1.

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
#Step 2: Load Dataset from URL
url= "https://raw.githubusercontent.com/datasciencedojo/datasets/master/titanic.csv" # Titanic
dataset
df=pd.read_csv(url)
# Step 3: Initial Exploration of Data
print("In=====-========== Dataset Overview -===============\n")
print(df.head().to_string())

print("In================ Dataset Summary =========\n")
df.info()
print("In=== Basic Statistics\n")
print(df.describe().to_string())
#Step 4: Data Preprocessing - Handling Missing Values
print("In=====Missing Values======\n")
print(df.isnull().sum())

# Filling missing values
#Age: Filling with median value
df["Age"] = df["Age"].fillna(df["Age"].median())
#Embarked: Filling with most frequent value
df["Embarked"] = df["Embarked"].fillna(df["Embarked"].mode()[0])
# Drop Cabin column due to too many missing values
if "Cabin" in df.columns:
   df.drop(columns=["Cabin"], inplace=True)
print(df.isnull().sum())
# Convert categorical variables to appropriate types
df['Sex'] = df['Sex'].astype("category")
df['Embarked'] = df['Embarked'].astype("category")
print(df.dtypes)
```

2.

```
# pract2
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from scipy import stats
# Create a sample "Academic performance" dataset
data= {
'Student_ID': range(1, 11), 'Maths_Score': [85, 92, np.nan, 78, 88, 85, 79, 93, 105, 74],
   'English_Score': [78, 85, 90, 82, np.nan, 88, 74, 91, 86, 80], 'Science_Score': [89, 91, 94, 76, 88,
np.nan, 72, 85, 98, 74]
}
df= pd.DataFrame(data)
# Display the initial dataset
print("Initial Dataset:")
print(df)
#1. Scan all variables for missing values and inconsistencies
# Check for missing values
missing_values = df.isnull().sum()
print("InMissing values in each column:")
print(missing_values)
```

```
# Handling missing values by filling them with the mean of respective columns
df['Maths_Score'].fillna(df['Maths_Score'].mean(), inplace=True)
df['English_Score'].fillna(df['English_Score'].mean(), inplace=True)
df['Science_Score'].fillna(df['Science_Score'].mean(), inplace=True)
# Display dataset after handling missing values
print("InDataset after handling missing values:")
print(df)
#2.Scan all numeric variables for outliers
# Detect outliers using z-score
z_scores =np.abs(stats.zscore(df[['Maths_Score','Science_Score','English_Score' ]]))
print("nz-scores of the dataset:")
print(z_scores)
# Set a threshold for Z-score, consider any z-score greater than 3 as an outlier
outliers = (z_scores > 3)
print("Inoutliers detected (True means outlier):")
print(outliers)
# 3. Apply data transformations on at least one variable
# Applying a log transformation to 'Maths _Score" to decrease skewness
df[ 'Maths_Score_log']=np.log(df['Maths_Score']+1)
plt.figure(figsize=(10, 6))
sns.histplot(df['Maths_Score'], kde=True, color='blue', label='Original Maths_Score')
sns.histplot(df['Maths_Score_log'], kde=True, color='red', label='Log-transformed Maths_Score')
plt.legend()
plt.title("Effect of Log Transformation on Maths_Score")
plt.show()
print(df)
```

3.

```
import pandas as pd
import numpy as np
# Create a sample employee dataset
data = {
'Employee ID': [1,2,3,4,5,6,7,8,9,10],
'Employee ID': range(1,11), 'Age': [25, 30, 35, 40, 28,45,50,60, 30,20],
'Salary': [45000, 55000, 60000,75000, 48000, 80000,90000,100000,300000,2000000]
}
dff=pd.DataFrame(data)
# Create a DataFrame
# dataframe will use dictionary (data) and will convert, df = pd.DataFrame(data)
# Display the whole dataset
print(dff)
# df.head() - first S rows will be displayed
# df.tail() - last S rows will be displayed
# Define a function to categorize age groups
def categorize_age(age):
   if 20 <= age < 30:
      return '20-30'
   elif 30 <= age< 40:
      return '30-40'
   elif 40 < age < 50:
    return '40-50'
      elif 50 <= age < 60:
         return '50-60'
      elif 60 <= age < 70:
         return '60-70'
      else:
         return 'Unknown'
```

```
#In case there are ages outside the range
# Apply the function to the 'Age' column to create a new 'Age Group
dff['AgeGroup'] = dff['Age'].apply(categorize_age)
# Group the data by Age Group and calculate summary statistics for
summary_stats = dff.groupby('AgeGroup')['Salary'].describe()
# Display the summary statistics
print(summary_stats)

import seaborn as sns
import pandas as pd
# Load the Iris dataset from seaborn
dfff = pd.read_csv("C:/Users/DELL/Downloads/archive/Iris.csv")

#Group by species and calculate summary statistics
summary_stats = dfff.groupby('Species').describe()

# Print the summary statistics for each feature
for Species in dfff['Species'].unique():
    print(f"Summary statistics for {Species}:")
    Species_stats = summary_stats.loc[Species]
    print(Species_stats[['SepalLengthCm', 'SepalWidthCm','PetalLengthCm','PetalWidthCm']])
    print("\n" + "-"*50 + "\n")
```

In [17]:
```
## 1. Import Libraries

import pandas as pd
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 [23]:
```
## 2. Load Dataset
df = pd.read_csv("HousingData.csv")
df = df.dropna()
```

In [24]:
```
## 3. Train-Test Split
X = df.drop('MEDV', axis=1)
y = df['MEDV']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

In [26]:
```
## 4. Train the Model
model = LinearRegression()
model.fit(X_train, y_train)
```

Out[26]:

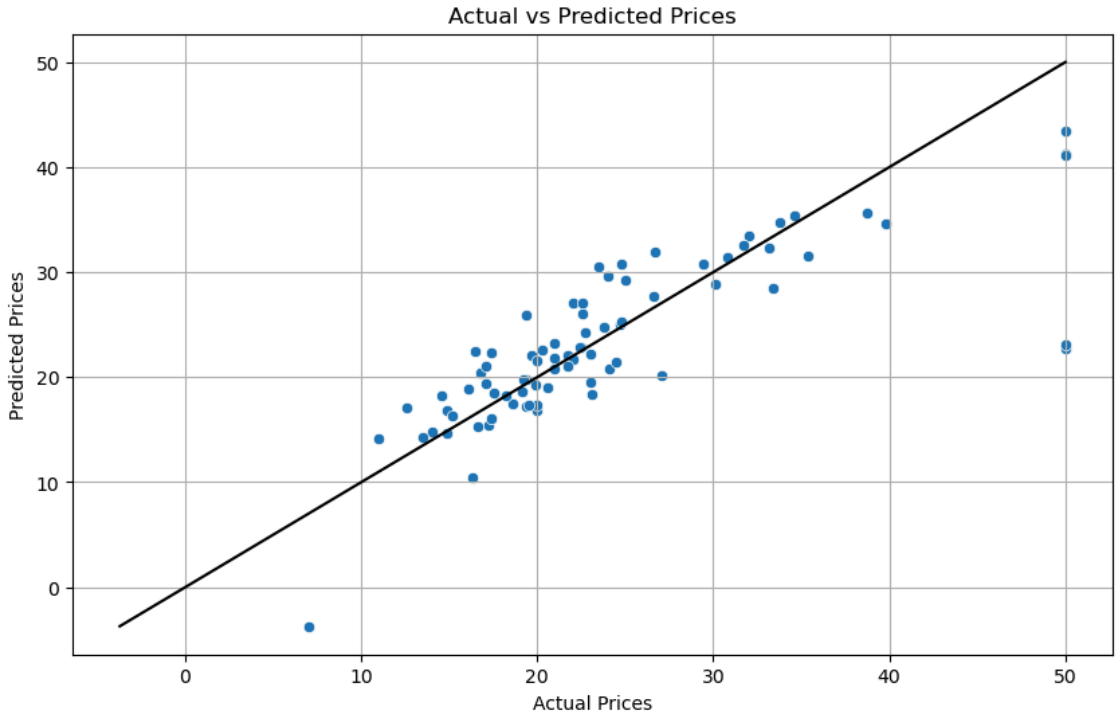
▾ LinearRegression

LinearRegression()

In [27]:
```
## 5. Evaluate the Model
y_pred = model.predict(X_test)
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)
print(f"Mean Squared Error: {mse}")
print(f"R^2 Score: {r2}")
```

```
Mean Squared Error: 31.45404766495081
R^2 Score: 0.6270849941673199
```

In [34]:
```
plt.figure(figsize=(10,6))

sns.scatterplot(x=y_test, y=y_pred)

min_val = min(y_test.min(), y_pred.min())
max_val = max(y_test.max(), y_pred.max())
plt.plot([min_val, max_val],
         [min_val, max_val],
         linewidth=1.5,
         color='black')    # solid line

plt.xlabel("Actual Prices")
plt.ylabel("Predicted Prices")
plt.title("Actual vs Predicted Prices")
plt.grid(True)

plt.show()
```



In [ ]:

In [1]:
```
# Import necessary libraries
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.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import confusion_matrix, accuracy_score, classification_report
```

In [2]:
```
# Load dataset
df = pd.read_csv('Social_Network_Ads.csv')
df.head()
```

Out[2]:

	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 [3]:
```
# Data preprocessing
# Assuming columns: 'Age', 'EstimatedSalary', 'Purchased'
X = df[['Age', 'EstimatedSalary']]
y = df['Purchased']

# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=0)

# Feature scaling
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

In [4]:
```
# Logistic Regression Model
classifier = LogisticRegression(random_state=0)
classifier.fit(X_train, y_train)
```

Out[4]:

LogisticRegression

LogisticRegression(random_state=0)

In [5]:
```
# Predictions
y_pred = classifier.predict(X_test)

# Evaluation
print(confusion_matrix(y_test, y_pred))
print(accuracy_score(y_test, y_pred))
print(classification_report(y_test, y_pred))
```

```
[[65  3]
 [ 8 24]]
0.89
              precision    recall  f1-score   support

           0       0.89      0.96      0.92        68
           1       0.89      0.75      0.81        32

    accuracy                           0.89       100
   macro avg       0.89      0.85      0.87       100
weighted avg       0.89      0.89      0.89       100
```

Naive Bayes Classification on Iris Dataset

This notebook demonstrates how to implement a simple Naive Bayes classification algorithm using Python on the `iris.csv` dataset.

In [1]:
```
# Import necessary libraries
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
```

In [6]:
```
# Load dataset
df = pd.read_csv('iris.csv')
```

```
Index(['Id', 'SepalLengthCm', 'SepalWidthCm', 'PetalLengthCm', 'PetalWidthCm',
       'Species'],
      dtype='object')
```

In [7]:
```
# Split dataset into features and target
X = df.drop('Species', axis=1)
y = df['Species']

# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
```

In [8]:
```
# Train Naive Bayes model
model = GaussianNB()
model.fit(X_train, y_train)
```

Out[8]:
```
GaussianNB()
```

In [9]:
```
# Predict and evaluate
y_pred = model.predict(X_test)

print('Accuracy:', accuracy_score(y_test, y_pred))
print('Confusion Matrix:\n', confusion_matrix(y_test, y_pred))
print('Classification Report:\n', classification_report(y_test, y_pred))
```

```
Accuracy: 1.0
Confusion Matrix:
 [[19  0  0]
 [ 0 13  0]
 [ 0  0 13]]
Classification Report:
                 precision    recall  f1-score   support

    Iris-setosa       1.00      1.00      1.00        19
Iris-versicolor       1.00      1.00      1.00        13
 Iris-virginica       1.00      1.00      1.00        13

       accuracy                           1.00        45
      macro avg       1.00      1.00      1.00        45
   weighted avg       1.00      1.00      1.00        45
```

In [3]:
```
import os
import nltk
from nltk.tokenize import word_tokenize, sent_tokenize
from nltk.tag import pos_tag
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer, WordNetLemmatizer
from nltk.probability import FreqDist
import math
import matplotlib.pyplot as plt
import pandas as pd
```

```
WARNING: Skipping C:\Users\Adnan\anaconda3\Lib\site-packages\numpy-2.2.4.dist-inf
o due to invalid metadata entry 'name'
WARNING: Skipping C:\Users\Adnan\anaconda3\Lib\site-packages\numpy-2.2.4.dist-inf
o due to invalid metadata entry 'name'
WARNING: Skipping C:\Users\Adnan\anaconda3\Lib\site-packages\numpy-2.2.4.dist-inf
o due to invalid metadata entry 'name'
WARNING: Skipping C:\Users\Adnan\anaconda3\Lib\site-packages\numpy-2.2.4.dist-inf
o due to invalid metadata entry 'name'
WARNING: Skipping C:\Users\Adnan\anaconda3\Lib\site-packages\numpy-2.2.4.dist-inf
o due to invalid metadata entry 'name'
WARNING: Skipping C:\Users\Adnan\anaconda3\Lib\site-packages\numpy-2.2.4.dist-inf
o due to invalid metadata entry 'name'
Requirement already satisfied: nltk in c:\users\adnan\anaconda3\lib\site-packages
(3.8.1)
Requirement already satisfied: click in c:\users\adnan\anaconda3\lib\site-package
s (from nltk) (8.1.7)
Requirement already satisfied: joblib in c:\users\adnan\anaconda3\lib\site-packag
es (from nltk) (1.4.2)
Requirement already satisfied: regex>=2021.8.3 in c:\users\adnan\anaconda3\lib\si
te-packages (from nltk) (2023.10.3)
Requirement already satisfied: tqdm in c:\users\adnan\anaconda3\lib\site-packages
(from nltk) (4.66.4)
Requirement already satisfied: colorama in c:\users\adnan\anaconda3\lib\site-pack
ages (from click->nltk) (0.4.6)
```

In [45]:
```
sent = "Sachin is considered to be one of the greatest cricket players. Virat is
print(word_tokenize(sent))
print(sent_tokenize(sent))
```

```
['Sachin', 'is', 'considered', 'to', 'be', 'one', 'of', 'the', 'greatest', 'crick
et', 'players', '.', 'Virat', 'is', 'the', 'captain', 'of', 'the', 'Indian', 'cri
cket', 'team']
['Sachin is considered to be one of the greatest cricket players.', 'Virat is the
captain of the Indian cricket team']
```

In [47]:
```
nltk.download('stopwords')
stop_words = stopwords.words('english')
print(stop_words)
```

```
['a', 'about', 'above', 'after', 'again', 'against', 'ain', 'all', 'am', 'an', 'a
nd', 'any', 'are', 'aren', "aren't", 'as', 'at', 'be', 'because', 'been', 'befor
e', 'being', 'below', 'between', 'both', 'but', 'by', 'can', 'couldn', "could
n't", 'd', 'did', 'didn', "didn't", 'do', 'does', 'doesn', "doesn't", 'doing', 'd
on', "don't", 'down', 'during', 'each', 'few', 'for', 'from', 'further', 'had',
'hadn', "hadn't", 'has', 'hasn', "hasn't", 'have', 'haven', "haven't", 'having',
'he', "he'd", "he'll", 'her', 'here', 'hers', 'herself', "he's", 'him', 'himsel
f', 'his', 'how', 'i', "i'd", 'if', "i'll", "i'm", 'in', 'into', 'is', 'isn', "is
n't", 'it', "it'd", "it'll", "it's", 'its', 'itself', "i've", 'just', 'll', 'm',
'ma', 'me', 'mightn', "mightn't", 'more', 'most', 'mustn', "mustn't", 'my', 'myse
lf', 'needn', "needn't", 'no', 'nor', 'not', 'now', 'o', 'of', 'off', 'on', 'onc
e', 'only', 'or', 'other', 'our', 'ours', 'ourselves', 'out', 'over', 'own', 'r
e', 's', 'same', 'shan', "shan't", 'she', "she'd", "she'll", "she's", 'should',
'shouldn', "shouldn't", "should've", 'so', 'some', 'such', 't', 'than', 'that',
"that'll", 'the', 'their', 'theirs', 'them', 'themselves', 'then', 'there', 'thes
e', 'they', "they'd", "they'll", "they're", "they've", 'this', 'those', 'throug
h', 'to', 'too', 'under', 'until', 'up', 've', 'very', 'was', 'wasn', "wasn't",
'we', "we'd", "we'll", "we're", 'were', 'weren', "weren't", "we've", 'what', 'whe
n', 'where', 'which', 'while', 'who', 'whom', 'why', 'will', 'with', 'won', "wo
n't", 'wouldn', "wouldn't", 'y', 'you', "you'd", "you'll", 'your', "you're", 'you
rs', 'yourself', 'yourselves', "you've"]
[nltk_data] Downloading package stopwords to
[nltk_data]     C:\Users\Adnan\nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
```

In [49]:
```
token = word_tokenize(sent)
cleaned_token = []
for word in token:
 if word not in stop_words:
    cleaned_token.append(word)

print("This is the unclean version : ",token)
print("This is the cleaned version : ",cleaned_token)
```

```
This is the unclean version :  ['Sachin', 'is', 'considered', 'to', 'be', 'one',
'of', 'the', 'greatest', 'cricket', 'players', '.', 'Virat', 'is', 'the', 'captai
n', 'of', 'the', 'Indian', 'cricket', 'team']
This is the cleaned version :  ['Sachin', 'considered', 'one', 'greatest', 'crick
et', 'players', '.', 'Virat', 'captain', 'Indian', 'cricket', 'team']
```

In [53]:
```
words = [cleaned_token.lower() for cleaned_token in cleaned_token if cleaned_tok
print(words)
```

```
['sachin', 'considered', 'one', 'greatest', 'cricket', 'players', 'virat', 'capta
in', 'indian', 'cricket', 'team']
```

In [57]:
```
stemmer = PorterStemmer()
port_stemmer_output = [stemmer.stem(words) for words in words]
print(port_stemmer_output)
```

```
['sachin', 'consid', 'one', 'greatest', 'cricket', 'player', 'virat', 'captain',
'indian', 'cricket', 'team']
```

In [59]:
```
nltk.download('wordnet')
lemmatizer = WordNetLemmatizer()
lemmatizer_output = [lemmatizer.lemmatize(words) for words in words]
print(lemmatizer_output)
```

```
['sachin', 'considered', 'one', 'greatest', 'cricket', 'player', 'virat', 'captai
n', 'indian', 'cricket', 'team']
```

```
[nltk_data] Downloading package wordnet to C:\Users\Adnan\nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
```

In [61]:
```
nltk.download('averaged_perceptron_tagger')
token = word_tokenize(sent)
cleaned_token = []
for word in token:
 if word not in stop_words:
    cleaned_token.append(word)
tagged = pos_tag(cleaned_token)
print(tagged)
```

```
[('Sachin', 'NNP'), ('considered', 'VBD'), ('one', 'CD'), ('greatest', 'JJS'),
('cricket', 'NN'), ('players', 'NNS'), ('.', '.'), ('Virat', 'NNP'), ('captain',
'NN'), ('Indian', 'JJ'), ('cricket', 'NN'), ('team', 'NN')]
```

```
[nltk_data] Downloading package averaged_perceptron_tagger to
[nltk_data]     C:\Users\Adnan\nltk_data...
[nltk_data]   Package averaged_perceptron_tagger is already up-to-
[nltk_data]       date!
```

In [63]:
```
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import cosine_similarity
import pandas as pd
```

In [65]:
```
docs = [ "Sachin is considered to be one of the greatest cricket players",
 "Federer is considered one of the greatest tennis players",
 "Nadal is considered one of the greatest tennis players",
 "Virat is the captain of the Indian cricket team"]
```

In [67]:
```
vectorizer = TfidfVectorizer(analyzer = "word", norm = None , use_idf = True , s
Mat = vectorizer.fit(docs)
print(Mat.vocabulary_)
```

```
{'sachin': 12, 'is': 7, 'considered': 2, 'to': 16, 'be': 0, 'one': 10, 'of': 9,
'the': 15, 'greatest': 5, 'cricket': 3, 'players': 11, 'federer': 4, 'tennis': 1
4, 'nadal': 8, 'virat': 17, 'captain': 1, 'indian': 6, 'team': 13}
```

In [69]:
```
tfidfMat = vectorizer.fit_transform(docs)
```

In [71]:
```
features_names = vectorizer.get_feature_names_out()
print(features_names)
```

```
['be' 'captain' 'considered' 'cricket' 'federer' 'greatest' 'indian' 'is'
 'nadal' 'of' 'one' 'players' 'sachin' 'team' 'tennis' 'the' 'to' 'virat']
```

In [73]:
```
dense = tfidfMat.todense()
denselist = dense.tolist()
df = pd.DataFrame(denselist , columns = features_names)
```

In [75]:
```
df
```

Out[75]:

	be	captain	considered	cricket	federer	greatest	indian	is	nadal
0	1.916291	0.000000	1.223144	1.510826	0.000000	1.223144	0.000000	1.0	0.000000
1	0.000000	0.000000	1.223144	0.000000	1.916291	1.223144	0.000000	1.0	0.000000
2	0.000000	0.000000	1.223144	0.000000	0.000000	1.223144	0.000000	1.0	1.916291
3	0.000000	1.916291	0.000000	1.510826	0.000000	0.000000	1.916291	1.0	0.000000

In [79]:
```
features_names = sorted(vectorizer.get_feature_names_out())
```

In [81]:
```
docList = ['Doc 1','Doc 2','Doc 3','Doc 4']
skDocsIfIdfdf = pd.DataFrame(tfidfMat.todense(),index = sorted(docList), columns
print(skDocsIfIdfdf)
```

```
            be   captain  considered   cricket   federer  greatest    indian  \
Doc 1  1.916291  0.000000    1.223144  1.510826  0.000000  1.223144  0.000000
Doc 2  0.000000  0.000000    1.223144  0.000000  1.916291  1.223144  0.000000
Doc 3  0.000000  0.000000    1.223144  0.000000  0.000000  1.223144  0.000000
Doc 4  0.000000  1.916291    0.000000  1.510826  0.000000  0.000000  1.916291

        is     nadal   of       one   players    sachin      team    tennis  \
Doc 1  1.0  0.000000  1.0  1.223144  1.223144  1.916291  0.000000  0.000000
Doc 2  1.0  0.000000  1.0  1.223144  1.223144  0.000000  0.000000  1.510826
Doc 3  1.0  1.916291  1.0  1.223144  1.223144  0.000000  0.000000  1.510826
Doc 4  1.0  0.000000  1.0  0.000000  0.000000  0.000000  1.916291  0.000000

       the        to     virat
Doc 1  1.0  1.916291  0.000000
Doc 2  1.0  0.000000  0.000000
Doc 3  1.0  0.000000  0.000000
Doc 4  2.0  0.000000  1.916291
```

In [83]:
```
csim = cosine_similarity(tfidfMat,tfidfMat)
```

In [85]:
```
csimDf = pd.DataFrame(csim,index=sorted(docList),columns=sorted(docList))
```

In [87]:
```
print(csimDf)
```

```
          Doc 1     Doc 2     Doc 3     Doc 4
Doc 1  1.000000  0.492416  0.492416  0.277687
Doc 2  0.492416  1.000000  0.754190  0.215926
Doc 3  0.492416  0.754190  1.000000  0.215926
Doc 4  0.277687  0.215926  0.215926  1.000000
```

In [5]:
```
#!pip install seaborn
import seaborn as sns
import pandas as pd

# Load the titanic dataset
titanic = sns.load_dataset("titanic")

# Verify it worked with a simple plot
titanic.head(1)
```

Out[5]:

	survived	pclass	sex	age	sibsp	parch	fare	embarked	class	who	adult_male
0	0	3	male	22.0	1	0	7.25	S	Third	man	True

In [7]:
```
titanic.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
```

In [9]:
```
titanic.describe()
```

Out[9]:

	survived	pclass	age	sibsp	parch	fare
count	891.000000	891.000000	714.000000	891.000000	891.000000	891.000000
mean	0.383838	2.308642	29.699118	0.523008	0.381594	32.204208
std	0.486592	0.836071	14.526497	1.102743	0.806057	49.693429
min	0.000000	1.000000	0.420000	0.000000	0.000000	0.000000
25%	0.000000	2.000000	20.125000	0.000000	0.000000	7.910400
50%	0.000000	3.000000	28.000000	0.000000	0.000000	14.454200
75%	1.000000	3.000000	38.000000	1.000000	0.000000	31.000000
max	1.000000	3.000000	80.000000	8.000000	6.000000	512.329200

In [11]:
```
df=titanic.drop(['pclass','embarked','deck','embark_town'],axis=1)
df.head(2)
```

Out[11]:

	survived	sex	age	sibsp	parch	fare	class	who	adult_male	alive	alon
0	0	male	22.0	1	0	7.2500	Third	man	True	no	Fals
1	1	female	38.0	1	0	71.2833	First	woman	False	yes	Fals

In [15]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 11 columns):
 #   Column      Non-Null Count  Dtype   
---  ------      --------------  -----   
 0   survived    891 non-null    int64   
 1   sex         891 non-null    object  
 2   age         714 non-null    float64 
 3   sibsp       891 non-null    int64   
 4   parch       891 non-null    int64   
 5   fare        891 non-null    float64 
 6   class       891 non-null    category
 7   who         891 non-null    object  
 8   adult_male  891 non-null    bool    
 9   alive       891 non-null    object  
 10  alone       891 non-null    bool    
dtypes: bool(2), category(1), float64(2), int64(3), object(3)
memory usage: 58.6+ KB
```

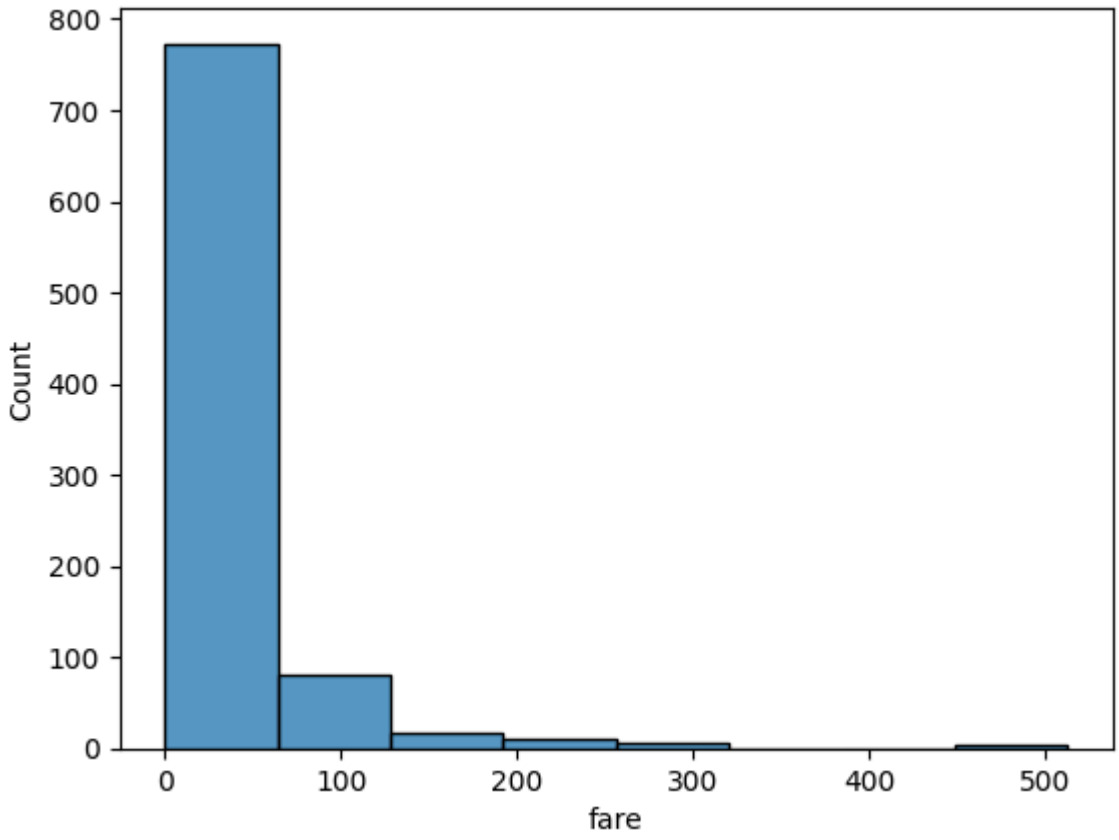
In [17]:
```
df.isnull().sum()
```

Out[17]:
```
survived        0
sex             0
age           177
sibsp           0
parch           0
fare            0
class           0
who             0
adult_male      0
alive           0
alone           0
dtype: int64
```

In [21]:
```
numeric_df = df.select_dtypes(include=['float64', 'int64'])
correlation = numeric_df.corr(method='pearson')
```

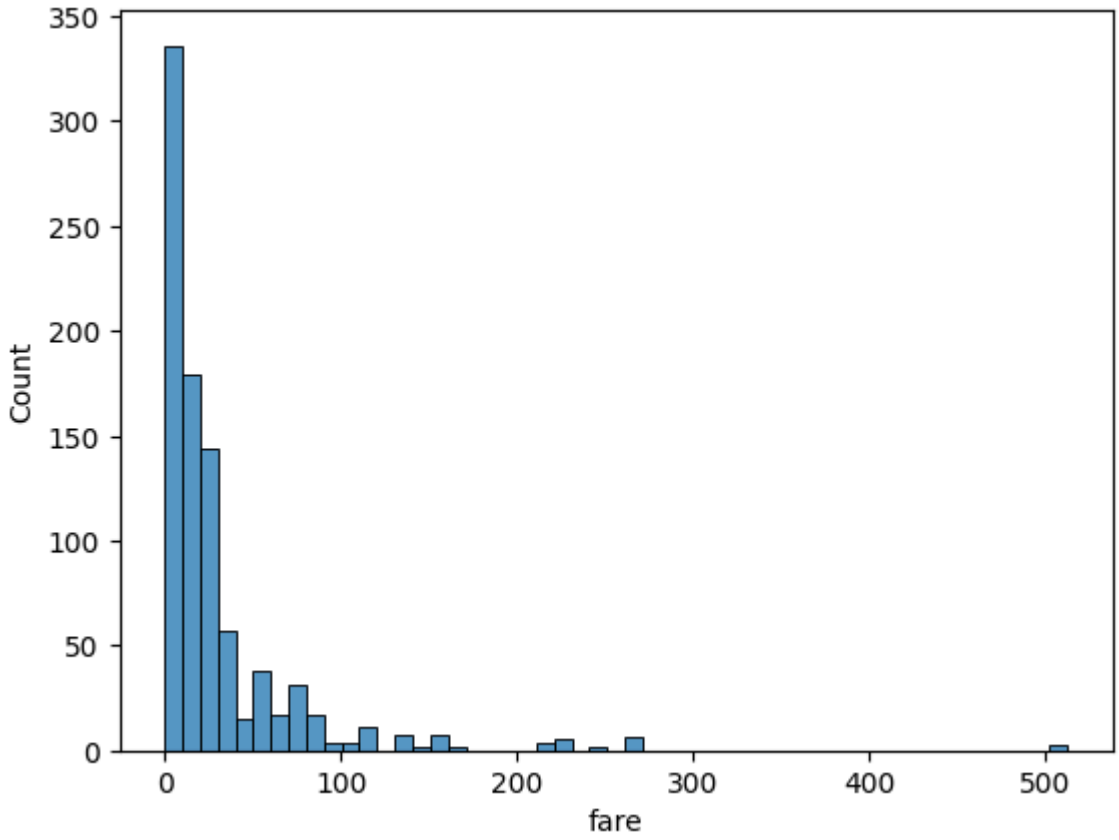
In [23]:
```
sns.histplot(data=titanic,x="fare",bins=8)
```

Out[23]:
```
<Axes: xlabel='fare', ylabel='Count'>
```



In [30]:
```
sns.histplot(data=titanic,x="fare",bins=20,binwidth=10)
```

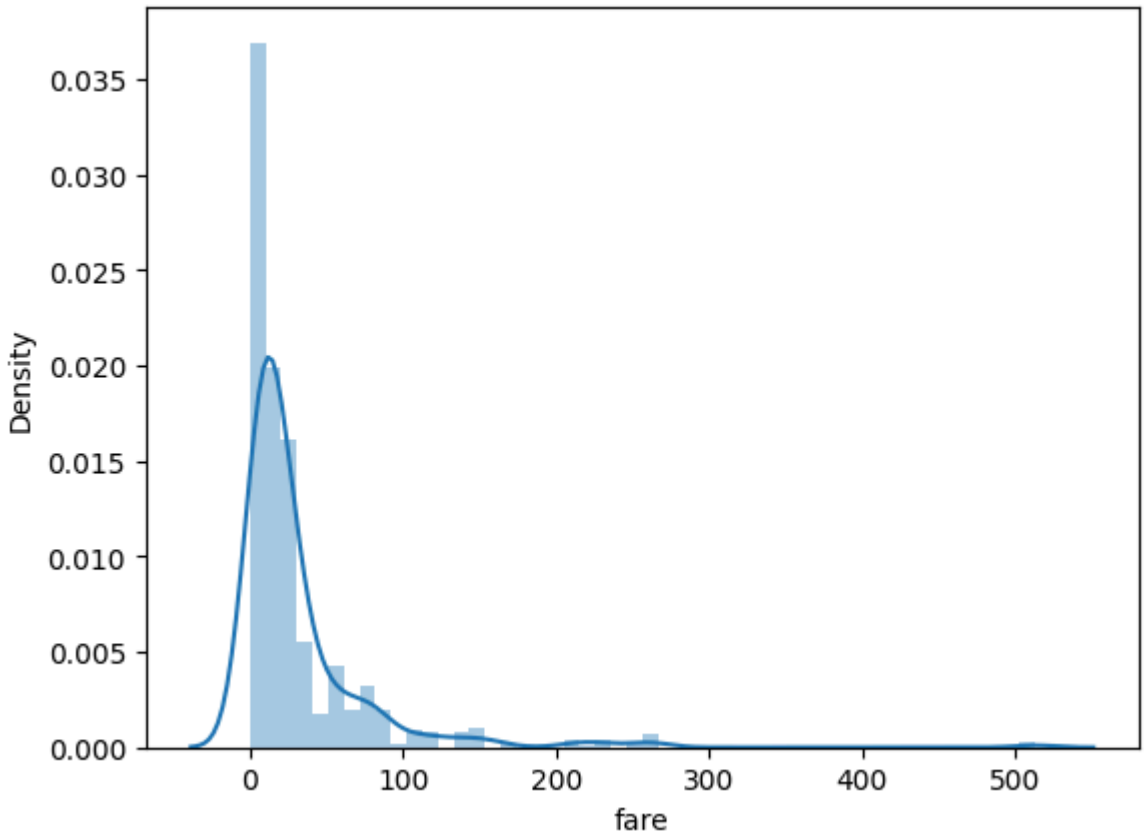
Out[30]:
```
<Axes: xlabel='fare', ylabel='Count'>
```



In [34]:
```
sns.distplot(df['fare'])
```

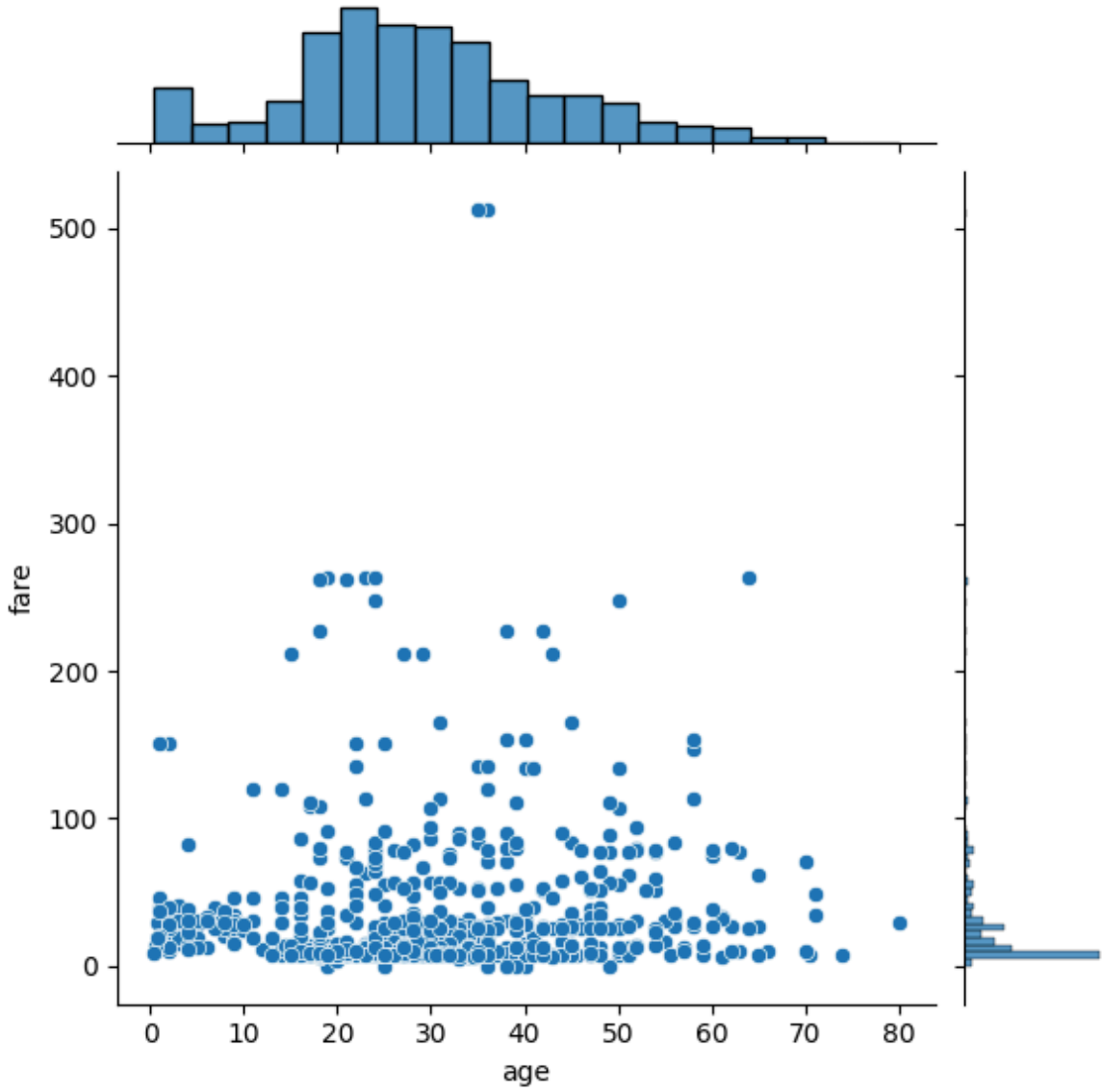
```
C:\Users\Adnan\AppData\Local\Temp\ipykernel_23152\1195996103.py:1: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(df['fare'])
```

Out[34]:
```
<Axes: xlabel='fare', ylabel='Density'>
```



In [44]:
```
sns.jointplot(x='age', y='fare', data=df)
```

Out[44]:
```
<seaborn.axisgrid.JointGrid at 0x1bf37616990>
```


500
400
300
200
100
0
fare
0
10
20
30
40
50
60
70
80
age

In [4]:
```
import seaborn as sns
import pandas as pd

# Load the titanic dataset
titanic = sns.load_dataset("titanic")

# Verify it worked with a simple plot
titanic.head(1)
```

Out[4]:

	survived	pclass	sex	age	sibsp	parch	fare	embarked	class	who	adult_male
0	0	3	male	22.0	1	0	7.25	S	Third	man	True

In [6]:
```
titanic.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
```

In [8]:
```
titanic.describe()
```

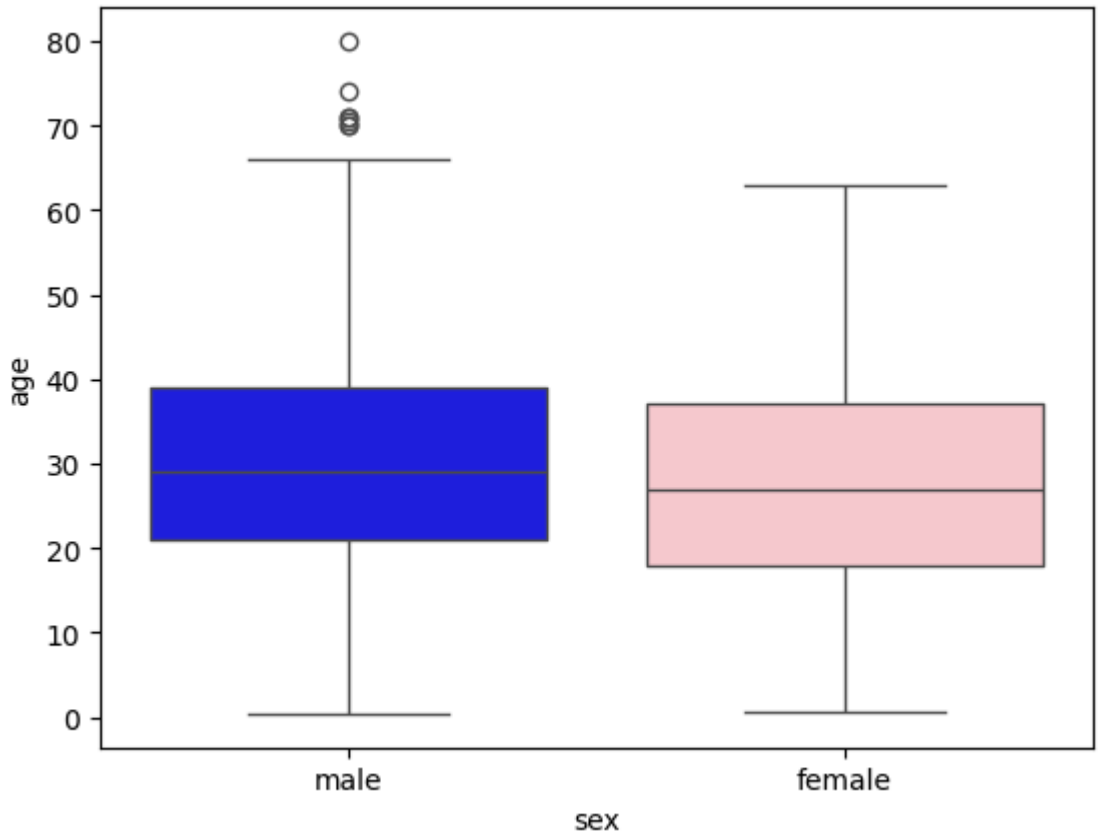
Out[8]:

	survived	pclass	age	sibsp	parch	fare
count	891.000000	891.000000	714.000000	891.000000	891.000000	891.000000
mean	0.383838	2.308642	29.699118	0.523008	0.381594	32.204208
std	0.486592	0.836071	14.526497	1.102743	0.806057	49.693429
min	0.000000	1.000000	0.420000	0.000000	0.000000	0.000000
25%	0.000000	2.000000	20.125000	0.000000	0.000000	7.910400
50%	0.000000	3.000000	28.000000	0.000000	0.000000	14.454200
75%	1.000000	3.000000	38.000000	1.000000	0.000000	31.000000
max	1.000000	3.000000	80.000000	8.000000	6.000000	512.329200

In [16]:
```
sns.boxplot(x="sex", y="age", data=titanic, palette={"male": "blue", "female": "
```

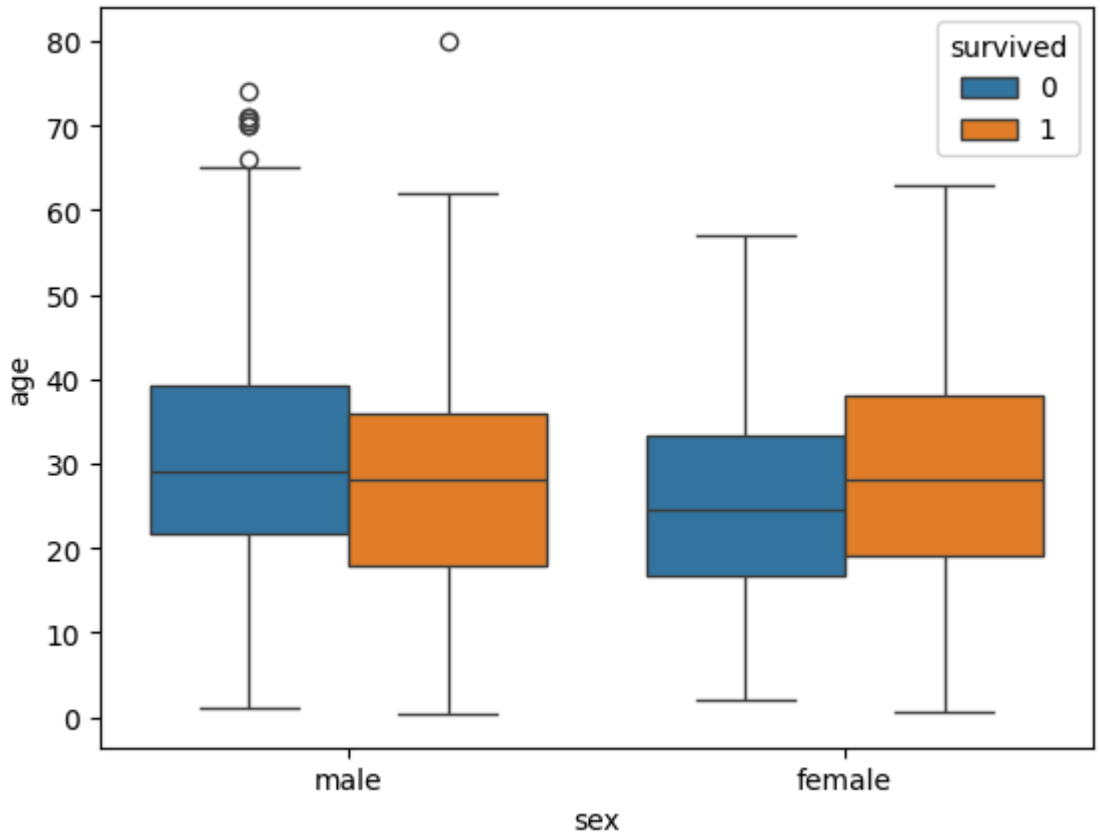
```
C:\Users\Adnan\AppData\Local\Temp\ipykernel_15576\3551433936.py:1: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v
0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same effe
ct.

  sns.boxplot(x="sex", y="age", data=titanic, palette={"male": "blue", "female":
"pink"})
```

Out[16]: <Axes: xlabel='sex', ylabel='age'>



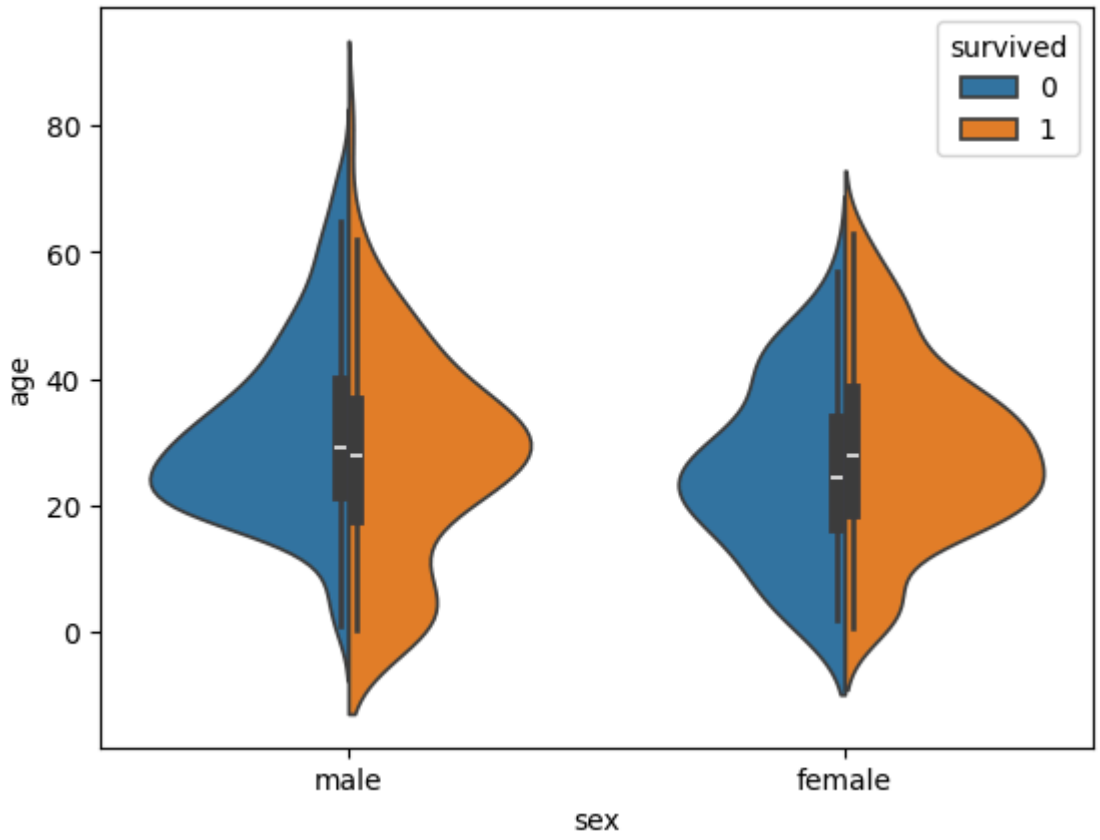
In [18]:
```
sns.boxplot(x="sex",y="age",data=titanic,hue="survived")
```

Out[18]: <Axes: xlabel='sex', ylabel='age'>



In [22]:
```
sns.violinplot(x='sex', y='age', data=titanic, hue='survived', split=True)
```

Out[22]:
```
<Axes: xlabel='sex', ylabel='age'>
```



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

In [3]:
```
df = pd.read_csv("https://raw.githubusercontent.com/shrikant-temburwar/Iris-Data
df.head(3)
```

Out[3]:

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
0	1	5.1	3.5	1.4	0.2	Iris-setosa
1	2	4.9	3.0	1.4	0.2	Iris-setosa
2	3	4.7	3.2	1.3	0.2	Iris-setosa

In [5]:
```
df.describe()
```

Out[5]:

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm
count	150.000000	150.000000	150.000000	150.000000	150.000000
mean	75.500000	5.843333	3.054000	3.758667	1.198667
std	43.445368	0.828066	0.433594	1.764420	0.763161
min	1.000000	4.300000	2.000000	1.000000	0.100000
25%	38.250000	5.100000	2.800000	1.600000	0.300000
50%	75.500000	5.800000	3.000000	4.350000	1.300000
75%	112.750000	6.400000	3.300000	5.100000	1.800000
max	150.000000	7.900000	4.400000	6.900000	2.500000

In [7]:
```
df.shape
```

Out[7]:
```
(150, 6)
```

In [9]:
```
df["Species"].unique()
```

Out[9]:
```
array(['Iris-setosa', 'Iris-versicolor', 'Iris-virginica'], dtype=object)
```

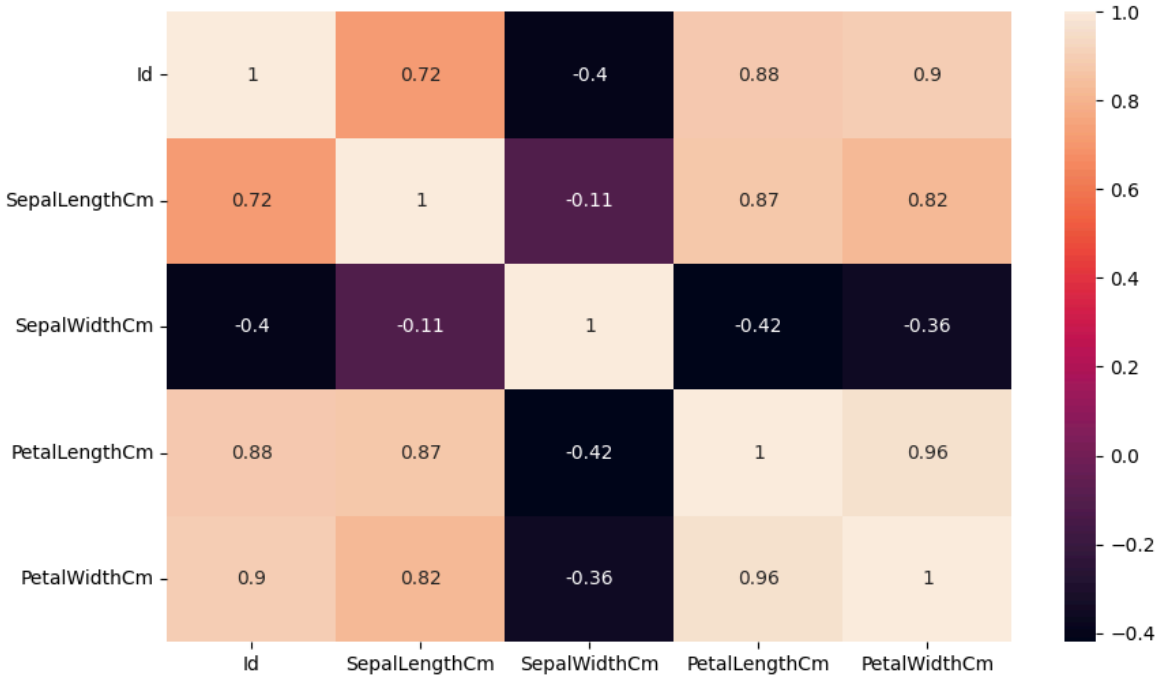
In [11]:
```
df.groupby("Species").size()
```

Out[11]:
```
Species
Iris-setosa        50
Iris-versicolor    50
Iris-virginica     50
dtype: int64
```

In [15]:
```
numeric_df = df.select_dtypes(include=['float64', 'int64'])

# Now calculate correlation only on numerical columns
corr = numeric_df.corr()

# Plot the heatmap
```

```
plt.subplots(figsize=(10,6))
sns.heatmap(corr, annot=True)
```

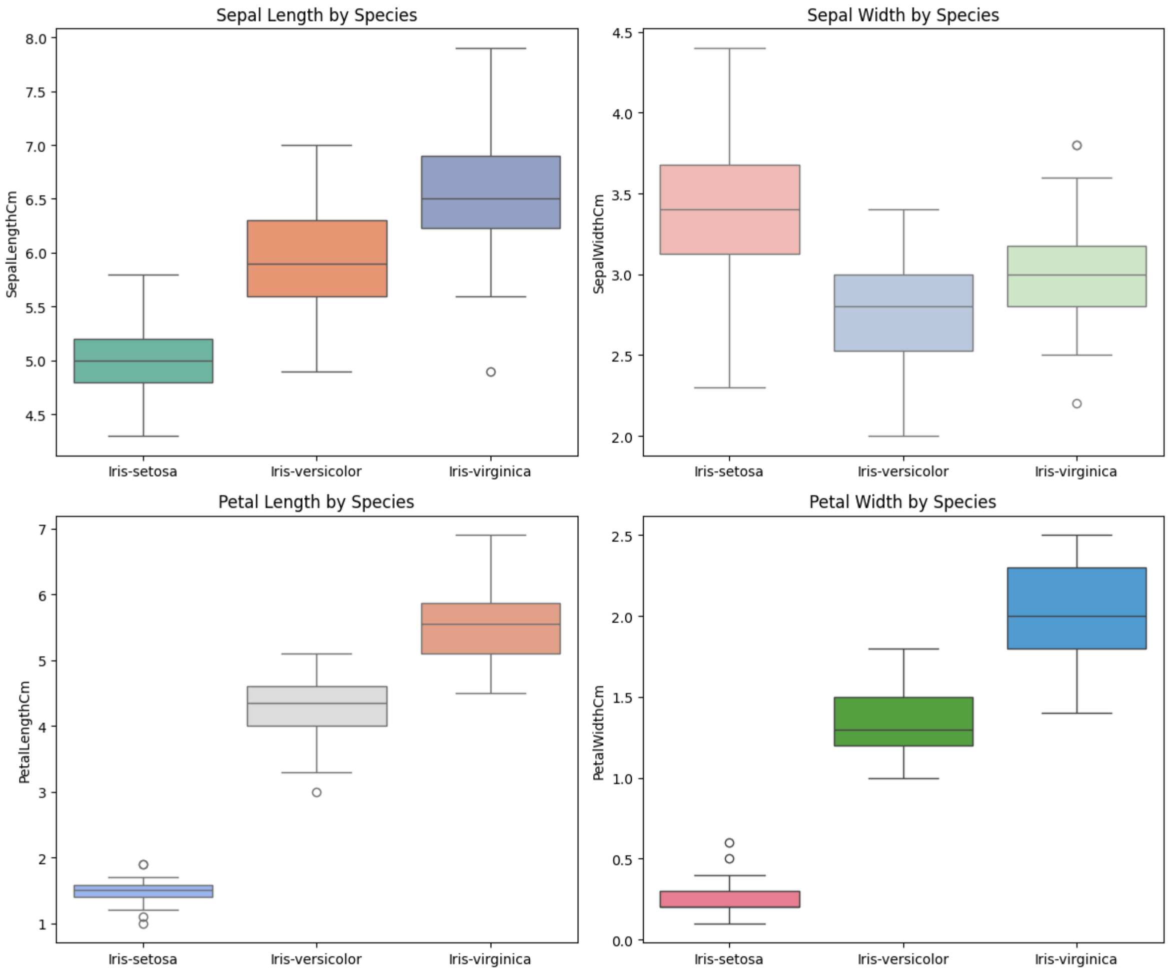
Out[15]: <Axes: >



In [21]:

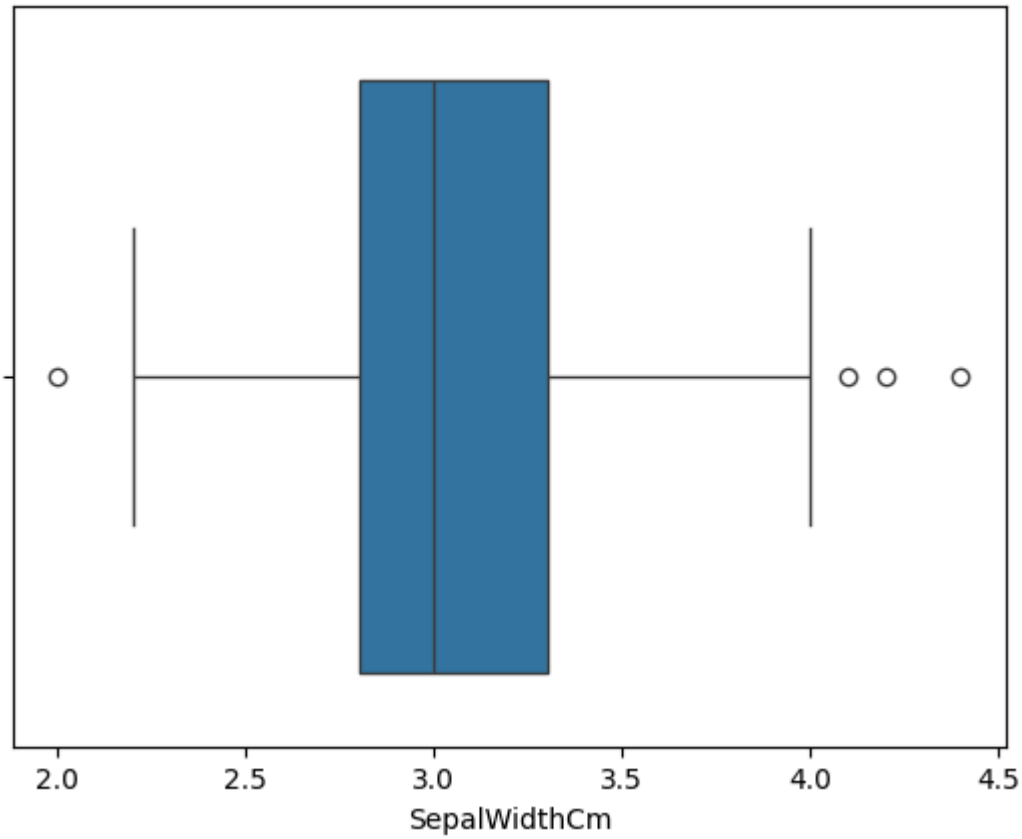
```
def graph(y, position, title, palette):
    plt.subplot(2, 2, position)
    sns.boxplot(x="Species", y=y, data=df, palette=palette)
    plt.title(title)
    plt.xlabel("")
    plt.ylabel(y)

# Features, titles, and custom palettes for variety
features = ['SepalLengthCm', 'SepalWidthCm', 'PetalLengthCm', 'PetalWidthCm']
titles = ['Sepal Length by Species', 'Sepal Width by Species',
          'Petal Length by Species', 'Petal Width by Species']
palettes = ['Set2', 'Pastel1', 'coolwarm', 'husl']

plt.figure(figsize=(12, 10))

for i, (feature, title, palette) in enumerate(zip(features, titles, palettes), 1
    graph(feature, i, title, palette)

plt.tight_layout()
plt.show()
```

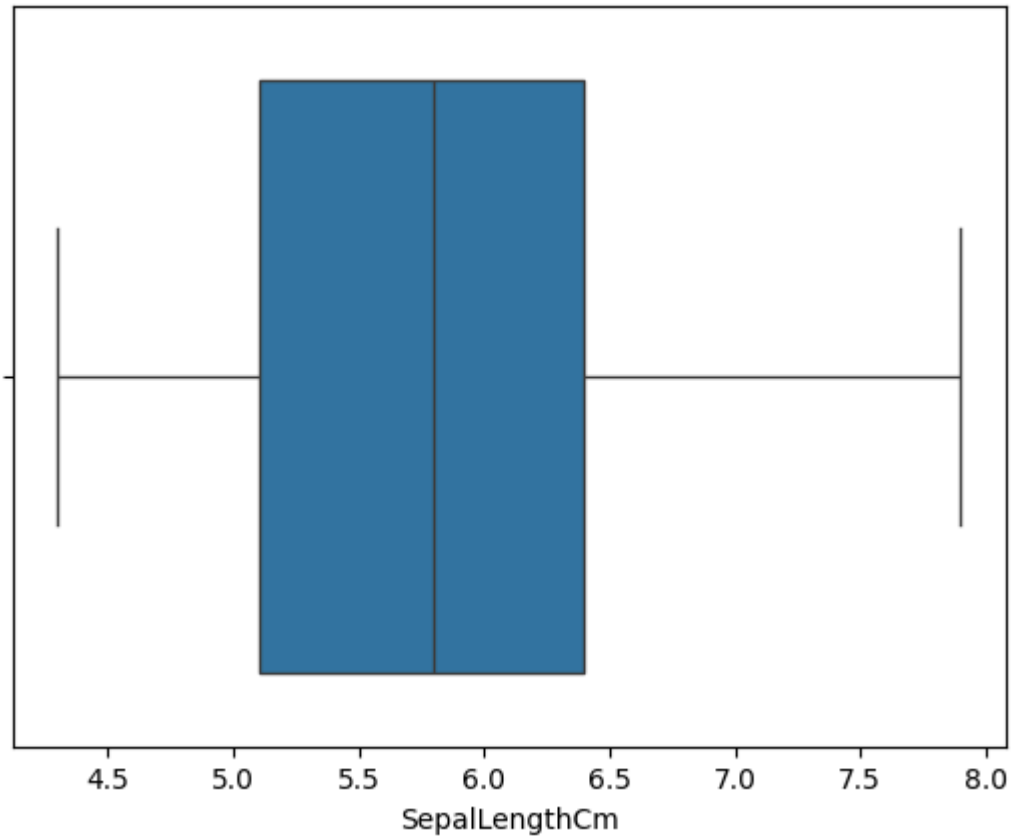
```
C:\Users\Adnan\AppData\Local\Temp\ipykernel_21992\2327543419.py:3: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v
0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same effe
ct.

  sns.boxplot(x="Species", y=y, data=df, palette=palette)
C:\Users\Adnan\AppData\Local\Temp\ipykernel_21992\2327543419.py:3: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v
0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same effe
ct.

  sns.boxplot(x="Species", y=y, data=df, palette=palette)
C:\Users\Adnan\AppData\Local\Temp\ipykernel_21992\2327543419.py:3: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v
0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same effe
ct.

  sns.boxplot(x="Species", y=y, data=df, palette=palette)
C:\Users\Adnan\AppData\Local\Temp\ipykernel_21992\2327543419.py:3: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v
0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same effe
ct.

  sns.boxplot(x="Species", y=y, data=df, palette=palette)
```

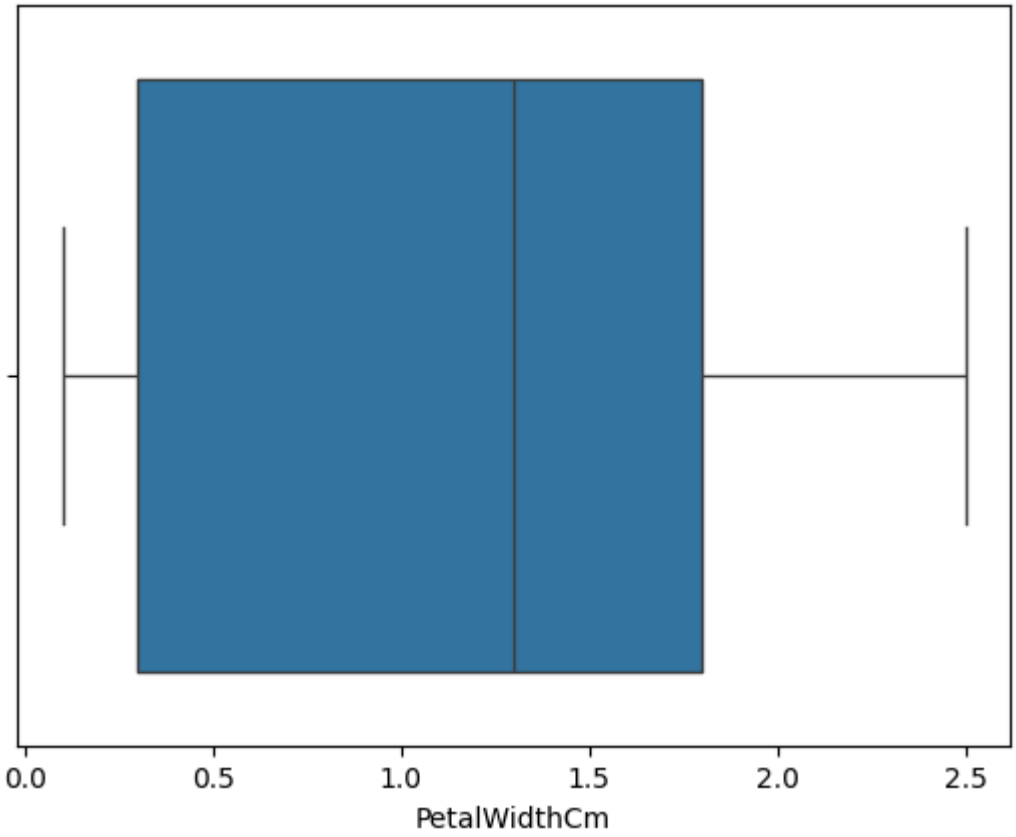


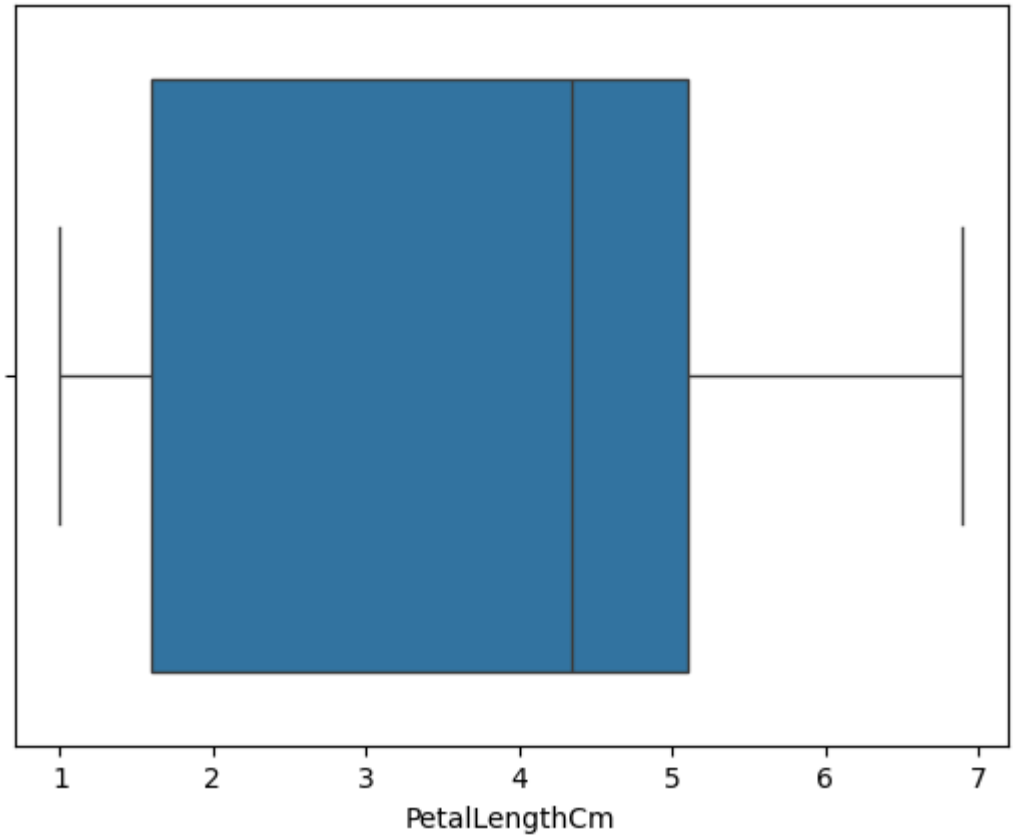
In [23]:
```python
sns.boxplot(x='SepalWidthCm', data=df)
plt.show()
```

```
sns.boxplot(x='SepalLengthCm', data=df)
plt.show()
sns.boxplot(x='PetalWidthCm', data=df)
plt.show()
sns.boxplot(x='PetalLengthCm', data=df)
plt.show()
```









In [ ]:

```
import java.io.IOException;
import java.util.StringTokenizer;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.Reducer;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
public class WordCount {
 public static class TokenizerMapper
 extends Mapper<Object, Text, Text, IntWritable>{
 private final static IntWritable one = new IntWritable(1);
 private Text word = new Text();
 /*
 The Mapper implementation, via the map method, processes one line at a time, as
provided by the specified TextInputFormat. It then splits the line into tokens separated by
whitespaces,
 via the StringTokenizer, and emits a key-value pair of < <word>, 1>.
 For the given sample input the first map emits:
 < Hello, 1>
 < World, 1>
 < Bye, 1>
 < World, 1>
 */
 public void map(Object key, Text value, Context context
```

```
) throws IOException, InterruptedException {
StringTokenizer itr = new StringTokenizer(value.toString());
while (itr.hasMoreTokens()) {
word.set(itr.nextToken());
context.write(word, one);
}
}
}
public static class IntSumReducer
extends Reducer<Text,IntWritable,Text,IntWritable> {
private IntWritable result = new IntWritable();
public void reduce(Text key, Iterable<IntWritable> values,
Context context
) throws IOException, InterruptedException {
int sum = 0;
for (IntWritable val : values) {
sum += val.get();
}
result.set(sum);
context.write(key, result);
}
}
public static void main(String[] args) throws Exception {
Configuration conf = new Configuration();
Job job = Job.getInstance(conf, "word count");
job.setJarByClass(WordCount.class);
job.setMapperClass(TokenizerMapper.class);
job.setCombinerClass(IntSumReducer.class);
```

```
 job.setReducerClass(IntSumReducer.class);
 job.setOutputKeyClass(Text.class);
 job.setOutputValueClass(IntWritable.class);
 FileInputFormat.addInputPath(job, new Path(args[0]));
 FileOutputFormat.setOutputPath(job, new Path(args[1]));
 System.exit(job.waitForCompletion(true) ? 0 : 1);
 }
}
```

Output:

Hello 1

Java 2

World 1

```
import java.io.IOException;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.FloatWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.Reducer;
import org.apache.hadoop.mapreduce.lib.input.TextInputFormat;
import org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;
public class WeatherDataAverage {
 public static class TokenizerMapper extends Mapper<Object, Text, Text, FloatWritable> {
 private Text category = new Text();
 private FloatWritable temperature = new FloatWritable();
 private FloatWritable windSpeed = new FloatWritable();
 private FloatWritable dewPoint = new FloatWritable();
 public void map(Object key, Text value, Context context) throws IOException,
InterruptedException {
 String[] cols = value.toString().split(" ");
 float temp = Float.parseFloat(cols[0]);
 float wind = Float.parseFloat(cols[1]);
 float dew = Float.parseFloat(cols[2]);
 category.set("Temperature");
 temperature.set(temp);
 context.write(category, temperature);
 category.set("WindSpeed");
 windSpeed.set(wind);
 context.write(category, windSpeed);
```

```
category.set("DewPoint");
dewPoint.set(dew);
context.write(category, dewPoint);
}
}
public static class FloatAverageReducer extends Reducer<Text, FloatWritable, Text,
FloatWritable> {
private FloatWritable result = new FloatWritable();
public void reduce(Text key, Iterable<FloatWritable> values, Context context)
throws IOException, InterruptedException {
float sum = 0;
int count = 0;
for (FloatWritable a : values){sum += a.get();count++;}
float avg = sum / count;
result.set(avg);
context.write(key, result);
}
}
public static void main(String[] args) throws Exception {
Configuration conf = new Configuration();
Job job = Job.getInstance(conf, "weather data average");
job.setJarByClass(WeatherDataAverage.class);
job.setMapperClass(TokenizerMapper.class);
job.setReducerClass(FloatAverageReducer.class);
job.setMapOutputKeyClass(Text.class);
job.setMapOutputValueClass(FloatWritable.class);
job.setOutputKeyClass(Text.class);
job.setOutputValueClass(FloatWritable.class);
```

```
 job.setInputFormatClass(TextInputFormat.class);
 job.setOutputFormatClass(TextOutputFormat.class);
 TextInputFormat.addInputPath(job, new Path(args[0]));
 TextOutputFormat.setOutputPath(job, new Path(args[1]));
 System.exit(job.waitForCompletion(true) ? 0 : 1);
 }
}
```

Output:

DewPoint 61.94638

Temperature 54.65593

WindSpeed 1007.7881

```
// This opens up an interactive shell
// Create a new input1.txt file
val input = sc.textFile("passage.txt")

// Create a new RDD by splitting the input RDD on the basis of space
val words = input.flatMap(x => x.split(" "))

// Create a new RDD by mapping each word to a tuple of (word, 1)
val counts = words.map(x => (x, 1))

// Create a new RDD by reducing the tuples by key
val reducedCounts = counts.reduceByKey((x, y) => x + y)

// Save the RDD to a file
reducedCounts.saveAsTextFile("output.txt")

// Print the contents of the file
reducedCounts.foreach(println)
```

Output:

(Hello,1)

(Shubham,1)