```python
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

1. Read the Auto data

```python
import pandas as pd
path = "/content/drive/MyDrive/CSV /Auto.csv"
df = pd.read_csv(path)

df.info()
print("Get the number of rows: ", len(df))
print("Get the number of columns: ", len(df.columns))
print("Get the number of rows and columns: ", df.shape)
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 392 entries, 0 to 391
Data columns (total 9 columns):
 #   Column        Non-Null Count  Dtype  
---  ------        --------------  -----  
 0   mpg           392 non-null    float64
 1   cylinders     392 non-null    int64  
 2   displacement  392 non-null    float64
 3   horsepower    392 non-null    int64  
 4   weight        392 non-null    int64  
 5   acceleration  391 non-null    float64
 6   year          390 non-null    float64
 7   origin        392 non-null    int64  
 8   name          392 non-null    object 
dtypes: float64(4), int64(4), object(1)
memory usage: 27.7+ KB
Get the number of rows:  392
Get the number of columns:  9
Get the number of rows and columns:  (392, 9)
```

1. Data exploration with code

```python
mpg = pd.Series(df.mpg)
year = pd.Series(df.year)
weight = pd.Series(df.weight)

data = pd.DataFrame({"mpg": mpg, "weight": weight, "year": year})
data[["mpg","weight","year"]].describe(include="all")
#The range of mpg: 9 - 46.6 Avg: 22.75
#The range of weight: 1613 - 5140 Avg: 2803.5
#The range of year: 70 - 82 Avg: 76
```

```
              mpg       weight        year
count  392.000000   392.000000  390.000000
mean    23.445918  2977.584184   76.010256
std      7.805007   849.402560    3.668093
min      9.000000  1613.000000   70.000000
```

```
25%     17.000000  2225.250000   73.000000
50%     22.750000  2803.500000   76.000000
75%     29.000000  3614.750000   79.000000
max     46.600000  5140.000000   82.000000
```

1. Explore data types

```
dt = pd.DataFrame(df)
print(dt.dtypes)

dcy = df.cylinders.astype("category").cat.codes

print(dcy)
print(dcy.dtypes)

dor = pd.Series(df.origin,dtype="category")
print(dor)
print(dor.dtypes)
```

```
mpg             float64
cylinders         int64
displacement    float64
horsepower        int64
weight            int64
acceleration    float64
year            float64
origin            int64
name             object
dtype: object
0      4
1      4
2      4
3      4
4      4
      ..
387    1
388    1
389    1
390    1
391    1
Length: 392, dtype: int8
int8
0      1
1      1
2      1
3      1
4      1
      ..
387    1
388    2
389    1
```

```
390    1
391    1
Name: origin, Length: 392, dtype: category
Categories (3, int64): [1, 2, 3]
category
```

1. Deal with NAs

```
import pandas as pd
path = "/content/drive/MyDrive/CSV /Auto.csv"
df = pd.read_csv(path)

df_new = df.dropna()

df_new.info()
print("Get the number of rows: ", len(df_new))
print("Get the number of columns: ", len(df_new.columns))
print("Get the number of rows and columns: ", df_new.shape)
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 389 entries, 0 to 391
Data columns (total 9 columns):
 #   Column        Non-Null Count  Dtype  
---  ------        --------------  -----  
 0   mpg           389 non-null    float64
 1   cylinders     389 non-null    int64  
 2   displacement  389 non-null    float64
 3   horsepower    389 non-null    int64  
 4   weight        389 non-null    int64  
 5   acceleration  389 non-null    float64
 6   year          389 non-null    float64
 7   origin        389 non-null    int64  
 8   name          389 non-null    object 
dtypes: float64(4), int64(4), object(1)
memory usage: 30.4+ KB
Get the number of rows:  389
Get the number of columns:  9
Get the number of rows and columns:  (389, 9)
```

1. Modify columns

```
import pandas as pd
path = "/content/drive/MyDrive/CSV /Auto.csv"
df = pd.read_csv(path)
avg_mpg = df['mpg'].mean()

# create new column mpg_high
df['mpg_high'] = (df['mpg'] > avg_mpg).astype(int)

# drop mpg and name columns
df.drop(['mpg', 'name'], axis=1, inplace=True)
```

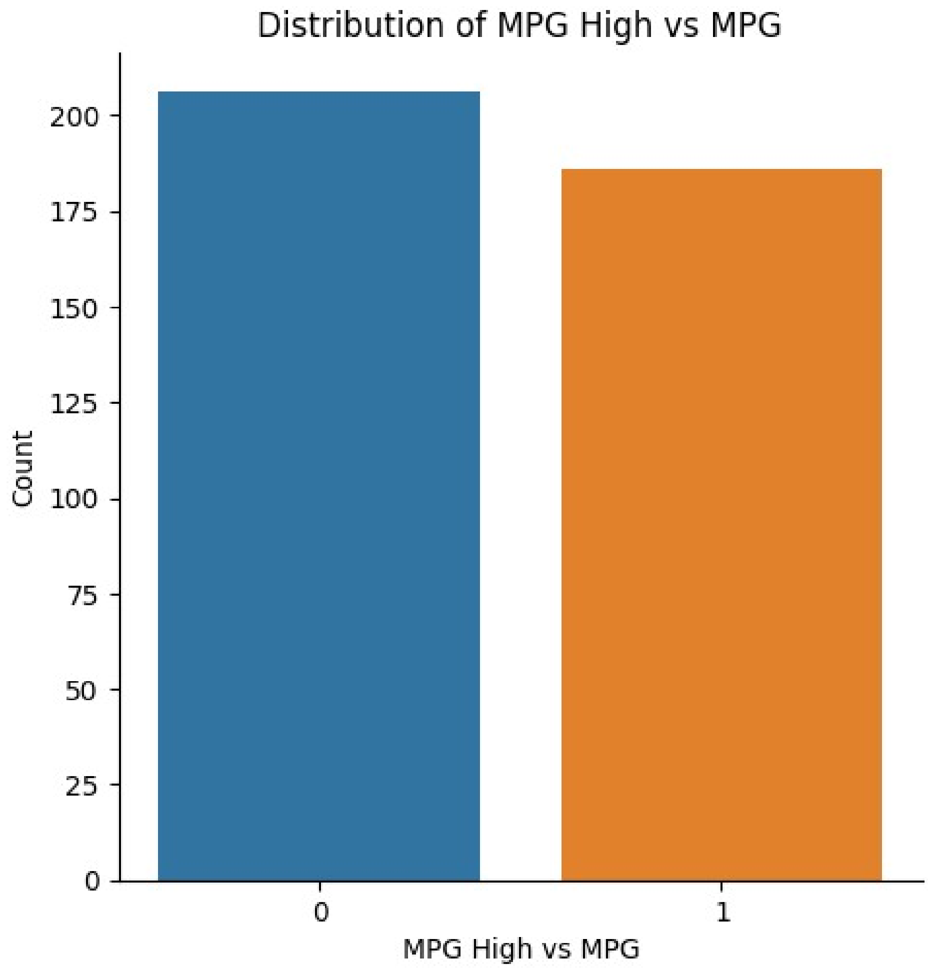
```
print(df)
```

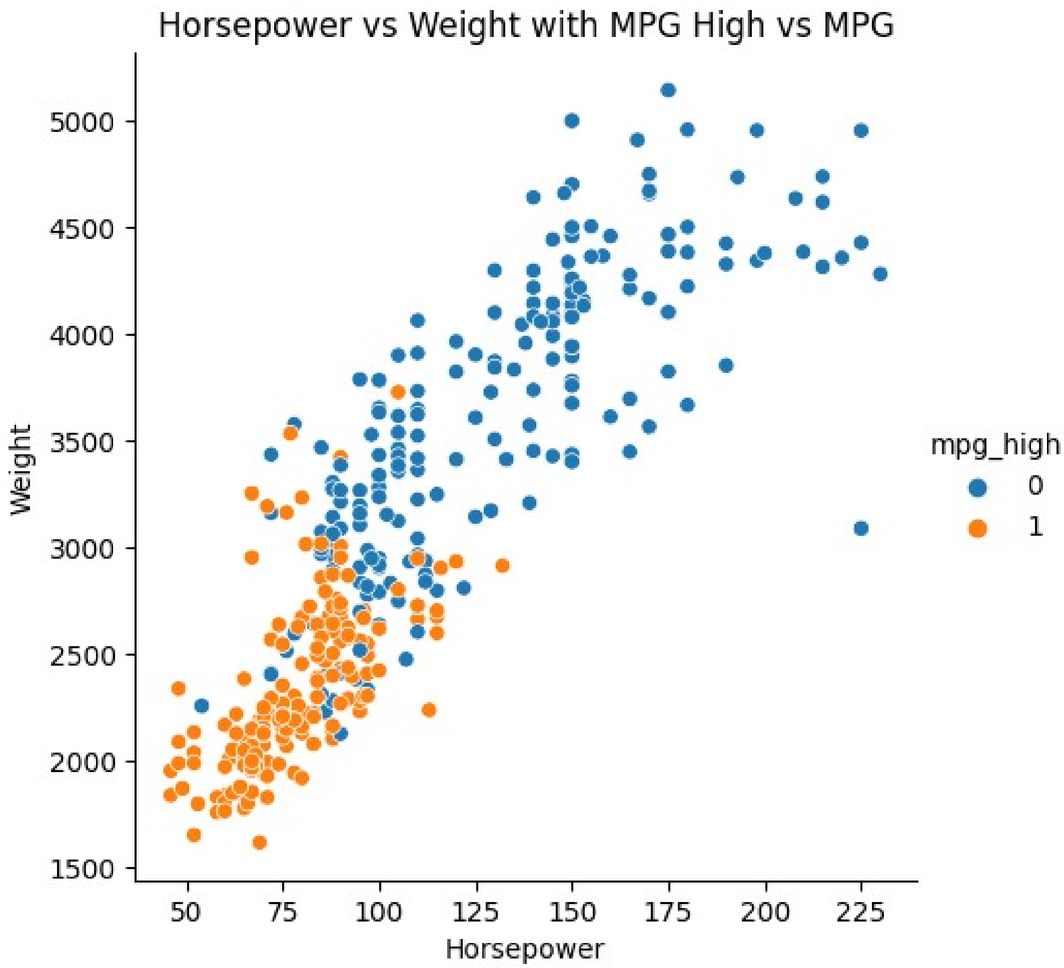
```
     cylinders  displacement  horsepower  weight  acceleration  year
origin  \
0            8         307.0         130    3504          12.0  70.0
1
1            8         350.0         165    3693          11.5  70.0
1
2            8         318.0         150    3436          11.0  70.0
1
3            8         304.0         150    3433          12.0  70.0
1
4            8         302.0         140    3449           NaN  70.0
1
..         ...           ...         ...     ...           ...   ...
...
387          4         140.0          86    2790          15.6  82.0
1
388          4          97.0          52    2130          24.6  82.0
2
389          4         135.0          84    2295          11.6  82.0
1
390          4         120.0          79    2625          18.6  82.0
1
391          4         119.0          82    2720          19.4  82.0
1

     mpg_high
0           0
1           0
2           0
3           0
4           0
..        ...
387         1
388         1
389         1
390         1
391         1

[392 rows x 8 columns]
```

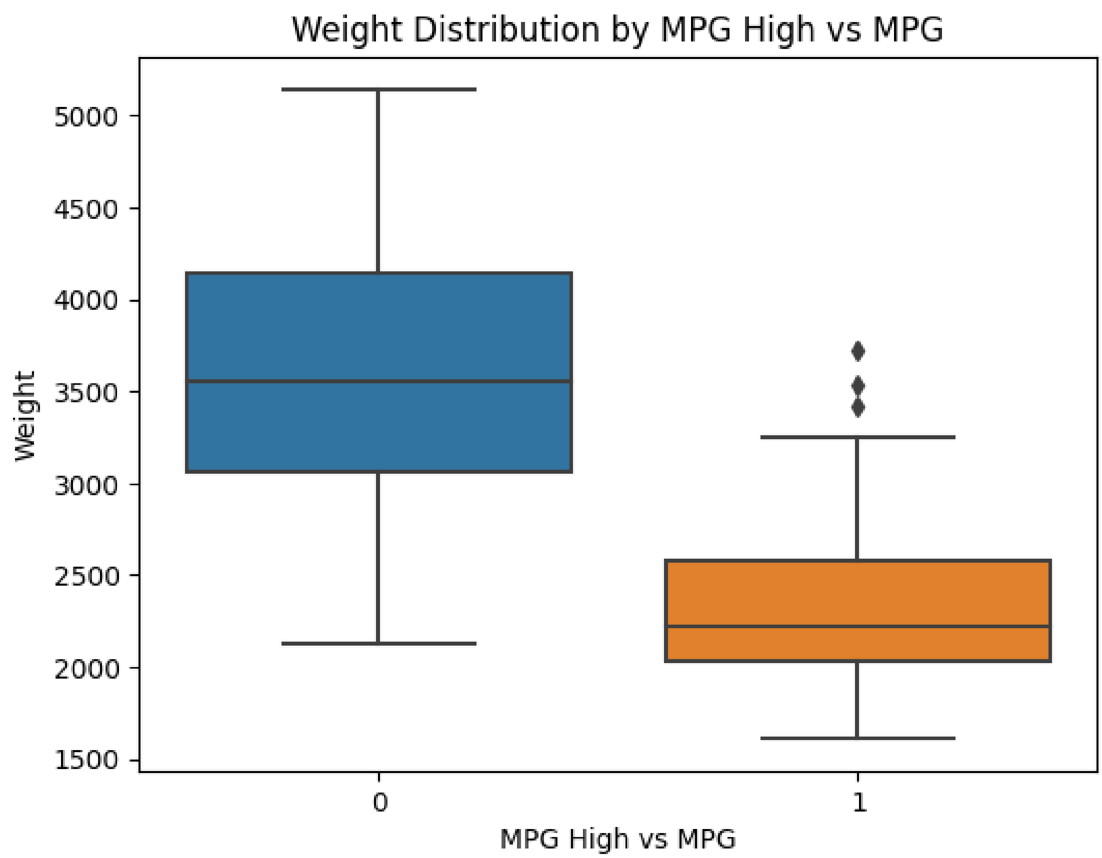
1. Data exploration with graphs

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

# seaborn catplot on the mpg_high column
# The majority of cars mpg < average mpg in the dataset
sns.catplot(x='mpg_high', kind='count', data=df)
```

```python
plt.title('Distribution of MPG High vs MPG')
plt.xlabel('MPG High vs MPG')
plt.ylabel('Count')
plt.show()

# seaborn relplot with horsepower on the x axis, weight on the y axis, setting hue to mpg_high
# Cars with higher horsepower tend to be heavier, and that cars with higher MPG tend to have lower horsepower and lower weight.
sns.relplot(x='horsepower', y='weight', hue='mpg_high', data=df)
plt.title('Horsepower vs Weight with MPG High vs MPG')
plt.xlabel('Horsepower')
plt.ylabel('Weight')
plt.show()

# seaborn boxplot with mpg_high on the x axis and weight on the y axis
# Cars with high MPG tend to have lower weight, and that there is more variation in weight for cars with low MPG.
sns.boxplot(x='mpg_high', y='weight', data=df)
plt.title('Weight Distribution by MPG High vs MPG')
plt.xlabel('MPG High vs MPG')
plt.ylabel('Weight')
plt.show()
```


Distribution of MPG High vs MPG
Count
200
175
150
125
100
75
50
25
0
0
1
MPG High vs MPG


Horsepower vs Weight with MPG High vs MPG
Weight
5000
4500
4000
3500
3000
2500
2000
1500
50
75
100
125
150
175
200
225
Horsepower
mpg_high
0
1



1. Train/test split

```
import numpy as np
from sklearn.model_selection import train_test_split

df_new = df.dropna()
X_train, X_test, y_train, y_test =
train_test_split(df_new.drop('mpg_high', axis=1), df_new['mpg_high'],
test_size=0.2, random_state=1234)

print("Train set dimensions:", X_train.shape)
print("Test set dimensions:", X_test.shape)
```

```
Train set dimensions: (311, 7)
Test set dimensions: (78, 7)
```

1. Logistic Regression

```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report

# Define logistic regression model with lbfgs solver
logis = LogisticRegression(solver='lbfgs')
```

```
logis.fit(X_train, y_train)

y_pred = logis.predict(X_test)

# Using the classification report
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.98      0.80      0.88        50
           1       0.73      0.96      0.83        28

    accuracy                           0.86        78
   macro avg       0.85      0.88      0.85        78
weighted avg       0.89      0.86      0.86        78


/usr/local/lib/python3.9/dist-packages/sklearn/linear_model/
_logistic.py:458: ConvergenceWarning: lbfgs failed to converge
(status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as
shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:

https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
```

1. Decision Tree

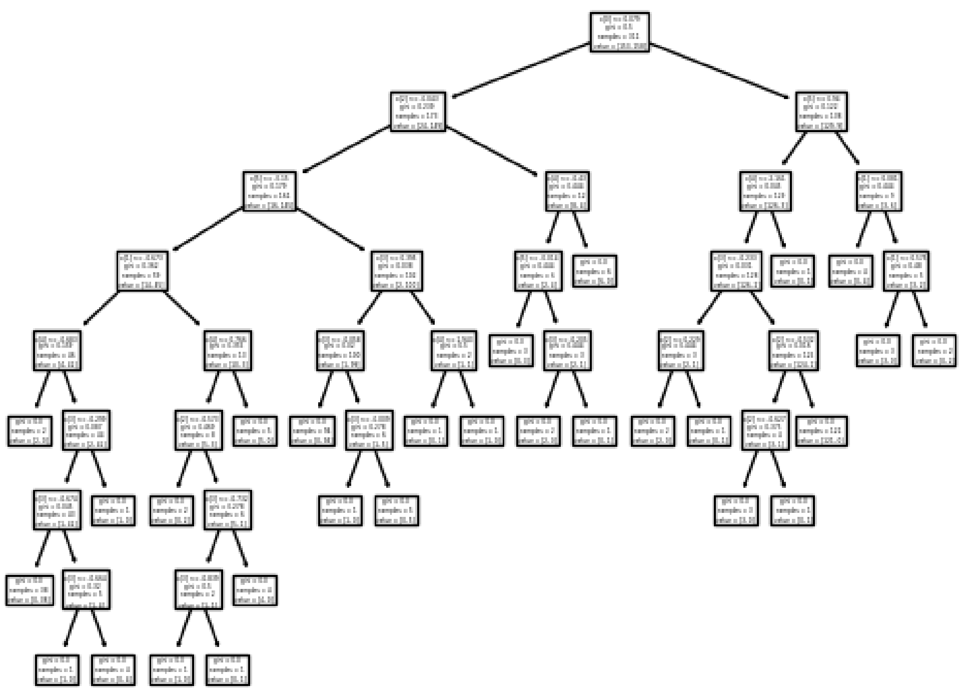
```
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import classification_report
from sklearn import tree
# Define decision tree model
dt = DecisionTreeClassifier(random_state=1234)

dt.fit(X_train, y_train)

y_pred = dt.predict(X_test)

print(classification_report(y_test, y_pred))

tree.plot_tree(dt)
```

```
              precision    recall  f1-score   support

           0       0.96      0.92      0.94        50
           1       0.87      0.93      0.90        28
```

```
    accuracy                           0.92        78
   macro avg       0.91      0.92      0.92        78
weighted avg       0.93      0.92      0.92        78


[Text(0.6433823529411765, 0.9444444444444444, 'x[0] <= 0.079\ngini =
0.5\nsamples = 311\nvalue = [153, 158]'),
 Text(0.4338235294117647, 0.8333333333333334, 'x[2] <= -0.043\ngini =
0.239\nsamples = 173\nvalue = [24, 149]'),
 Text(0.27941176470588236, 0.7222222222222222, 'x[5] <= -0.15\ngini =
0.179\nsamples = 161\nvalue = [16, 145]'),
 Text(0.14705882352941177, 0.6111111111111112, 'x[1] <= -0.673\ngini =
0.362\nsamples = 59\nvalue = [14, 45]'),
 Text(0.058823529411764705, 0.5, 'x[4] <= -0.683\ngini = 0.159\
nsamples = 46\nvalue = [4, 42]'),
 Text(0.029411764705882353, 0.3888888888888889, 'gini = 0.0\nsamples =
2\nvalue = [2, 0]'),
 Text(0.08823529411764706, 0.3888888888888889, 'x[3] <= -0.299\ngini =
0.087\nsamples = 44\nvalue = [2, 42]'),
 Text(0.058823529411764705, 0.2777777777777778, 'x[3] <= -0.674\ngini
= 0.045\nsamples = 43\nvalue = [1, 42]'),
 Text(0.029411764705882353, 0.16666666666666666, 'gini = 0.0\nsamples
= 38\nvalue = [0, 38]'),
 Text(0.08823529411764706, 0.16666666666666666, 'x[3] <= -0.664\ngini
= 0.32\nsamples = 5\nvalue = [1, 4]'),
 Text(0.058823529411764705, 0.05555555555555555, 'gini = 0.0\nsamples
= 1\nvalue = [1, 0]'),
 Text(0.11764705882352941, 0.05555555555555555, 'gini = 0.0\nsamples =
4\nvalue = [0, 4]'),
 Text(0.11764705882352941, 0.2777777777777778, 'gini = 0.0\nsamples =
1\nvalue = [1, 0]'),
 Text(0.23529411764705882, 0.5, 'x[4] <= 0.766\ngini = 0.355\nsamples
= 13\nvalue = [10, 3]'),
 Text(0.20588235294117646, 0.3888888888888889, 'x[2] <= -0.573\ngini =
0.469\nsamples = 8\nvalue = [5, 3]'),
 Text(0.17647058823529413, 0.2777777777777778, 'gini = 0.0\nsamples =
2\nvalue = [0, 2]'),
 Text(0.23529411764705882, 0.2777777777777778, 'x[3] <= -0.732\ngini =
0.278\nsamples = 6\nvalue = [5, 1]'),
 Text(0.20588235294117646, 0.16666666666666666, 'x[3] <= -0.839\ngini
= 0.5\nsamples = 2\nvalue = [1, 1]'),
 Text(0.17647058823529413, 0.05555555555555555, 'gini = 0.0\nsamples =
1\nvalue = [1, 0]'),
 Text(0.23529411764705882, 0.05555555555555555, 'gini = 0.0\nsamples =
1\nvalue = [0, 1]'),
 Text(0.2647058823529412, 0.16666666666666666, 'gini = 0.0\nsamples =
4\nvalue = [4, 0]'),
 Text(0.2647058823529412, 0.3888888888888889, 'gini = 0.0\nsamples =
5\nvalue = [5, 0]'),
```

```
 Text(0.4117647058823529, 0.6111111111111112, 'x[3] <= 0.395\ngini =
0.038\nsamples = 102\nvalue = [2, 100]'),
 Text(0.35294117647058826, 0.5, 'x[3] <= -0.058\ngini = 0.02\nsamples
= 100\nvalue = [1, 99]'),
 Text(0.3235294117647059, 0.3888888888888889, 'gini = 0.0\nsamples =
94\nvalue = [0, 94]'),
 Text(0.38235294117647056, 0.3888888888888889, 'x[3] <= -0.009\ngini =
0.278\nsamples = 6\nvalue = [1, 5]'),
 Text(0.35294117647058826, 0.2777777777777778, 'gini = 0.0\nsamples =
1\nvalue = [1, 0]'),
 Text(0.4117647058823529, 0.2777777777777778, 'gini = 0.0\nsamples =
5\nvalue = [0, 5]'),
 Text(0.47058823529411764, 0.5, 'x[4] <= 1.943\ngini = 0.5\nsamples =
2\nvalue = [1, 1]'),
 Text(0.4411764705882353, 0.3888888888888889, 'gini = 0.0\nsamples =
1\nvalue = [0, 1]'),
 Text(0.5, 0.3888888888888889, 'gini = 0.0\nsamples = 1\nvalue = [1,
0]'),
 Text(0.5882352941176471, 0.7222222222222222, 'x[4] <= -0.43\ngini =
0.444\nsamples = 12\nvalue = [8, 4]'),
 Text(0.5588235294117647, 0.6111111111111112, 'x[5] <= -0.014\ngini =
0.444\nsamples = 6\nvalue = [2, 4]'),
 Text(0.5294117647058824, 0.5, 'gini = 0.0\nsamples = 3\nvalue = [0,
3]'),
 Text(0.5882352941176471, 0.5, 'x[3] <= -0.205\ngini = 0.444\nsamples
= 3\nvalue = [2, 1]'),
 Text(0.5588235294117647, 0.3888888888888889, 'gini = 0.0\nsamples =
2\nvalue = [2, 0]'),
 Text(0.6176470588235294, 0.3888888888888889, 'gini = 0.0\nsamples =
1\nvalue = [0, 1]'),
 Text(0.6176470588235294, 0.6111111111111112, 'gini = 0.0\nsamples =
6\nvalue = [6, 0]'),
 Text(0.8529411764705882, 0.8333333333333334, 'x[5] <= 0.94\ngini =
0.122\nsamples = 138\nvalue = [129, 9]'),
 Text(0.7941176470588235, 0.7222222222222222, 'x[4] <= 2.161\ngini =
0.045\nsamples = 129\nvalue = [126, 3]'),
 Text(0.7647058823529411, 0.6111111111111112, 'x[3] <= -0.233\ngini =
0.031\nsamples = 128\nvalue = [126, 2]'),
 Text(0.7058823529411765, 0.5, 'x[2] <= 0.229\ngini = 0.444\nsamples =
3\nvalue = [2, 1]'),
 Text(0.6764705882352942, 0.3888888888888889, 'gini = 0.0\nsamples =
2\nvalue = [2, 0]'),
 Text(0.7352941176470589, 0.3888888888888889, 'gini = 0.0\nsamples =
1\nvalue = [0, 1]'),
 Text(0.8235294117647058, 0.5, 'x[2] <= -0.532\ngini = 0.016\nsamples
= 125\nvalue = [124, 1]'),
 Text(0.7941176470588235, 0.3888888888888889, 'x[2] <= -0.627\ngini =
0.375\nsamples = 4\nvalue = [3, 1]'),
 Text(0.7647058823529411, 0.2777777777777778, 'gini = 0.0\nsamples =
3\nvalue = [3, 0]'),
```

```
 Text(0.8235294117647058, 0.2777777777777778, 'gini = 0.0\nsamples =
1\nvalue = [0, 1]'),
 Text(0.8529411764705882, 0.3888888888888889, 'gini = 0.0\nsamples =
121\nvalue = [121, 0]'),
 Text(0.8235294117647058, 0.6111111111111112, 'gini = 0.0\nsamples =
1\nvalue = [0, 1]'),
 Text(0.9117647058823529, 0.7222222222222222, 'x[1] <= 0.081\ngini =
0.444\nsamples = 9\nvalue = [3, 6]'),
 Text(0.8823529411764706, 0.6111111111111112, 'gini = 0.0\nsamples =
4\nvalue = [0, 4]'),
 Text(0.9411764705882353, 0.6111111111111112, 'x[1] <= 0.575\ngini =
0.48\nsamples = 5\nvalue = [3, 2]'),
 Text(0.9117647058823529, 0.5, 'gini = 0.0\nsamples = 3\nvalue = [3,
0]'),
 Text(0.9705882352941176, 0.5, 'gini = 0.0\nsamples = 2\nvalue = [0,
2]')]
```



1. Neural Network

```python
import numpy as np
import pandas as pd
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import accuracy_score, classification_report

mlp = MLPClassifier(hidden_layer_sizes=(10,), max_iter=150,
batch_size=15, random_state=1234)
mlp.fit(X_train, y_train)
```

```python
y_pred = mlp.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
report = classification_report(y_test, y_pred)

print("Accuracy: ", accuracy)
print("Classification Report: ", report)
```

```
Accuracy:  0.8846153846153846
Classification Report:                precision    recall  f1-score
support

           0       1.00      0.82      0.90        50
           1       0.76      1.00      0.86        28

    accuracy                           0.88        78
   macro avg       0.88      0.91      0.88        78
weighted avg       0.91      0.88      0.89        78


/usr/local/lib/python3.9/dist-packages/sklearn/neural_network/
_multilayer_perceptron.py:686: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (150) reached and the optimization
hasn't converged yet.
  warnings.warn(
```

1. Analysis

Having used both R and scikit-learn for data analysis, I felt that scikit-learn is a more user-friendly and efficient library for my needs. One of its main advantages is its simple and consistent API, which is easy to learn and use, particularly for beginners. Additionally, scikit-learn provides a comprehensive set of algorithms for both supervised and unsupervised learning, making it a valuable tool for tackling complex machine learning projects. I believe that scikit-learn offers a solid foundation for data analysis and machine learning, and is a great choice for those seeking a user-friendly and versatile library.