Introduction to Machine Learning Lab

February 16, 2026

1 Lab 01: Introduction to Machine Learning and Python Environment

Importing Libraries

[1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings('ignore')
```

[2]:
```
from sklearn.datasets import load_iris
```

Loading Iris Dataset

[3]:
```
iris = load_iris()
```

[4]:
```
df = pd.DataFrame(iris.data, columns=iris.feature_names)
df["species"] = iris.target
```

Displaying basic statistics and first few records

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

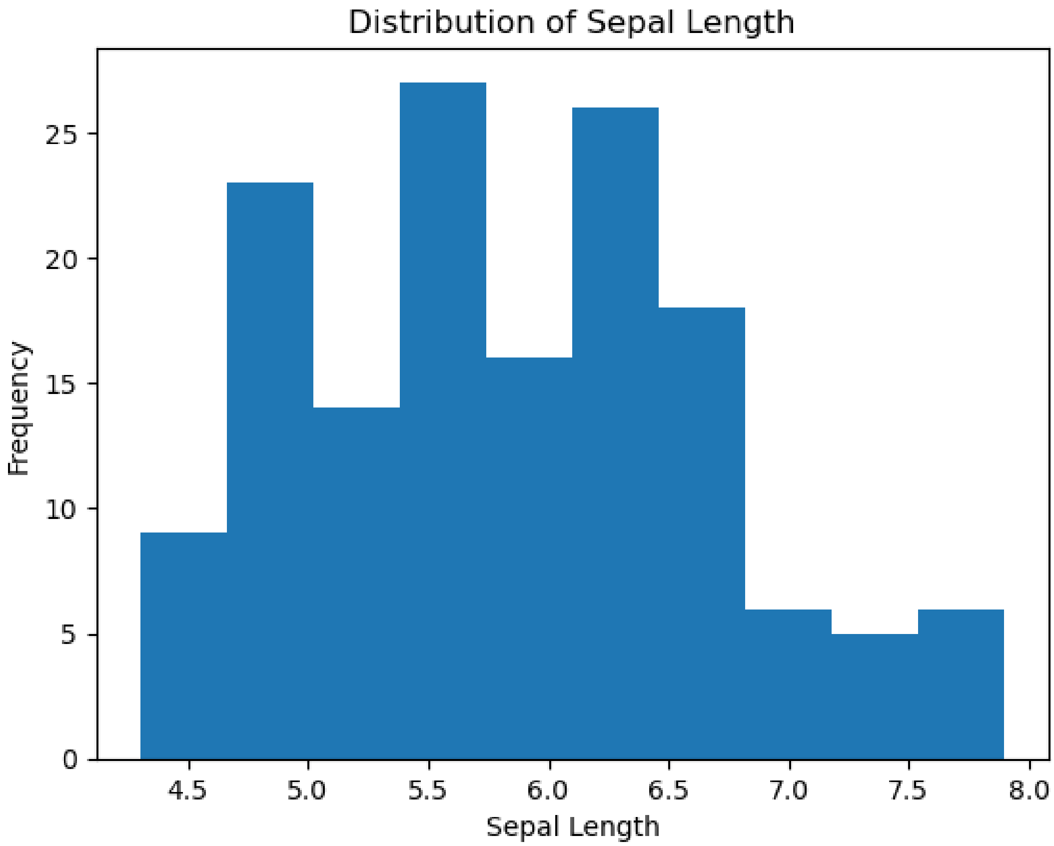
[5]:
```
   sepal length (cm)  sepal width (cm)  petal length (cm)  petal width (cm)  \
0                5.1               3.5                1.4               0.2
1                4.9               3.0                1.4               0.2
2                4.7               3.2                1.3               0.2
3                4.6               3.1                1.5               0.2
4                5.0               3.6                1.4               0.2

   species
0        0
1        0
2        0
3        0
4        0
```

[6]:
```
df.describe()
```

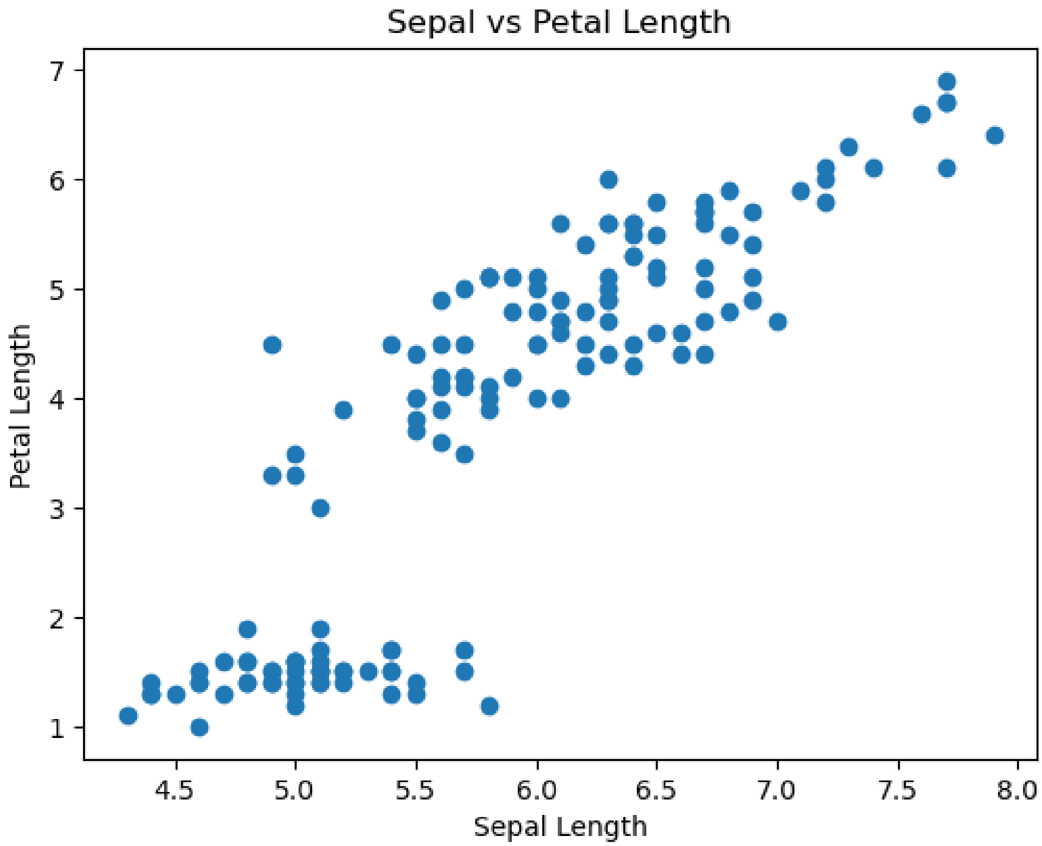
[6]:
```
       sepal length (cm)  sepal width (cm)  petal length (cm)  \
count         150.000000        150.000000         150.000000
mean            5.843333          3.057333           3.758000
std             0.828066          0.435866           1.765298
min             4.300000          2.000000           1.000000
25%             5.100000          2.800000           1.600000
50%             5.800000          3.000000           4.350000
75%             6.400000          3.300000           5.100000
max             7.900000          4.400000           6.900000

       petal width (cm)     species
count        150.000000  150.000000
mean           1.199333    1.000000
std            0.762238    0.819232
min            0.100000    0.000000
25%            0.300000    0.000000
50%            1.300000    1.000000
75%            1.800000    2.000000
max            2.500000    2.000000
```

Ploting histogram and scatter using matplotlib

[7]:
```
plt.hist(df["sepal length (cm)"])
plt.title("Distribution of Sepal Length")
plt.xlabel("Sepal Length")
plt.ylabel("Frequency")
plt.show()
```

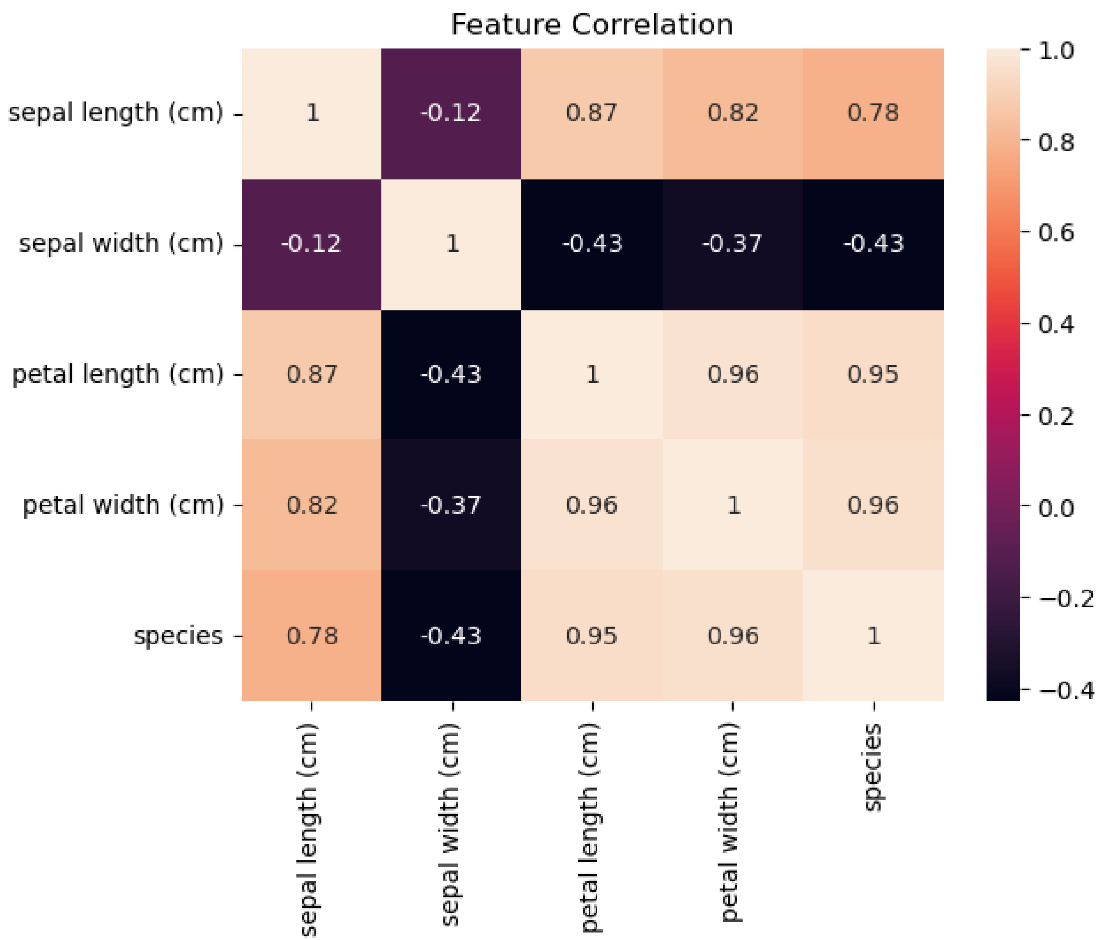


[8]:
```
plt.scatter(df["sepal length (cm)"], df["petal length (cm)"])
plt.xlabel("Sepal Length")
plt.ylabel("Petal Length")
plt.title("Sepal vs Petal Length")
plt.show()
```



Finding correlation and showing heatmap using seaborn

[9]:
```
corr = df.corr()
sns.heatmap(corr, annot=True)
plt.title("Feature Correlation")
plt.show()
```



Splitting data into test and train set

[10]:
```
from sklearn.model_selection import train_test_split

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

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.
 ↪2,random_state=42)

print("X_train shape:", X_train.shape)
print("X_test shape:", X_test.shape)
print("y_train shape:", y_train.shape)
print("y_test shape:", y_test.shape)
```

```
X_train shape: (120, 4)
X_test shape: (30, 4)
y_train shape: (120,)
```

```
y_test shape: (30,)
```

2 Lab – 02: Data Preprocessing

Loading Titanic Dataset

[11]:
```python
df = sns.load_dataset("titanic")
df.head()
```

[11]:
```
   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
```

Checking features null values

[12]:
```python
df.isnull().sum()
```

[12]:
```
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
```

Droping Null values

[13]:
```python
df_drop = df.dropna()
print("Original shape:", df.shape)
```

```python
print("After drop:", df_drop.shape)
```

```
Original shape: (891, 15)
After drop: (182, 15)
```

Filling Null values with median

[14]:
```python
df["age"].fillna(df["age"].median(), inplace=True)
```

Identifying categorical columns.

[15]:
```python
df.select_dtypes(include=["object", "category"]).head()
```

[15]:
```
      sex embarked  class    who deck  embark_town alive
0    male        S  Third    man  NaN  Southampton    no
1  female        C  First  woman    C    Cherbourg   yes
2  female        S  Third  woman  NaN  Southampton   yes
3  female        S  First  woman    C  Southampton   yes
4    male        S  Third    man  NaN  Southampton    no
```

Applying Label Encoder

[16]:
```python
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
df["class_encoded"] = le.fit_transform(df["class"])
df.head()
```

[16]:
```
   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  class_encoded
0    man        True  NaN  Southampton    no  False              2
1  woman       False    C    Cherbourg   yes  False              0
2  woman       False  NaN  Southampton   yes   True              2
3  woman       False    C  Southampton   yes  False              0
4    man        True  NaN  Southampton    no   True              2
```

Applying One Hot Encoding technique

[17]:
```python
df_encoded = pd.get_dummies(df, columns=["sex", "embarked"], drop_first=True)
df_encoded.head()
```

[17]:
```
   survived  pclass   age  sibsp  parch     fare  class    who  adult_male  \
0         0       3  22.0      1      0   7.2500  Third    man        True
1         1       1  38.0      1      0  71.2833  First  woman       False
2         1       3  26.0      0      0   7.9250  Third  woman       False
```

```
3         1        1  35.0       1       0  53.1000  First  woman        False
4         0        3  35.0       0       0   8.0500  Third    man         True

  deck  embark_town alive  alone  class_encoded  sex_male  embarked_Q  \
0  NaN  Southampton    no  False              2      True       False
1    C    Cherbourg   yes  False              0     False       False
2  NaN  Southampton   yes   True              2     False       False
3    C  Southampton   yes  False              0     False       False
4  NaN  Southampton    no   True              2      True       False

   embarked_S
0        True
1       False
2        True
3        True
4        True
```

3 Lab – 03: Exploratory Data Analysis (EDA)

Inspecting data using `head()`, `tail()`, `info()`, `describe()` methods

[18]:
```
df.head()
```

[18]:
```
   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  class_encoded
0    man        True  NaN  Southampton    no  False              2
1  woman       False    C    Cherbourg   yes  False              0
2  woman       False  NaN  Southampton   yes   True              2
3  woman       False    C  Southampton   yes  False              0
4    man        True  NaN  Southampton    no   True              2
```

[19]:
```
df.tail()
```

[19]:
```
     survived  pclass     sex   age  sibsp  parch   fare embarked   class  \
886         0       2    male  27.0      0      0  13.00        S  Second
887         1       1  female  19.0      0      0  30.00        S   First
888         0       3  female  28.0      1      2  23.45        S   Third
889         1       1    male  26.0      0      0  30.00        C   First
890         0       3    male  32.0      0      0   7.75        Q   Third

       who  adult_male deck  embark_town alive  alone  class_encoded
```

```
886    man        True  NaN  Southampton   no   True               1
887  woman       False    B  Southampton  yes   True               0
888  woman       False  NaN  Southampton   no  False               2
889    man        True    C    Cherbourg  yes   True               0
890    man        True  NaN   Queenstown   no   True               2
```

[20]:
```python
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 16 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            891 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
 15  class_encoded  891 non-null    int64
dtypes: bool(2), category(2), float64(2), int64(5), object(5)
memory usage: 87.6+ KB
```

[21]:
```python
df.describe()
```

[21]:
```
         survived      pclass         age       sibsp       parch        fare  \
count  891.000000  891.000000  891.000000  891.000000  891.000000  891.000000
mean     0.383838    2.308642   29.361582    0.523008    0.381594   32.204208
std      0.486592    0.836071   13.019697    1.102743    0.806057   49.693429
min      0.000000    1.000000    0.420000    0.000000    0.000000    0.000000
25%      0.000000    2.000000   22.000000    0.000000    0.000000    7.910400
50%      0.000000    3.000000   28.000000    0.000000    0.000000   14.454200
75%      1.000000    3.000000   35.000000    1.000000    0.000000   31.000000
max      1.000000    3.000000   80.000000    8.000000    6.000000  512.329200

       class_encoded
count     891.000000
mean        1.308642
std         0.836071
```

```
min          0.000000
25%          1.000000
50%          2.000000
75%          2.000000
max          2.000000
```

Calculate mean, median, mode, standard deviation, variance, and correlation for numerical features.

[22]:
```
df.mean(numeric_only=True)
```

[22]:
```
survived          0.383838
pclass            2.308642
age              29.361582
sibsp             0.523008
parch             0.381594
fare             32.204208
adult_male        0.602694
alone             0.602694
class_encoded     1.308642
dtype: float64
```

[23]:
```
df.median(numeric_only=True)
```

[23]:
```
survived          0.0000
pclass            3.0000
age              28.0000
sibsp             0.0000
parch             0.0000
fare             14.4542
adult_male        1.0000
alone             1.0000
class_encoded     2.0000
dtype: float64
```

[24]:
```
df.mode(numeric_only=True)
```

[24]:
```
   survived  pclass   age  sibsp  parch  fare  adult_male  alone  \
0         0       3  28.0      0      0  8.05        True   True

   class_encoded
0              2
```

[25]:
```
df.std(numeric_only=True)
```

[25]:
```
survived          0.486592
pclass            0.836071
age              13.019697
```

```
sibsp              1.102743
parch              0.806057
fare              49.693429
adult_male         0.489615
alone              0.489615
class_encoded      0.836071
dtype: float64
```

[26]:
```python
df.var(numeric_only=True)
```

[26]:
```
survived              0.236772
pclass                0.699015
age                 169.512498
sibsp                 1.216043
parch                 0.649728
fare               2469.436846
adult_male            0.239723
alone                 0.239723
class_encoded         0.699015
dtype: float64
```

[27]:
```python
df.corr(numeric_only=True)
```

[27]:
```
               survived    pclass       age     sibsp     parch      fare  \
survived       1.000000 -0.338481 -0.064910 -0.035322  0.081629  0.257307
pclass        -0.338481  1.000000 -0.339898  0.083081  0.018443 -0.549500
age           -0.064910 -0.339898  1.000000 -0.233296 -0.172482  0.096688
sibsp         -0.035322  0.083081 -0.233296  1.000000  0.414838  0.159651
parch          0.081629  0.018443 -0.172482  0.414838  1.000000  0.216225
fare           0.257307 -0.549500  0.096688  0.159651  0.216225  1.000000
adult_male    -0.557080  0.094035  0.247704 -0.253586 -0.349943 -0.182024
alone         -0.203367  0.135207  0.171647 -0.584471 -0.583398 -0.271832
class_encoded -0.338481  1.000000 -0.339898  0.083081  0.018443 -0.549500

               adult_male     alone  class_encoded
survived        -0.557080 -0.203367      -0.338481
pclass           0.094035  0.135207       1.000000
age              0.247704  0.171647      -0.339898
sibsp           -0.253586 -0.584471       0.083081
parch           -0.349943 -0.583398       0.018443
fare            -0.182024 -0.271832      -0.549500
adult_male       1.000000  0.404744       0.094035
alone            0.404744  1.000000       0.135207
class_encoded    0.094035  0.135207       1.000000
```

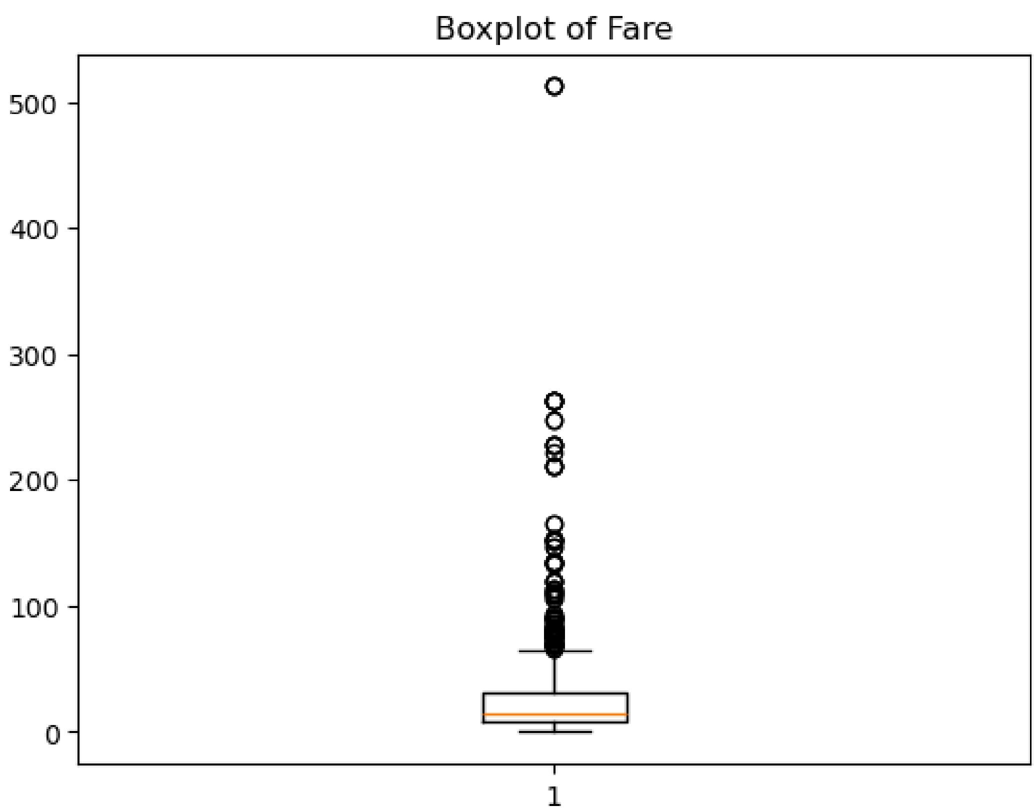
Identifying missing values

[28]:
```python
df.isnull().sum()
```

[28]:
```
survived           0
pclass             0
sex                0
age                0
sibsp              0
parch              0
fare               0
embarked           2
class              0
who                0
adult_male         0
deck             688
embark_town        2
alive              0
alone              0
class_encoded      0
dtype: int64
```

Detecting Outliers using `Boxplot`

[29]:
```
plt.boxplot(df["fare"])
plt.title("Boxplot of Fare")
plt.show()
```



Detecting Outliers using IQR

[30]:
```python
Q1 = df["fare"].quantile(0.25)
Q3 = df["fare"].quantile(0.75)
IQR = Q3 - Q1

lower = Q1 - 1.5 * IQR
upper = Q3 + 1.5 * IQR

outliers = df[(df["fare"] < lower) | (df["fare"] > upper)]
print("Number of outliers:", outliers.shape[0])
```

```
Number of outliers: 116
```

Handling Outliers (Capping Method)

[31]:
```python
df["fare"] = np.where(df["fare"] > upper, upper, df["fare"])
df["fare"] = np.where(df["fare"] < lower, lower, df["fare"])
```

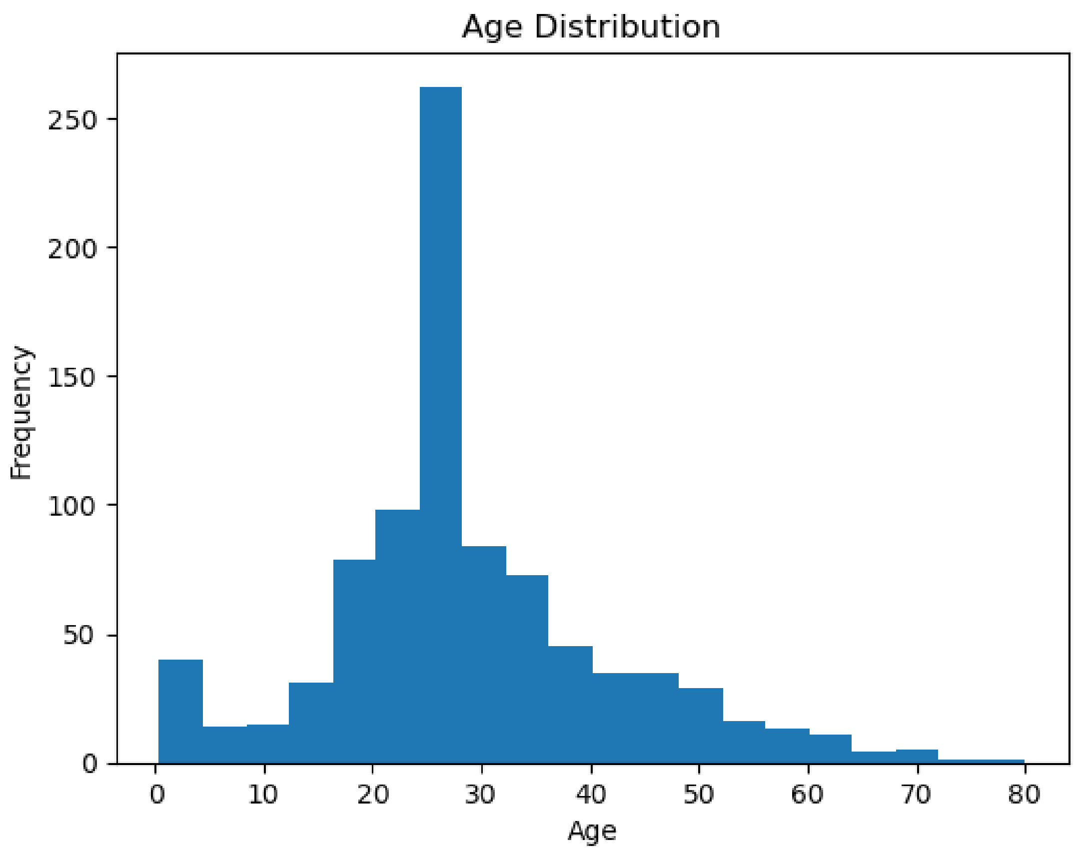
[32]:
```python
df.head()
```

[32]:
```
   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  65.6344        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  class_encoded
0    man        True  NaN  Southampton    no  False              2
1  woman       False    C    Cherbourg   yes  False              0
2  woman       False  NaN  Southampton   yes   True              2
3  woman       False    C  Southampton   yes  False              0
4    man        True  NaN  Southampton    no   True              2
```

Ploting histograms, scatter plots, and bar charts using matplotlib or seaborn.

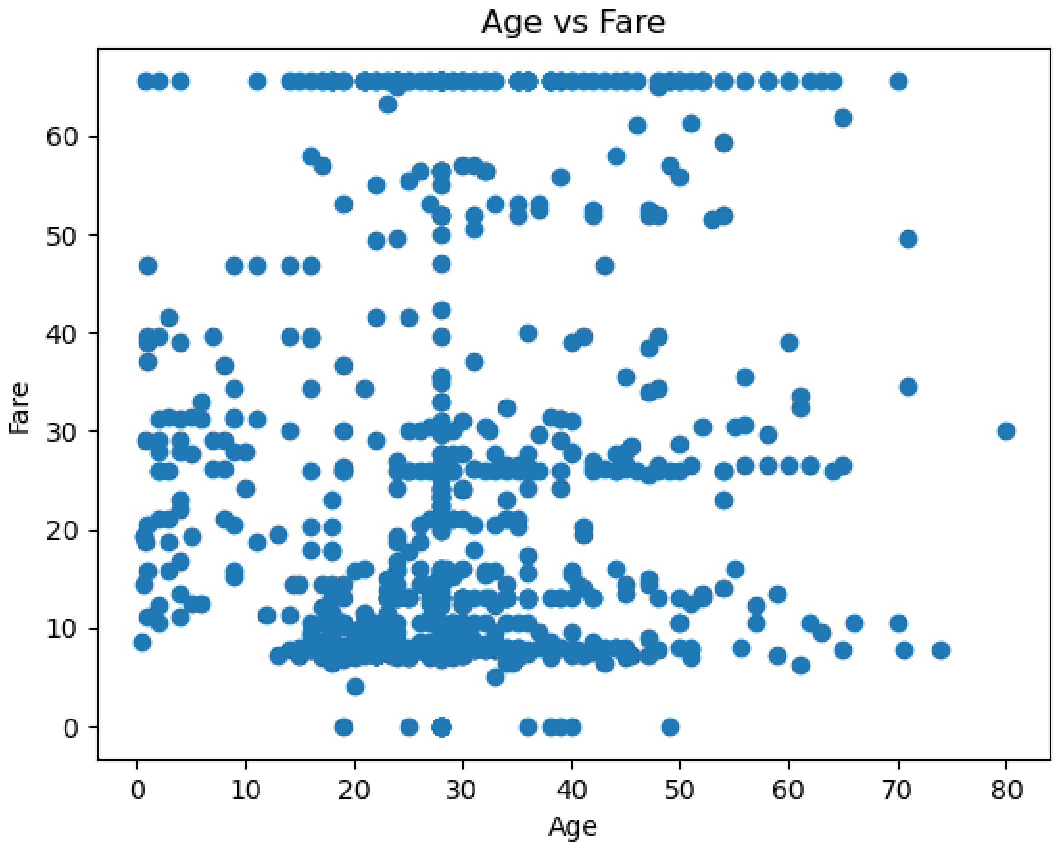
Histogram

[33]:
```python
plt.hist(df["age"], bins=20)
plt.title("Age Distribution")
plt.xlabel("Age")
plt.ylabel("Frequency")
plt.show()
```



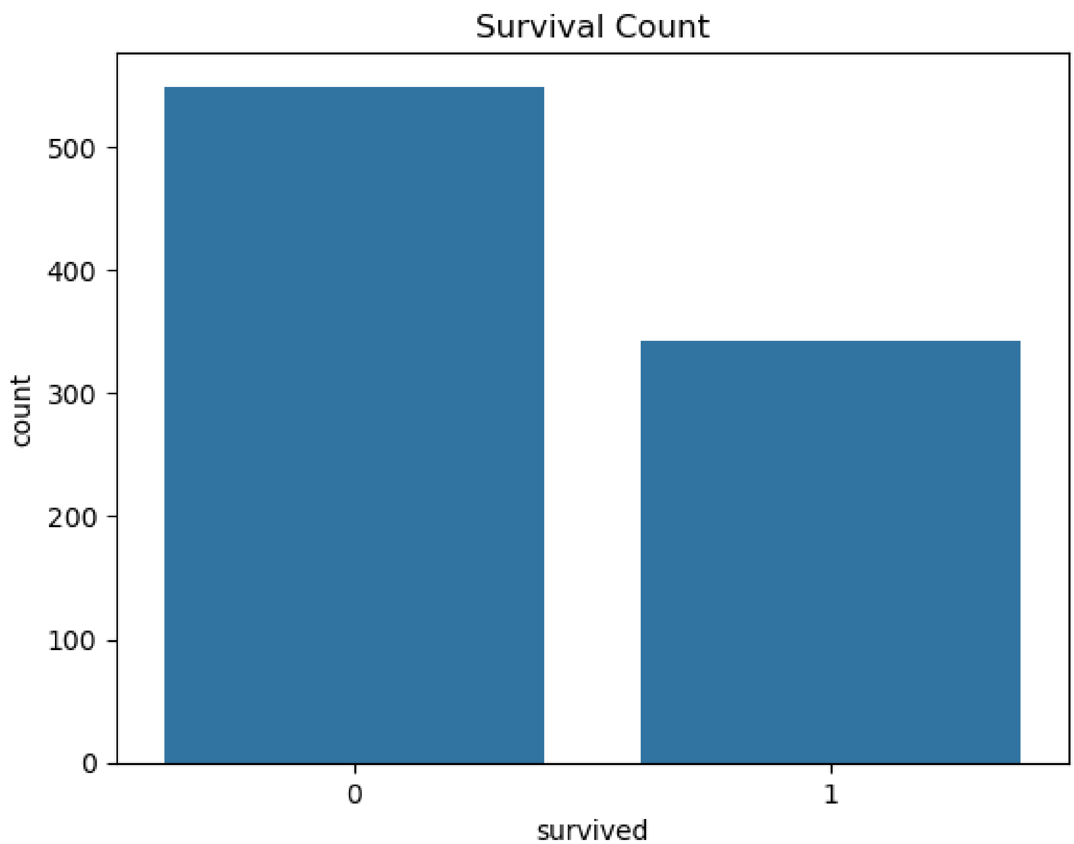
Scatter Plot

[34]:
```
plt.scatter(df["age"], df["fare"])
plt.xlabel("Age")
plt.ylabel("Fare")
plt.title("Age vs Fare")
plt.show()
```



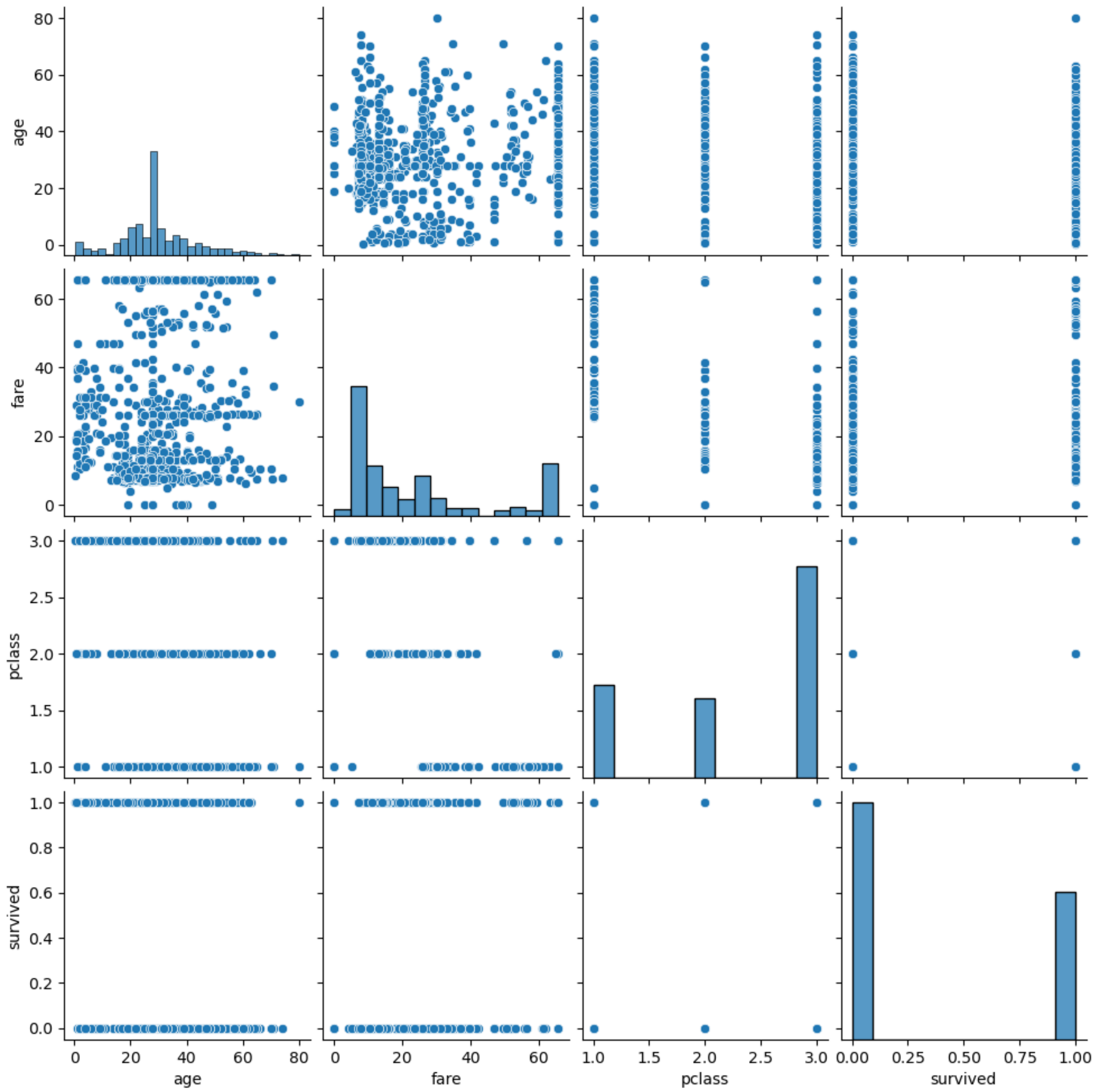
Bar Plot

[35]:
```
sns.countplot(x="survived", data=df)
plt.title("Survival Count")
plt.show()
```



Pairplot

[36]:
```
sns.pairplot(df[["age", "fare", "pclass", "survived"]])
plt.show()
```



Categorical Variable Analysis

[37]:
```
df["sex"].value_counts()
```

[37]:
```
sex
male      577
female    314
Name: count, dtype: int64
```

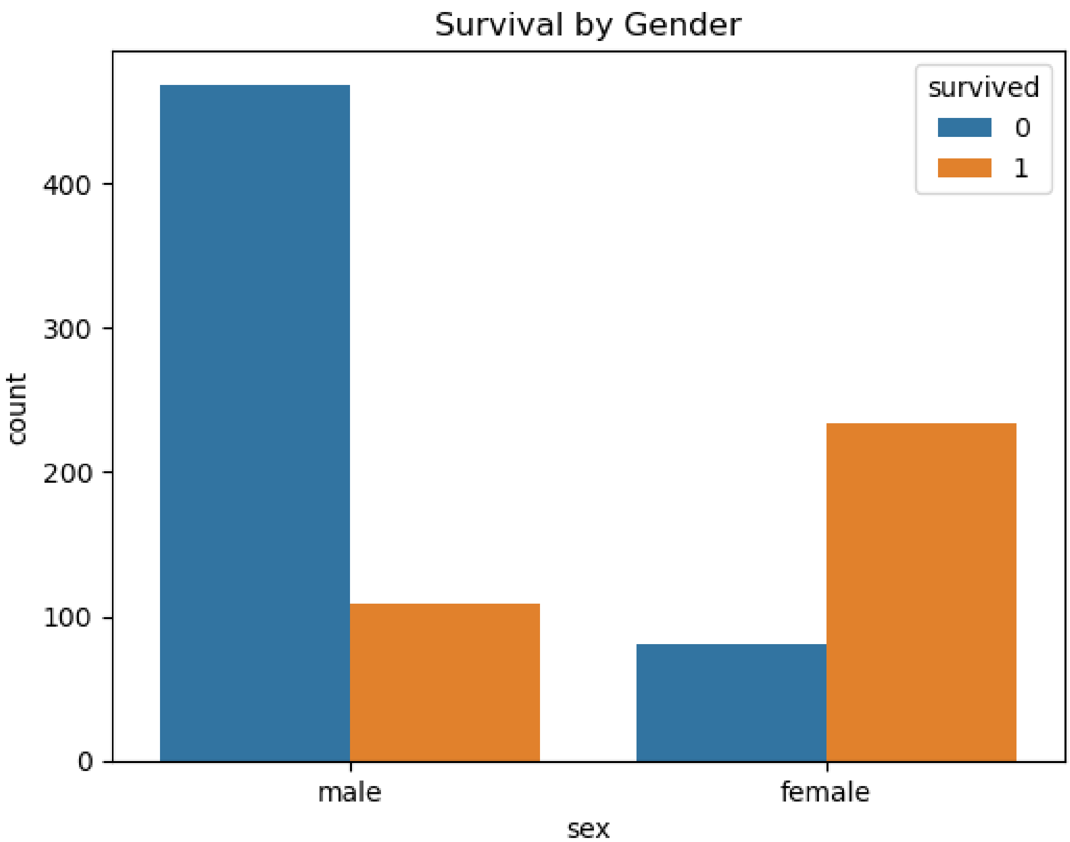
[38]:
```
df["class"].value_counts()
```

[38]:
```
class
Third     491
```

```
First      216
Second     184
Name: count, dtype: int64
```

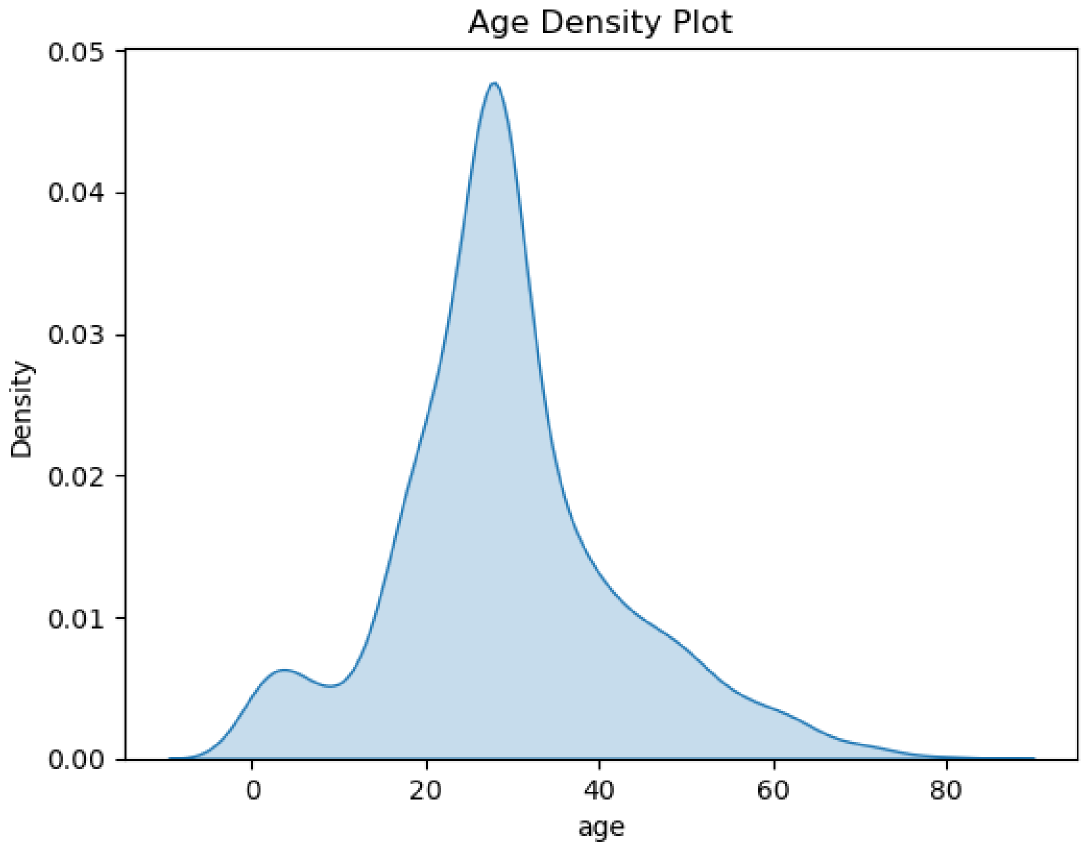
Bar Plot for categorical variables

[39]:
```
sns.countplot(x="sex", hue="survived", data=df)
plt.title("Survival by Gender")
plt.show()
```



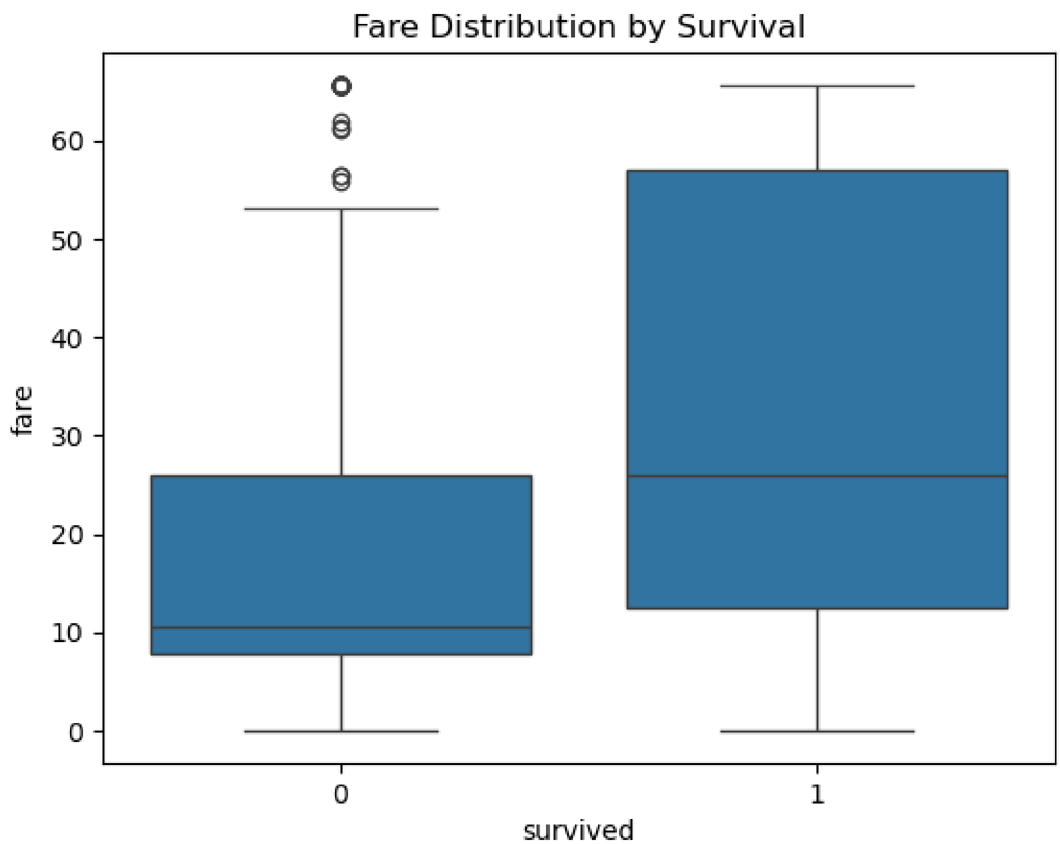
Numerical Feature Distribution (Density Plot)

[40]:
```
sns.kdeplot(df["age"], fill=True)
plt.title("Age Density Plot")
plt.show()
```



Boxplot for Numerical Feature

[41]:
```
sns.boxplot(x="survived", y="fare", data=df)
plt.title("Fare Distribution by Survival")
plt.show()
```



4 Lab – 04: Linear Regression

Loading Advertising Dataset from local device

[42]:
```
df = pd.read_csv("archive/Advertising.csv")
df.head()
```

[42]:

	Unnamed: 0	TV	Radio	Newspaper	Sales
0	1	230.1	37.8	69.2	22.1
1	2	44.5	39.3	45.1	10.4
2	3	17.2	45.9	69.3	9.3
3	4	151.5	41.3	58.5	18.5
4	5	180.8	10.8	58.4	12.9

[43]:
```
df.drop(columns='Unnamed: 0', inplace=True)
```

[44]:
```
df.head()
```

[44]:

	TV	Radio	Newspaper	Sales
0	230.1	37.8	69.2	22.1
1	44.5	39.3	45.1	10.4
2	17.2	45.9	69.3	9.3
3	151.5	41.3	58.5	18.5
4	180.8	10.8	58.4	12.9

For simple linear regression, taking single feature `TV` and target feature `Sales`

[45]:
```
X = df[['TV']]
y = df['Sales']
```

Splitting dataset in train and test

[46]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
↪random_state=42)
```

Training Model

[47]:
```
from sklearn.linear_model import LinearRegression
model_simple = LinearRegression()
model_simple.fit(X_train, y_train)
```

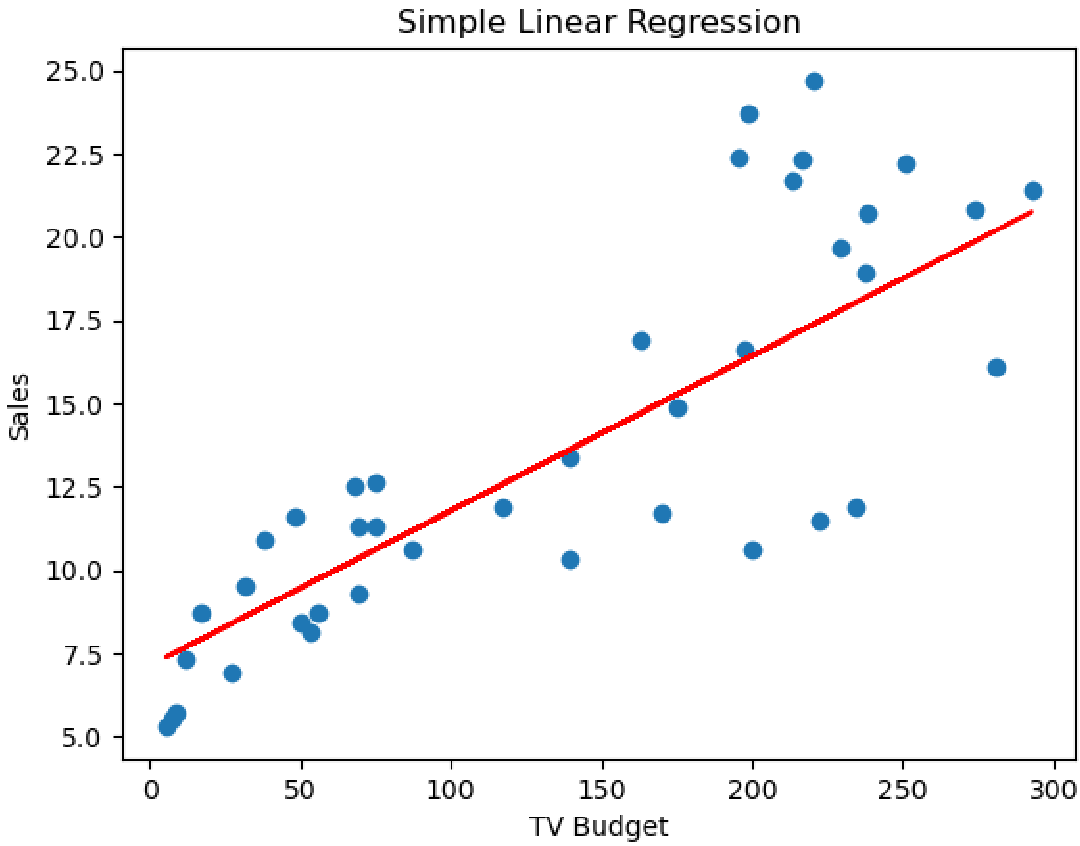
[47]: LinearRegression()

Predicting

[48]:
```
y_pred_simple = model_simple.predict(X_test)
```

Visualize Regression Line

[49]:
```
plt.scatter(X_test, y_test)
plt.plot(X_test, y_pred_simple, color="red")
plt.xlabel("TV Budget")
plt.ylabel("Sales")
plt.title("Simple Linear Regression")
plt.show()
```



Evaluating Model

[50]:
```
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score

print("MAE:", mean_absolute_error(y_test, y_pred_simple))
print("MSE:", mean_squared_error(y_test, y_pred_simple))
print("RMSE:", np.sqrt(mean_squared_error(y_test, y_pred_simple)))
print("R2:", r2_score(y_test, y_pred_simple))
```

```
MAE: 2.444420003751042
MSE: 10.204654118800956
RMSE: 3.194472431998898
R2: 0.6766954295627077
```

For Multiple Linear regression

[51]:
```
X_multi = df[["TV", "Radio", "Newspaper"]]
y_multi = df["Sales"]
```

Splitting data

[52]:
```
X_train_m, X_test_m, y_train_m, y_test_m = train_test_split(X_multi, y_multi,␣
↪test_size=0.2, random_state=42)
```

Train model

[53]:
```
model_multi = LinearRegression()
model_multi.fit(X_train_m, y_train_m)
```

[53]:
```
LinearRegression()
```

Prediction

[54]:
```
y_pred_multi = model_multi.predict(X_test_m)
```

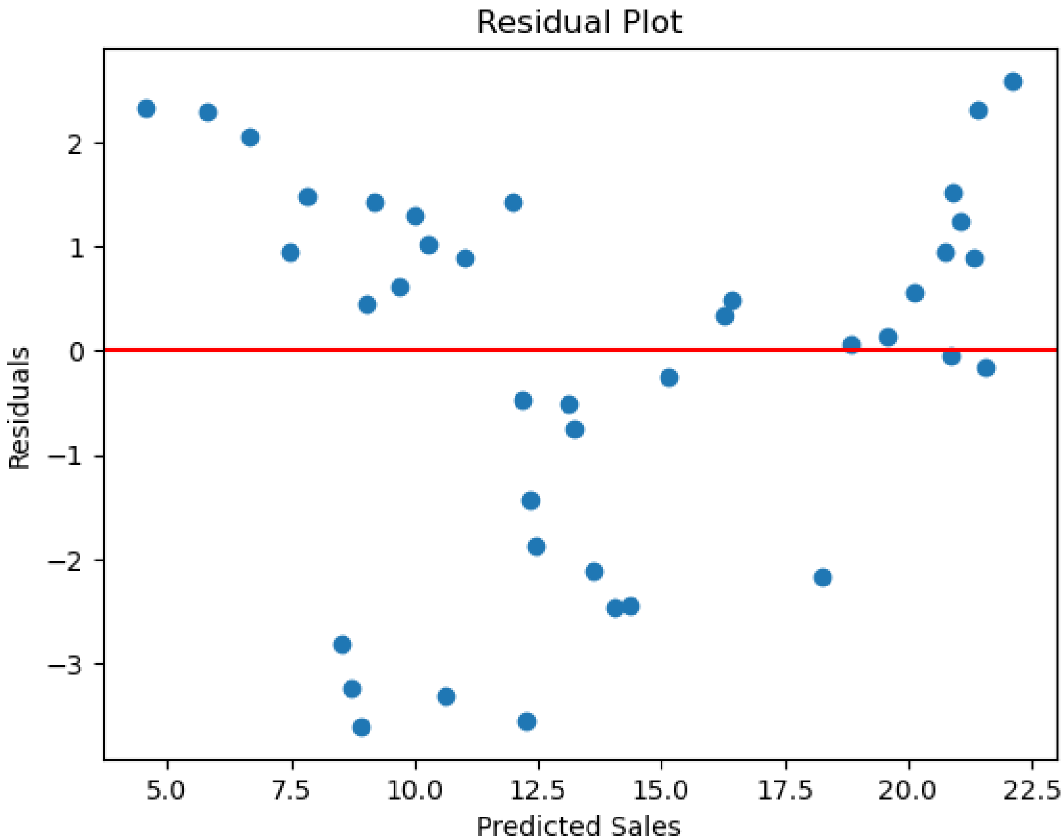
Evaluate Multiple Model

[55]:
```
print("MAE:", mean_absolute_error(y_test_m, y_pred_multi))
print("MSE:", mean_squared_error(y_test_m, y_pred_multi))
print("RMSE:", np.sqrt(mean_squared_error(y_test_m, y_pred_multi)))
print("R2:", r2_score(y_test_m, y_pred_multi))
```

```
MAE: 1.4607567168117606
MSE: 3.1740973539761046
RMSE: 1.7815996615334502
R2: 0.899438024100912
```

Residual Plot

[56]:
```
residuals = y_test_m - y_pred_multi
plt.scatter(y_pred_multi, residuals)
plt.axhline(y=0, color="red")
plt.xlabel("Predicted Sales")
plt.ylabel("Residuals")
plt.title("Residual Plot")
plt.show()
```



Predict on New Data

[57]:
```
new_data = [[200, 40, 50]]
prediction = model_multi.predict(new_data)
print("Predicted Sales:", prediction)
```

```
Predicted Sales: [19.63082872]
```

5 Lab – 05: Logistic Regression

Load iris dataset

[58]:
```
iris = load_iris()
df = pd.DataFrame(iris.data, columns=iris.feature_names)
df["species"] = iris.target
df["binary_species"] = np.where(df["species"]==0, 0, 1) # 0 = setosa, 1 =␣
↪non-setosa
df.head()
```

[58]:
```
   sepal length (cm)  sepal width (cm)  petal length (cm)  petal width (cm)  \
0                5.1               3.5                1.4               0.2
1                4.9               3.0                1.4               0.2
2                4.7               3.2                1.3               0.2
3                4.6               3.1                1.5               0.2
4                5.0               3.6                1.4               0.2

   species  binary_species
0        0               0
1        0               0
2        0               0
3        0               0
4        0               0
```

Splitting the data

[59]:
```
X = df[iris.feature_names]
y = df["binary_species"]
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=42)
```

Train Logistic model

[60]:
```
from sklearn.linear_model import LogisticRegression

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

[60]:
```
LogisticRegression()
```

Making Prediction

[61]:
```
y_pred = log_model.predict(X_test)
```

Accuracy Score

[62]:
```
from sklearn.metrics import accuracy_score

accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
```

```
Accuracy: 1.0
```

Confusion Matrix & Classification Report

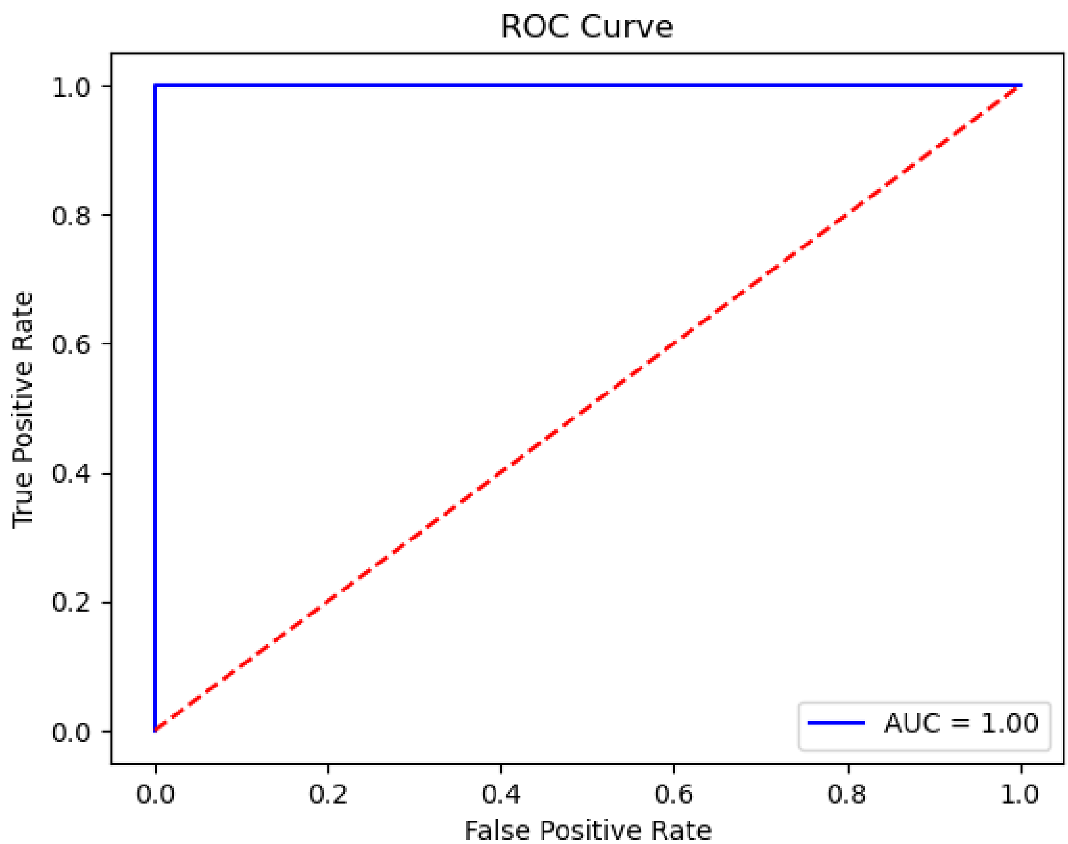
[63]:
```
from sklearn.metrics import confusion_matrix, classification_report

cm = confusion_matrix(y_test, y_pred)
print("Confusion Matrix:\n", cm)
cr = classification_report(y_test, y_pred)
```

```
print("Classification Report:\n", cr)
```

```
Confusion Matrix:
 [[10  0]
 [ 0 20]]
Classification Report:
               precision    recall  f1-score   support

           0       1.00      1.00      1.00        10
           1       1.00      1.00      1.00        20

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

[64]:

```
from sklearn.metrics import roc_curve, roc_auc_score
y_prob = log_model.predict_proba(X_test)[:,1]  # probability for class 1

fpr, tpr, thresholds = roc_curve(y_test, y_prob)
auc_score = roc_auc_score(y_test, y_prob)

plt.plot(fpr, tpr, color="blue", label=f"AUC = {auc_score:.2f}")
plt.plot([0,1], [0,1], color="red", linestyle="--")
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("ROC Curve")
plt.legend()
plt.show()
```



Multi Class Classification

[65]:
```
X_multi = df[iris.feature_names]
y_multi = df["species"]

X_train_m, X_test_m, y_train_m, y_test_m = train_test_split(
    X_multi, y_multi, test_size=0.2, random_state=42
)

log_model_multi = LogisticRegression(multi_class="ovr", max_iter=200)
log_model_multi.fit(X_train_m, y_train_m)

y_pred_m = log_model_multi.predict(X_test_m)
```

Multiclass Evaluation

[66]:
```
print("Accuracy:", accuracy_score(y_test_m, y_pred_m))
print(classification_report(y_test_m, y_pred_m))
```

```
Accuracy: 0.9666666666666667
              precision    recall  f1-score   support
```

```
             0       1.00      1.00      1.00        10
             1       1.00      0.89      0.94         9
             2       0.92      1.00      0.96        11

      accuracy                           0.97        30
     macro avg       0.97      0.96      0.97        30
  weighted avg       0.97      0.97      0.97        30
```

Predicting New Data

[67]:
```
# Example new samples (sepal length, sepal width, petal length, petal width)
new_samples = [[5.1, 3.5, 1.4, 0.2],   # likely setosa
               [6.0, 2.9, 4.5, 1.5],   # likely versicolor
               [6.9, 3.1, 5.4, 2.1]]   # likely virginica

predicted_classes = log_model_multi.predict(new_samples)
predicted_names = [iris.target_names[i] for i in predicted_classes]

predicted_names
```

[67]:
```
[np.str_('setosa'), np.str_('versicolor'), np.str_('virginica')]
```

6 Lab – 06: Decision Trees

[68]:
```
df = df.drop(columns=['binary_species'])
df.head()
```

[68]:
```
   sepal length (cm)  sepal width (cm)  petal length (cm)  petal width (cm)  \
0                5.1               3.5                1.4               0.2
1                4.9               3.0                1.4               0.2
2                4.7               3.2                1.3               0.2
3                4.6               3.1                1.5               0.2
4                5.0               3.6                1.4               0.2

   species
0        0
1        0
2        0
3        0
4        0
```

[69]:
```
from sklearn.tree import DecisionTreeClassifier, plot_tree
```

Splitting the data

[70]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
↪random_state=42)
```

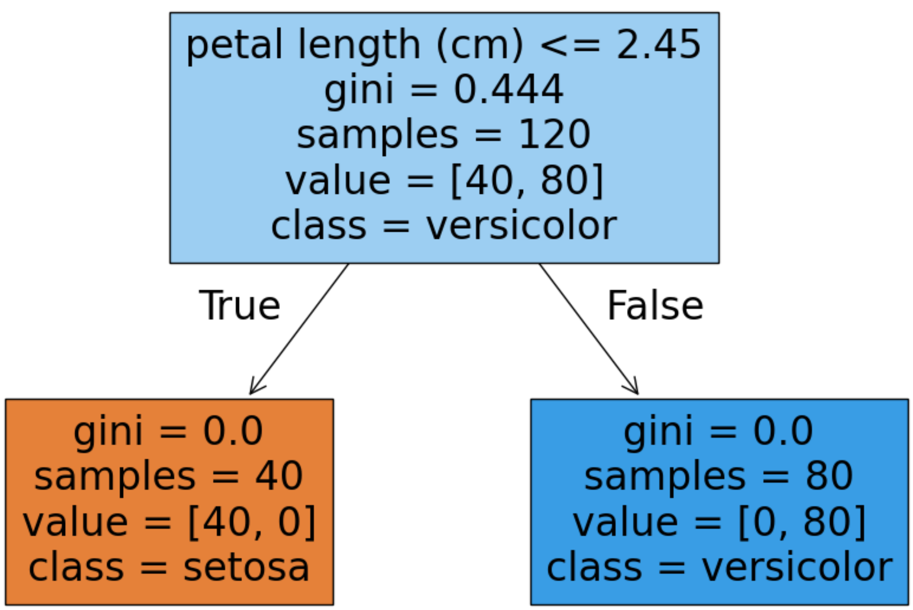
Train Decision Tree

[71]:
```
dt_model = DecisionTreeClassifier(random_state=42)
dt_model.fit(X_train, y_train)
```

[71]:
```
DecisionTreeClassifier(random_state=42)
```

Visualize Tree

[72]:
```
plt.figure(figsize=(12,8))
plot_tree(
    dt_model,
    feature_names=iris.feature_names,
    class_names=iris.target_names,
    filled=True
)
plt.show()
```



Predict

[73]:
```
y_pred = dt_model.predict(X_test)
```

Performance Evaluation

[74]:
```
from sklearn.metrics import accuracy_score, precision_score, recall_score,␣
↪f1_score, classification_report

print("Accuracy:", accuracy_score(y_test, y_pred))
print("Precision:", precision_score(y_test, y_pred, average='weighted'))
print("Recall:", recall_score(y_test, y_pred, average='weighted'))
print("F1 Score:", f1_score(y_test, y_pred, average='weighted'))

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

```
Accuracy: 1.0
Precision: 1.0
Recall: 1.0
F1 Score: 1.0

Classification Report:

              precision    recall  f1-score   support

           0       1.00      1.00      1.00        10
           1       1.00      1.00      1.00        20

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

Gini Vs Entropy

[75]:
```
dt_gini = DecisionTreeClassifier(criterion="gini", random_state=42)
dt_gini.fit(X_train, y_train)

dt_entropy = DecisionTreeClassifier(criterion="entropy", random_state=42)
dt_entropy.fit(X_train, y_train)
```

[75]:
```
DecisionTreeClassifier(criterion='entropy', random_state=42)
```

Compare Accuracy

[76]:
```
print("Gini Accuracy:", accuracy_score(y_test, dt_gini.predict(X_test)))

print("Entropy Accuracy:", accuracy_score(y_test, dt_entropy.predict(X_test)))
# Usually same for iris dataset
```

```
Gini Accuracy: 1.0
Entropy Accuracy: 1.0
```

Pruned Tree

[77]:
```
dt_pruned = DecisionTreeClassifier(
    max_depth=3,
    min_samples_split=4,
    min_samples_leaf=2,
    random_state=42
)

dt_pruned.fit(X_train, y_train)
```

[77]:
```
DecisionTreeClassifier(max_depth=3, min_samples_leaf=2, min_samples_split=4,
                       random_state=42)
```

Compare Accuracy

[78]:
```
print("Unpruned Accuracy:", accuracy_score(y_test, dt_model.predict(X_test)))
print("Pruned Accuracy:", accuracy_score(y_test, dt_pruned.predict(X_test)))
```

```
Unpruned Accuracy: 1.0
Pruned Accuracy: 1.0
```

Tree Complexity

[79]:
```
print("Unpruned depth:", dt_model.get_depth())
print("Pruned depth:", dt_pruned.get_depth())
```

```
Unpruned depth: 1
Pruned depth: 1
```

Predicting in new samples

[80]:
```
new_samples = [
    [5.1, 3.5, 1.4, 0.2],   # likely setosa
    [6.0, 2.9, 4.5, 1.5],   # likely versicolor
    [6.9, 3.1, 5.4, 2.1]    # likely virginica
]

predictions = dt_pruned.predict(new_samples)

predicted_names = [iris.target_names[i] for i in predictions]

predicted_names
```

[80]:
```
[np.str_('setosa'), np.str_('versicolor'), np.str_('versicolor')]
```

7 Lab – 07: Random Forests and Ensemble Methods

Load Titanic dataset

[81]:
```python
df = sns.load_dataset("titanic")
df = df[["survived", "pclass", "sex", "age", "sibsp", "parch", "fare",
 "embarked"]]
df.head()
```

[81]:

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

Basic Preprocessin

[82]:
```python
# Fill missing values
df["age"].fillna(df["age"].median(), inplace=True)
df["embarked"].fillna(df["embarked"].mode()[0], inplace=True)

# One-hot encoding
df = pd.get_dummies(df, columns=["sex", "embarked"], drop_first=True)
```

Train-Test Split

[83]:
```python
X = df.drop("survived", axis=1)
y = df["survived"]
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
 random_state=42)
```

Train Random Forest Classifier

[84]:
```python
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score

rf_model = RandomForestClassifier(n_estimators=100, random_state=42)

rf_model.fit(X_train, y_train)

y_pred = rf_model.predict(X_test)

print("Random Forest Accuracy:", accuracy_score(y_test, y_pred))
```

```
Random Forest Accuracy: 0.8212290502793296
```

Important feature analysis

[85]:
```python
importances = rf_model.feature_importances_

importance_df = pd.DataFrame({
    "Feature": X.columns,
    "Importance": importances
```

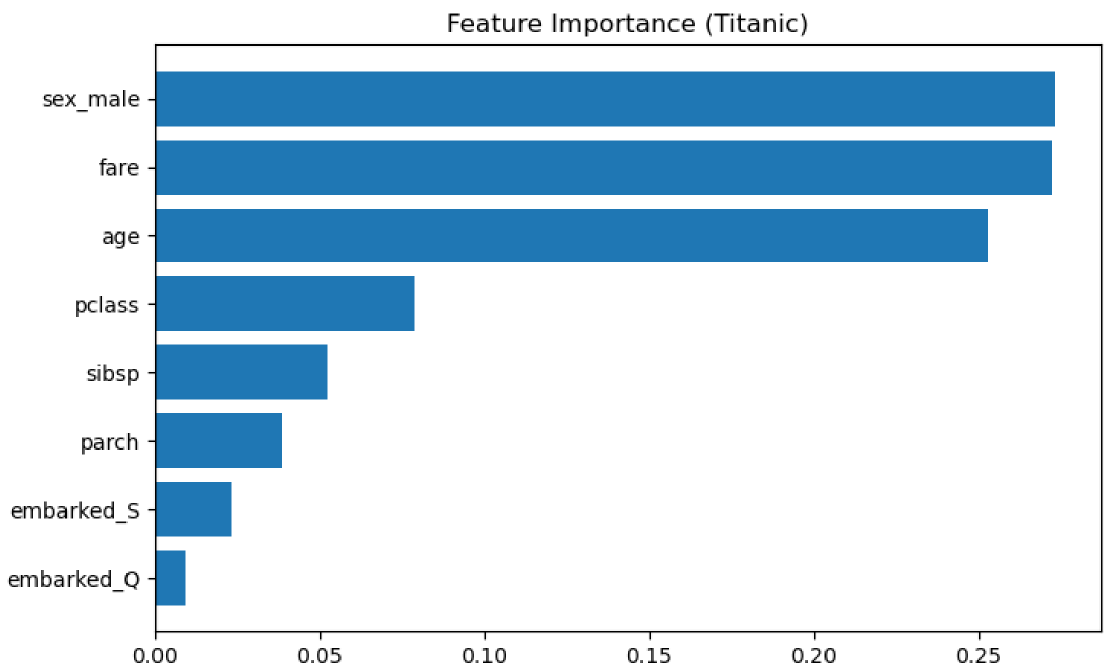
```
}).sort_values(by="Importance", ascending=False)

importance_df.head()
```

[85]:

	Feature	Importance
5	sex_male	0.273316
4	fare	0.272058
1	age	0.252745
0	pclass	0.078616
2	sibsp	0.052192

Visualizing Important feature

[86]:
```
plt.figure(figsize=(8,5))
plt.barh(importance_df["Feature"], importance_df["Importance"])
plt.gca().invert_yaxis()
plt.title("Feature Importance (Titanic)")
plt.show()
```



Hyperparameter Tuning

Model_1: small

[87]:
```
rf_small = RandomForestClassifier(n_estimators=50, max_depth=3,random_state=42)

rf_small.fit(X_train, y_train)
```

```
print("Small Forest Accuracy:", accuracy_score(y_test, rf_small.
 ↪predict(X_test)))
```

```
Small Forest Accuracy: 0.8044692737430168
```

Model_2: Large

[88]:
```
rf_large = RandomForestClassifier(
    n_estimators=200,
    max_depth=None,
    min_samples_split=2,
    min_samples_leaf=1,
    random_state=42
)

rf_large.fit(X_train, y_train)

print("Large Forest Accuracy:", accuracy_score(y_test, rf_large.
 ↪predict(X_test)))
```

```
Large Forest Accuracy: 0.8100558659217877
```

Observation - More trees more stable predictions - Lower depth less overfitting - Very deep trees may overfit

7.1 Random Forest Regressor

[89]:
```
from sklearn.datasets import fetch_california_housing
from sklearn.ensemble import RandomForestRegressor
```

[90]:
```
data = fetch_california_housing()

X = data.data
y = data.target

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

rf_reg = RandomForestRegressor(n_estimators=100, random_state=42)
rf_reg.fit(X_train, y_train)

y_pred = rf_reg.predict(X_test)

print("MSE:", mean_squared_error(y_test, y_pred))
print("RMSE:", np.sqrt(mean_squared_error(y_test, y_pred)))
print("R2:", r2_score(y_test, y_pred))
```

```
MSE: 0.2553684927247781
RMSE: 0.5053399773665033
```

```
R2: 0.8051230593157366
```

8 Lab – 08: Support Vector Machine (SVM)

Loading Iris Data

[91]:
```python
iris = load_iris()
X = iris.data
y = iris.target

# Keep only class 0 and 1
X = X[y != 2]
y = y[y != 2]

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

Train Linear SVM

[92]:
```python
from sklearn.svm import SVC
svm_linear = SVC(kernel="linear")
svm_linear.fit(X_train, y_train)

y_pred = svm_linear.predict(X_test)

print("Linear SVM Accuracy:", accuracy_score(y_test, y_pred))
```

```
Linear SVM Accuracy: 1.0
```

Using Different Kernels

[93]:
```python
kernels = ["linear", "poly", "rbf"]

for k in kernels:
    model = SVC(kernel=k)
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    print(f"{k} Kernel Accuracy:", accuracy_score(y_test, y_pred))
```

```
linear Kernel Accuracy: 1.0
poly Kernel Accuracy: 1.0
rbf Kernel Accuracy: 1.0
```

Hyperparameter Tuning

[94]:
```python
for c in [0.1, 1, 10, 100]:
    model = SVC(kernel="linear", C=c)
```

```
    model.fit(X_train, y_train)
    print(f"C={c} Accuracy:",
          accuracy_score(y_test, model.predict(X_test)))
```

```
C=0.1 Accuracy: 1.0
C=1 Accuracy: 1.0
C=10 Accuracy: 1.0
C=100 Accuracy: 1.0
```

Effect of Gamma (RBF)

[95]:
```
for g in [0.01, 0.1, 1, 10]:
    model = SVC(kernel="rbf", gamma=g)
    model.fit(X_train, y_train)
    print(f"gamma={g} Accuracy:",
          accuracy_score(y_test, model.predict(X_test)))
```

```
gamma=0.01 Accuracy: 1.0
gamma=0.1 Accuracy: 1.0
gamma=1 Accuracy: 1.0
gamma=10 Accuracy: 0.95
```

Multi-class Classification

[96]:
```
# Full dataset
X_full = iris.data
y_full = iris.target

X_train, X_test, y_train, y_test = train_test_split(
    X_full, y_full,
    test_size=0.2,
    random_state=42
)

svm_multi = SVC(kernel="rbf")
svm_multi.fit(X_train, y_train)

print("Multiclass Accuracy:",
      accuracy_score(y_test, svm_multi.predict(X_test)))
```

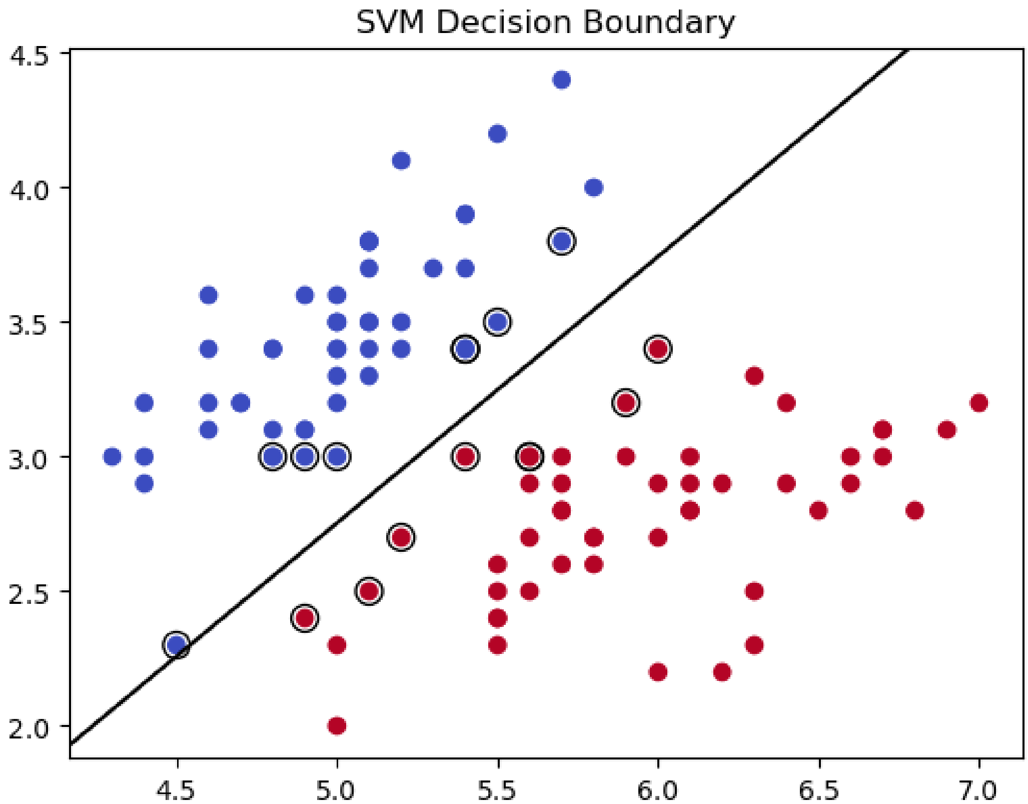
```
Multiclass Accuracy: 1.0
```

Visualizing Decision Boundary (2D)

[97]:
```
X_2d = X[:, :2]   # first two features
y_2d = y

svm_vis = SVC(kernel="linear")
svm_vis.fit(X_2d, y_2d)
```

[97]: SVC(kernel='linear')

[98]:
```
def plot_svm(model, X, y):
    plt.scatter(X[:,0], X[:,1], c=y, cmap="coolwarm")

    ax = plt.gca()
    xlim = ax.get_xlim()
    ylim = ax.get_ylim()

    xx = np.linspace(xlim[0], xlim[1], 30)
    yy = np.linspace(ylim[0], ylim[1], 30)
    YY, XX = np.meshgrid(yy, xx)
    xy = np.vstack([XX.ravel(), YY.ravel()]).T
    Z = model.decision_function(xy).reshape(XX.shape)

    ax.contour(XX, YY, Z, levels=[0], colors="black")

    # Support vectors
    ax.scatter(model.support_vectors_[:,0],
               model.support_vectors_[:,1],
               s=100, facecolors="none",
               edgecolors="k")

    plt.title("SVM Decision Boundary")
    plt.show()

plot_svm(svm_vis, X_2d, y_2d)
```



9 Lab – 09: K-Nearest Neighbors (KNN)

[99]:
```
iris = load_iris()

X = iris.data
y = iris.target
```

Feature Scaling

[100]:
```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
```

Train test split

[101]:
```
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2,␣
↪random_state=42)
```

KNN classification

[102]:
```python
from sklearn.neighbors import KNeighborsClassifier

knn = KNeighborsClassifier(n_neighbors=5)
knn.fit(X_train, y_train)

y_pred = knn.predict(X_test)

print("Accuracy:", accuracy_score(y_test, y_pred))
print(classification_report(y_test, y_pred))
```

```
Accuracy: 1.0
              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
```

Compare Different K Values

[103]:
```python
for k in [1, 3, 5, 7, 9]:
    model = KNeighborsClassifier(n_neighbors=k)
    model.fit(X_train, y_train)
    print(f"K={k} Accuracy:", accuracy_score(y_test, model.predict(X_test)))
```

```
K=1 Accuracy: 0.9666666666666667
K=3 Accuracy: 1.0
K=5 Accuracy: 1.0
K=7 Accuracy: 1.0
K=9 Accuracy: 1.0
```

Compare Distance Metrics

[104]:
```python
for metric in ["euclidean", "manhattan", "minkowski"]:
    model = KNeighborsClassifier(n_neighbors=5, metric=metric)
    model.fit(X_train, y_train)
    print(metric, "Accuracy:", accuracy_score(y_test, model.predict(X_test)))
```

```
euclidean Accuracy: 1.0
manhattan Accuracy: 1.0
minkowski Accuracy: 1.0
```

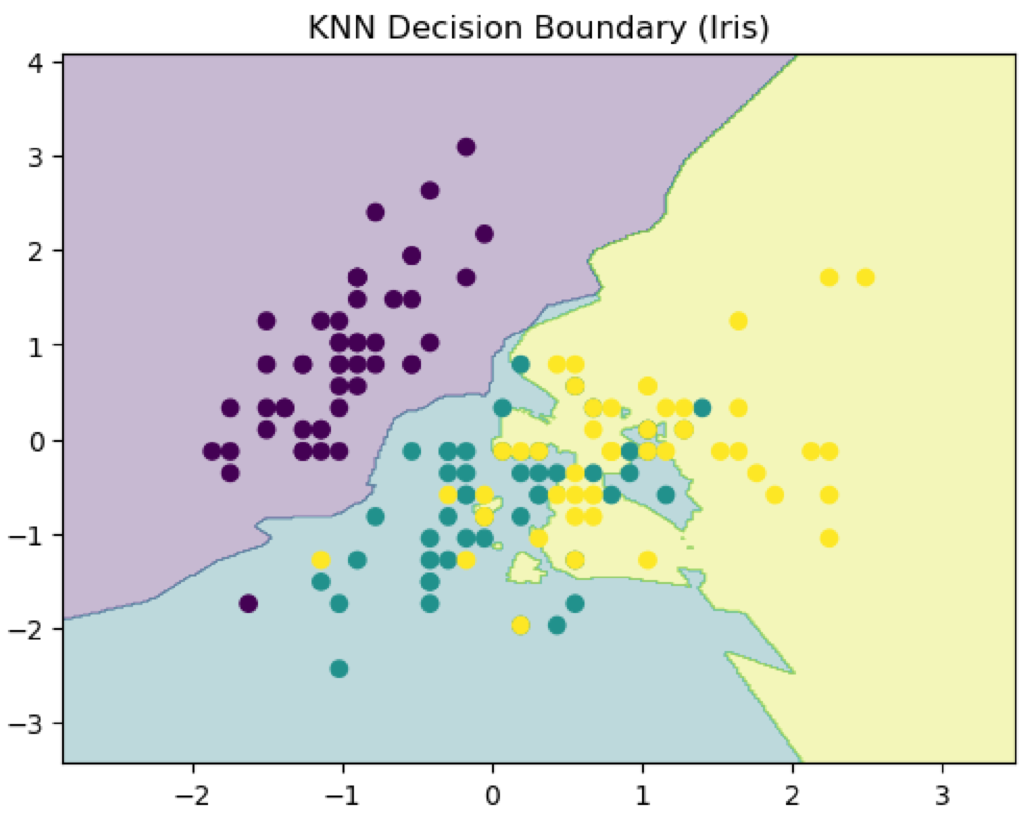
Decision Boundary (2D Visualization)

[105]:
```python
X_2d = X_scaled[:, :2]   # first 2 features
```

```python
knn_2d = KNeighborsClassifier(n_neighbors=5)
knn_2d.fit(X_2d, y)
```

[105]: KNeighborsClassifier()

[106]:
```python
h = 0.02
x_min, x_max = X_2d[:, 0].min() - 1, X_2d[:, 0].max() + 1
y_min, y_max = X_2d[:, 1].min() - 1, X_2d[:, 1].max() + 1

xx, yy = np.meshgrid(
    np.arange(x_min, x_max, h),
    np.arange(y_min, y_max, h)
)

Z = knn_2d.predict(np.c_[xx.ravel(), yy.ravel()])
Z = Z.reshape(xx.shape)

plt.contourf(xx, yy, Z, alpha=0.3)
plt.scatter(X_2d[:, 0], X_2d[:, 1], c=y)
plt.title("KNN Decision Boundary (Iris)")
plt.show()
```



10 Lab – 10: Unsupervised Learning – Clustering

[107]:
```
from sklearn.cluster import KMeans, DBSCAN
from sklearn.metrics import silhouette_score, davies_bouldin_score
from scipy.cluster.hierarchy import dendrogram, linkage
```

Load Dataset (Without Labels)

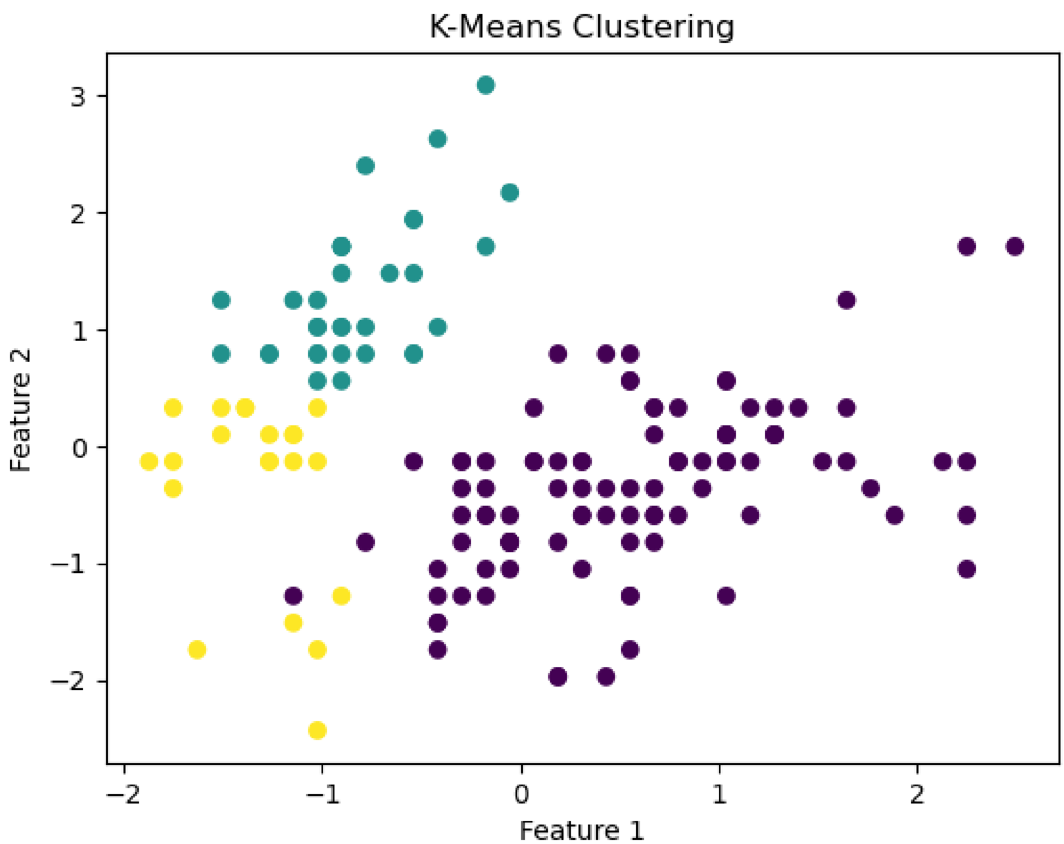
[108]:
```
iris = load_iris()
X = iris.data   # only features

scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
```

K-Means Clustering

[109]:
```
kmeans = KMeans(n_clusters=3, random_state=42)
clusters = kmeans.fit_predict(X_scaled)
```

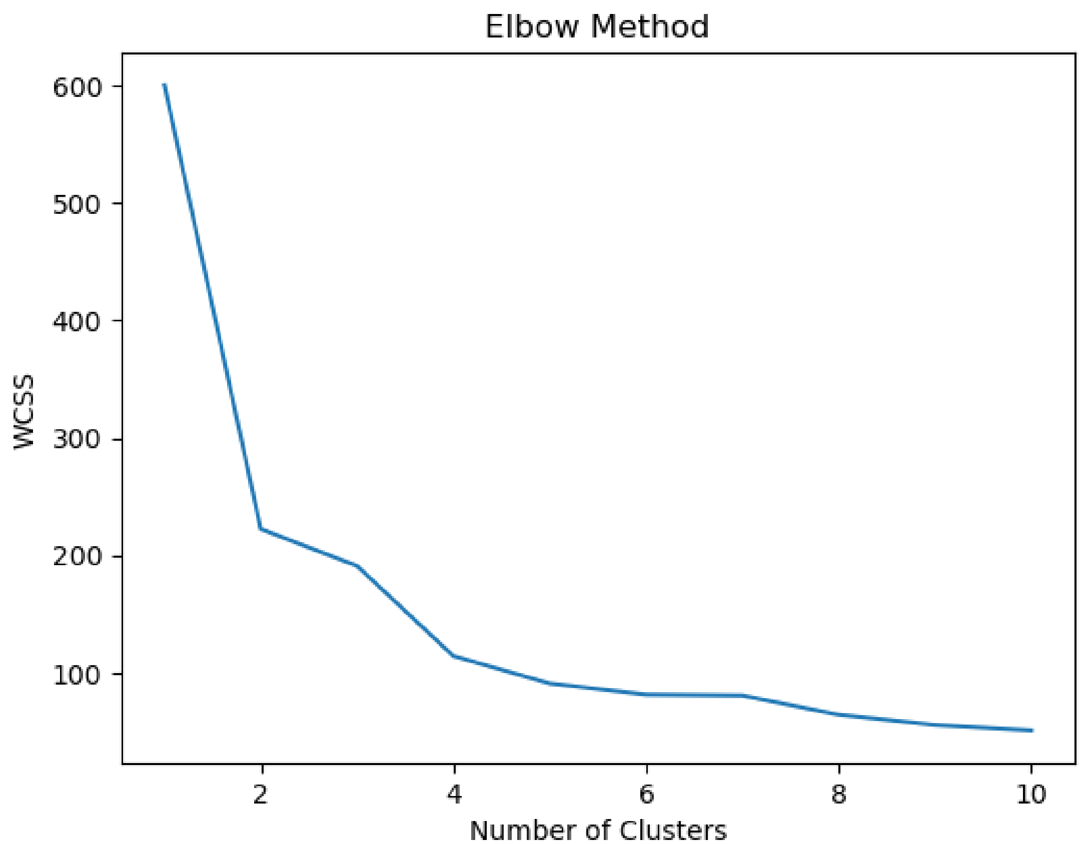
Visualize Clusters

[110]:
```
plt.scatter(X_scaled[:, 0], X_scaled[:, 1], c=clusters)
plt.title("K-Means Clustering")
plt.xlabel("Feature 1")
plt.ylabel("Feature 2")
plt.show()
```



Elbow Method (Optimal K)

[111]:
```
wcss = []

for k in range(1, 11):
    model = KMeans(n_clusters=k, random_state=42)
    model.fit(X_scaled)
    wcss.append(model.inertia_)

plt.plot(range(1, 11), wcss)
plt.xlabel("Number of Clusters")
plt.ylabel("WCSS")
plt.title("Elbow Method")
plt.show()
```



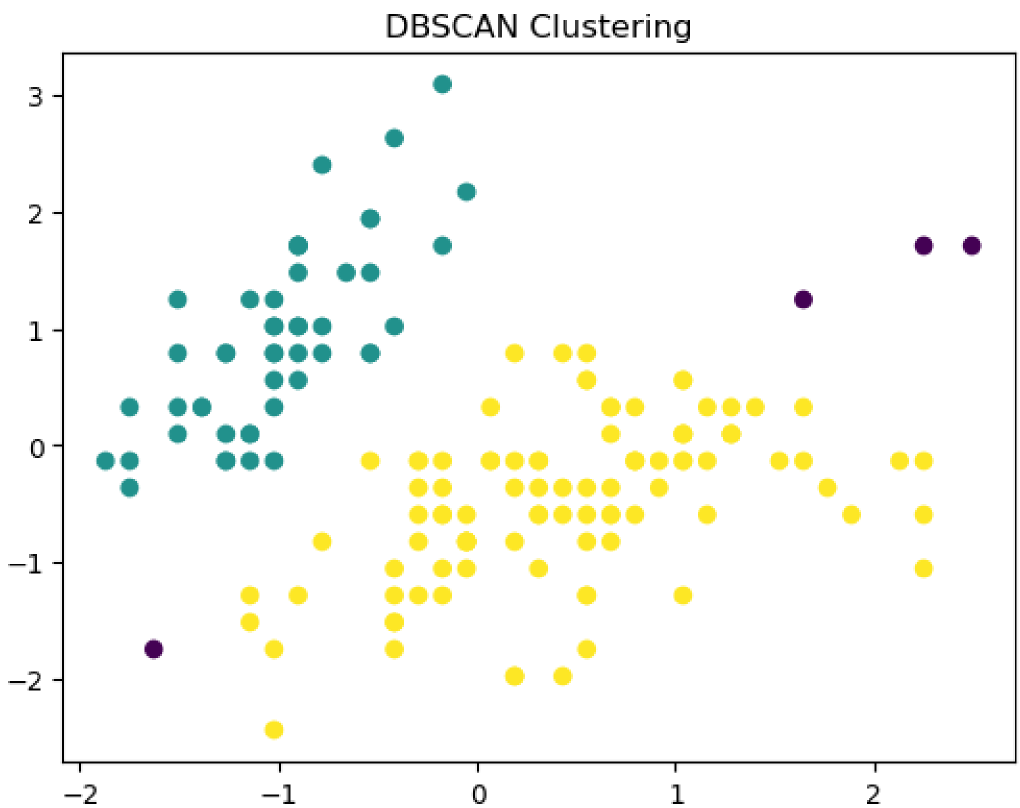
Hierarchical Clustering

linked = linkage(X_scaled, method='ward')

plt.figure(figsize=(8,5)) dendrogram(linked) plt.title("Hierarchical Clustering Dendrogram") plt.show()

DBSCAN Clustering

[112]:
```
dbscan = DBSCAN(eps=0.8, min_samples=5)
db_clusters = dbscan.fit_predict(X_scaled)

plt.scatter(X_scaled[:, 0], X_scaled[:, 1], c=db_clusters)
plt.title("DBSCAN Clustering")
plt.show()
```



Silhouette Score

[113]:
```
print("KMeans Silhouette:", silhouette_score(X_scaled, clusters))
print("DBSCAN Silhouette:", silhouette_score(X_scaled, db_clusters))
```

```
KMeans Silhouette: 0.4798814508199817
DBSCAN Silhouette: 0.5216965052515835
```

Davies-Bouldin Index

[114]:
```
print("KMeans DB Index:", davies_bouldin_score(X_scaled, clusters))
print("DBSCAN DB Index:", davies_bouldin_score(X_scaled, db_clusters))
```

```
KMeans DB Index: 0.7893630242997912
DBSCAN DB Index: 1.9432005358011466
```

11 Lab – 11: Dimensionality Reduction

[115]:
```
from sklearn.datasets import load_digits
from sklearn.decomposition import PCA
from sklearn.manifold import TSNE
```

Load High-Dimensional Dataset

[116]:
```
digits = load_digits()
X = digits.data      # 64 features
y = digits.target    # 0-9 classes

print("Original shape:", X.shape)
```

```
Original shape: (1797, 64)
```

Standardize Data

[117]:
```
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
```

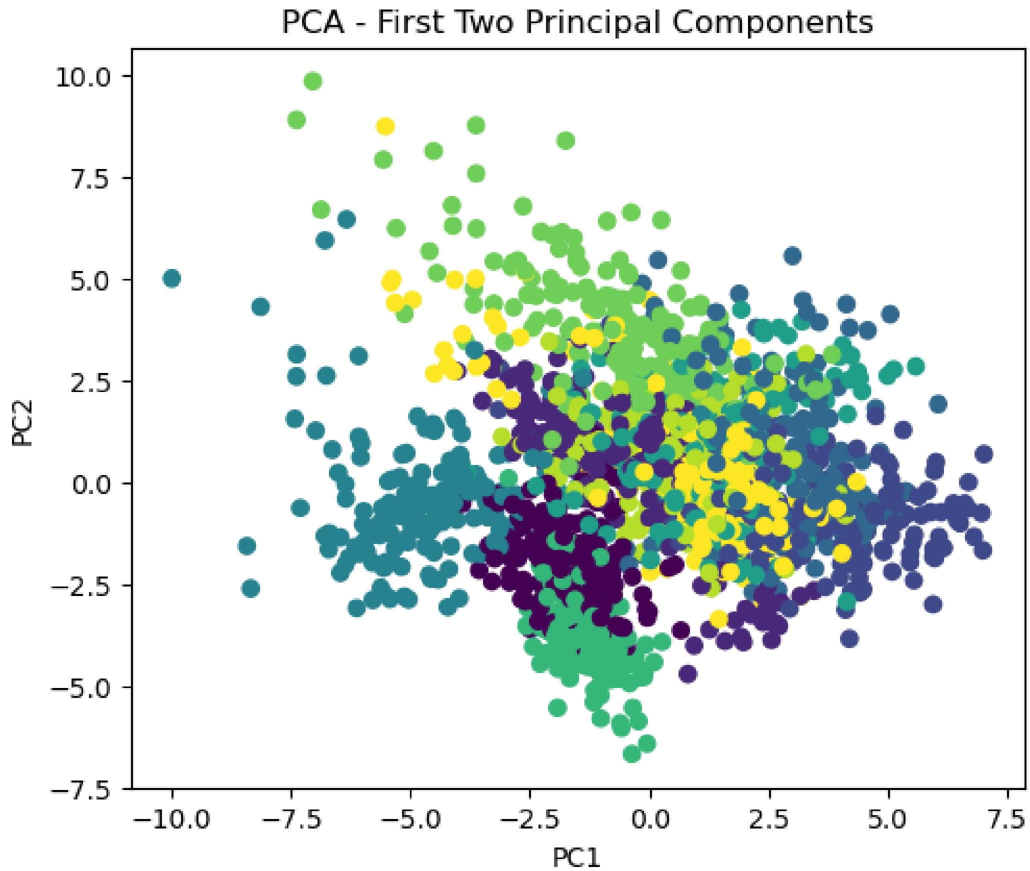
Apply PCA (2 Components for Visualization)

[118]:
```
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X_scaled)

print("Reduced shape:", X_pca.shape)
```

```
Reduced shape: (1797, 2)
```

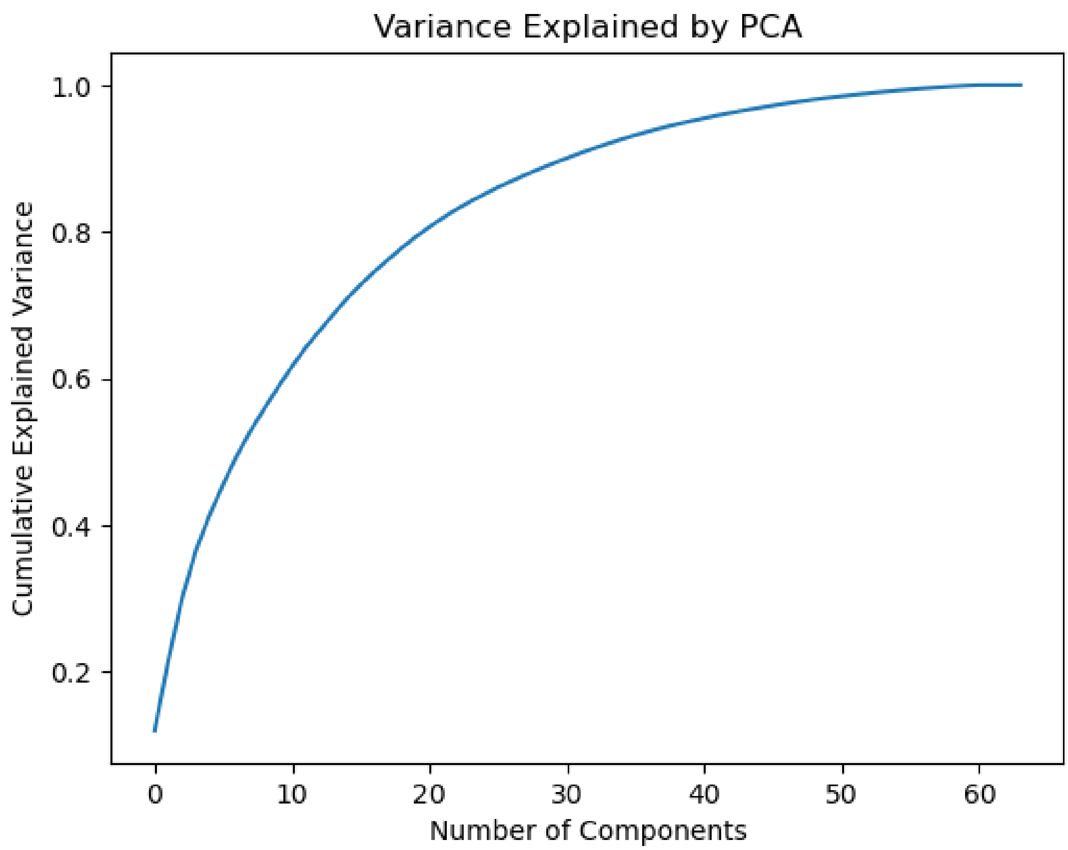
Visualize First Two Principal Components

[119]:
```
plt.figure(figsize=(6,5))
plt.scatter(X_pca[:,0], X_pca[:,1], c=y)
plt.title("PCA - First Two Principal Components")
plt.xlabel("PC1")
plt.ylabel("PC2")
plt.show()
```



Variance

[120]:
```
pca_full = PCA()
pca_full.fit(X_scaled)

cumulative_variance = np.cumsum(pca_full.explained_variance_ratio_)

plt.plot(cumulative_variance)
plt.xlabel("Number of Components")
plt.ylabel("Cumulative Explained Variance")
plt.title("Variance Explained by PCA")
plt.show()
```



Number of Components for 95% Variance

[121]:
```
n_components_95 = np.argmax(cumulative_variance >= 0.95) + 1
print("Components for 95% variance:", n_components_95)
```

```
Components for 95% variance: 40
```

Reduce to 95% Variance

[122]:
```
pca_95 = PCA(n_components=n_components_95)
X_reduced = pca_95.fit_transform(X_scaled)
```

Train Classifier (Logistic Regression)

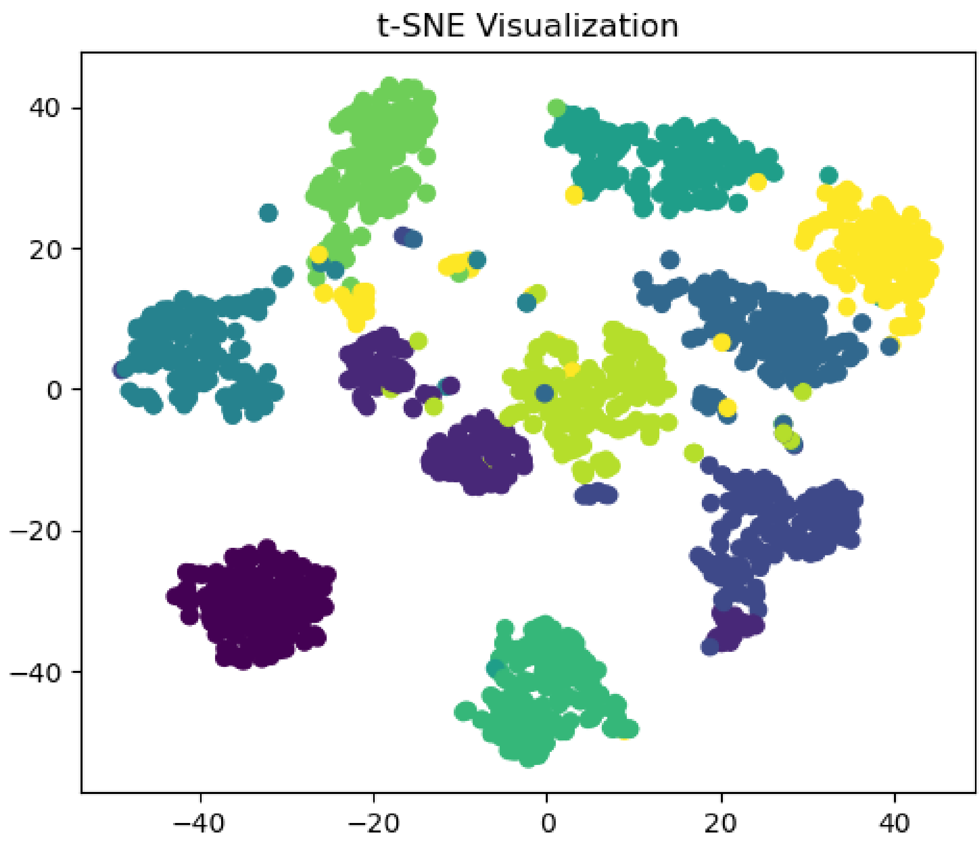
[123]:
```
X_train, X_test, y_train, y_test = train_test_split(
    X_reduced, y,
    test_size=0.2,
    random_state=42
)

clf = LogisticRegression(max_iter=2000)
clf.fit(X_train, y_train)
```

```
y_pred = clf.predict(X_test)

print("Accuracy with PCA:", accuracy_score(y_test, y_pred))
```

```
Accuracy with PCA: 0.9611111111111111
```

t-SNE Visualization

[124]:
```
tsne = TSNE(n_components=2, random_state=42)
X_tsne = tsne.fit_transform(X_scaled)

plt.figure(figsize=(6,5))
plt.scatter(X_tsne[:,0], X_tsne[:,1], c=y)
plt.title("t-SNE Visualization")
plt.show()
```



Observations: - t-SNE gives better visual cluster separation. - PCA is suitable for feature reduction before ML. - t-SNE is mainly for visualization.

12 Lab – 12: Model Evaluation & Cross-Validation

[125]:
```
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import KFold, StratifiedKFold, cross_val_score,␣
↪GridSearchCV
```

Load and Scale the dataset

[126]:
```
data = load_breast_cancer()
X = data.data
y = data.target

scaler = StandardScaler()
X = scaler.fit_transform(X)
```

Train-Test Split Evaluation

[127]:
```
X_train, X_test, y_train, y_test = train_test_split(
    X, y,
    test_size=0.2,
    random_state=42
)

model = LogisticRegression(max_iter=2000)
model.fit(X_train, y_train)

y_pred = model.predict(X_test)

print("Accuracy:", accuracy_score(y_test, y_pred))
print("Precision:", precision_score(y_test, y_pred))
print("Recall:", recall_score(y_test, y_pred))
print("F1 Score:", f1_score(y_test, y_pred))
```

```
Accuracy: 0.9736842105263158
Precision: 0.9722222222222222
Recall: 0.9859154929577465
F1 Score: 0.9790209790209791
```

K-Fold Cross-Validation

[128]:
```
kf = KFold(n_splits=5, shuffle=True, random_state=42)

cv_scores = cross_val_score(model, X, y, cv=kf)

print("KFold Accuracy Scores:", cv_scores)
print("Average Accuracy:", np.mean(cv_scores))
```

```
KFold Accuracy Scores: [0.97368421 0.98245614 0.96491228 0.99122807 0.97345133]
Average Accuracy: 0.9771464058376029
```

Stratified K-Fold Cross-Validation

[129]:
```python
skf = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)

strat_scores = cross_val_score(model, X, y, cv=skf)

print("StratifiedKFold Scores:", strat_scores)
print("Average Accuracy:", np.mean(strat_scores))
```

```
StratifiedKFold Scores: [0.97368421 0.94736842 0.96491228 0.99122807 0.99115044]
Average Accuracy: 0.9736686849868033
```

Hyperparameter Tuning with Grid Search

[130]:
```python
param_grid = {
    'C': [0.1, 1, 10],
    'gamma': ['scale', 0.01, 0.001],
    'kernel': ['rbf']
}

grid = GridSearchCV(
    SVC(probability=True),
    param_grid,
    cv=5,
    scoring='accuracy'
)

grid.fit(X_train, y_train)

print("Best Parameters:", grid.best_params_)
print("Best CV Score:", grid.best_score_)
```

```
Best Parameters: {'C': 1, 'gamma': 'scale', 'kernel': 'rbf'}
Best CV Score: 0.9736263736263737
```

Evaluate on test set:

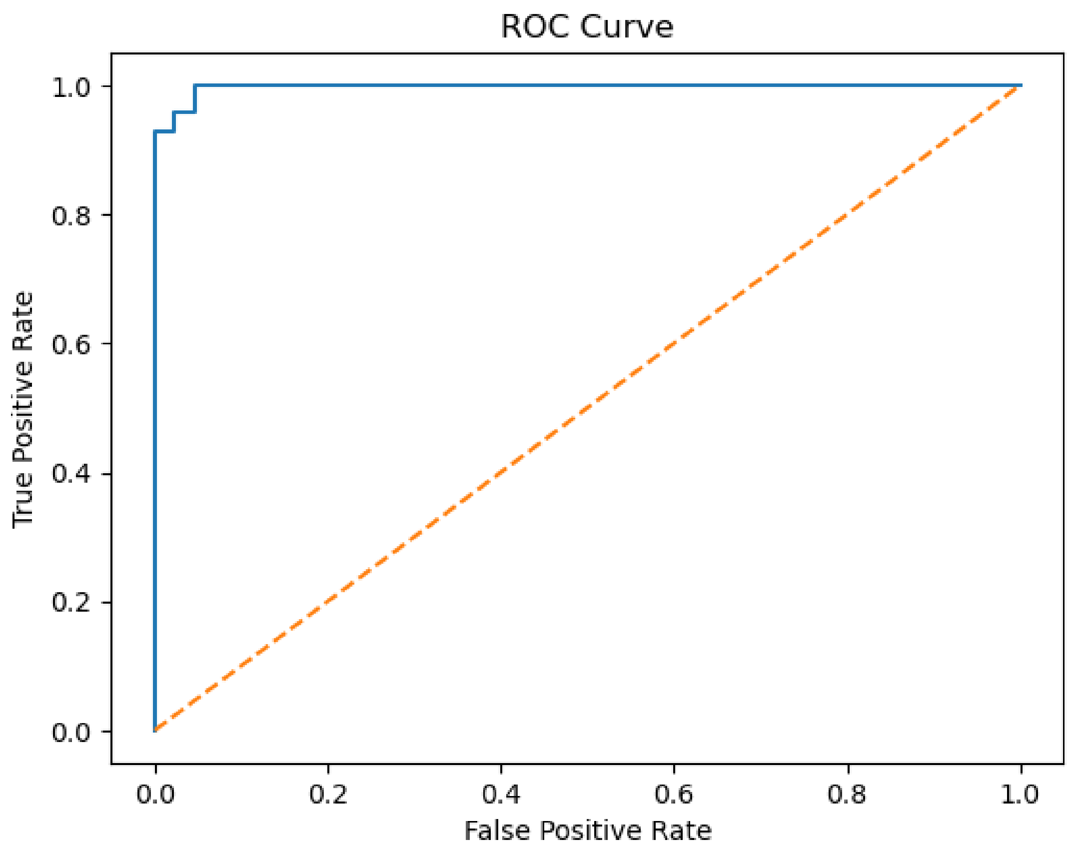
[131]:
```python
best_model = grid.best_estimator_
y_pred = best_model.predict(X_test)

print("Test Accuracy:", accuracy_score(y_test, y_pred))
```

```
Test Accuracy: 0.9736842105263158
```

ROC Curve and AUC

[132]:
```python
y_prob = best_model.predict_proba(X_test)[:, 1]

fpr, tpr, thresholds = roc_curve(y_test, y_prob)
auc_score = roc_auc_score(y_test, y_prob)

plt.plot(fpr, tpr)
```

```
plt.plot([0,1], [0,1], linestyle='--')
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("ROC Curve")
plt.show()

print("AUC Score:", auc_score)
```




```
AUC Score: 0.99737962659679
```

[ ]: