

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

```
In [2]: df = pd.read_csv("dsbda.csv")
print("First 5 rows:\n", df.head())
```

First 5 rows:

	StudentID	Age	Gender	Ethnicity	ParentalEducation	StudyTimeWeekly	\
0	1001	17	1	0	2	19.833723	
1	1002	18	0	0	1	15.408756	
2	1003	15	0	2	3	4.210570	
3	1004	17	1	0	3	10.028829	
4	1005	17	1	0	2	4.672495	

	Absences	Tutoring	ParentalSupport	Extracurricular	Sports	Music	\
0	7	1	2	0	0	1	
1	0	0	1	0	0	0	
2	26	0	2	0	0	0	
3	14	0	3	1	0	0	
4	17	1	3	0	0	0	

	Volunteering	GPA	GradeClass
0	0	2.929196	2.0
1	0	3.042915	1.0
2	0	0.112602	4.0
3	0	2.054218	3.0
4	0	1.288061	4.0

```
In [3]: print("\nMissing values:\n", df.isnull().sum())
print("\nStatistical Summary:\n", df.describe())
```

Missing values:

```

StudentID      0
Age            0
Gender         0
Ethnicity      0
ParentalEducation  0
StudyTimeWeekly  0
Absences       0
Tutoring       0
ParentalSupport  0
Extracurricular  0
Sports         0
Music          0
Volunteering   0
GPA            0
GradeClass     0
dtype: int64

```

Statistical Summary:

	StudentID	Age	Gender	Ethnicity	ParentalEducation \
count	2392.000000	2392.000000	2392.000000	2392.000000	2392.000000
mean	2196.500000	16.468645	0.510870	0.877508	1.746237
std	690.655244	1.123798	0.499986	1.028476	1.000411
min	1001.000000	15.000000	0.000000	0.000000	0.000000
25%	1598.750000	15.000000	0.000000	0.000000	1.000000
50%	2196.500000	16.000000	1.000000	0.000000	2.000000
75%	2794.250000	17.000000	1.000000	2.000000	2.000000
max	3392.000000	18.000000	1.000000	3.000000	4.000000

	StudyTimeWeekly	Absences	Tutoring	ParentalSupport \
count	2392.000000	2392.000000	2392.000000	2392.000000
mean	9.771992	14.541388	0.301421	2.122074
std	5.652774	8.467417	0.458971	1.122813
min	0.001057	0.000000	0.000000	0.000000
25%	5.043079	7.000000	0.000000	1.000000
50%	9.705363	15.000000	0.000000	2.000000
75%	14.408410	22.000000	1.000000	3.000000
max	19.978094	29.000000	1.000000	4.000000

	Extracurricular	Sports	Music	Volunteering	GPA \
count	2392.000000	2392.000000	2392.000000	2392.000000	2392.000000
mean	0.383361	0.303512	0.196906	0.157191	1.906186
std	0.486307	0.459870	0.397744	0.364057	0.915156
min	0.000000	0.000000	0.000000	0.000000	0.000000
25%	0.000000	0.000000	0.000000	0.000000	1.174803
50%	0.000000	0.000000	0.000000	0.000000	1.893393
75%	1.000000	1.000000	0.000000	0.000000	2.622216
max	1.000000	1.000000	1.000000	1.000000	4.000000

	GradeClass
count	2392.000000
mean	2.983696
std	1.233908
min	0.000000
25%	2.000000
50%	4.000000

```
75%      4.000000
max      4.000000
```

```
In [4]: print("\nRows:", df.shape[0])
        print("Columns:", df.shape[1])
        print("\nData Types:\n", df.dtypes)
```

```
Rows: 2392
Columns: 15
```

```
Data Types:
StudentID      int64
Age            int64
Gender         int64
Ethnicity      int64
ParentalEducation  int64
StudyTimeWeekly float64
Absences       int64
Tutoring       int64
ParentalSupport int64
Extracurricular int64
Sports         int64
Music          int64
Volunteering   int64
GPA            float64
GradeClass     float64
dtype: object
```

```
In [5]: df['Gender'] = df['Gender'].astype('category')
        df['Sports'] = df['Sports'].astype('category')
        df['Music'] = df['Music'].astype('category')
        df['Tutoring'] = df['Tutoring'].astype('category')

        print("\nUpdated Data Types:\n", df.dtypes)
```

```
Updated Data Types:
StudentID      int64
Age            int64
Gender         category
Ethnicity      int64
ParentalEducation  int64
StudyTimeWeekly float64
Absences       int64
Tutoring       category
ParentalSupport int64
Extracurricular int64
Sports         category
Music          category
Volunteering   int64
GPA            float64
GradeClass     float64
dtype: object
```

```
In [6]: df_encoded = pd.get_dummies(df, columns=[
        'Gender',
        'Sports',
        'Music',
```

```
'Tutoring']])
```

```
print("\nEncoded Data:\n", df_encoded.head())
```

Encoded Data:

	StudentID	Age	Ethnicity	ParentalEducation	StudyTimeWeekly	Absences	\
0	1001	17	0	2	19.833723	7	
1	1002	18	0	1	15.408756	0	
2	1003	15	2	3	4.210570	26	
3	1004	17	0	3	10.028829	14	
4	1005	17	0	2	4.672495	17	

	ParentalSupport	Extracurricular	Volunteering	GPA	GradeClass	\
0	2	0	0	2.929196	2.0	
1	1	0	0	3.042915	1.0	
2	2	0	0	0.112602	4.0	
3	3	1	0	2.054218	3.0	
4	3	0	0	1.288061	4.0	

	Gender_0	Gender_1	Sports_0	Sports_1	Music_0	Music_1	Tutoring_0	\
0	False	True	True	False	False	True	False	
1	True	False	True	False	True	False	True	
2	True	False	True	False	True	False	True	
3	False	True	True	False	True	False	True	
4	False	True	True	False	True	False	False	

	Tutoring_1
0	True
1	False
2	False
3	False
4	True

```
In [7]: print("\nFinal Shape:", df_encoded.shape)
```

Final Shape: (2392, 19)

```
In [ ]:
```

```
In [17]: import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

```
In [18]: data = {
    'StudentID': [1,2,3,4,5,6,7,8,9,10],
    'MathScore': [88, 92, np.nan, 70, 65, 85, 90, 87, 100, 55],
    'EnglishScore': [78, 85, 90, 68, 60, 82, 75, 88, 80, 94],
    'ScienceScore': [85, 90, 92, np.nan, 66, 88, 80, 95, 99, 60],
    'StudyHours': [10, 12, 15, 8, 6, 11, 9, 14, 13, 20],
    'Attendance': [95, 98, 100, 85, 80, 92, 90, 99, 97, 105]
}
```

```
In [19]: df = pd.DataFrame(data)

print("Original Data:\n", df.head())
```

Original Data:

	StudentID	MathScore	EnglishScore	ScienceScore	StudyHours	Attendance
0	1	88.0	78	85.0	10	95
1	2	92.0	85	90.0	12	98
2	3	NaN	90	92.0	15	100
3	4	70.0	68	NaN	8	85
4	5	65.0	60	66.0	6	80

```
In [20]: print("\nMissing values:\n", df.isnull().sum())
df['MathScore'] = df['MathScore'].fillna(df['MathScore'].mean())
df['ScienceScore'] = df['ScienceScore'].fillna(df['ScienceScore'].mean())
```

Missing values:

StudentID	0
MathScore	1
EnglishScore	0
ScienceScore	1
StudyHours	0
Attendance	0

dtype: int64

```
In [21]: df['Attendance'] = df['Attendance'].clip(0,100)
print("\nAfter cleaning:\n", df)
```

After cleaning:

	StudentID	MathScore	EnglishScore	ScienceScore	StudyHours	Attendance
0	1	88.000000	78	85.000000	10	95
1	2	92.000000	85	90.000000	12	98
2	3	81.333333	90	92.000000	15	100
3	4	70.000000	68	83.888889	8	85
4	5	65.000000	60	66.000000	6	80
5	6	85.000000	82	88.000000	11	92
6	7	90.000000	75	80.000000	9	90
7	8	87.000000	88	95.000000	14	99
8	9	100.000000	80	99.000000	13	97
9	10	55.000000	94	60.000000	20	100

```
In [22]: outliers = df[df['StudyHours'] > 15]
print("\nOutliers in StudyHours:\n", outliers)
df['StudyHours'] = np.where(df['StudyHours'] > 15, 15, df['StudyHours'])
```

Outliers in StudyHours:

	StudentID	MathScore	EnglishScore	ScienceScore	StudyHours	Attendance
9	10	55.0	94	60.0	20	100

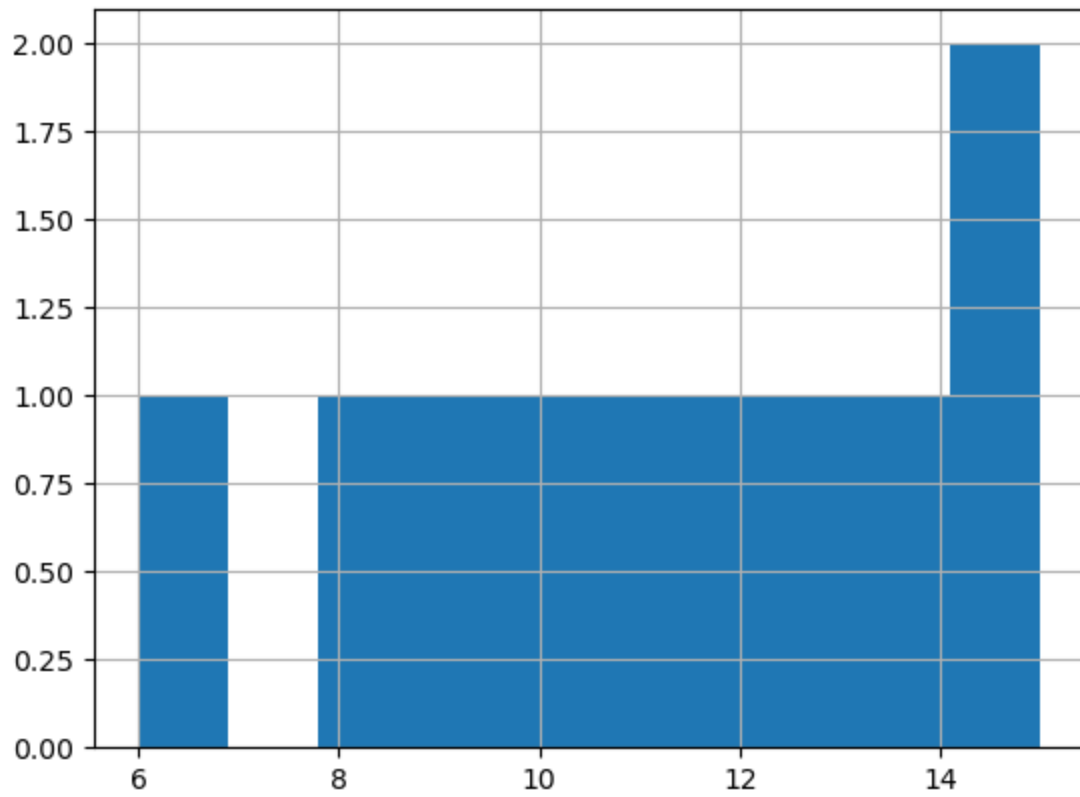
```
In [23]: df['StudyHours_log'] = np.log1p(df['StudyHours'])
print("\nFinal Data:\n", df)
```

Final Data:

	StudentID	MathScore	EnglishScore	ScienceScore	StudyHours	Attendance	\
0	1	88.000000	78	85.000000	10	95	
1	2	92.000000	85	90.000000	12	98	
2	3	81.333333	90	92.000000	15	100	
3	4	70.000000	68	83.888889	8	85	
4	5	65.000000	60	66.000000	6	80	
5	6	85.000000	82	88.000000	11	92	
6	7	90.000000	75	80.000000	9	90	
7	8	87.000000	88	95.000000	14	99	
8	9	100.000000	80	99.000000	13	97	
9	10	55.000000	94	60.000000	15	100	

	StudyHours_log
0	2.397895
1	2.564949
2	2.772589
3	2.197225
4	1.945910
5	2.484907
6	2.302585
7	2.708050
8	2.639057
9	2.772589

```
In [24]: df['StudyHours'].hist()
plt.show()
```



```
In [25]: df.to_csv("academic_performance.csv", index=False)
```

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

```
In [2]: df = pd.read_csv("dsbda.csv")
print("Dataset:\n", df.head())
```

Dataset:

	StudentID	Age	Gender	Ethnicity	ParentalEducation	StudyTimeWeekly	\
0	1001	17	1	0	2	19.833723	
1	1002	18	0	0	1	15.408756	
2	1003	15	0	2	3	4.210570	
3	1004	17	1	0	3	10.028829	
4	1005	17	1	0	2	4.672495	

	Absences	Tutoring	ParentalSupport	Extracurricular	Sports	Music	\
0	7	1	2	0	0	1	
1	0	0	1	0	0	0	
2	26	0	2	0	0	0	
3	14	0	3	1	0	0	
4	17	1	3	0	0	0	

	Volunteering	GPA	GradeClass
0	0	2.929196	2.0
1	0	3.042915	1.0
2	0	0.112602	4.0
3	0	2.054218	3.0
4	0	1.288061	4.0

```
In [3]: col = df["Age"]
print("\nBasic Statistics for Age:")
print("Mean:", col.mean())
print("Median:", col.median())
print("Maximum:", col.max())
print("Minimum:", col.min())
print("Standard Deviation:", col.std())
print("\nPercentiles:")
print(col.quantile([0.25, 0.5, 0.75]))
```

Basic Statistics for Age:

Mean: 16.468645484949832

Median: 16.0

Maximum: 18

Minimum: 15

Standard Deviation: 1.1237983798555635

Percentiles:

0.25 15.0

0.50 16.0

0.75 17.0

Name: Age, dtype: float64

```
In [4]: grouped = df.groupby("Ethnicity")["Age"]
summary = grouped.agg(['mean', 'median', 'min', 'max', 'std'])
print("\nSummary Statistics of Age grouped by Ethnicity:\n")
print(summary)
```

Summary Statistics of Age grouped by Ethnicity:

	mean	median	min	max	std
Ethnicity					
0	16.499586	16.0	15	18	1.126991
1	16.423935	16.0	15	18	1.110187
2	16.491489	17.0	15	18	1.132450
3	16.351351	16.0	15	18	1.114679

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 import metrics
```

```
In [10]: df = pd.read_csv(r"D:\Pradnya\TE\DSBDA\Practicals\new\boston-housing\train.csv")
print("First 5 rows:\n", df.head())
```

First 5 rows:

	ID	crim	zn	indus	chas	nox	rm	age	dis	rad	tax	\
0	1	0.00632	18.0	2.31	0	0.538	6.575	65.2	4.0900	1	296	
1	2	0.02731	0.0	7.07	0	0.469	6.421	78.9	4.9671	2	242	
2	4	0.03237	0.0	2.18	0	0.458	6.998	45.8	6.0622	3	222	
3	5	0.06905	0.0	2.18	0	0.458	7.147	54.2	6.0622	3	222	
4	7	0.08829	12.5	7.87	0	0.524	6.012	66.6	5.5605	5	311	

	ptratio	black	lstat	medv
0	15.3	396.90	4.98	24.0
1	17.8	396.90	9.14	21.6
2	18.7	394.63	2.94	33.4
3	18.7	396.90	5.33	36.2
4	15.2	395.60	12.43	22.9

```
In [11]: print("\nDataset Info:\n")
print(df.info())

print("\nStatistical Summary:\n")
print(df.describe())
```

Dataset Info:

<class 'pandas.core.frame.DataFrame'>

RangeIndex: 333 entries, 0 to 332

Data columns (total 15 columns):

#	Column	Non-Null Count	Dtype
0	ID	333 non-null	int64
1	crim	333 non-null	float64
2	zn	333 non-null	float64
3	indus	333 non-null	float64
4	chas	333 non-null	int64
5	nox	333 non-null	float64
6	rm	333 non-null	float64
7	age	333 non-null	float64
8	dis	333 non-null	float64
9	rad	333 non-null	int64
10	tax	333 non-null	int64
11	ptratio	333 non-null	float64
12	black	333 non-null	float64
13	lstat	333 non-null	float64
14	medv	333 non-null	float64

dtypes: float64(11), int64(4)

memory usage: 39.2 KB

None

Statistical Summary:

	ID	crim	zn	indus	chas	nox \
count	333.000000	333.000000	333.000000	333.000000	333.000000	333.000000
mean	250.951952	3.360341	10.689189	11.293483	0.060060	0.557144
std	147.859438	7.352272	22.674762	6.998123	0.237956	0.114955
min	1.000000	0.006320	0.000000	0.740000	0.000000	0.385000
25%	123.000000	0.078960	0.000000	5.130000	0.000000	0.453000
50%	244.000000	0.261690	0.000000	9.900000	0.000000	0.538000
75%	377.000000	3.678220	12.500000	18.100000	0.000000	0.631000
max	506.000000	73.534100	100.000000	27.740000	1.000000	0.871000

	rm	age	dis	rad	tax	ptratio \
count	333.000000	333.000000	333.000000	333.000000	333.000000	333.000000
mean	6.265619	68.226426	3.709934	9.633634	409.279279	18.448048
std	0.703952	28.133344	1.981123	8.742174	170.841988	2.151821
min	3.561000	6.000000	1.129600	1.000000	188.000000	12.600000
25%	5.884000	45.400000	2.122400	4.000000	279.000000	17.400000
50%	6.202000	76.700000	3.092300	5.000000	330.000000	19.000000
75%	6.595000	93.800000	5.116700	24.000000	666.000000	20.200000
max	8.725000	100.000000	10.710300	24.000000	711.000000	21.200000

	black	lstat	medv
count	333.000000	333.000000	333.000000
mean	359.466096	12.515435	22.768769
std	86.584567	7.067781	9.173468
min	3.500000	1.730000	5.000000
25%	376.730000	7.180000	17.400000
50%	392.050000	10.970000	21.600000

```

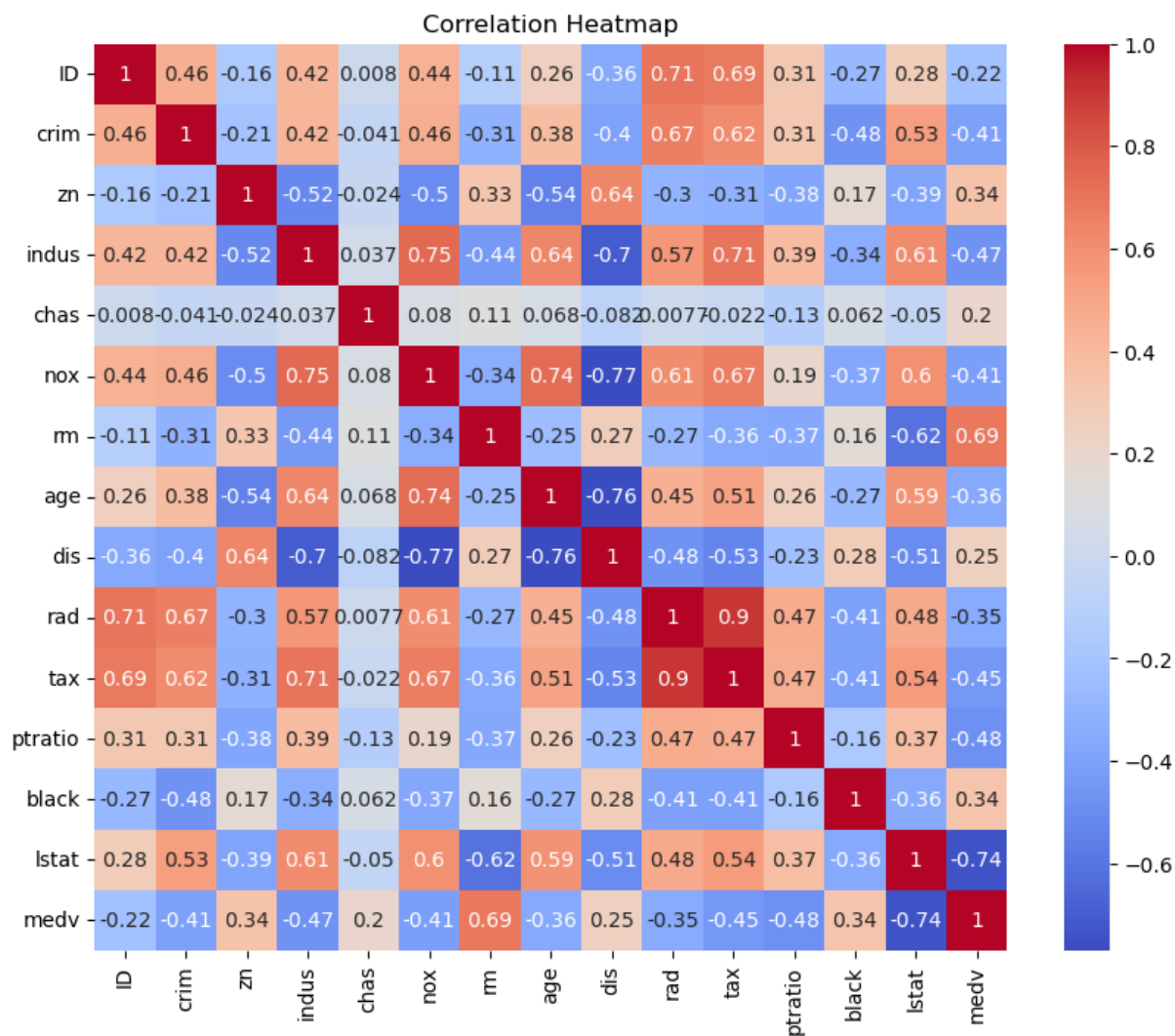
75%    396.240000    16.420000    25.000000
max    396.900000    37.970000    50.000000

```

```

In [12]: plt.figure(figsize=(10,8))
sns.heatmap(df.corr(), annot=True, cmap='coolwarm')
plt.title("Correlation Heatmap")
plt.show()

```



```

In [15]: target_col = 'MEDV' if 'MEDV' in df.columns else 'medv'
X = df.drop(target_col, axis=1)
y = df[target_col]
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)
coeff_df = pd.DataFrame(model.coef_, X.columns, columns=['Coefficient'])
print("\nModel Coefficients:\n", coeff_df)

```

Model Coefficients:

	Coefficient
ID	-0.004902
crim	-0.070217
zn	0.061337
indus	-0.024444
chas	4.174129
nox	-14.787057
rm	3.397953
age	-0.014945
dis	-1.863079
rad	0.421680
tax	-0.013171
ptratio	-0.734327
black	0.008077
lstat	-0.637599

```
In [16]: y_pred = model.predict(X_test)
print("\nModel Evaluation:")
print("Mean Absolute Error:", metrics.mean_absolute_error(y_test, y_pred))
print("Mean Squared Error:", metrics.mean_squared_error(y_test, y_pred))
print("Root Mean Squared Error:", np.sqrt(metrics.mean_squared_error(y_test, y_pred)))
print("R2 Score:", metrics.r2_score(y_test, y_pred))
```

Model Evaluation:

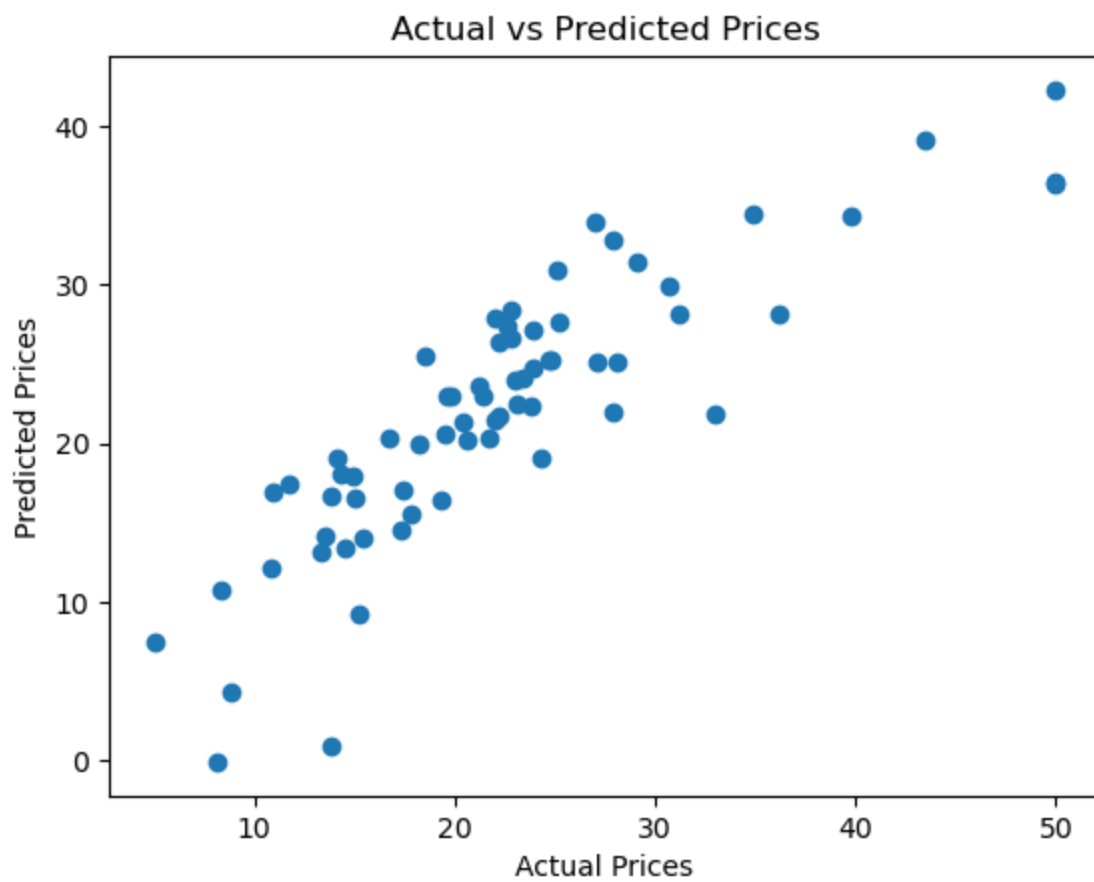
Mean Absolute Error: 3.6594858160279498

Mean Squared Error: 23.411701826598417

Root Mean Squared Error: 4.8385640252660105

R2 Score: 0.7398653051224349

```
In [17]: plt.scatter(y_test, y_pred)
plt.xlabel("Actual Prices")
plt.ylabel("Predicted Prices")
plt.title("Actual vs Predicted Prices")
plt.show()
```



In []:

```
In [1]: import pandas as pd
        from sklearn.model_selection import train_test_split
        from sklearn.linear_model import LogisticRegression
        from sklearn.metrics import confusion_matrix, accuracy_score, precision_score, recall_score
```

```
In [6]: df = pd.read_csv(r"D:\Pradnya\TE\DSBDA\Practicals\new\soclnetwork\Social_Network.csv")
        print(df.head())
```

	User ID	Gender	Age	EstimatedSalary	Purchased
0	15624510	Male	19	19000	0
1	15810944	Male	35	20000	0
2	15668575	Female	26	43000	0
3	15603246	Female	27	57000	0
4	15804002	Male	19	76000	0

```
In [11]: X = df[['Age', 'EstimatedSalary']]
        y = df['Purchased']
        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)

        y_pred = model.predict(X_test)
```

```
In [8]: cm = confusion_matrix(y_test, y_pred)
        print("\nConfusion Matrix:\n", cm)
```

Confusion Matrix:

```
[[50  2]
 [ 7 21]]
```

```
In [9]: TN, FP, FN, TP = cm.ravel()
        print("\nTN:", TN)
        print("FP:", FP)
        print("FN:", FN)
        print("TP:", TP)
```

TN: 50

FP: 2

FN: 7

TP: 21

```
In [10]: print("\nAccuracy:", accuracy_score(y_test, y_pred))
        print("Error Rate:", 1 - accuracy_score(y_test, y_pred))
        print("Precision:", precision_score(y_test, y_pred))
        print("Recall:", recall_score(y_test, y_pred))
```

Accuracy: 0.8875

Error Rate: 0.11250000000000004

Precision: 0.9130434782608695

Recall: 0.75

```
In [ ]:
```

```
In [2]: import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import confusion_matrix, accuracy_score, precision_score, recall_score
```

```
In [11]: df = pd.read_csv(r"D:\Pradnya\TE\DSBDA\Practicals\new\iris\iris.csv")

# Features and target
X = df.drop('Species', axis=1)
y = df['Species']

# Split dataset
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42
)
# Train Naïve Bayes model
model = GaussianNB()
model.fit(X_train, y_train)

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

```
In [12]: # Confusion Matrix
cm = confusion_matrix(y_test, y_pred)

# Function to compute TP, FP, TN, FN
def get_confusion_metrics(conf_matrix):
    FP = conf_matrix.sum(axis=0) - np.diag(conf_matrix)
    FN = conf_matrix.sum(axis=1) - np.diag(conf_matrix)
    TP = np.diag(conf_matrix)
    TN = conf_matrix.sum() - (FP + FN + TP)
    return TP, FP, TN, FN

tp, fp, tn, fn = get_confusion_metrics(cm)

# Performance metrics
accuracy = accuracy_score(y_test, y_pred)
error_rate = 1 - accuracy
precision = precision_score(y_test, y_pred, average='macro')
recall = recall_score(y_test, y_pred, average='macro')
```

```
In [13]: print("--- Confusion Matrix ---")
print(cm)

print("\nTrue Positives:", tp)
print("False Positives:", fp)
print("True Negatives:", tn)
print("False Negatives:", fn)

print("\n--- Performance Metrics ---")
```

```
print("Accuracy:", accuracy)
print("Error Rate:", error_rate)
print("Precision:", precision)
print("Recall:", recall)
```

--- Confusion Matrix ---

```
[[10  0  0]
 [ 0  9  0]
 [ 0  0 11]]
```

True Positives: [10 9 11]

False Positives: [0 0 0]

True Negatives: [20 21 19]

False Negatives: [0 0 0]

--- Performance Metrics ---

Accuracy: 1.0

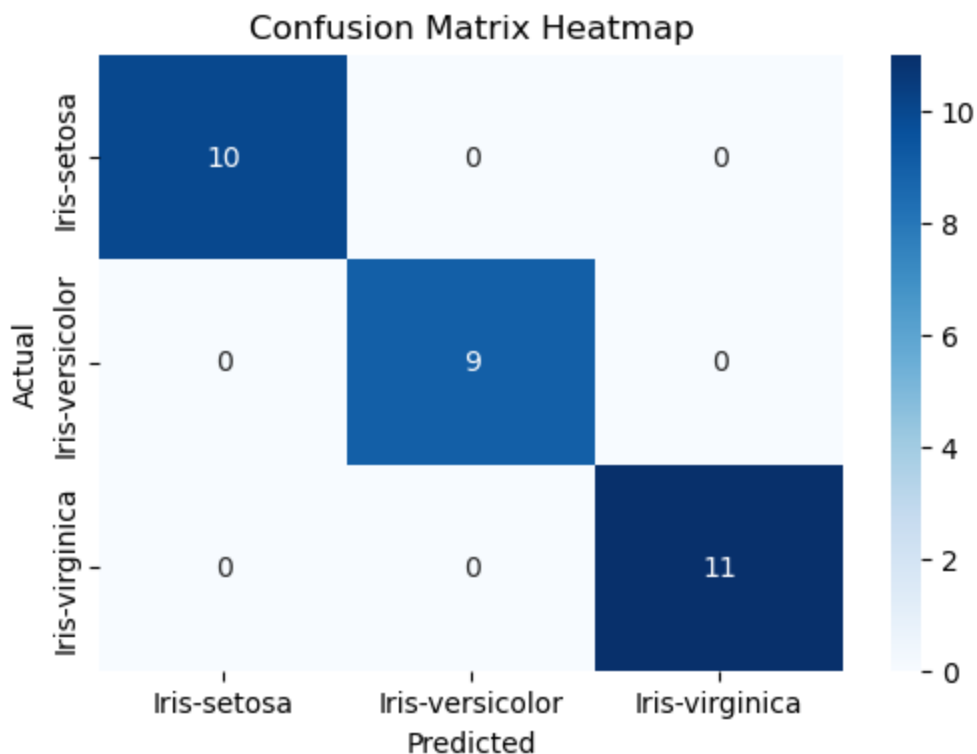
Error Rate: 0.0

Precision: 1.0

Recall: 1.0

```
In [14]: # Plot confusion matrix
plt.figure(figsize=(6,4))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
            xticklabels=model.classes_,
            yticklabels=model.classes_)

plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix Heatmap')
plt.show()
```



```
In [4]: import nltk
from nltk.tokenize import word_tokenize
from nltk.corpus import stopwords
from nltk import pos_tag
from nltk.stem import PorterStemmer, WordNetLemmatizer
from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer
```

```
In [13]: import nltk

nltk.download('punkt')
nltk.download('punkt_tab')
nltk.download('stopwords')
nltk.download('averaged_perceptron_tagger')
nltk.download('averaged_perceptron_tagger_eng')
nltk.download('wordnet')
```

```
[nltk_data] Downloading package punkt to
[nltk_data]   C:\Users\ADMIN\AppData\Roaming\nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package punkt_tab to
[nltk_data]   C:\Users\ADMIN\AppData\Roaming\nltk_data...
[nltk_data]   Package punkt_tab is already up-to-date!
[nltk_data] Downloading package stopwords to
[nltk_data]   C:\Users\ADMIN\AppData\Roaming\nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
[nltk_data] Downloading package averaged_perceptron_tagger to
[nltk_data]   C:\Users\ADMIN\AppData\Roaming\nltk_data...
[nltk_data]   Package averaged_perceptron_tagger is already up-to-
[nltk_data]   date!
[nltk_data] Downloading package averaged_perceptron_tagger_eng to
[nltk_data]   C:\Users\ADMIN\AppData\Roaming\nltk_data...
[nltk_data]   Unzipping taggers\averaged_perceptron_tagger_eng.zip.
[nltk_data] Downloading package wordnet to
[nltk_data]   C:\Users\ADMIN\AppData\Roaming\nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
```

```
Out[13]: True
```

```
In [14]: document = "Natural Language Processing is a fascinating field. It deals with under
```

```
In [15]: # 1. Tokenization
tokens = word_tokenize(document)
print("Word Tokens:", tokens)
```

```
Word Tokens: ['Natural', 'Language', 'Processing', 'is', 'a', 'fascinating', 'fiel
d', '.', 'It', 'deals', 'with', 'understanding', 'and', 'generating', 'human', 'lang
uage', 'using', 'computers', '.']
```

```
In [16]: # 2. POS Tagging
pos_tags = pos_tag(tokens)
print("\nPOS Tags:", pos_tags)
```

```
POS Tags: [('Natural', 'JJ'), ('Language', 'NNP'), ('Processing', 'NNP'), ('is', 'VBZ'), ('a', 'DT'), ('fascinating', 'JJ'), ('field', 'NN'), ('.', '.'), ('It', 'PRP'), ('deals', 'VBZ'), ('with', 'IN'), ('understanding', 'JJ'), ('and', 'CC'), ('generating', 'VBG'), ('human', 'JJ'), ('language', 'NN'), ('using', 'VBG'), ('computers', 'NNS'), ('.', '.')]

```

```
In [17]: # 3. Stop Words Removal
stop_words = set(stopwords.words('english'))
filtered_words = [word.lower() for word in tokens if word.lower() not in stop_words]
print("\nAfter Stop Words Removal:", filtered_words)

```

After Stop Words Removal: ['natural', 'language', 'processing', 'fascinating', 'field', 'deals', 'understanding', 'generating', 'human', 'language', 'using', 'computers']

```
In [18]: # 4. Stemming
stemmer = PorterStemmer()
stemmed_words = [stemmer.stem(word) for word in filtered_words]
print("\nStemmed Words:", stemmed_words)

```

Stemmed Words: ['natur', 'languag', 'process', 'fascin', 'field', 'deal', 'understand', 'gener', 'human', 'languag', 'use', 'comput']

```
In [19]: # 5. Lemmatization
lemmatizer = WordNetLemmatizer()
lemmatized_words = [lemmatizer.lemmatize(word) for word in filtered_words]
print("\nLemmatized Words:", lemmatized_words)

```

Lemmatized Words: ['natural', 'language', 'processing', 'fascinating', 'field', 'deal', 'understanding', 'generating', 'human', 'language', 'using', 'computer']

```
In [20]: # 6. Term Frequency (TF)
cleaned_text = " ".join(lemmatized_words)

count_vectorizer = CountVectorizer()
tf_matrix = count_vectorizer.fit_transform([cleaned_text])

print("\nTerm Frequency Matrix:\n", tf_matrix.toarray())
print("TF Feature Names:", count_vectorizer.get_feature_names_out())

```

Term Frequency Matrix:

```
[[1 1 1 1 1 2 1 1 1 1]]
```

TF Feature Names: ['computer' 'deal' 'fascinating' 'field' 'generating' 'human' 'language' 'natural' 'processing' 'understanding' 'using']

```
In [21]: # 7. TF-IDF
tfidf_vectorizer = TfidfVectorizer()
tfidf_matrix = tfidf_vectorizer.fit_transform([cleaned_text])

print("\nTF-IDF Matrix:\n", tfidf_matrix.toarray())
print("TF-IDF Feature Names:", tfidf_vectorizer.get_feature_names_out())

```

TF-IDF Matrix:

```
[[0.26726124 0.26726124 0.26726124 0.26726124 0.26726124 0.26726124  
 0.53452248 0.26726124 0.26726124 0.26726124 0.26726124]]
```

TF-IDF Feature Names: ['computer' 'deal' 'fascinating' 'field' 'generating' 'human'
'language'
'natural' 'processing' 'understanding' 'using']

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

```
In [2]: df = sns.load_dataset('titanic')
print(df.head())
print("\nDataset Info:\n")
print(df.info())
```

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

	who	adult_male	deck	embark_town	alive	alone
0	man	True	NaN	Southampton	no	False
1	woman	False	C	Cherbourg	yes	False
2	woman	False	NaN	Southampton	yes	True
3	woman	False	C	Southampton	yes	False
4	man	True	NaN	Southampton	no	True

Dataset Info:

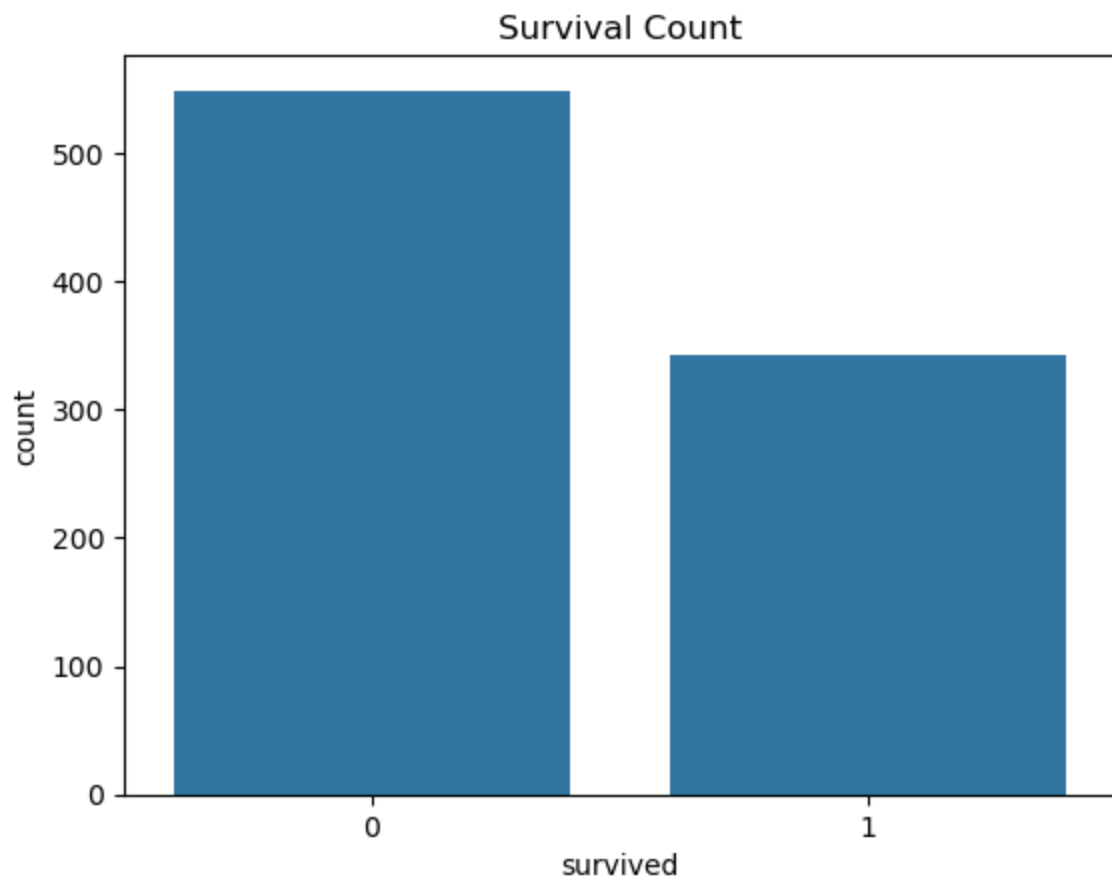
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 15 columns):
#   Column          Non-Null Count  Dtype
---  -
0   survived        891 non-null    int64
1   pclass          891 non-null    int64
2   sex             891 non-null    object
3   age            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
None
```

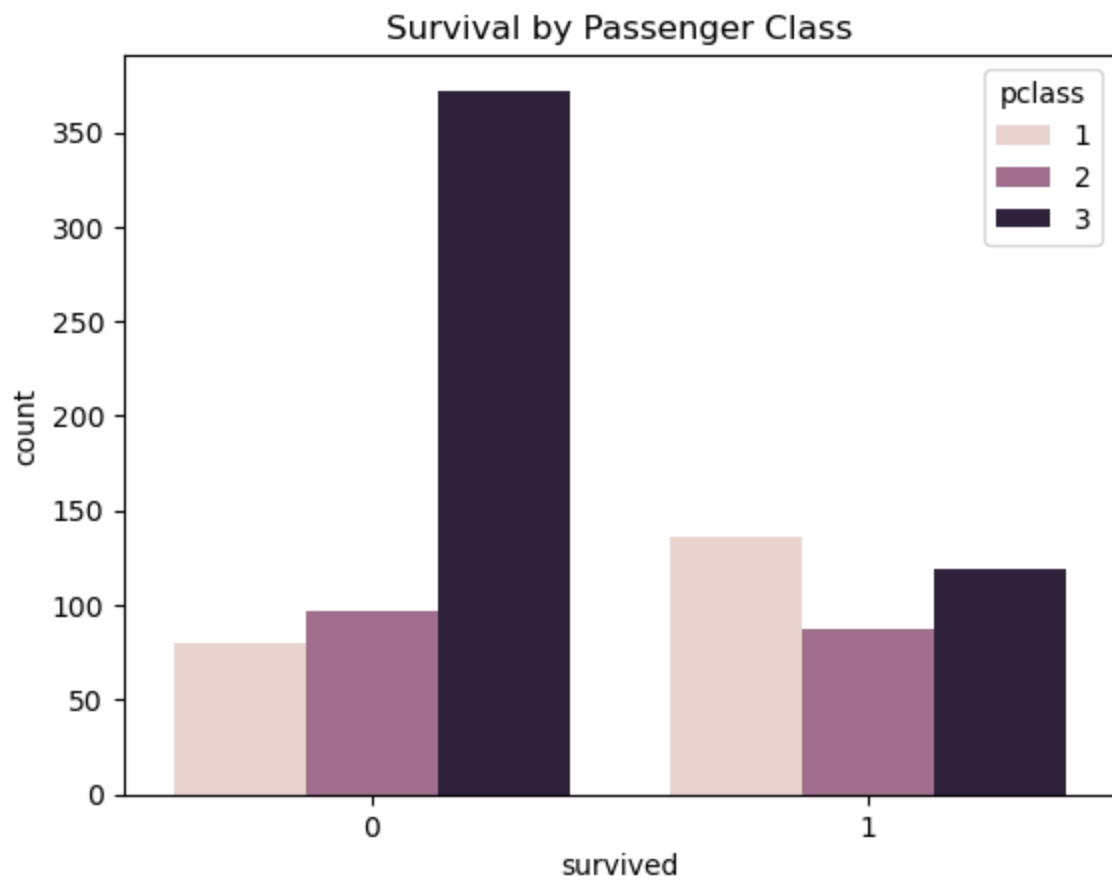
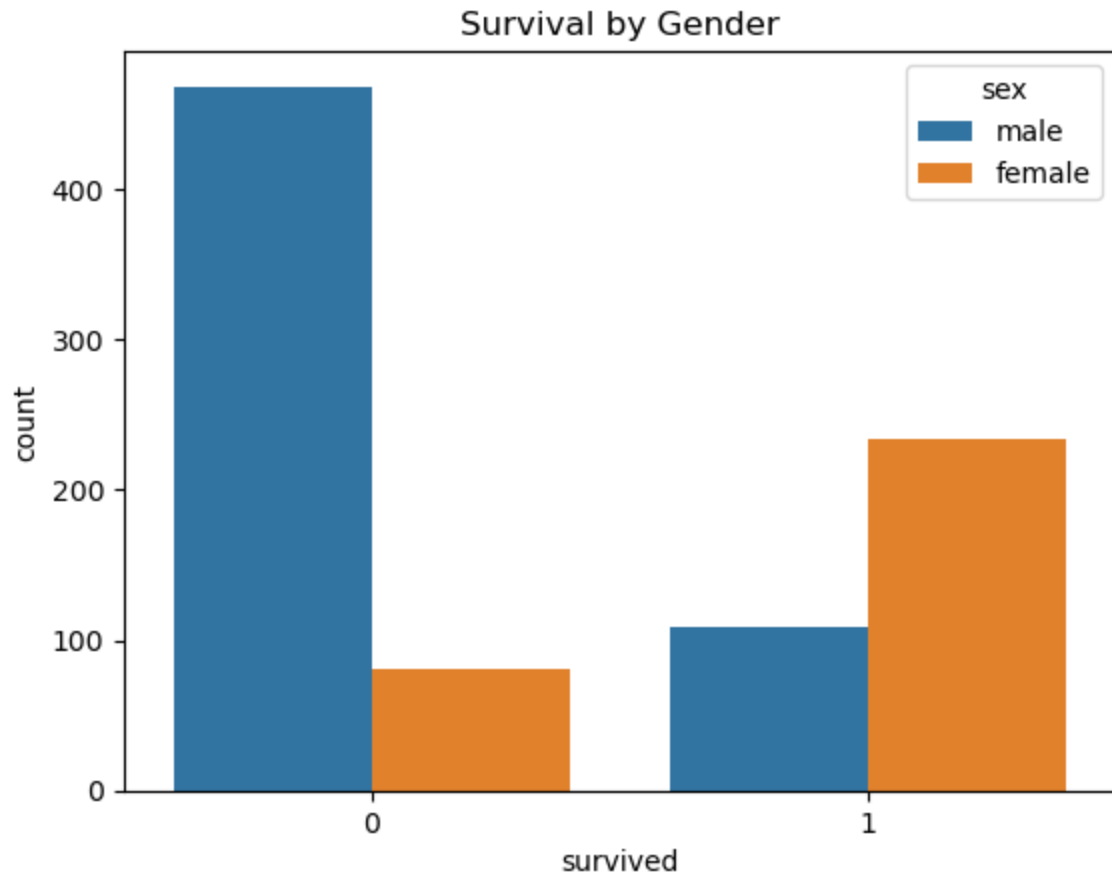
```
In [5]: # Survival count
sns.countplot(x='survived', data=df)
plt.title("Survival Count")
plt.show()
```

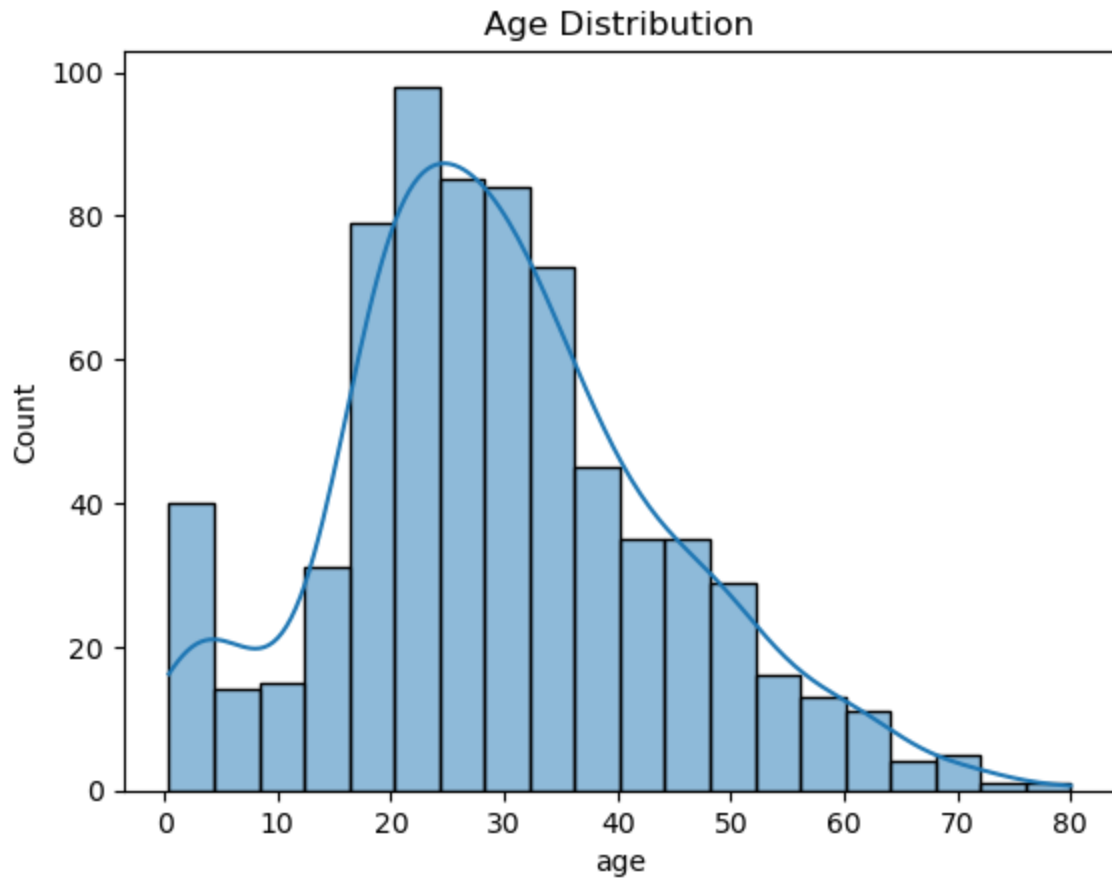
```
# Survival based on gender
sns.countplot(x='survived', hue='sex', data=df)
plt.title("Survival by Gender")
plt.show()

# Survival based on passenger class
sns.countplot(x='survived', hue='pclass', data=df)
plt.title("Survival by Passenger Class")
plt.show()

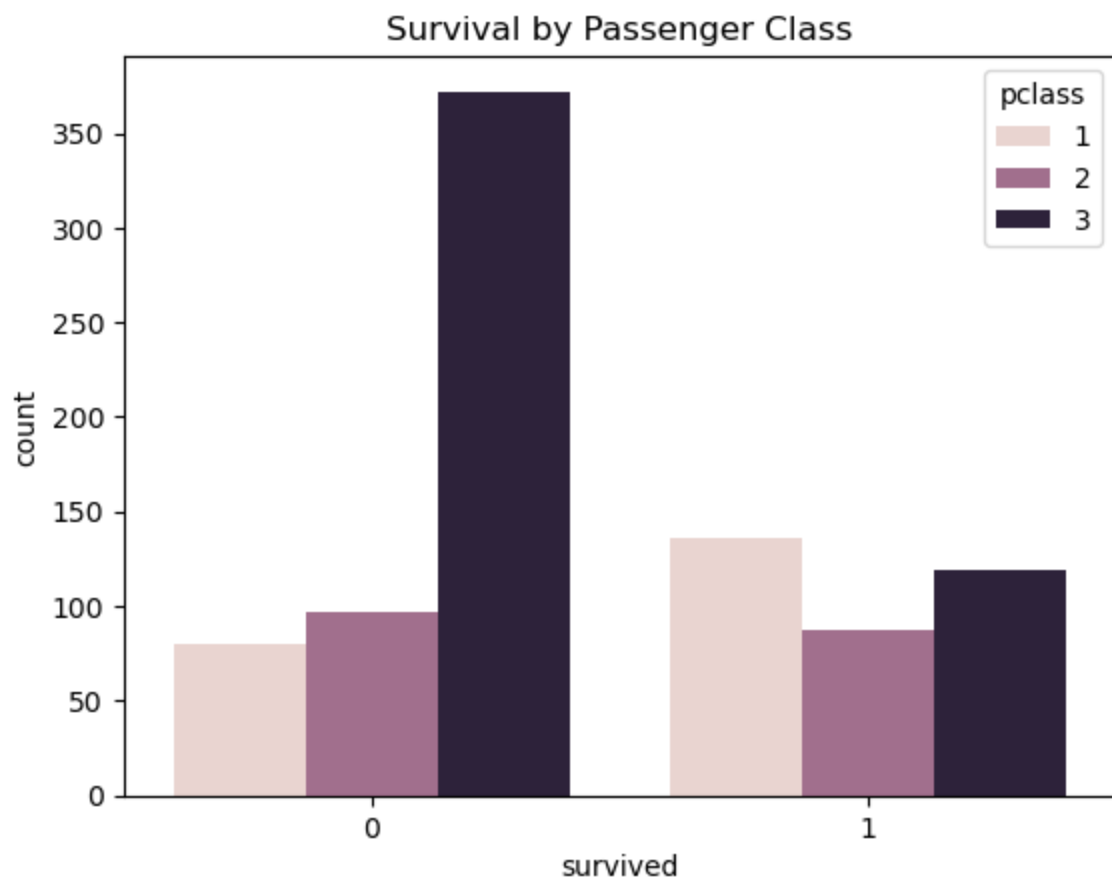
# Age distribution
sns.histplot(df['age'], kde=True)
plt.title("Age Distribution")
plt.show()
```

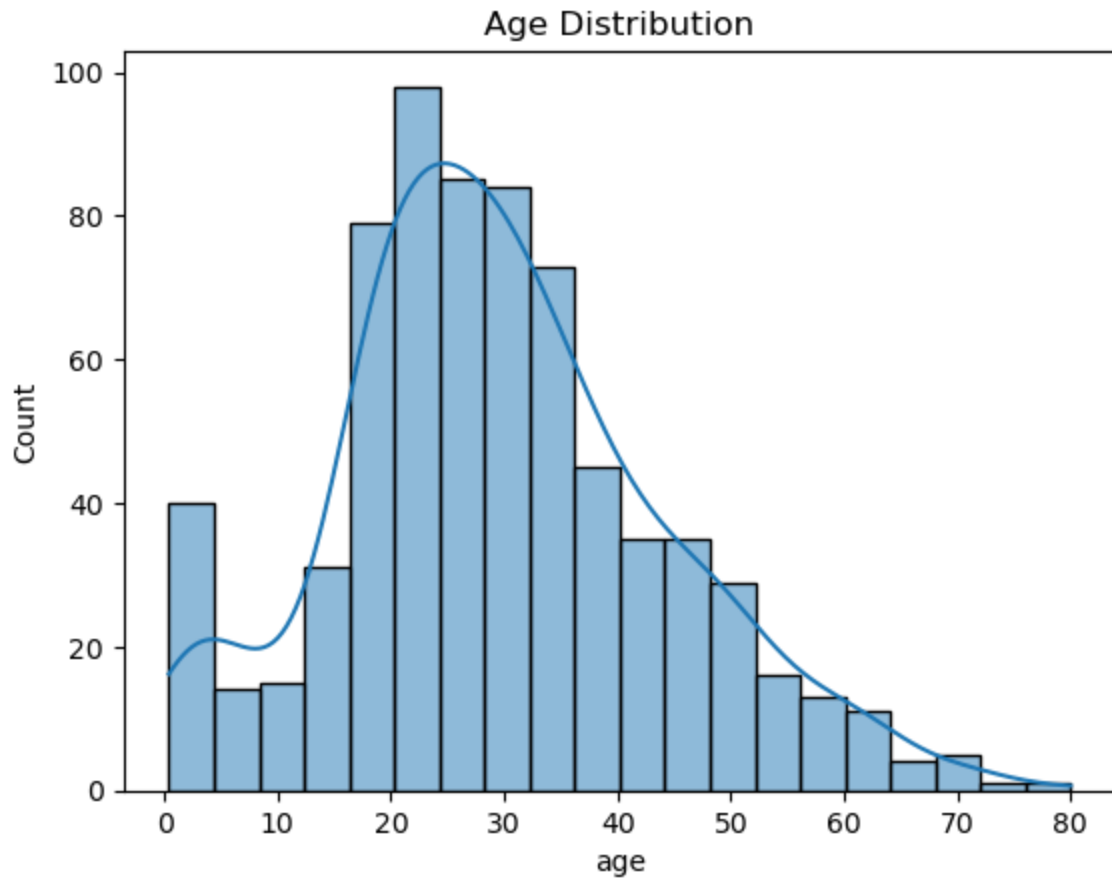




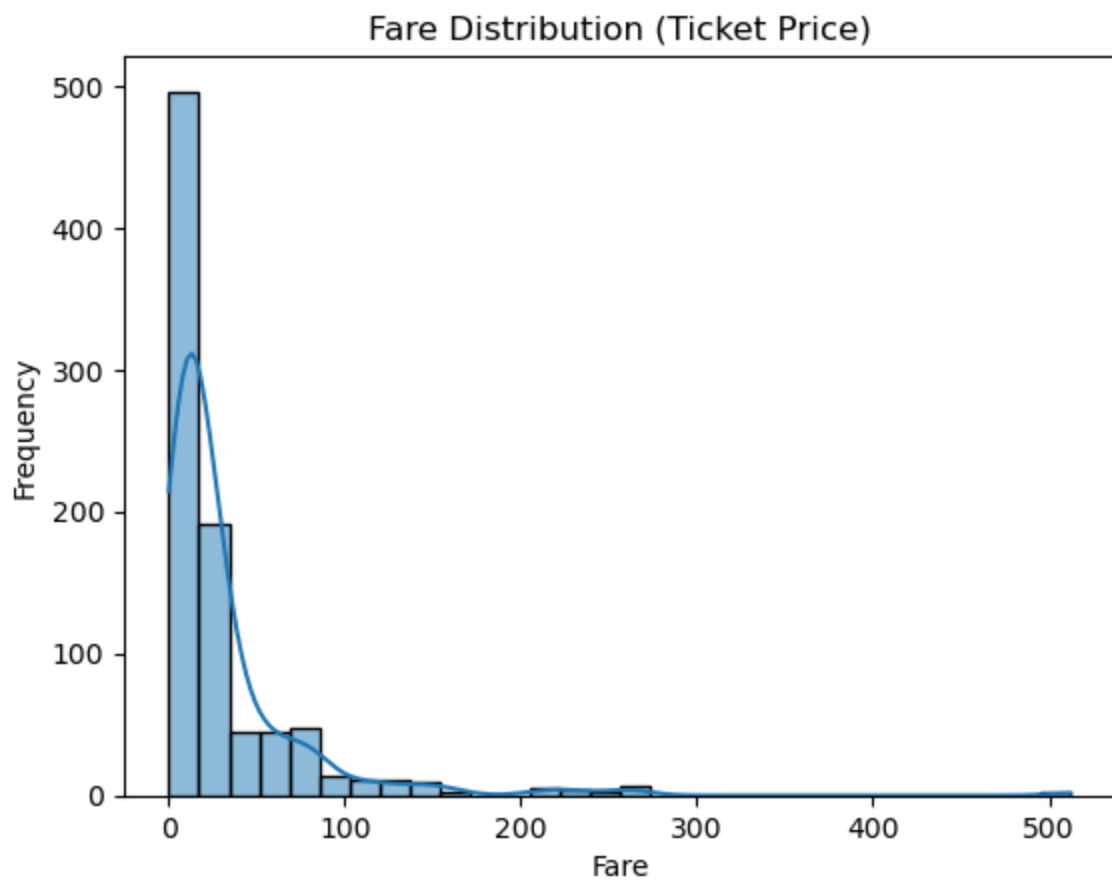


In [4]:





```
In [6]: # 3. Fare Distribution
sns.histplot(df['fare'], bins=30, kde=True)
plt.title("Fare Distribution (Ticket Price)")
plt.xlabel("Fare")
plt.ylabel("Frequency")
plt.show()
```



In []:

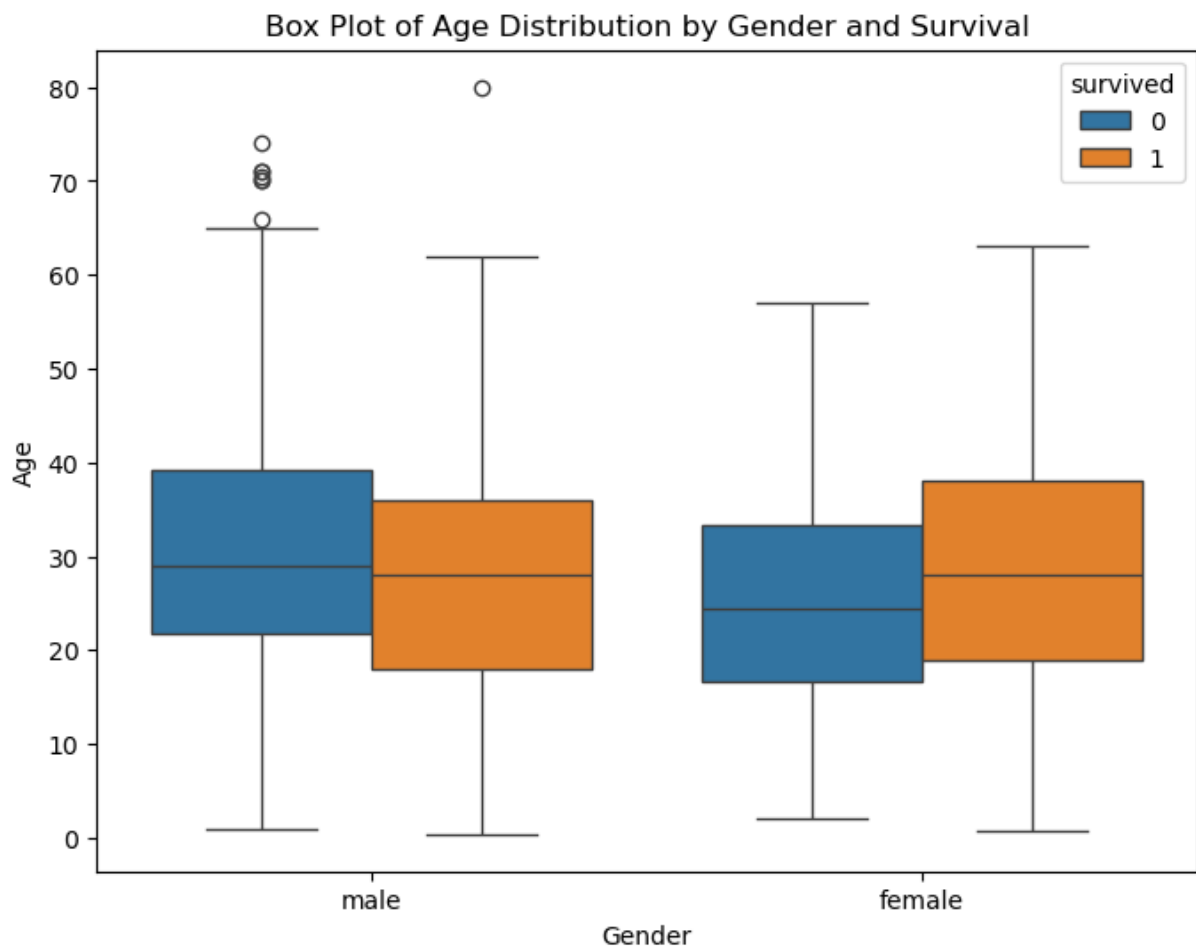
```
In [1]: import seaborn as sns
import matplotlib.pyplot as plt
```

```
In [3]: df = sns.load_dataset('titanic')
print(df.head())
```

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

	who	adult_male	deck	embark_town	alive	alone
0	man	True	NaN	Southampton	no	False
1	woman	False	C	Cherbourg	yes	False
2	woman	False	NaN	Southampton	yes	True
3	woman	False	C	Southampton	yes	False
4	man	True	NaN	Southampton	no	True

```
In [5]: plt.figure(figsize=(8,6))
sns.boxplot(x='sex', y='age', hue='survived', data=df)
plt.title("Box Plot of Age Distribution by Gender and Survival")
plt.xlabel("Gender")
plt.ylabel("Age")
plt.show()
```



In []:

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

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

```
In [3]: print("First 5 rows:\n", df.head())
print("\nFeature Types:\n")
print(df.dtypes)
```

First 5 rows:

	sepal_length	sepal_width	petal_length	petal_width	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

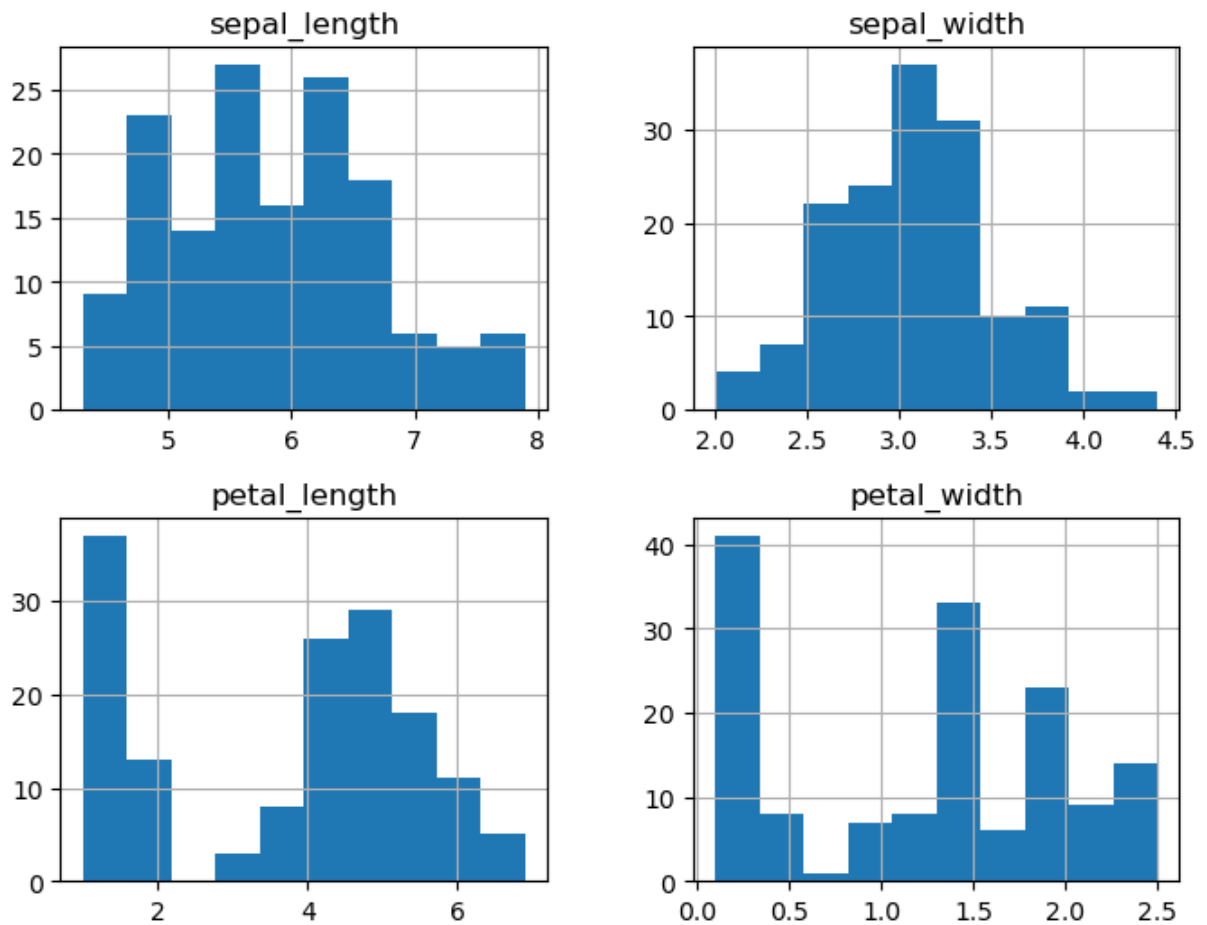
Feature Types:

```
sepal_length    float64
sepal_width     float64
petal_length    float64
petal_width     float64
species         object
dtype: object
```

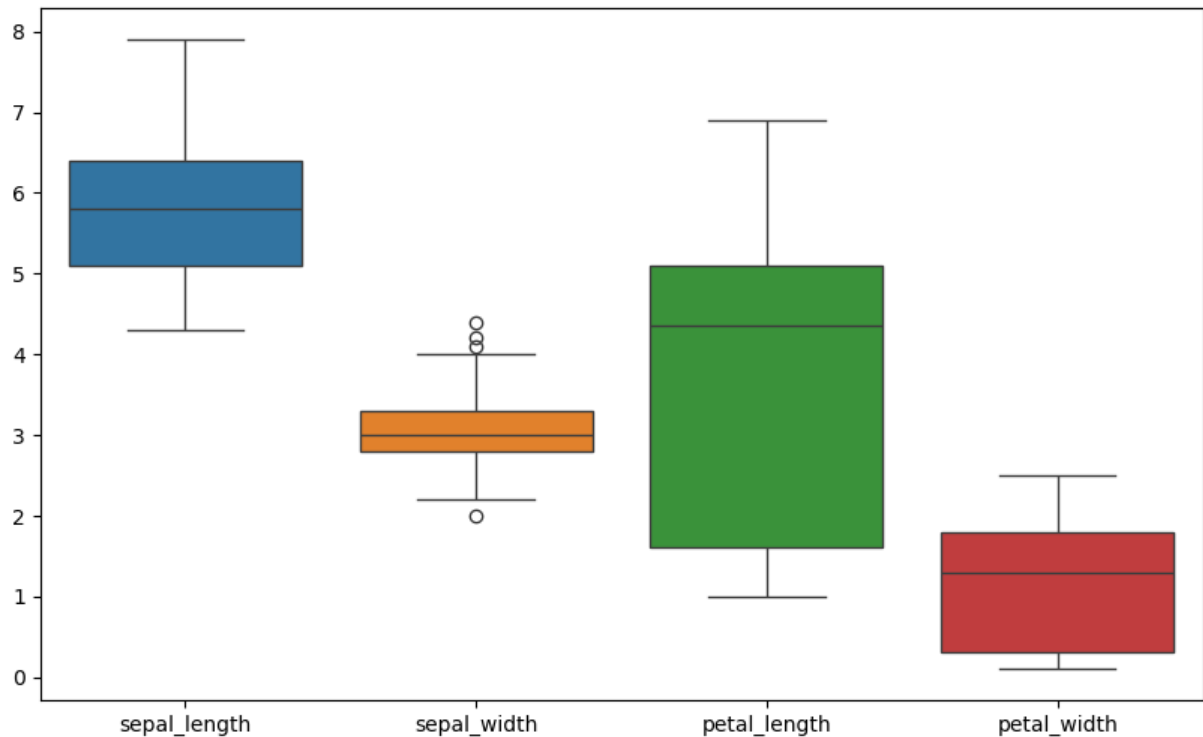
```
In [4]: # 2. Histograms for each feature
df.hist(figsize=(8,6))
plt.suptitle("Histograms of Iris Features")
plt.show()

# 3. Box plots for each feature
plt.figure(figsize=(10,6))
sns.boxplot(data=df)
plt.title("Box Plot of Iris Features")
plt.show()
```

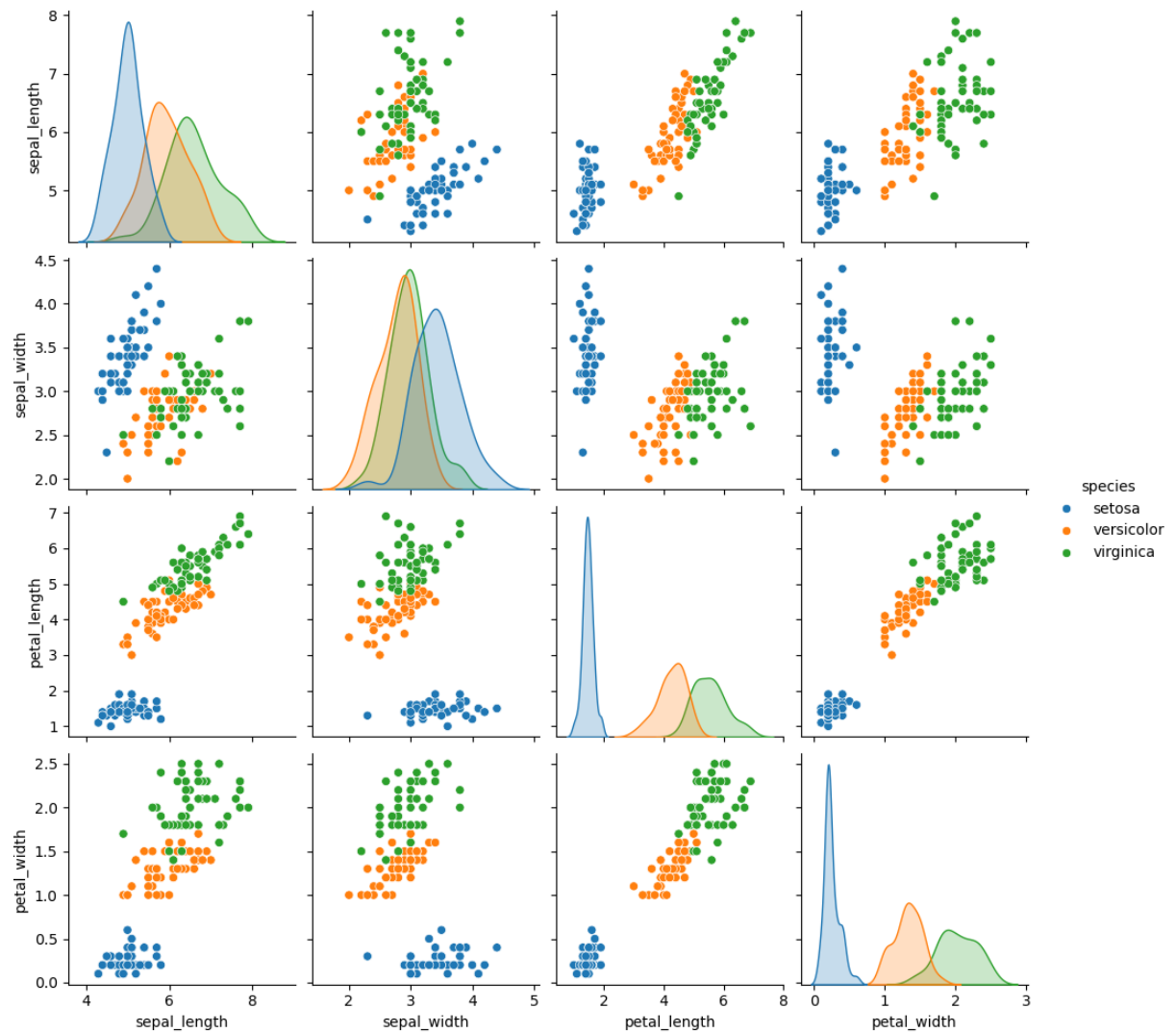
Histograms of Iris Features



Box Plot of Iris Features



```
In [5]: # 4. Compare distributions (pairplot)
sns.pairplot(df, hue='species')
plt.show()
```



INPUT:

```
import java.util.*;

public class Main {
    public static void main(String[] args) {

        String input = "Hello welcome to big data and big data analytics";

        // Map Phase
        String[] words = input.split("\\s+");
        HashMap<String, Integer> map = new HashMap<>();

        for (String word : words) {
            word = word.toLowerCase();
            map.put(word, map.getOrDefault(word, 0) + 1);
        }

        // Reduce Phase (printing result)
        System.out.println("Word Count:");
        for (Map.Entry<String, Integer> entry : map.entrySet()) {
            System.out.println(entry.getKey() + " -> " + entry.getValue());
        }
    }
}
```

OUTPUT:

Output

Word Count:

analytics -> 1

big -> 2

data -> 2

and -> 1

hello -> 1

to -> 1

welcome -> 1

=== Code Execution Successful ===

```
In [1]: # Sample log data (you can also read from file)
logs = [
    "2026-04-10 10:00:01 INFO User login successful",
    "2026-04-10 10:01:15 ERROR Database connection failed",
    "2026-04-10 10:02:20 WARN Disk space low",
    "2026-04-10 10:03:10 INFO File uploaded",
    "2026-04-10 10:04:05 ERROR Timeout occurred"
]

# -----
# MAP PHASE
# -----
mapped = []

for line in logs:
    parts = line.split()
    if len(parts) >= 3:
        level = parts[2]
        mapped.append((level, 1))

print("Mapped Output:")
print(mapped)

# -----
# SHUFFLE + SORT PHASE
# -----
shuffle = {}

for key, value in mapped:
    if key not in shuffle:
        shuffle[key] = []
    shuffle[key].append(value)

print("\nShuffled Output:")
print(shuffle)

# -----
# REDUCE PHASE
# -----
reduced = {}

for key in shuffle:
    reduced[key] = sum(shuffle[key])

print("\nFinal Output (Log Counts):")
for key, value in reduced.items():
    print(key, value)
```

Mapped Output:

```
[('INFO', 1), ('ERROR', 1), ('WARN', 1), ('INFO', 1), ('ERROR', 1)]
```

Shuffled Output:

```
{'INFO': [1, 1], 'ERROR': [1, 1], 'WARN': [1]}
```

Final Output (Log Counts):

```
INFO 2
```

```
ERROR 2
```

```
WARN 1
```

In []:

```
In [1]: with open("sample_weather.txt", "w") as f:
        f.write("""2026-04-01 Pune 32 22 18 12
2026-04-01 Mumbai 34 25 20 15
2026-04-01 Delhi 36 20 17 10
2026-04-02 Pune 30 21 16 11
2026-04-02 Mumbai 33 26 21 14
2026-04-02 Delhi 37 22 19 13
2026-04-03 Pune 29 20 15 10
2026-04-03 Mumbai 32 25 18 16
2026-04-03 Delhi 38 23 20 12
2026-04-04 Pune 31 21 17 13
2026-04-04 Mumbai 35 27 22 17
2026-04-04 Delhi 39 24 21 14
""")
```

```
In [2]: # -----  
# MAP PHASE  
# -----  
mapped = []  
  
with open("sample_weather.txt", "r") as file:  
    for line in file:  
        parts = line.split()  
  
        # Extract values  
        temp = float(parts[2])  
        dew = float(parts[3])  
        wind = float(parts[5])  
  
        # Emit key-value pairs  
        mapped.append(("temp", temp))  
        mapped.append(("dew", dew))  
        mapped.append(("wind", wind))  
  
print("Mapped Data:")  
print(mapped)  
  
# -----  
# SHUFFLE & SORT PHASE  
# -----  
shuffle = {}  
  
for key, value in mapped:  
    if key not in shuffle:  
        shuffle[key] = []  
    shuffle[key].append(value)  
  
print("\nShuffled Data:")  
print(shuffle)  
  
# -----  
# REDUCE PHASE  
# -----  
result = {}  
  
for key in shuffle:  
    values = shuffle[key]  
    avg = sum(values) / len(values)  
    result[key] = avg  
  
print("\nFinal Averages:")  
print("Average Temperature:", result["temp"])  
print("Average Dew Point:", result["dew"])  
print("Average Wind Speed:", result["wind"])
```

Mapped Data:

```
[('temp', 32.0), ('dew', 22.0), ('wind', 12.0), ('temp', 34.0), ('dew', 25.0), ('wind', 15.0), ('temp', 36.0), ('dew', 20.0), ('wind', 10.0), ('temp', 30.0), ('dew', 21.0), ('wind', 11.0), ('temp', 33.0), ('dew', 26.0), ('wind', 14.0), ('temp', 37.0), ('dew', 22.0), ('wind', 13.0), ('temp', 29.0), ('dew', 20.0), ('wind', 10.0), ('temp', 32.0), ('dew', 25.0), ('wind', 16.0), ('temp', 38.0), ('dew', 23.0), ('wind', 12.0), ('temp', 31.0), ('dew', 21.0), ('wind', 13.0), ('temp', 35.0), ('dew', 27.0), ('wind', 17.0), ('temp', 39.0), ('dew', 24.0), ('wind', 14.0)]
```

Shuffled Data:

```
{'temp': [32.0, 34.0, 36.0, 30.0, 33.0, 37.0, 29.0, 32.0, 38.0, 31.0, 35.0, 39.0], 'dew': [22.0, 25.0, 20.0, 21.0, 26.0, 22.0, 20.0, 25.0, 23.0, 21.0, 27.0, 24.0], 'wind': [12.0, 15.0, 10.0, 11.0, 14.0, 13.0, 10.0, 16.0, 12.0, 13.0, 17.0, 14.0]}
```

Final Averages:

Average Temperature: 33.833333333333336

Average Dew Point: 23.0

Average Wind Speed: 13.083333333333334

In []: