1.

```
import pandas as pd
import numpy as np
dataset = pd.read_csv("/content/BMW sales data (2010-2024) (1).csv")
dataset
```

Output:

	Model	Year	Region	Color	Fuel_Type	Transmission	Engine_Size_L	Mileage_KM	Price_USD	Sales_Volume	Sales_Classification
0	5 Series	2016	Asia	Red	Petrol	Manual	3.5	151748	98740	8300	High
1	i8	2013	North America	Red	Hybrid	Automatic	1.6	121671	79219	3428	Low
2	5 Series	2022	North America	Blue	Petrol	Automatic	4.5	10991	113265	6994	Low
3	X3	2024	Middle East	Blue	Petrol	Automatic	1.7	27255	60971	4047	Low
4	7 Series	2020	South America	Black	Diesel	Manual	2.1	122131	49898	3080	Low
...	...	...	...	...	...	...	...	...	...	...	...
49995	i3	2014	Asia	Red	Hybrid	Manual	4.6	151030	42932	8182	High
49996	i3	2023	Middle East	Silver	Electric	Manual	4.2	147396	48714	9816	High
49997	5 Series	2010	Middle East	Red	Petrol	Automatic	4.5	174939	46126	8280	High
49998	i3	2020	Asia	White	Electric	Automatic	3.8	3379	58566	9486	High
49999	X1	2020	North America	Blue	Diesel	Manual	3.3	171003	77492	1764	Low

50000 rows × 11 columns

```
dataset['Fuel_Type'].value_counts()
```

Output:

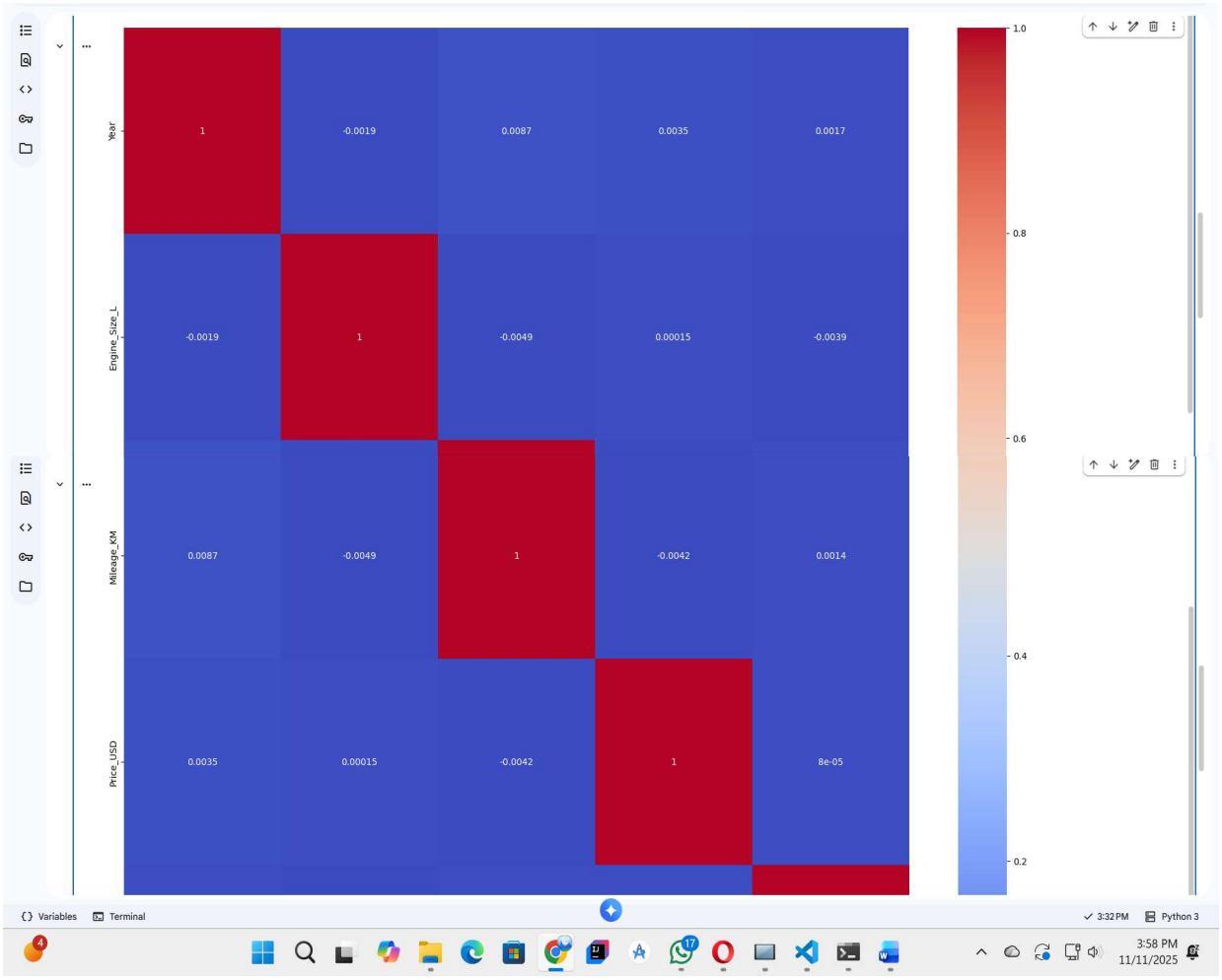
Fuel_Type	count
Hybrid	12716
Petrol	12550
Electric	12471
Diesel	12263

dtype: int64

```
import matplotlib
import matplotlib.pyplot as plt
import seaborn as sns
```

```
correlations = dataset.drop(columns=['Model', 'Region', 'Color', 'Fuel_Type', 'Transmission',
'Sales_Classification']).corr()
plt.figure(figsize = (20, 20))
sns.heatmap(correlations, annot=True, cmap='coolwarm')
```

Output:



2.

```
import pandas as pd
data = [10, 20, 30, 40]
s = pd.Series(data, index=['a', 'b', 'c', 'd'])
print(s)
```

Output:

```
a    10
b    20
c    30
d    40
dtype: int64
```

```
data = {
   'Name': ['Alice', 'Bob', 'Charlie', 'David'],
   'Age': [24, 27, 22, 32],
   'City': ['New York', 'Los Angeles', 'Chicago', 'Houston']
}
df = pd.DataFrame(data)
print(df)
```

Output:

```
      Name  Age         City
0    Alice   24     New York
1      Bob   27  Los Angeles
2  Charlie   22      Chicago
3    David   32      Houston
```

```
column = df['Age']
print(column)
```

Output:

```
0    24
1    27
2    22
3    32
Name: Age, dtype: int64
```

```
row = df.loc[1]
print(row)
```

Output:

```
Name              Bob
Age                27
City    Los Angeles
Name: 1, dtype: object
```

```
subset = df.loc[1:3, ['Name', 'City']]
print(subset)
```

Output:

```
      Name         City
1      Bob  Los Angeles
2  Charlie      Chicago
3    David      Houston
```

```
df.isnull()
```

Output:

	Name	Age	City
0	False	False	False
1	False	False	False
2	False	False	False
3	False	False	False

```
df.fillna(0, inplace=True)
df
```

Output:

	Name	Age	City
0	Alice	24	New York
1	Bob	27	Los Angeles
2	Charlie	22	Chicago
3	David	32	Houston

```
df.dropna(inplace=True)
grouped = df.groupby('City').agg({'Age': ['mean']})
print(grouped)
```

output:

```
             Age
            mean
City
Chicago     22.0
Houston     32.0
Los Angeles 27.0
New York    24.0
```

```
grouped = df.groupby('City').agg({'Age': ['mean', 'min', 'max']})
print(grouped)
```

Output:

	Age		
	mean	min	max
City			
Chicago	22.0	22	22
Houston	32.0	32	32
Los Angeles	27.0	27	27
New York	24.0	24	24

3.1

```
import numpy as np
arr = np.array([1, 2, 3, 4, 5])
print(arr)
```

Output:

```
[1 2 3 4 5]
```

```
matrix = np.array([[1, 2, 3, 4], [4, 5, 6, 5], [7, 8, 9, 5]])
print(matrix)
```

Output:

```
[[1 2 3 4]
 [4 5 6 5]
 [7 8 9 5]]
```

```
zeros = np.zeros((3, 3))
ones = np.ones((2, 2))
eye = np.eye(3)
print(zeros)
print(ones)
print(eye)
```

Output:

```
[[0. 0. 0.]
 [0. 0. 0.]
 [0. 0. 0.]]
[[1. 1.]
 [1. 1.]]
[[1. 0. 0.]
 [0. 1. 0.]
 [0. 0. 1.]]
```

```
ones = np.ones((2, 2))
print(ones)
```

Output:

```
[[1. 1.]
 [1. 1.]]
```

```
eye = np.eye(3)
print(eye)
```

Output:

```
[[1. 0. 0.]
 [0. 1. 0.]
 [0. 0. 1.]]
```

```
zeros = np.zeros((3, 3))
print(zeros)
```

Output:

```
[[0. 0. 0.]
 [0. 0. 0.]
 [0. 0. 0.]]
```

```
arr = np.arange(0, 50, 5)
print(arr)
```

Output:

```
[ 0  5 10 15 20 25 30 35 40 45]
```

```
arr = np.linspace(0, 50, 5)
print(arr)
```

Output:

```
[ 0.  12.5 25.  37.5 50. ]
```

```
arr = np.array([1, 2, 3, 4])
print(arr + 2)
print(arr * 3)
```

Output:

```
[3 4 5 6]
[ 3  6  9 12]
```

```
matrix1 = np.array([[1, 2], [3, 4]])
matrix2 = np.array([[5, 6], [7, 8]])
product = np.dot(matrix1, matrix2)
print(product)
```

Output:

```
[[19 22]
 [43 50]]
```

```
arr = np.array([10, 20, 30, 40, 50])
print(arr[1])
print(arr[0:3])
```

Output:

```
20
[10 20 30]
```

```
matrix = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
print(matrix[1, 2])
print(matrix[2,:])
```

Output:

```
6
[7 8 9]
```

```
arr = np.array([1, 2, 3, 4, 5])
print(np.mean(arr))
print(np.sum(arr))
print(np.max(arr))
print(np.min(arr))
print(np.std(arr))
```

Output:

```
3.0
15
5
1
1.4142135623730951
```

```
arr = np.arange(1, 10)
reshaped = arr.reshape((3, 3))  # Reshape to 3x3 matrix
print(reshaped)
```

Output:

```
[[1 2 3]
 [4 5 6]
 [7 8 9]]
```

```
transposed = reshaped.T
print(transposed)
```

Output:

```
[[1 4 7]
 [2 5 8]
 [3 6 9]]
```

```
arr = np.array([1, 2, 3])
matrix = np.array([[1], [2], [3]])
result = arr + matrix
print(result)
```

Output:

```
[[2 3 4]
 [3 4 5]
 [4 5 6]]
```

```
random_arr = np.random.rand(3, 3)
print(random_arr)
random_int_arr = np.random.randint(0, 10, (3, 3))
print(random_int_arr)
```

Output:

```
[[0.04354263 0.90765938 0.67974755]
 [0.90230963 0.04194419 0.19479115]
 [0.1013152  0.09291469 0.69581315]]
[[0 4 2]
 [8 9 0]
 [3 7 5]]
```

3.2

```
from sklearn.datasets import load_iris
data = load_iris()
X, y = data.data, data.target
print(X.shape, y.shape)
```

Output:

```
(150, 4) (150,)
```

data

Output:

```
{'data': array([[5.1, 3.5, 1.4, 0.2],
        [4.9, 3. , 1.4, 0.2],
        [4.7, 3.2, 1.3, 0.2],
        [4.6, 3.1, 1.5, 0.2],
        [5. , 3.6, 1.4, 0.2],
        [5.4, 3.9, 1.7, 0.4],
        [4.6, 3.4, 1.4, 0.3],
        [5. , 3.4, 1.5, 0.2],
        [4.4, 2.9, 1.4, 0.2],
        [4.9, 3.1, 1.5, 0.1],
        [5.4, 3.7, 1.5, 0.2],
        [4.8, 3.4, 1.6, 0.2],
        [4.8, 3. , 1.4, 0.1],
        [4.3, 3. , 1.1, 0.1],
        [5.8, 4. , 1.2, 0.2],
        [5.7, 4.4, 1.5, 0.4],
        [5.4, 3.9, 1.3, 0.4],
        [5.1, 3.5, 1.4, 0.3],
        [5.7, 3.8, 1.7, 0.3],
        [5.1, 3.8, 1.5, 0.3],
        [5.4, 3.4, 1.7, 0.2],
        [5.1, 3.7, 1.5, 0.4],
```

'target': array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,

0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,

0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,

1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,

```
    1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2,
    2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
    2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2])
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
print(X_train.shape, X_test.shape)
```

Output:

```
(105, 4) (45, 4)
```

```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

```
from sklearn.linear_model import LogisticRegression
clf = LogisticRegression()
clf.fit(X_train, y_train)
y_pred = clf.predict(X_test)
```

```
from sklearn.ensemble import RandomForestClassifier
clf = RandomForestClassifier(n_estimators=100)
clf.fit(X_train, y_train)
y_pred = clf.predict(X_test)
```

```
from sklearn.svm import SVC
clf = SVC(kernel='linear')
clf.fit(X_train, y_train)
y_pred = clf.predict(X_test)
```

```
from sklearn.neighbors import KNeighborsClassifier
clf = KNeighborsClassifier(n_neighbors=3)
clf.fit(X_train, y_train)
y_pred = clf.predict(X_test)

from sklearn.tree import DecisionTreeClassifier
clf = DecisionTreeClassifier()
clf.fit(X_train, y_train)
y_pred = clf.predict(X_test)


from sklearn.cluster import KMeans
kmeans = KMeans(n_clusters=3)
kmeans.fit(X)
print(kmeans.labels_)
```

Output:

```
[1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 0 0 2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 2 0 2 2 2 2 0 2 2 2 2
 2 2 0 0 2 2 2 2 0 2 0 2 0 2 2 0 0 2 2 2 2 0 2 2 2 2 0 2 2 2 0 2 2 2 0 2 2
 2 0]
```

```
from sklearn.metrics import accuracy_score, confusion_matrix
accuracy = accuracy_score(y_test, y_pred)
cm = confusion_matrix(y_test, y_pred)
print(f"Accuracy: {accuracy}")
print("Confusion Matrix:\n", cm)
```

Output:

```
Accuracy: 1.0
Confusion Matrix:
 [[19  0  0]
 [ 0 13  0]
 [ 0  0 13]]
```

```
from sklearn.decomposition import PCA
pca = PCA(n_components=2)
X_reduced = pca.fit_transform(X)
print(X_reduced.shape)  # Output: (150, 2)
```

Output:

```
(150, 2)
```

```
from sklearn.pipeline import Pipeline
pipeline = Pipeline([
   ('scaler', StandardScaler()),
   ('logreg', LogisticRegression())
])
pipeline.fit(X_train, y_train)
y_pred = pipeline.predict(X_test)
from sklearn.feature_selection import SelectKBest, chi2
selector = SelectKBest(chi2, k=2)
X_new = selector.fit_transform(X, y)
print(X_new.shape)  # Output: (150, 2)
```

Output:

```
(150, 2)
```

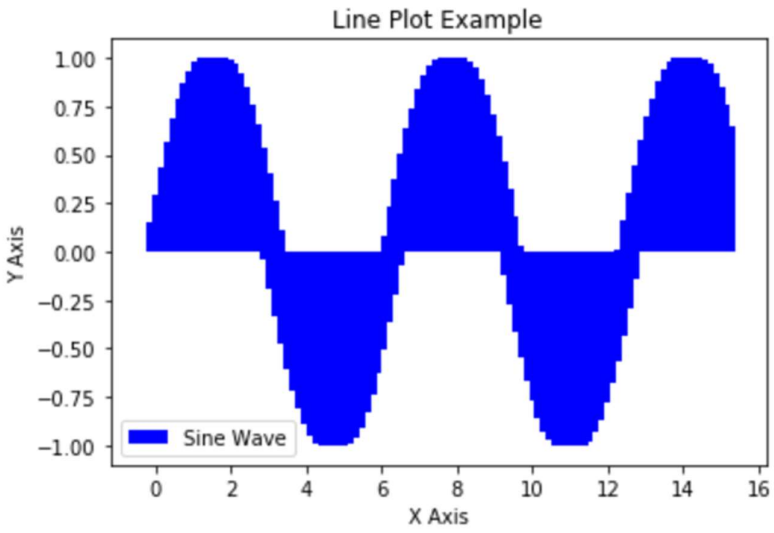
```
from sklearn.manifold import TSNE
tsne = TSNE(n_components=2)
X_tsne = tsne.fit_transform(X)
```

```
import joblib
joblib.dump(clf, 'model.pkl')
clf = joblib.load('model.pkl')
```

4.

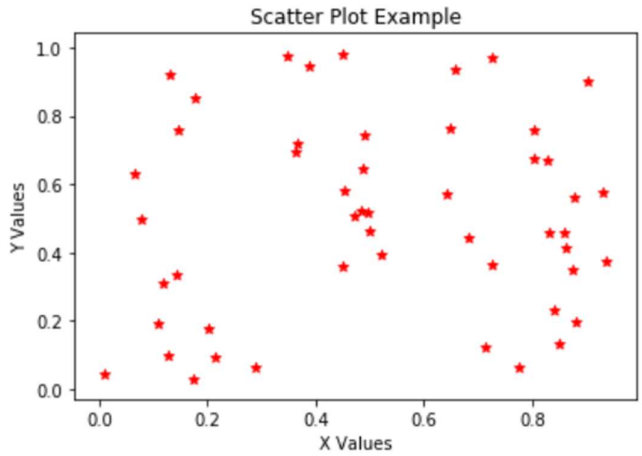
```
import matplotlib.pyplot as plt
import numpy as np
x = np.linspace(0, 15, 100)
y = np.sin(x)
plt.plot(x, y, label='Sine Wave',  color='blue', linestyle='--')
plt.xlabel('X Axis')
plt.ylabel('Y Axis')
plt.title('Line Plot Example')
plt.legend()
plt.show()
```

Output:




```
import matplotlib.pyplot as plt
import numpy as np
x = np.random.rand(50)
y = np.random.rand(50)
plt.scatter(x, y, color='red', marker='*')
plt.xlabel('X Values')
plt.ylabel('Y Values')
plt.title('Scatter Plot Example')
plt.show()
```

Output:

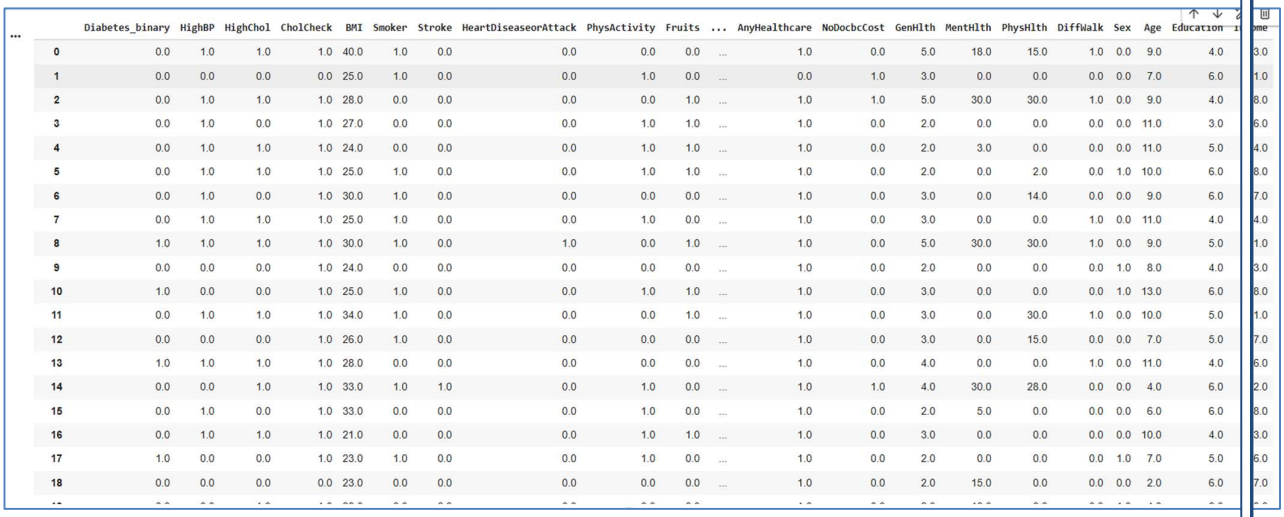



```
import matplotlib.pyplot as plt
import numpy as np
plt.hist(dataset['BMI'])
```

```
plt.xlabel('Value')
plt.ylabel('Frequency')
plt.title('Histogram Example')
plt.show()

import pandas as pd
dataset = pd.read_csv("C:/Users/souro/diabetes_binary_health_indicators_BRFSS2015.csv")
dataset
```

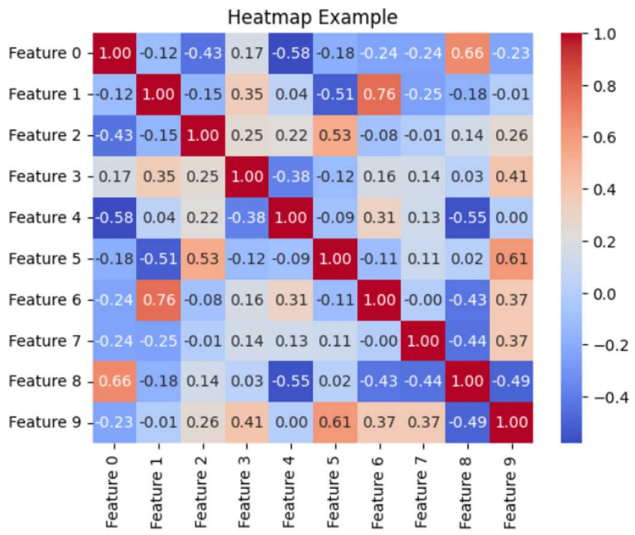
Output:



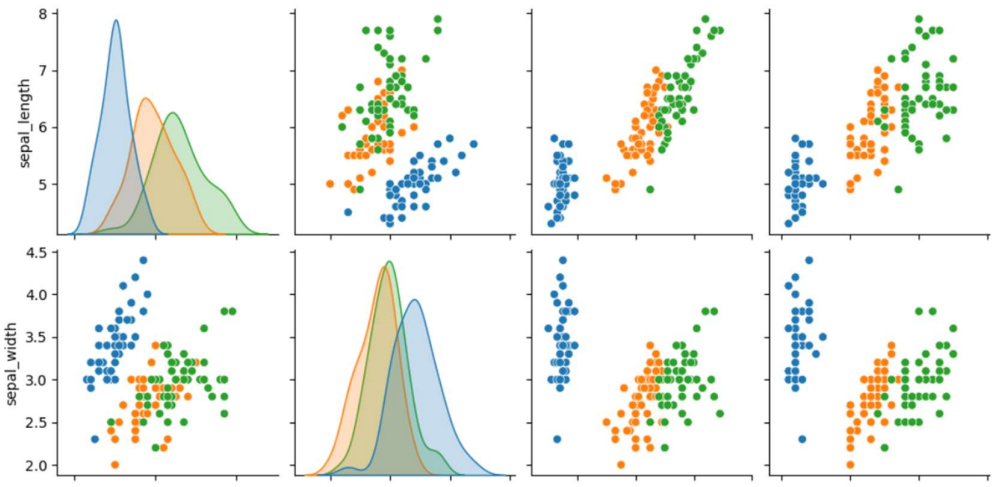
	Diabetes_binary	HighBP	HighChol	CholCheck	BMI	Smoker	Stroke	HeartDiseaseorAttack	PhysActivity	Fruits	...	AnyHealthcare	NoDocbcCost	GenHlth	MentHlth	PhysHlth	DiffWalk	Sex	Age	Education	I
0	0.0	1.0	1.0	1.0	40.0	1.0	0.0	0.0	0.0	0.0	...	1.0	0.0	5.0	18.0	15.0	1.0	0.0	9.0	4.0	3.0
1	0.0	0.0	0.0	0.0	25.0	1.0	0.0	0.0	1.0	0.0	...	0.0	1.0	3.0	0.0	0.0	0.0	0.0	7.0	6.0	1.0
2	0.0	1.0	1.0	1.0	28.0	0.0	0.0	0.0	0.0	1.0	...	1.0	1.0	5.0	30.0	30.0	1.0	0.0	9.0	4.0	8.0
3	0.0	1.0	0.0	1.0	27.0	0.0	0.0	0.0	1.0	1.0	...	1.0	0.0	2.0	0.0	0.0	0.0	0.0	11.0	3.0	6.0
4	0.0	1.0	1.0	1.0	24.0	0.0	0.0	0.0	1.0	1.0	...	1.0	0.0	2.0	3.0	0.0	0.0	0.0	11.0	5.0	4.0
5	0.0	1.0	1.0	1.0	25.0	1.0	0.0	0.0	1.0	1.0	...	1.0	0.0	2.0	0.0	2.0	0.0	1.0	10.0	6.0	8.0
6	0.0	1.0	0.0	1.0	30.0	1.0	0.0	0.0	0.0	0.0	...	1.0	0.0	3.0	0.0	14.0	0.0	0.0	9.0	6.0	7.0
7	0.0	1.0	1.0	1.0	25.0	1.0	0.0	0.0	1.0	0.0	...	1.0	0.0	3.0	0.0	0.0	1.0	0.0	11.0	4.0	4.0
8	1.0	1.0	1.0	1.0	30.0	1.0	0.0	1.0	0.0	1.0	...	1.0	0.0	5.0	30.0	30.0	1.0	0.0	9.0	5.0	1.0
9	0.0	0.0	0.0	1.0	24.0	0.0	0.0	0.0	0.0	0.0	...	1.0	0.0	2.0	0.0	0.0	0.0	1.0	8.0	4.0	3.0
10	1.0	0.0	0.0	1.0	25.0	1.0	0.0	0.0	1.0	1.0	...	1.0	0.0	3.0	0.0	0.0	0.0	1.0	13.0	6.0	8.0
11	0.0	1.0	1.0	1.0	34.0	1.0	0.0	0.0	0.0	1.0	...	1.0	0.0	3.0	0.0	30.0	1.0	0.0	10.0	5.0	1.0
12	0.0	0.0	0.0	1.0	26.0	1.0	0.0	0.0	0.0	0.0	...	1.0	0.0	3.0	0.0	15.0	0.0	0.0	7.0	5.0	7.0
13	1.0	1.0	1.0	1.0	28.0	0.0	0.0	0.0	0.0	0.0	...	1.0	0.0	4.0	0.0	0.0	1.0	0.0	11.0	4.0	6.0
14	0.0	0.0	1.0	1.0	33.0	1.0	1.0	0.0	1.0	0.0	...	1.0	1.0	4.0	30.0	28.0	0.0	0.0	4.0	6.0	2.0
15	0.0	1.0	0.0	1.0	33.0	0.0	0.0	0.0	1.0	0.0	...	1.0	0.0	2.0	5.0	0.0	0.0	0.0	6.0	6.0	8.0
16	0.0	1.0	1.0	1.0	21.0	0.0	0.0	0.0	1.0	1.0	...	1.0	0.0	3.0	0.0	0.0	0.0	0.0	10.0	4.0	3.0
17	1.0	0.0	0.0	1.0	23.0	1.0	0.0	0.0	1.0	0.0	...	1.0	0.0	2.0	0.0	0.0	0.0	1.0	7.0	5.0	6.0
18	0.0	0.0	0.0	0.0	23.0	0.0	0.0	0.0	0.0	0.0	...	1.0	0.0	2.0	15.0	0.0	0.0	0.0	2.0	6.0	7.0

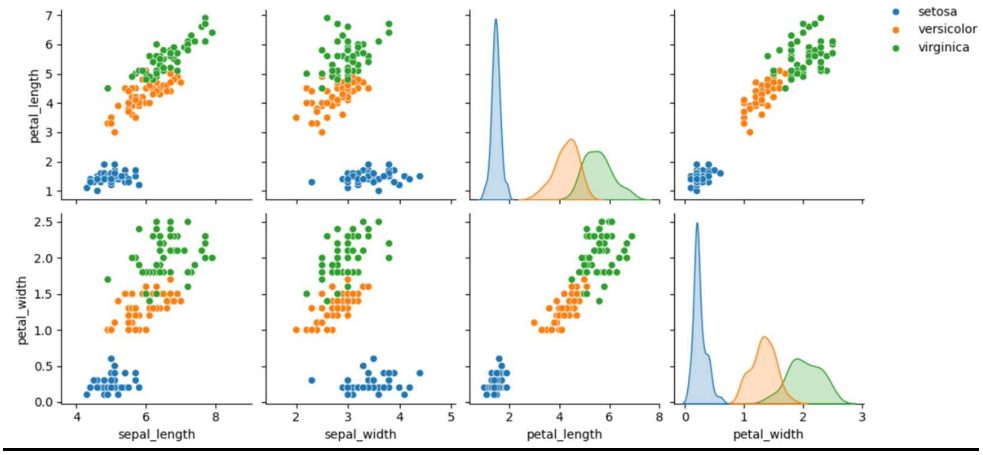
```
import seaborn as sns
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
data = np.random.rand(10, 10)
df = pd.DataFrame(data, columns=[f'Feature {i}' for i in range(10)])
corr_matrix = df.corr()
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm', fmt=".2f")
plt.title('Heatmap Example')
plt.show()
```

Output:



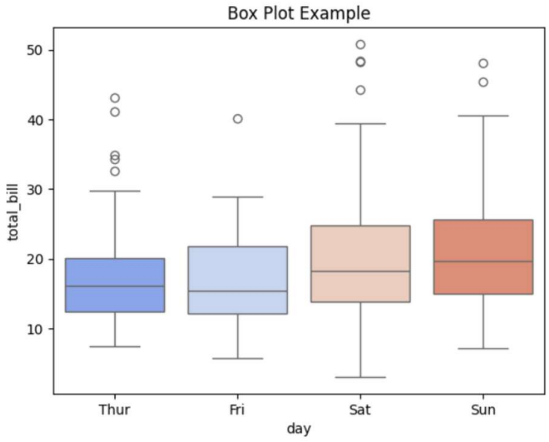

```
import seaborn as sns
import pandas as pd
import matplotlib.pyplot as plt
df = sns.load_dataset("iris")
sns.pairplot(df, hue="species")
  plt.show()
```






```
import seaborn as sns
import pandas as pd
import matplotlib.pyplot as plt
df = sns.load_dataset("tips")
sns.boxplot(x="day", y="total_bill", data=df, palette="coolwarm")
plt.title('Box Plot Example')
plt.show()
```

Output:



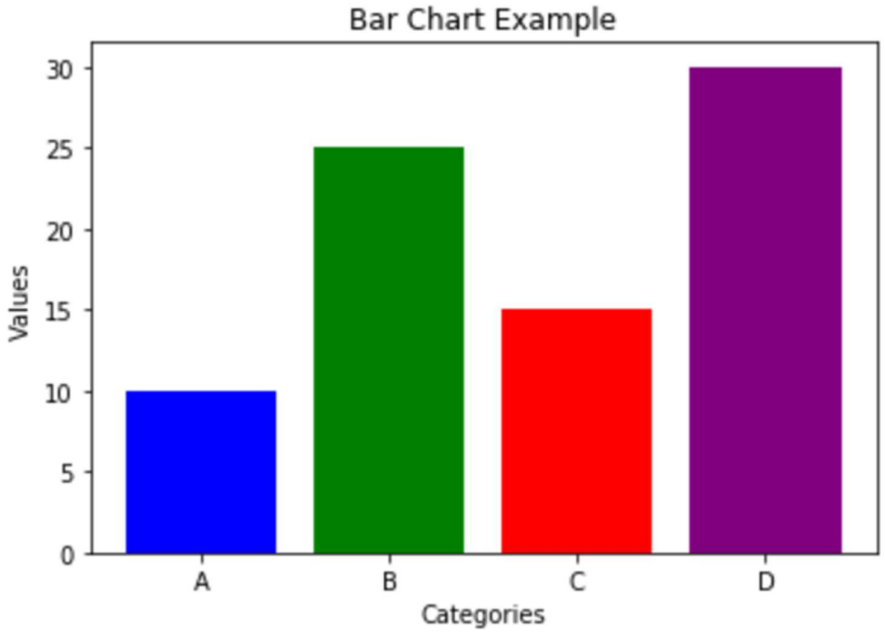
5.

#Bar Chart using Matplotlib

```
import matplotlib.pyplot as plt
import numpy as np
categories = ['A', 'B', 'C', 'D']
values = [10, 25, 15, 30]
plt.bar(categories, values, color=['blue', 'green', 'red', 'purple'])
```

```
plt.xlabel('Categories')
plt.ylabel('Values')
plt.title('Bar Chart Example')
plt.show()
```

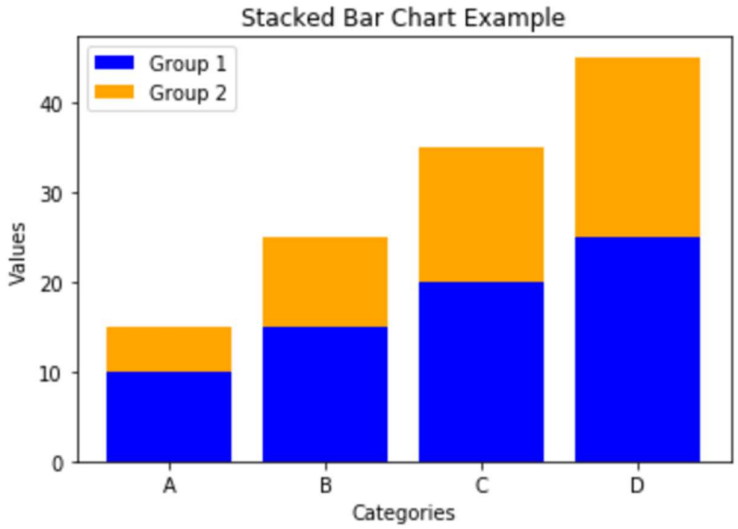
Output:



 # Stacked Bar Chart using Matplotlib

```
import matplotlib.pyplot as plt
import numpy as np
categories = ['A', 'B', 'C', 'D']
group1 = [10, 15, 20, 25]
group2 = [5, 10, 15, 20]
plt.bar(categories, group1, color='blue', label='Group 1')
plt.bar(categories, group2, color='orange', bottom=group1, label='Group 2')
plt.xlabel('Categories')
plt.ylabel('Values')
plt.title('Stacked Bar Chart Example')
plt.legend()
plt.show()
```

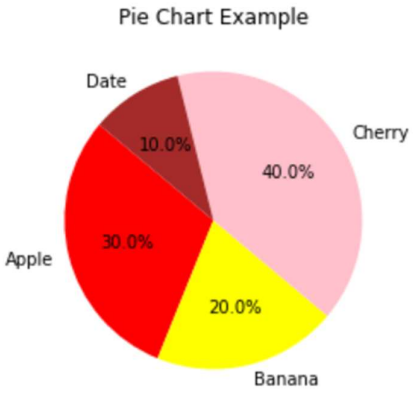
Output:



#Pie Chart using Matplotlib

```
import matplotlib.pyplot as plt
labels = ['Apple', 'Banana', 'Cherry', 'Date']
sizes = [30, 20, 40, 10]
colors = ['red', 'yellow', 'pink', 'brown']
plt.pie(sizes, labels=labels, colors=colors, autopct='%1.1f%%', startangle=140)
plt.title('Pie Chart Example')
plt.show()
```

Output:

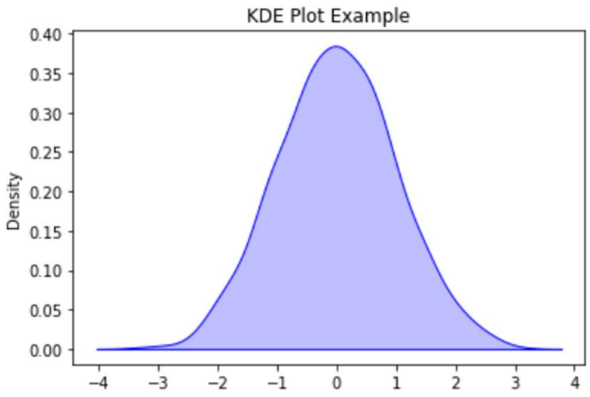


#KDE (Kernel Density Estimation) Plot using Seaborn

```
import seaborn as sns
import numpy as np
```

```
import matplotlib.pyplot as plt
data = np.random.randn(1000)
sns.kdeplot(data, shade=True, color="blue")
plt.title('KDE Plot Example')
plt.show()
```

Output:



#Swarm Plot using Seaborn

```
import seaborn as sns
import pandas as pd
import matplotlib.pyplot as plt
df = sns.load_dataset("tips")
sns.swarmplot(x="day", y="total_bill", data=df, palette="coolwarm")
plt.title('Swarm Plot Example')
plt.show()
```

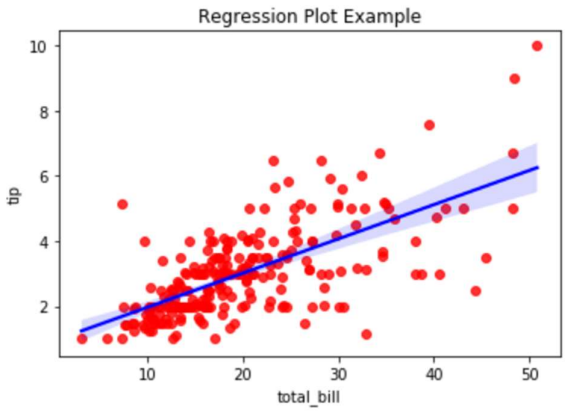
Output:

#Regression Plot using Seaborn

```
import seaborn as sns
import pandas as pd
import matplotlib.pyplot as plt
df = sns.load_dataset("tips")
```

```
sns.regplot(x="total_bill", y="tip", data=df, scatter_kws={"color": "red"}, line_kws={"color":
"blue"})
plt.title('Regression Plot Example')
plt.show()
```

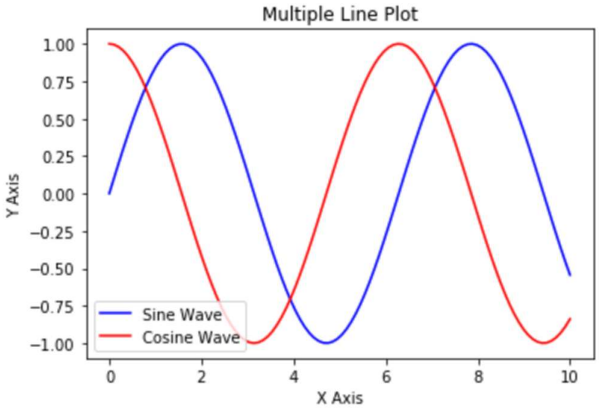
Output:



#Multiple Line Plot using Matplotlib

```
import matplotlib.pyplot as plt
import numpy as np
x = np.linspace(0, 10, 100)
y1 = np.sin(x)
y2 = np.cos(x)
plt.plot(x, y1, label='Sine Wave', color='blue')
plt.plot(x, y2, label='Cosine Wave', color='red')
plt.xlabel('X Axis')
plt.ylabel('Y Axis')
plt.title('Multiple Line Plot')plt.legend()
plt.show()
```

Output:



```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import confusion_matrix, accuracy_score, precision_score, recall_score
iris = load_iris()
df = pd.DataFrame(data=iris.data, columns=iris.feature_names)
df['species'] = iris.target
df['species'] = df['species'].map({0:'setosa', 1:'versicolor', 2:'virginica'})
print("Iris Dataset:")
print(df.head())
```

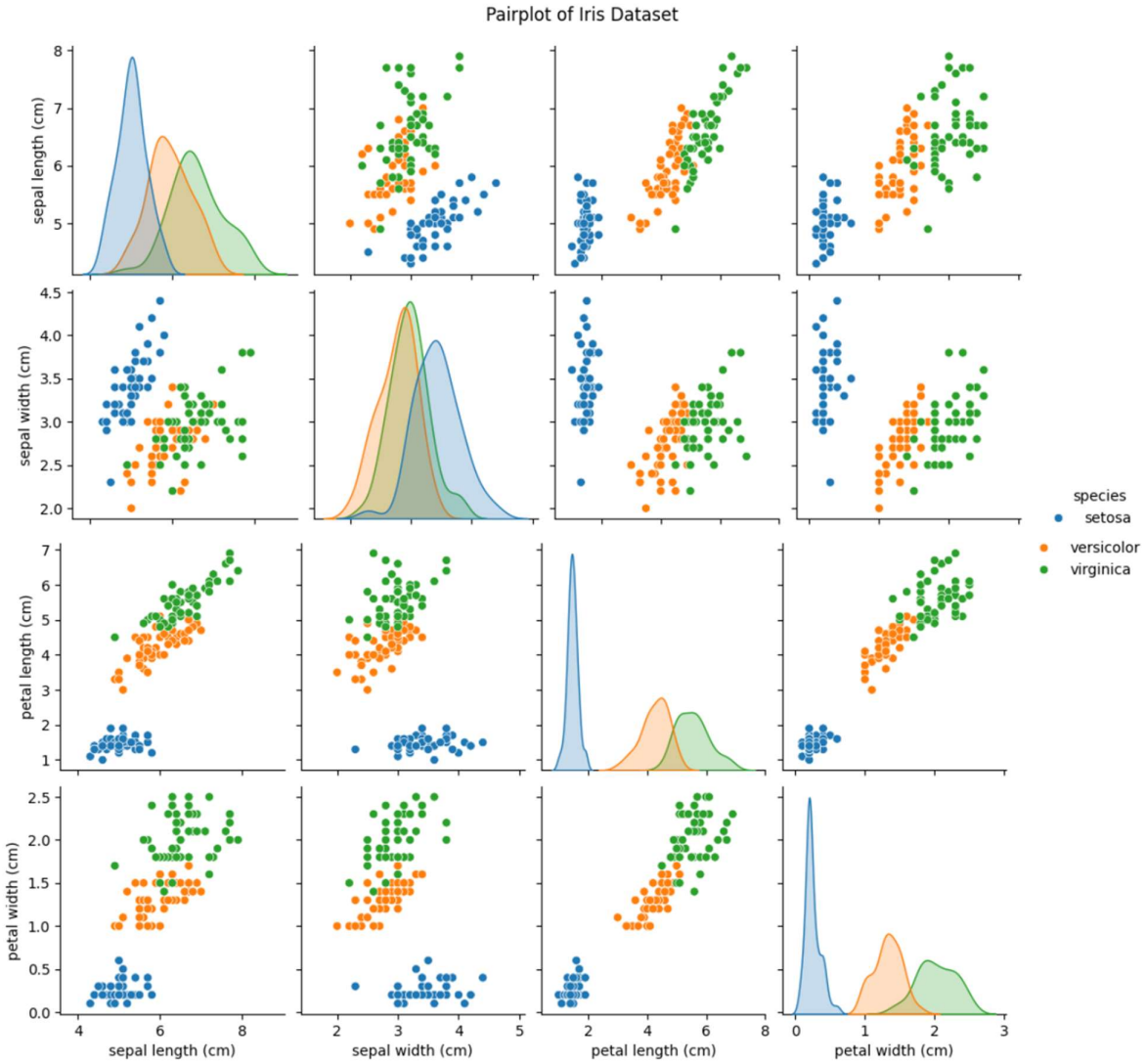
Output:

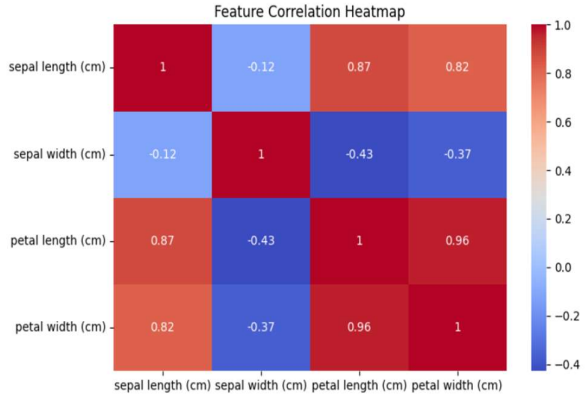
```
Iris Dataset:
   sepal length (cm)  sepal width (cm)  petal length (cm)  petal width (cm)
0                5.1               3.5                1.4               0.2
1                4.9               3.0                1.4               0.2
2                4.7               3.2                1.3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

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

```
sns.pairplot(df, hue='species')
plt.suptitle("Pairplot of Iris Dataset", y=1.02)
plt.show()
plt.figure(figsize=(8, 5))
sns.heatmap(df.iloc[:, :-1].corr(), annot=True, cmap='coolwarm')
plt.title("Feature Correlation Heatmap")
plt.show()
```

Output:






```
import numpy as np
import pandas as pd
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
```

```
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import confusion_matrix, accuracy_score, precision_score, recall_score
data = load_iris()
df = pd.DataFrame(data.data, columns=data.feature_names)
df['target'] = data.target
X = df.iloc[:, :-1].values
y = df.iloc[:, -1].values
X_train, X_test, y_train, y_test = train_test_split(
   X, y, test_size=0.3, random_state=42, stratify=y
)
# Standardize features (important for KNN)
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# ================= KNN Classifier =================
knn = KNeighborsClassifier(n_neighbors=5)
knn.fit(X_train, y_train)
y_pred_knn = knn.predict(X_test)

# ================= Decision Tree Classifier =================
dt = DecisionTreeClassifier(random_state=42)
dt.fit(X_train, y_train)
y_pred_dt = dt.predict(X_test)

# ================= Evaluation Function =================
def evaluate_model(y_test, y_pred, model_name):
   cm = confusion_matrix(y_test, y_pred, labels=np.unique(y_test))
   accuracy = accuracy_score(y_test, y_pred)
   precision = precision_score(y_test, y_pred, average='macro')
   recall = recall_score(y_test, y_pred, average='macro')

   # Specificity (per class, then average)
   TN = []
   FP = []
   for i in range(len(np.unique(y_test))):
      tn = np.sum(np.delete(np.delete(cm, i, axis=0), i, axis=1))
      fp = np.sum(cm[:, i]) - cm[i, i]
      TN.append(tn)
      FP.append(fp)
   specificity = np.mean(np.array(TN) / (np.array(TN) + np.array(FP)))

   print(f"\n===== {model_name} Evaluation =====")
```

```
    print("Confusion Matrix:\n", cm)
    print(f"Accuracy: {accuracy:.4f}")
    print(f"Precision: {precision:.4f}")
    print(f"Recall (Sensitivity): {recall:.4f}")
    print(f"Specificity: {specificity:.4f}")

# ================ Evaluate Both Models ================
evaluate_model(y_test, y_pred_knn, "KNN Classifier")
evaluate_model(y_test, y_pred_dt, "Decision Tree Classifier")
```

```
===== KNN Classifier Evaluation =====
Confusion Matrix:
 [[15  0  0]
 [ 0 15  0]
 [ 0  4 11]]
Accuracy: 0.9111
Precision: 0.9298
Recall (Sensitivity): 0.9111
Specificity: 0.9556

===== Decision Tree Classifier Evaluation =====
Confusion Matrix:
 [[15  0  0]
 [ 0 12  3]
 [ 0  1 14]]
Accuracy: 0.9111
Precision: 0.9155
Recall (Sensitivity): 0.9111
Specificity: 0.9556
```