get_x() + bar.get_width()/2, bar.get_height() + 30, f'{bar.get_
plt.show()
```

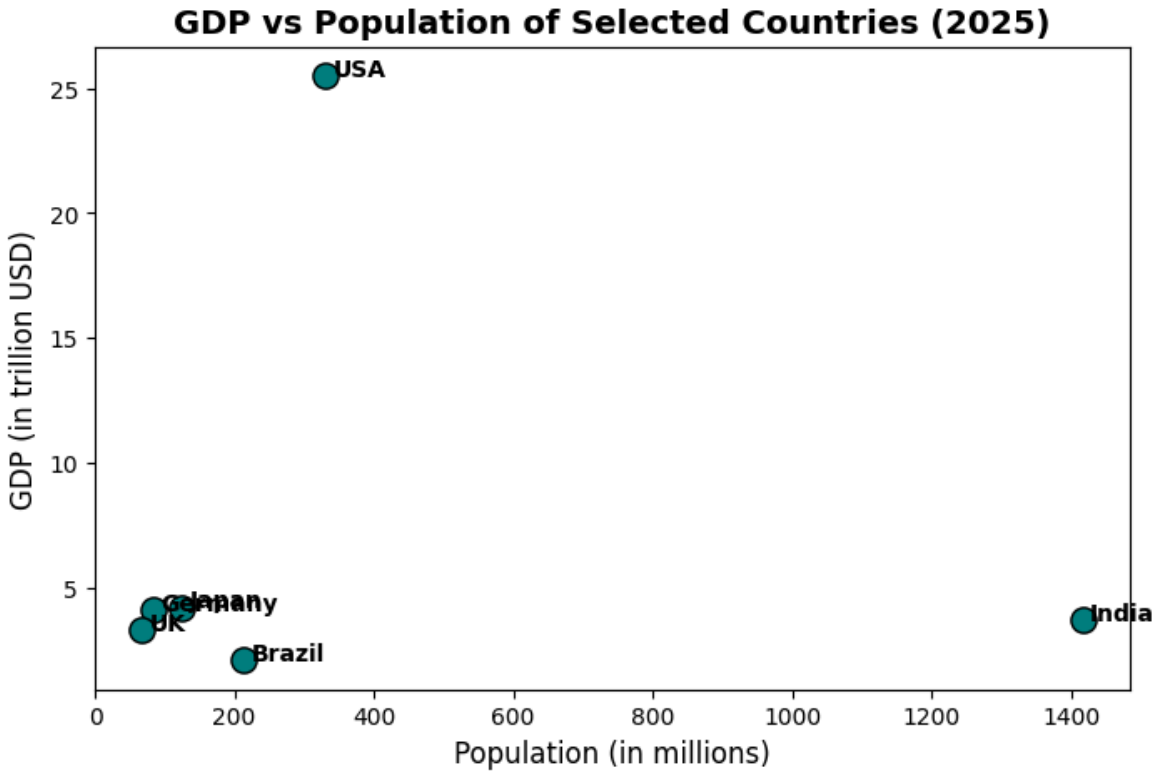


In [ ]:
```
```

In [1]:
```
import matplotlib.pyplot as plt
```

In [2]:
```
countries = ['USA', 'India', 'Japan', 'Germany', 'Brazil', 'UK']
gdp = [25.5, 3.7, 4.2, 4.1, 2.1, 3.3]
population = [331, 1417, 125, 83, 214, 67]
```

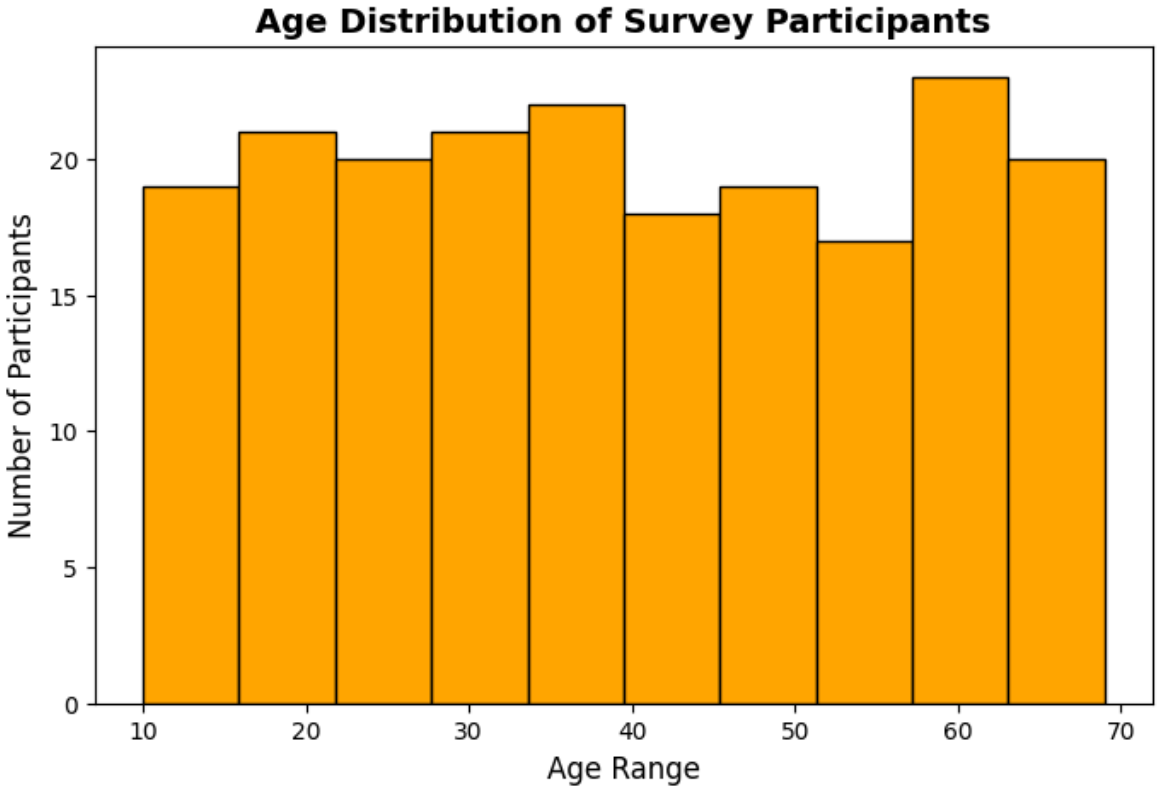
In [3]:
```
plt.figure(figsize=(8,5))
plt.scatter(population, gdp, color='teal', s=120, edgecolors='black')
plt.title('GDP vs Population of Selected Countries (2025)', fontsize=14, fontwei
plt.xlabel('Population (in millions)', fontsize=12)
plt.ylabel('GDP (in trillion USD)', fontsize=12)
for i, country in enumerate(countries):
    plt.text(population[i]+10, gdp[i], country, fontsize=10, fontweight='bold')
plt.show()
```



In [ ]:
```
```

In [1]:
```
import matplotlib.pyplot as plt
import numpy as np
```

In [3]:
```
ages = np.random.randint(10, 70, 200)
plt.figure(figsize=(8,5))
plt.hist(ages, bins=10, color='orange', edgecolor='black')
plt.title('Age Distribution of Survey Participants', fontsize=14, fontweight='bo
plt.xlabel('Age Range', fontsize=12)
plt.ylabel('Number of Participants', fontsize=12)
plt.show()
```

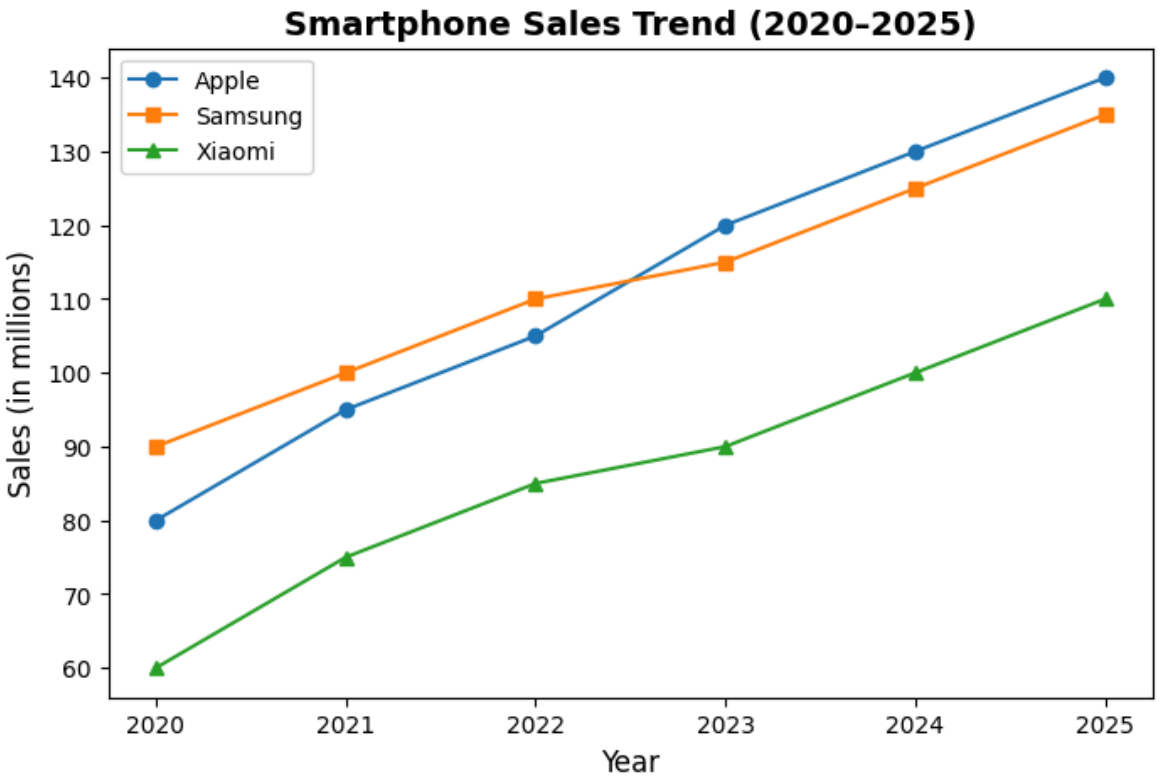


In [ ]:
```
```

In [1]:
```
import matplotlib.pyplot as plt
```

In [2]:
```
years = [2020, 2021, 2022, 2023, 2024, 2025]
apple_sales = [80, 95, 105, 120, 130, 140]
samsung_sales = [90, 100, 110, 115, 125, 135]
xiaomi_sales = [60, 75, 85, 90, 100, 110]
```

In [4]:
```
plt.figure(figsize=(8,5))
plt.plot(years, apple_sales, marker='o', label='Apple')
plt.plot(years, samsung_sales, marker='s', label='Samsung')
plt.plot(years, xiaomi_sales, marker='^', label='Xiaomi')
plt.title('Smartphone Sales Trend (2020-2025)', fontsize=14, fontweight='bold')
plt.xlabel('Year', fontsize=12)
plt.ylabel('Sales (in millions)', fontsize=12)
plt.legend()
plt.show()
```

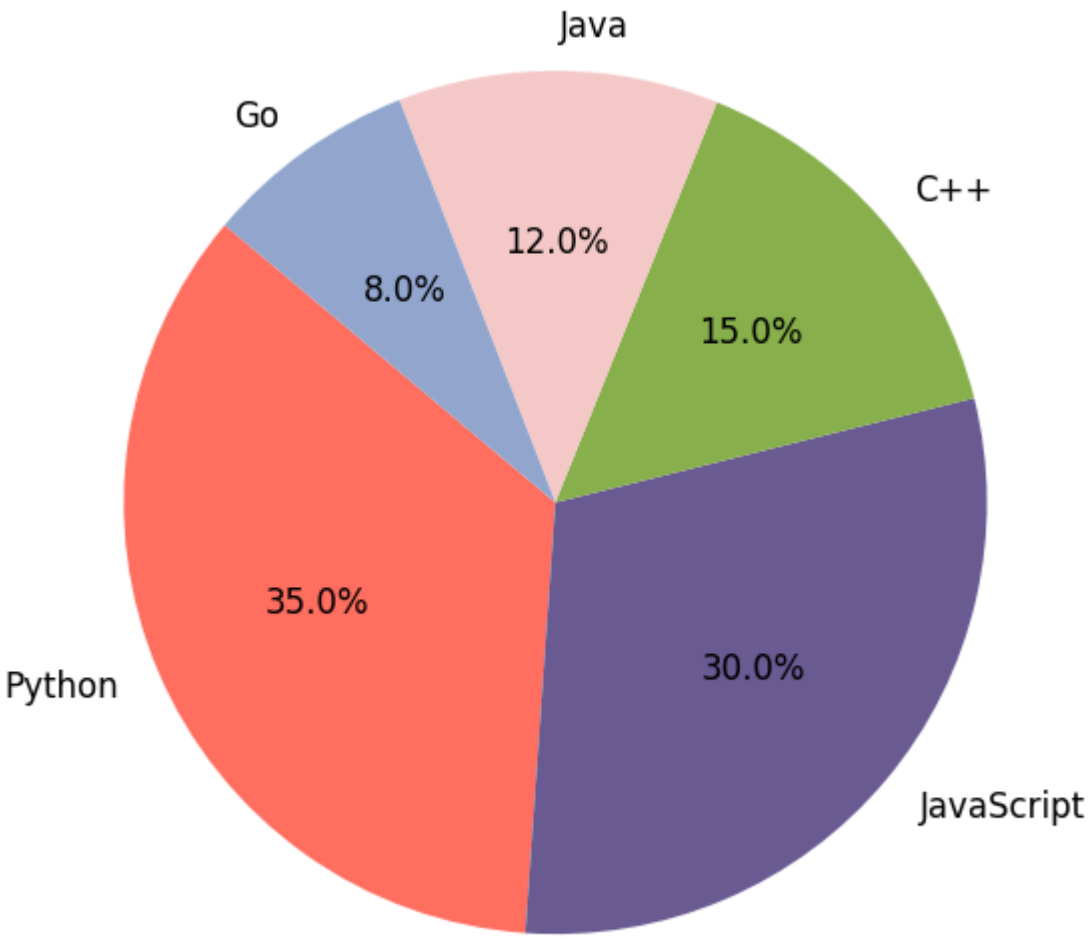


In [ ]:
```
```

In [1]:
```
import matplotlib.pyplot as plt
```

In [2]:
```
languages = ['Python', 'JavaScript', 'C++', 'Java', 'Go']
popularity = [35, 30, 15, 12, 8]
colors = ['#FF6F61', '#6B5B95', '#88B04B', '#F7CAC9', '#92A8D1']
```

In [3]:
```
plt.figure(figsize=(7,7))
plt.pie(popularity, labels=languages, colors=colors, autopct='%1.1f%%', startang
plt.show()
```



In [ ]:
```
```

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

In [2]:
```
sns.set_style("whitegrid")
%matplotlib inline
```

In [3]:
```
iris = load_iris(as_frame=True)
df = iris.frame
print("Dataset loaded directly from scikit-learn.")
```

```
Dataset loaded directly from scikit-learn.
```

In [4]:
```
print("\nInitial Data Exploration (First 5 Rows)")
display(df.head())
```

```
Initial Data Exploration (First 5 Rows)
```

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

In [5]:
```
print("\nData Info (Columns, Non-Null Counts, Dtypes)")
df.info()
```

```
Data Info (Columns, Non-Null Counts, Dtypes)
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 150 entries, 0 to 149
Data columns (total 5 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   sepal length (cm)  150 non-null    float64
 1   sepal width (cm)   150 non-null    float64
 2   petal length (cm)  150 non-null    float64
 3   petal width (cm)   150 non-null    float64
 4   target             150 non-null    int64  
dtypes: float64(4), int64(1)
memory usage: 6.0 KB
```

In [6]:
```
print("\nDescriptive Statistics")
display(df.describe())
```

```
Descriptive Statistics
```

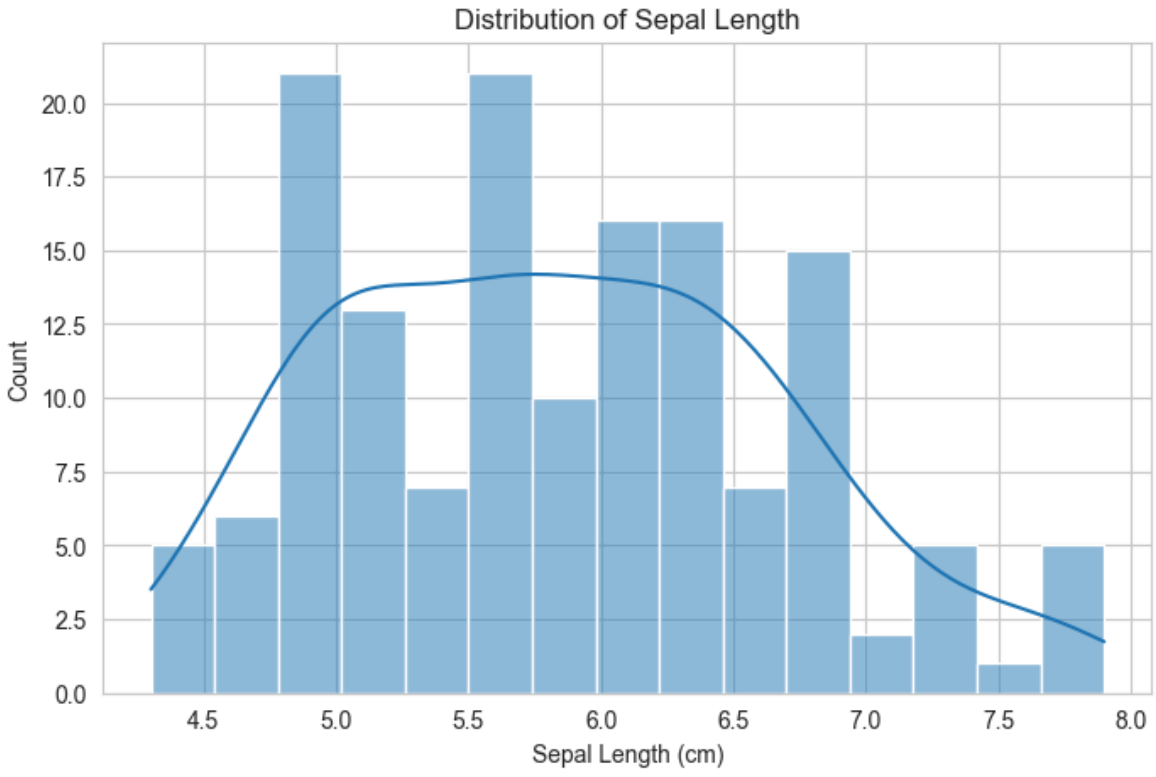
	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	target
count	150.000000	150.000000	150.000000	150.000000	150.000000
mean	5.843333	3.057333	3.758000	1.199333	1.000000
std	0.828066	0.435866	1.765298	0.762238	0.819232
min	4.300000	2.000000	1.000000	0.100000	0.000000
25%	5.100000	2.800000	1.600000	0.300000	0.000000
50%	5.800000	3.000000	4.350000	1.300000	1.000000
75%	6.400000	3.300000	5.100000	1.800000	2.000000
max	7.900000	4.400000	6.900000	2.500000	2.000000

In [7]:
```
print("\nCheck for Unique Values in 'target_names'")
print(iris.target_names)
```

```
Check for Unique Values in 'target_names'
['setosa' 'versicolor' 'virginica']
```

In [8]:
```
print("\nVisualization: Sepal Length Distribution")
plt.figure(figsize=(8, 5))
sns.histplot(df['sepal length (cm)'], kde=True, bins=15)
plt.title('Distribution of Sepal Length')
plt.xlabel('Sepal Length (cm)')
plt.show()
```

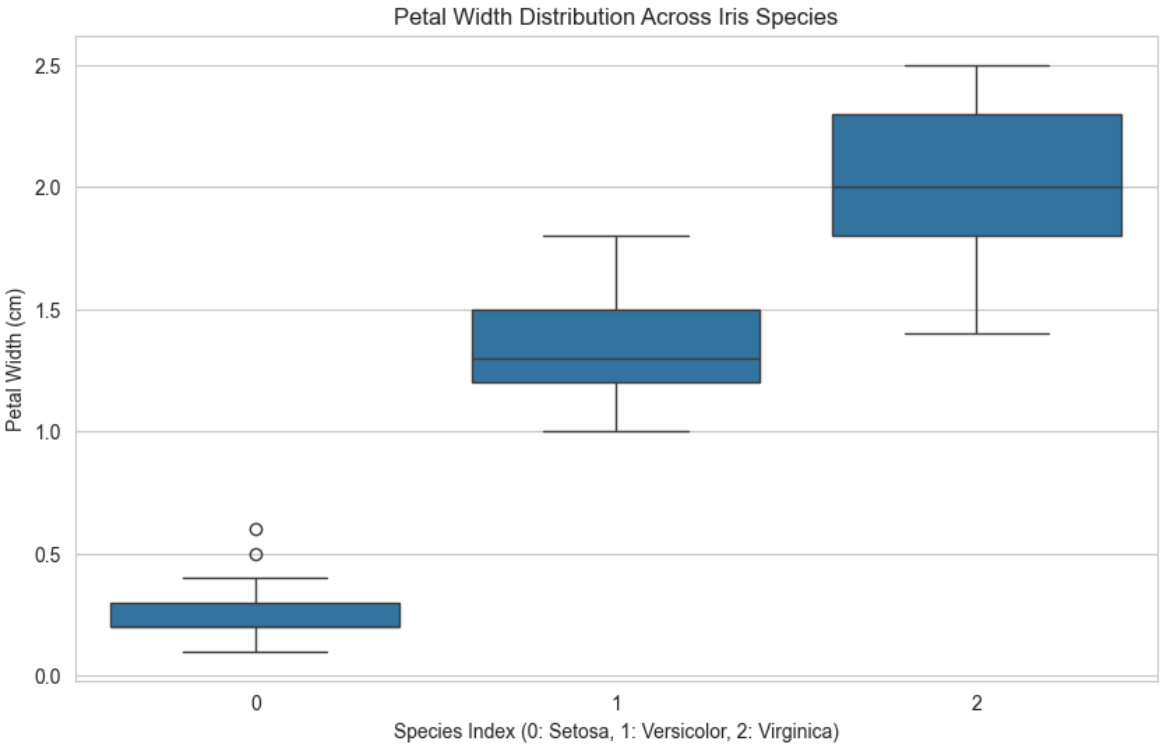
```
Visualization: Sepal Length Distribution
```



In [9]:
```
print("\nVisualization: Petal Width by Target Species")
plt.figure(figsize=(10, 6))
sns.boxplot(x='target', y='petal width (cm)', data=df)
plt.title('Petal Width Distribution Across Iris Species')
```

```
plt.xlabel('Species Index (0: Setosa, 1: Versicolor, 2: Virginica)')
plt.ylabel('Petal Width (cm)')
plt.show()
```

```
Visualization: Petal Width by Target Species
```



In [ ]:

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

sns.set_style("whitegrid")
%matplotlib inline
```

In [2]:
```
df = sns.load_dataset('titanic')

print("Dataset loaded directly from Seaborn.")
print("\nInitial Data Exploration (First 5 Rows)")
display(df.head())

print("\nData Info (Columns, Non-Null Counts, Dtypes)")
df.info()
```

```
Dataset loaded directly from Seaborn.

Initial Data Exploration (First 5 Rows)
```

	survived	pclass	sex	age	sibsp	parch	fare	embarked	class	who	adult_
0	0	3	male	22.0	1	0	7.2500	S	Third	man	
1	1	1	female	38.0	1	0	71.2833	C	First	woman	
2	1	3	female	26.0	0	0	7.9250	S	Third	woman	
3	1	1	female	35.0	1	0	53.1000	S	First	woman	
4	0	3	male	35.0	0	0	8.0500	S	Third	man	

```
Data Info (Columns, Non-Null Counts, Dtypes)
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 15 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   survived     891 non-null    int64
 1   pclass       891 non-null    int64
 2   sex          891 non-null    object
 3   age          714 non-null    float64
 4   sibsp        891 non-null    int64
 5   parch        891 non-null    int64
 6   fare         891 non-null    float64
 7   embarked     889 non-null    object
 8   class        891 non-null    category
 9   who          891 non-null    object
 10  adult_male   891 non-null    bool
 11  deck         203 non-null    category
 12  embark_town  889 non-null    object
 13  alive        891 non-null    object
 14  alone        891 non-null    bool
dtypes: bool(2), category(2), float64(2), int64(4), object(5)
memory usage: 80.7+ KB
```

In [4]:
```
print("Missing Value Counts:")
print(df.isnull().sum())
```

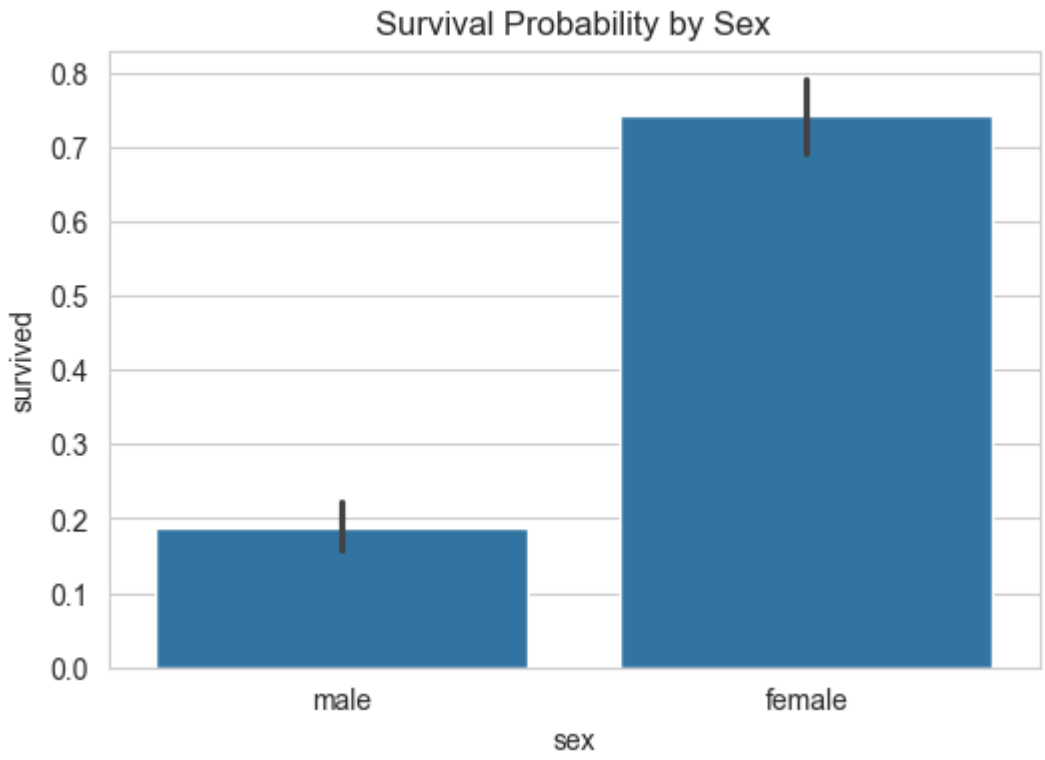
```
print("\nDropping columns with too many missing values ('deck' and other non-ess
df = df.drop(['deck', 'embark_town'], axis=1)

print("\nFilling missing 'age' values with the mean age.")
df['age'].fillna(df['age'].mean(), inplace=True)

print("\nFilling missing 'embarked' value with the mode (most frequent port).")
df['embarked'].fillna(df['embarked'].mode()[0], inplace=True)

print("\nFinal Check for Missing Values:")
print(df.isnull().sum())
```

```
Missing Value Counts:
survived         0
pclass           0
sex              0
age            177
sibsp            0
parch            0
fare             0
embarked         2
class            0
who              0
adult_male       0
deck           688
embark_town      2
alive            0
alone            0
dtype: int64

Dropping columns with too many missing values ('deck' and other non-essential fea
tures) for simplicity.

Filling missing 'age' values with the mean age.

Filling missing 'embarked' value with the mode (most frequent port).

Final Check for Missing Values:
survived      0
pclass        0
sex           0
age           0
sibsp         0
parch         0
fare          0
embarked      0
class         0
who           0
adult_male    0
alive         0
alone         0
dtype: int64
```

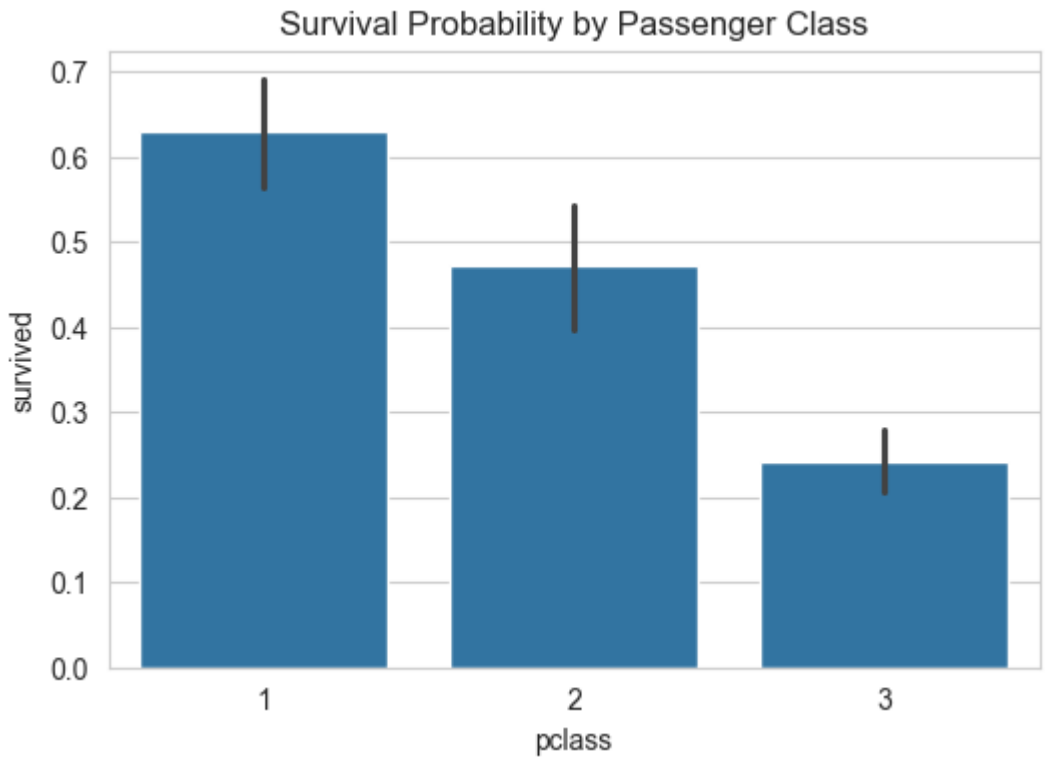
```
C:\Users\chith\AppData\Local\Temp\ipykernel_8736\298199241.py:9: FutureWarning: A
value is trying to be set on a copy of a DataFrame or Series through chained assi
gnment using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never work becau
se the intermediate object on which we are setting values always behaves as a cop
y.

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.meth
od({col: value}, inplace=True)' or df[col] = df[col].method(value) instead, to pe
rform the operation inplace on the original object.


  df['age'].fillna(df['age'].mean(), inplace=True)
C:\Users\chith\AppData\Local\Temp\ipykernel_8736\298199241.py:12: FutureWarning:
A value is trying to be set on a copy of a DataFrame or Series through chained as
signment using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never work becau
se the intermediate object on which we are setting values always behaves as a cop
y.

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.meth
od({col: value}, inplace=True)' or df[col] = df[col].method(value) instead, to pe
rform the operation inplace on the original object.


  df['embarked'].fillna(df['embarked'].mode()[0], inplace=True)
```

In [5]:
```
print("Visualization: Survival Rate by Sex")
plt.figure(figsize=(6, 4))
sns.barplot(x='sex', y='survived', data=df)
plt.title('Survival Probability by Sex')
plt.show()

print("\nVisualization: Survival Rate by Passenger Class (pclass)")
plt.figure(figsize=(6, 4))
sns.barplot(x='pclass', y='survived', data=df)
plt.title('Survival Probability by Passenger Class')
plt.show()
```

```
Visualization: Survival Rate by Sex
```



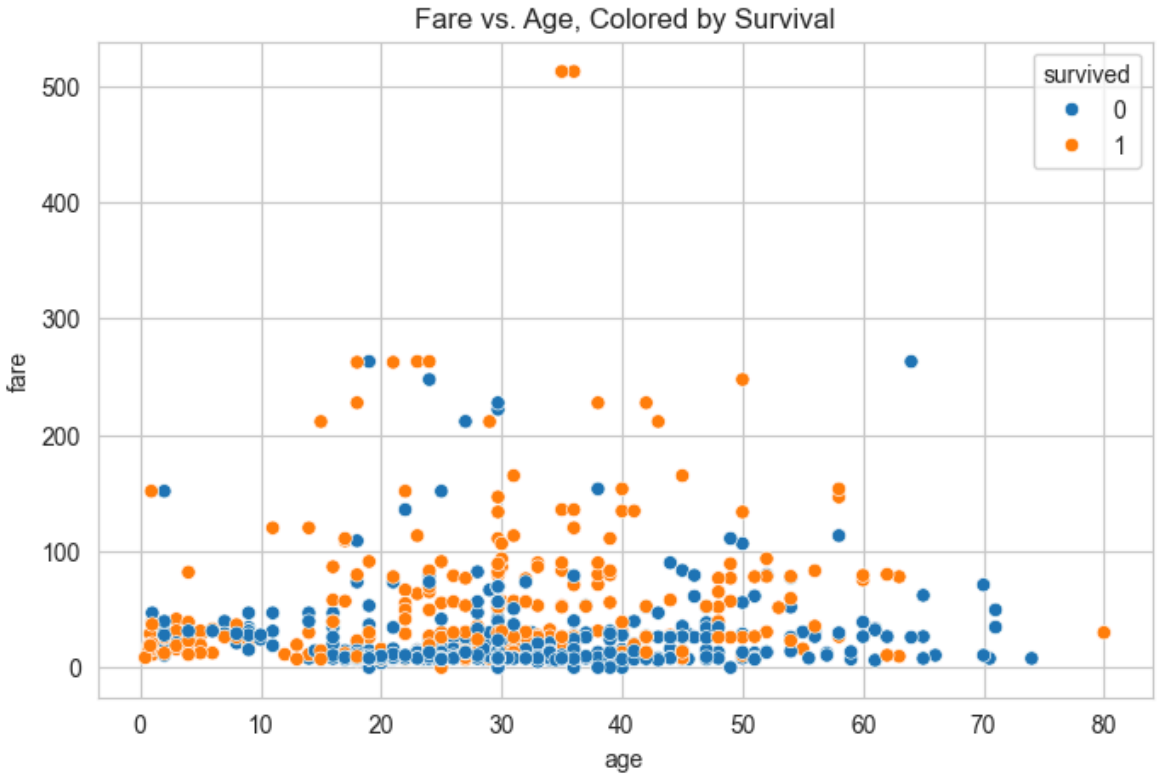
Visualization: Survival Rate by Passenger Class (pclass)



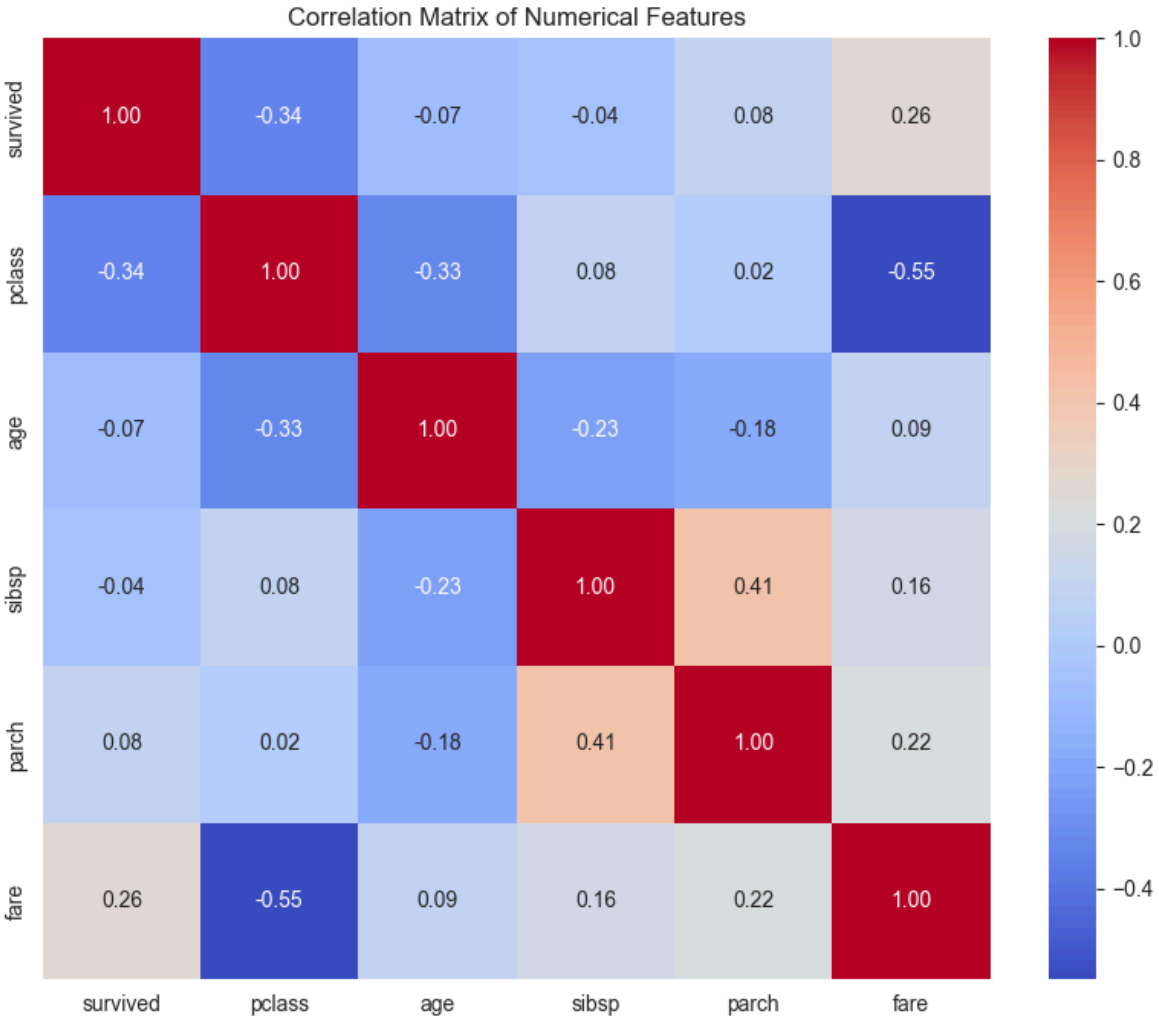
In [6]:
```
print("Visualization: Fare vs. Age")
plt.figure(figsize=(8, 5))
sns.scatterplot(x='age', y='fare', hue='survived', data=df)
plt.title('Fare vs. Age, Colored by Survival')
plt.show()

print("\nVisualization: Correlation Heatmap (Relationships)")
plt.figure(figsize=(10, 8))
correlation_matrix = df[['survived', 'pclass', 'age', 'sibsp', 'parch', 'fare']]
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt=".2f")
```

```
plt.title('Correlation Matrix of Numerical Features')
plt.show()
```

Visualization: Fare vs. Age



Visualization: Correlation Heatmap (Relationships)



In [ ]:

In [2]:
```
import pandas as pd
import numpy as np

data = {'Transaction_ID': [101, 102, 103, 104, 105],
        'Price_USD': [15000, 32000, 11000, 50000, 25000],
        'Units_Sold': [10, 5, 15, 2, 8],
        'Region': ['East', 'West', 'East', 'North', 'West']}
df_data = pd.DataFrame(data)

csv_file_name = 'scaling_data.csv'
df_data.to_csv(csv_file_name, index=False)

print(f"File '{csv_file_name}' successfully re-created.")
```

```
File 'scaling_data.csv' successfully re-created.
```

In [3]:
```
from sklearn.preprocessing import MinMaxScaler
import numpy as np

df = pd.read_csv('scaling_data.csv')

print("Original DataFrame loaded from CSV:")
display(df)
```

```
Original DataFrame loaded from CSV:
```

	Transaction_ID	Price_USD	Units_Sold	Region
0	101	15000	10	East
1	102	32000	5	West
2	103	11000	15	East
3	104	50000	2	North
4	105	25000	8	West

In [4]:
```
numerical_cols = ['Price_USD', 'Units_Sold']
scaler = MinMaxScaler()

df_scaled_features = scaler.fit_transform(df[numerical_cols])

df_scaled = pd.DataFrame(df_scaled_features, columns=numerical_cols)

for col in numerical_cols:
    df[col] = df_scaled[col]

print("Feature Scaling (Min-Max Normalization) complete.")

print("\nDataFrame with Scaled Features:")
display(df)
```

```
Feature Scaling (Min-Max Normalization) complete.

DataFrame with Scaled Features:
```

	Transaction_ID	Price_USD	Units_Sold	Region
0	101	0.102564	0.615385	East
1	102	0.538462	0.230769	West
2	103	0.000000	1.000000	East
3	104	1.000000	0.000000	North
4	105	0.358974	0.461538	West

In [ ]:

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

In [2]:
```
sns.set_style("whitegrid")
%matplotlib inline
```

In [3]:
```
iris = load_iris(as_frame=True)
df = iris.frame
print("Iris dataset loaded for outlier detection.")
```

```
Iris dataset loaded for outlier detection.
```

In [4]:
```
print("\nFirst 5 rows of data")
display(df.head())
```

```
First 5 rows of data
```

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

In [5]:
```
print("\nSummary statistics")
display(df.describe())
```

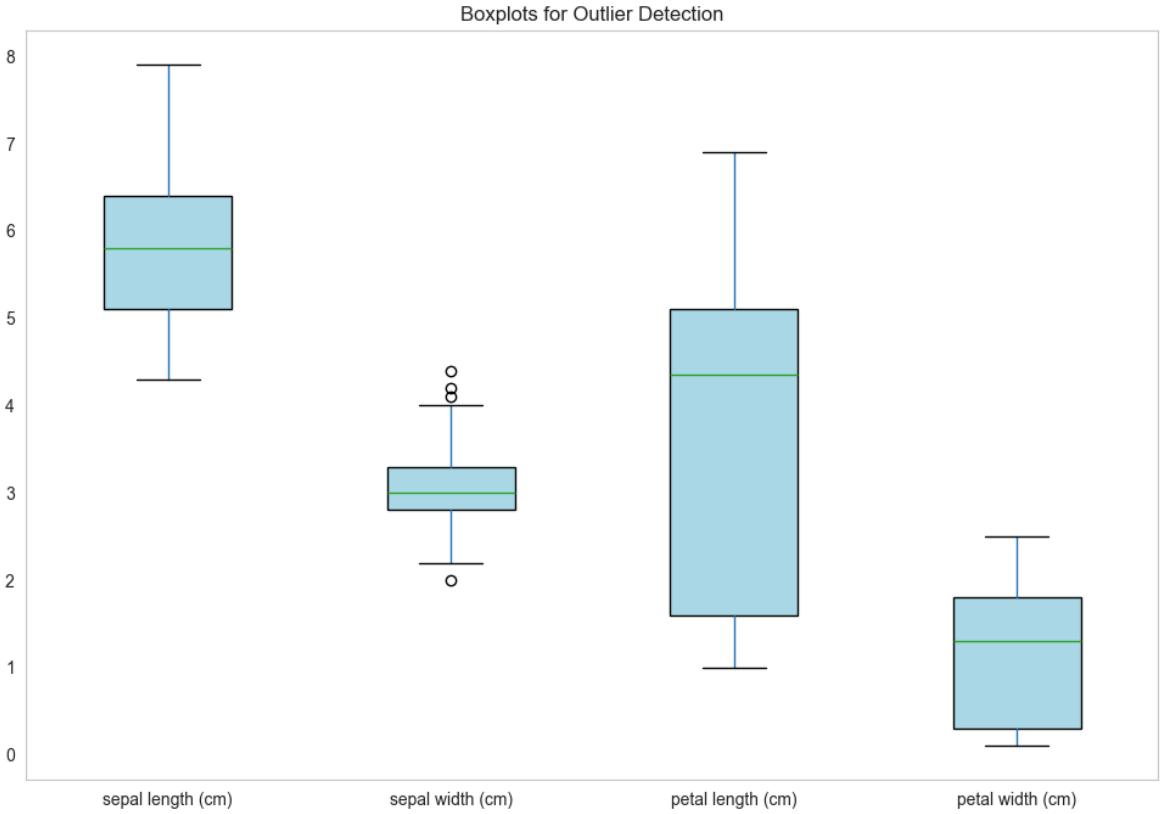
```
Summary statistics
```

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	target
count	150.000000	150.000000	150.000000	150.000000	150.000000
mean	5.843333	3.057333	3.758000	1.199333	1.000000
std	0.828066	0.435866	1.765298	0.762238	0.819232
min	4.300000	2.000000	1.000000	0.100000	0.000000
25%	5.100000	2.800000	1.600000	0.300000	0.000000
50%	5.800000	3.000000	4.350000	1.300000	1.000000
75%	6.400000	3.300000	5.100000	1.800000	2.000000
max	7.900000	4.400000	6.900000	2.500000	2.000000

In [6]:
```
print("\nBoxplots for detecting outliers in numeric features")
plt.figure(figsize=(12,8))
df[df.columns[:-1]].boxplot(grid=False, patch_artist=True, boxprops=dict(facecol
```

```
plt.title("Boxplots for Outlier Detection")
plt.show()
```

Boxplots for detecting outliers in numeric features

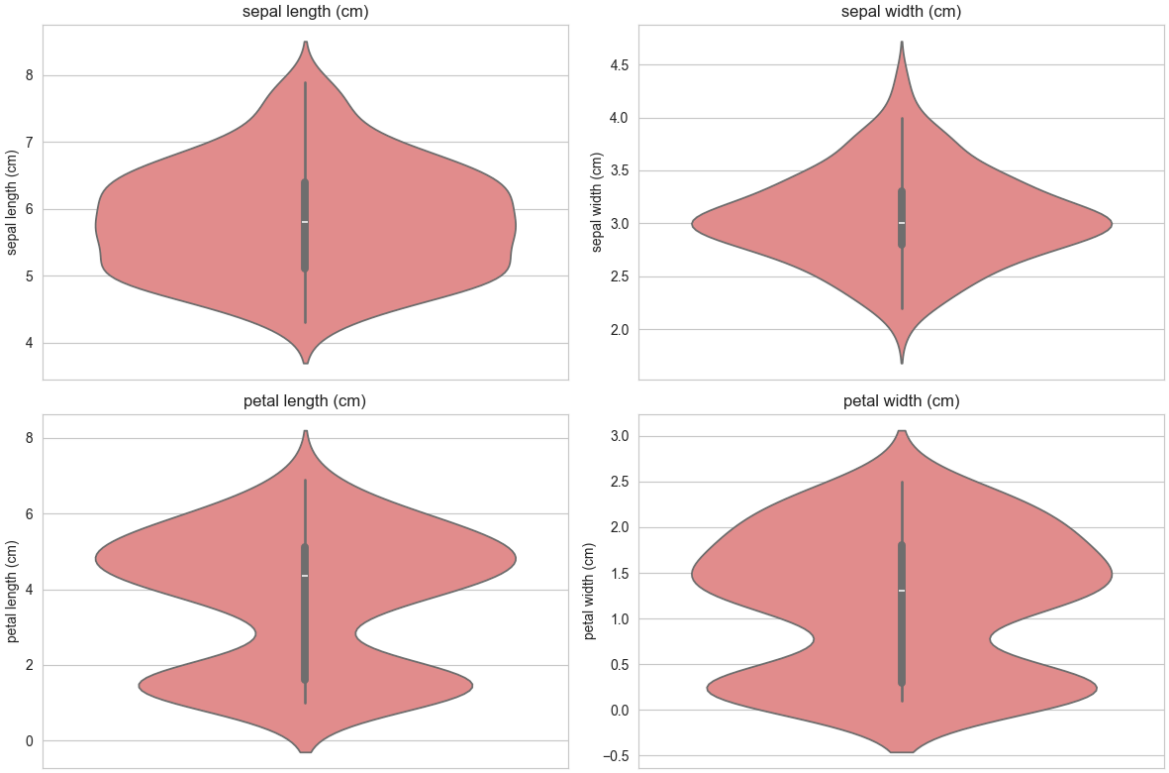


In [7]:
```
print("\nDetecting outliers using IQR method")
Q1 = df.quantile(0.25)
Q3 = df.quantile(0.75)
IQR = Q3 - Q1
outliers = ((df < (Q1 - 1.5 * IQR)) | (df > (Q3 + 1.5 * IQR)))
display(outliers.sum())
```

```
Detecting outliers using IQR method
sepal length (cm)    0
sepal width (cm)     4
petal length (cm)    0
petal width (cm)     0
target               0
dtype: int64
```

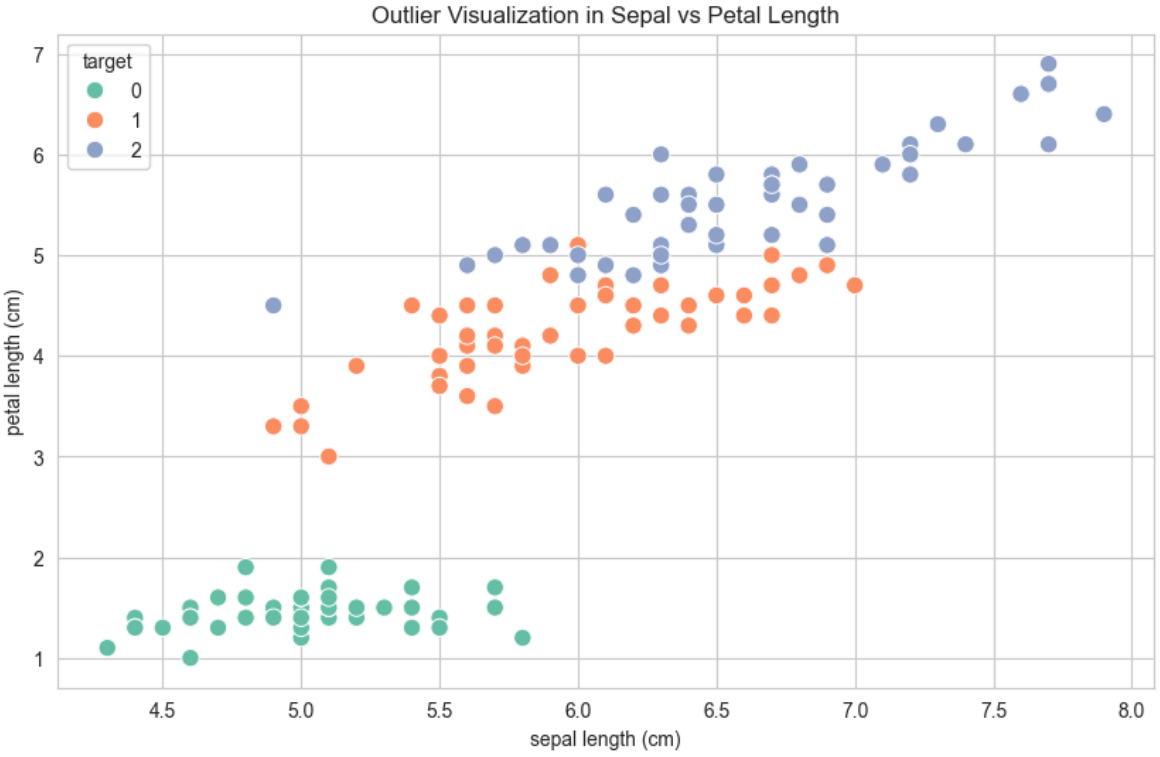
In [8]:
```
print("\nVisualizing outliers with violin plots")
plt.figure(figsize=(12,8))
for i, column in enumerate(df.columns[:-1], 1):
    plt.subplot(2,2,i)
    sns.violinplot(y=df[column], color='lightcoral')
    plt.title(column)
plt.tight_layout()
plt.show()
```

Visualizing outliers with violin plots



In [9]:
```
print("\nOutlier visualization using scatter plots")
plt.figure(figsize=(10,6))
sns.scatterplot(x='sepal length (cm)', y='petal length (cm)', data=df, hue='targ
plt.title("Outlier Visualization in Sepal vs Petal Length")
plt.show()
```

```
Outlier visualization using scatter plots
```



In [10]:
```
print("\nRemoving outliers for cleaner dataset")
df_clean = df[~((df < (Q1 - 1.5 * IQR)) | (df > (Q3 + 1.5 * IQR))).any(axis=1)]
print(f"Original dataset size: {df.shape[0]}")
print(f"After removing outliers: {df_clean.shape[0]}")
```

```
Removing outliers for cleaner dataset
Original dataset size: 150
After removing outliers: 146
```

In [ ]:

In [1]:
```
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.metrics import mean_squared_error, r2_score
```

In [2]:
```
from sklearn.datasets import fetch_california_housing
data = fetch_california_housing(as_frame=True)
df = data.frame
print("California Housing dataset loaded successfully.")
```

```
California Housing dataset loaded successfully.
```

In [3]:
```
display(df.head())
```

	MedInc	HouseAge	AveRooms	AveBedrms	Population	AveOccup	Latitude	Longitud
0	8.3252	41.0	6.984127	1.023810	322.0	2.555556	37.88	-122.2
1	8.3014	21.0	6.238137	0.971880	2401.0	2.109842	37.86	-122.2
2	7.2574	52.0	8.288136	1.073446	496.0	2.802260	37.85	-122.2
3	5.6431	52.0	5.817352	1.073059	558.0	2.547945	37.85	-122.2
4	3.8462	52.0	6.281853	1.081081	565.0	2.181467	37.85	-122.2

In [4]:
```
print("\nDataset Info")
df.info()
```

```
Dataset Info
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 20640 entries, 0 to 20639
Data columns (total 9 columns):
 #   Column       Non-Null Count  Dtype  
---  ------       --------------  -----  
 0   MedInc       20640 non-null  float64
 1   HouseAge     20640 non-null  float64
 2   AveRooms     20640 non-null  float64
 3   AveBedrms    20640 non-null  float64
 4   Population   20640 non-null  float64
 5   AveOccup     20640 non-null  float64
 6   Latitude     20640 non-null  float64
 7   Longitude    20640 non-null  float64
 8   MedHouseVal  20640 non-null  float64
dtypes: float64(9)
memory usage: 1.4 MB
```

In [5]:
```
print("\nChecking for Missing Values")
print(df.isnull().sum())
```

```
Checking for Missing Values
MedInc          0
HouseAge        0
AveRooms        0
AveBedrms       0
Population      0
AveOccup        0
Latitude        0
Longitude       0
MedHouseVal     0
dtype: int64
```

In [6]:
```
X = df[['MedInc']]
y = df['MedHouseVal']
```

In [7]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_
```

In [8]:
```
model = LinearRegression()
model.fit(X_train, y_train)
```

Out[8]:

▾ LinearRegression

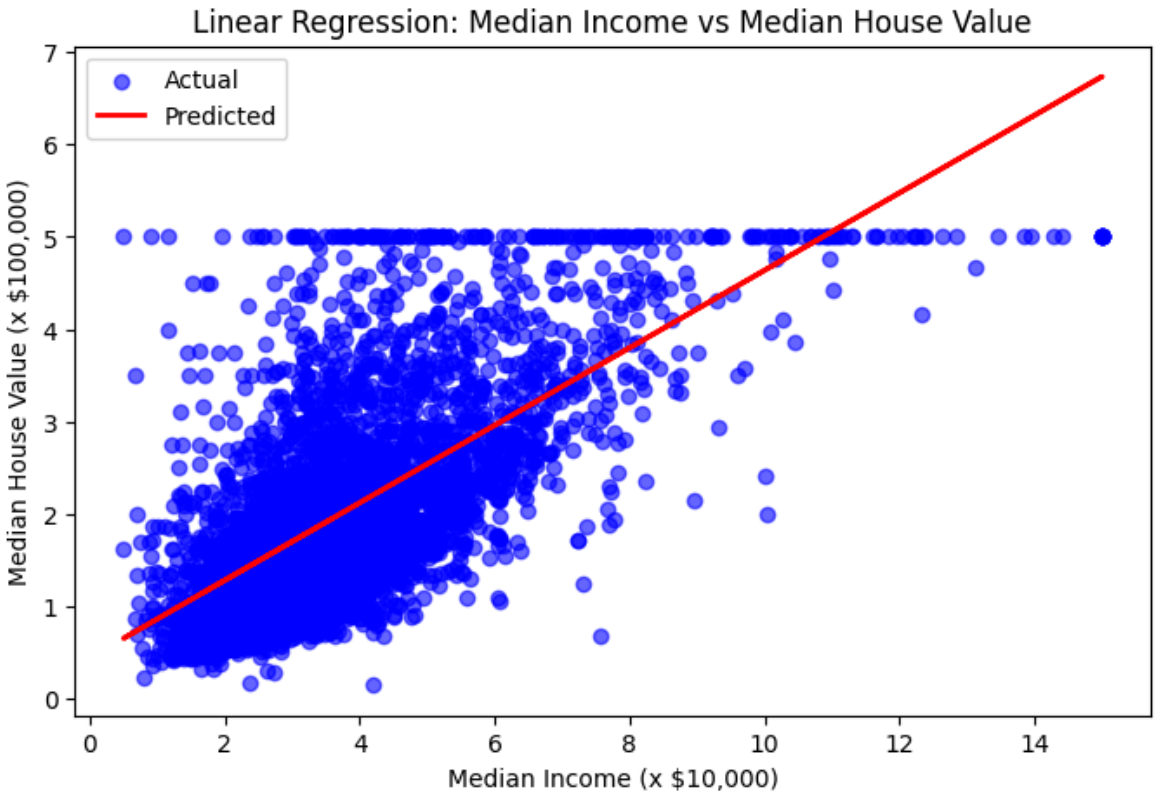
▶ Parameters

In [9]:
```
y_pred = model.predict(X_test)
```

In [10]:
```
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)
print(f"Mean Squared Error: {mse}")
print(f"R² Score: {r2}")
```

```
Mean Squared Error: 0.7091157771765548
R² Score: 0.45885918903846656
```

In [11]:
```
plt.figure(figsize=(8,5))
plt.scatter(X_test, y_test, color='blue', label='Actual', alpha=0.6)
plt.plot(X_test, y_pred, color='red', linewidth=2, label='Predicted')
plt.title('Linear Regression: Median Income vs Median House Value')
plt.xlabel('Median Income (x $10,000)')
plt.ylabel('Median House Value (x $100,000)')
plt.legend()
plt.show()
```


Linear Regression: Median Income vs Median House Value
Actual
Predicted
Median House Value (x $100,000)
Median Income (x $10,000)
0
1
2
3
4
5
6
7
0
2
4
6
8
10
12
14

In [ ]:

In [1]:
```
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 LogisticRegression
from sklearn.metrics import accuracy_score, confusion_matrix, classification_rep
from sklearn.datasets import load_breast_cancer
```

In [2]:
```
data = load_breast_cancer(as_frame=True)
df = data.frame
print("Breast Cancer dataset loaded successfully.")
```

```
Breast Cancer dataset loaded successfully.
```

In [3]:
```
display(df.head())
```

	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity	mean concave points	syn
0	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.3001	0.14710	
1	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.0869	0.07017	
2	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.1974	0.12790	
3	11.42	20.38	77.58	386.1	0.14250	0.28390	0.2414	0.10520	
4	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.1980	0.10430	

5 rows × 31 columns

In [4]:
```
print("\nDataset Info")
df.info()
```

```
Dataset Info
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 569 entries, 0 to 568
Data columns (total 31 columns):
 #   Column                   Non-Null Count  Dtype  
---  ------                   --------------  -----  
 0   mean radius              569 non-null    float64
 1   mean texture             569 non-null    float64
 2   mean perimeter           569 non-null    float64
 3   mean area                569 non-null    float64
 4   mean smoothness          569 non-null    float64
 5   mean compactness         569 non-null    float64
 6   mean concavity           569 non-null    float64
 7   mean concave points      569 non-null    float64
 8   mean symmetry            569 non-null    float64
 9   mean fractal dimension   569 non-null    float64
 10  radius error             569 non-null    float64
 11  texture error            569 non-null    float64
 12  perimeter error          569 non-null    float64
 13  area error               569 non-null    float64
 14  smoothness error         569 non-null    float64
 15  compactness error        569 non-null    float64
 16  concavity error          569 non-null    float64
 17  concave points error     569 non-null    float64
 18  symmetry error           569 non-null    float64
 19  fractal dimension error  569 non-null    float64
 20  worst radius             569 non-null    float64
 21  worst texture            569 non-null    float64
 22  worst perimeter          569 non-null    float64
 23  worst area               569 non-null    float64
 24  worst smoothness         569 non-null    float64
 25  worst compactness        569 non-null    float64
 26  worst concavity          569 non-null    float64
 27  worst concave points     569 non-null    float64
 28  worst symmetry           569 non-null    float64
 29  worst fractal dimension  569 non-null    float64
 30  target                   569 non-null    int64  
dtypes: float64(30), int64(1)
memory usage: 137.9 KB
```

In [5]:
```
print("\nChecking for Missing Values")
print(df.isnull().sum())
```

```
Checking for Missing Values
mean radius                0
mean texture               0
mean perimeter             0
mean area                  0
mean smoothness            0
mean compactness           0
mean concavity             0
mean concave points        0
mean symmetry              0
mean fractal dimension     0
radius error               0
texture error              0
perimeter error            0
area error                 0
smoothness error           0
compactness error          0
concavity error            0
concave points error       0
symmetry error             0
fractal dimension error    0
worst radius               0
worst texture              0
worst perimeter            0
worst area                 0
worst smoothness           0
worst compactness          0
worst concavity            0
worst concave points       0
worst symmetry             0
worst fractal dimension    0
target                     0
dtype: int64
```

In [6]:
```
X = df.drop('target', axis=1)
y = df['target']
```

In [7]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_
```

In [8]:
```
model = LogisticRegression(max_iter=5000)
model.fit(X_train, y_train)
```

Out[8]:

▾ LogisticRegression

▶ Parameters

In [9]:
```
y_pred = model.predict(X_test)
```

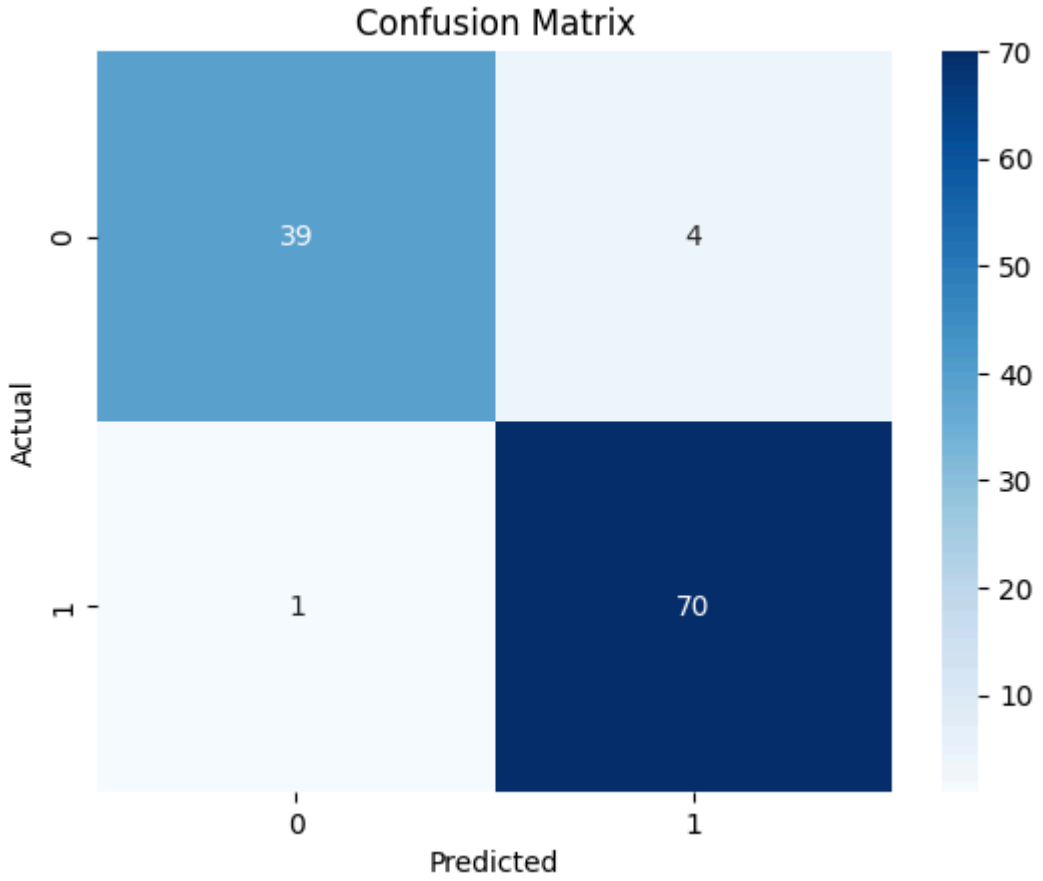
In [10]:
```
acc = accuracy_score(y_test, y_pred)
print(f"Accuracy: {acc}")
```

```
Accuracy: 0.956140350877193
```

In [11]:
```
print("\nConfusion Matrix")
cm = confusion_matrix(y_test, y_pred)
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')
plt.title('Confusion Matrix')
plt.xlabel('Predicted')
```

```
plt.ylabel('Actual')
plt.show()
```

```
Confusion Matrix
```



In [12]:
```
print("\nClassification Report")
print(classification_report(y_test, y_pred))
```

```
Classification Report
              precision    recall  f1-score   support

           0       0.97      0.91      0.94        43
           1       0.95      0.99      0.97        71

    accuracy                           0.96       114
   macro avg       0.96      0.95      0.95       114
weighted avg       0.96      0.96      0.96       114
```

In [ ]:
```

```

In [3]:
```
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.datasets import load_iris
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, confusion_matrix, classification_rep
```

In [4]:
```
iris = load_iris(as_frame=True)
df = iris.frame
print("Iris dataset loaded successfully.")
```

```
Iris dataset loaded successfully.
```

In [5]:
```
display(df.head())
```

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

In [6]:
```
print("\nDataset Info")
df.info()
```

```
Dataset Info
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 150 entries, 0 to 149
Data columns (total 5 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   sepal length (cm)  150 non-null    float64
 1   sepal width (cm)   150 non-null    float64
 2   petal length (cm)  150 non-null    float64
 3   petal width (cm)   150 non-null    float64
 4   target             150 non-null    int64  
dtypes: float64(4), int64(1)
memory usage: 6.0 KB
```

In [7]:
```
print("\nChecking for Missing Values")
print(df.isnull().sum())
```

```
Checking for Missing Values
sepal length (cm)    0
sepal width (cm)     0
petal length (cm)    0
petal width (cm)     0
target               0
dtype: int64
```

In [8]:
```
X = df.drop('target', axis=1)
y = df['target']
```

In [9]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_
```

In [10]:
```
knn = KNeighborsClassifier(n_neighbors=5)
knn.fit(X_train, y_train)
```

Out[10]:
```
▾ KNeighborsClassifier
▶ Parameters
```

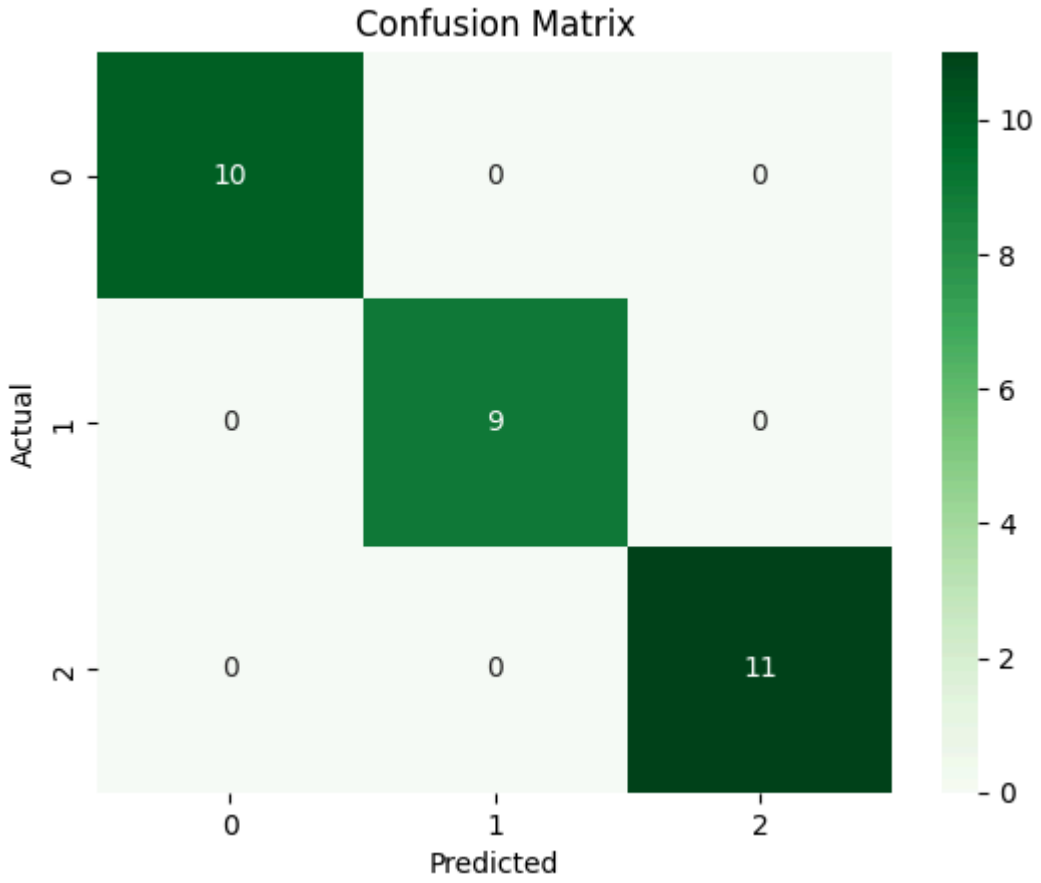
In [11]:
```
y_pred = knn.predict(X_test)
```

In [12]:
```
acc = accuracy_score(y_test, y_pred)
print(f"Accuracy: {acc}")
```

```
Accuracy: 1.0
```

In [13]:
```
print("\nConfusion Matrix")
cm = confusion_matrix(y_test, y_pred)
sns.heatmap(cm, annot=True, fmt='d', cmap='Greens')
plt.title('Confusion Matrix')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()
```

```
Confusion Matrix
```



In [14]:
```
print("\nClassification Report")
print(classification_report(y_test, y_pred))
```

```
Classification Report
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        10
           1       1.00      1.00      1.00         9
           2       1.00      1.00      1.00        11

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

In [ ]:

In [1]:
```
import pandas as pd
import numpy as np
from scipy import stats
```

In [2]:
```
np.random.seed(42)
group1 = np.random.normal(loc=50, scale=10, size=30)
group2 = np.random.normal(loc=55, scale=10, size=30)
```

In [3]:
```
print("Sample Mean of Group 1:", np.mean(group1))
print("Sample Mean of Group 2:", np.mean(group2))
```

```
Sample Mean of Group 1: 48.118531041489625
Sample Mean of Group 2: 53.788375297100565
```

In [4]:
```
t_stat, p_value = stats.ttest_ind(group1, group2)
print("T-statistic:", t_stat)
print("P-value:", p_value)
```

```
T-statistic: -2.3981151520102415
P-value: 0.019717941865758228
```

In [5]:
```
alpha = 0.05
if p_value < alpha:
    print("Reject Null Hypothesis: Significant difference between groups.")
else:
    print("Fail to Reject Null Hypothesis: No significant difference between gro
```

```
Reject Null Hypothesis: Significant difference between groups.
```

In [ ]:

In [1]:
```
import numpy as np
from statsmodels.stats.weightstats import ztest
```

In [2]:
```
np.random.seed(42)
sample1 = np.random.normal(loc=100, scale=15, size=50)
sample2 = np.random.normal(loc=105, scale=15, size=50)
```

In [3]:
```
z_stat, p_value = ztest(sample1, sample2)
print("Z-statistic:", z_stat)
print("P-value:", p_value)
```

```
Z-statistic: -3.187388009844473
P-value: 0.0014356404167852916
```

In [4]:
```
alpha = 0.05
if p_value < alpha:
    print("Reject Null Hypothesis: Significant difference between samples.")
else:
    print("Fail to Reject Null Hypothesis: No significant difference between sam
```

```
Reject Null Hypothesis: Significant difference between samples.
```

In [ ]:
```

```

In [1]:
```
import numpy as np
import pandas as pd

group1 = np.array([12,14,15,16,14,15,13,14])
group2 = np.array([10,11,12,13,12,11,10,12])

group1, group2
```

Out[1]:
```
(array([12, 14, 15, 16, 14, 15, 13, 14]),
 array([10, 11, 12, 13, 12, 11, 10, 12]))
```

In [2]:
```
var1 = group1.var(ddof=1)
var2 = group2.var(ddof=1)

var1, var2
```

Out[2]:
```
(np.float64(1.5535714285714286), np.float64(1.125))
```

In [3]:
```
F = var1 / var2
F
```

Out[3]:
```
np.float64(1.380952380952381)
```

In [4]:
```
from scipy.stats import f

df1 = len(group1) - 1
df2 = len(group2) - 1

p_value = 1 - f.cdf(F, df1, df2)
critical_value = f.ppf(0.95, df1, df2)

F, p_value, critical_value
```

Out[4]:
```
(np.float64(1.380952380952381),
 np.float64(0.3404622007754843),
 np.float64(3.7870435399280677))
```

In [5]:
```
alpha = 0.05
"Reject H0" if F > critical_value else "Fail to Reject H0"
```

Out[5]:
```
'Fail to Reject H0'
```

In [ ]:

In [1]:
```
import numpy as np
from scipy import stats
```

In [2]:
```
np.random.seed(42)
group1 = np.random.normal(50, 10, 30)
group2 = np.random.normal(55, 10, 30)
group3 = np.random.normal(60, 10, 30)
```

In [3]:
```
f_stat, p_value = stats.f_oneway(group1, group2, group3)
print("F-statistic:", f_stat)
print("P-value:", p_value)
```

```
F-statistic: 12.209525517972809
P-value: 2.1200748140507085e-05
```

In [4]:
```
alpha = 0.05
if p_value < alpha:
    print("Reject Null Hypothesis: At least one group mean is different.")
else:
    print("Fail to Reject Null Hypothesis: No significant difference between gro
```

```
Reject Null Hypothesis: At least one group mean is different.
```

In [ ]:

In [1]:
```
import pandas as pd

data = {
    "Gender": ["Male","Male","Male","Female","Female","Female","Female","Male","
    "Purchase": ["Yes","No","Yes","Yes","No","No","Yes","Yes","No","Yes"]
}

df = pd.DataFrame(data)
df
```

Out[1]:

	Gender	Purchase
0	Male	Yes
1	Male	No
2	Male	Yes
3	Female	Yes
4	Female	No
5	Female	No
6	Female	Yes
7	Male	Yes
8	Male	No
9	Female	Yes

In [2]:
```
table = pd.crosstab(df["Gender"], df["Purchase"])
table
```

Out[2]:

Purchase	No	Yes
Gender		
Female	2	3
Male	2	3

In [3]:
```
from scipy.stats import chi2_contingency

chi2, p, dof, expected = chi2_contingency(table)
chi2, p, dof
```

Out[3]:
```
(np.float64(0.0), np.float64(1.0), 1)
```

In [4]:
```
expected
```

Out[4]:
```
array([[2., 3.],
       [2., 3.]])
```

In [5]:
```
alpha = 0.05
"Reject H0" if p < alpha else "Fail to Reject H0"
```

Out[5]: 'Fail to Reject H0'

In [ ]: