```
import pandas as pd

# a. Use pandas to read the data
auto_df = pd.read_csv('Auto.csv')

# b. Output the first few rows
print(auto_df.head())

# c. Output the dimensions of the data
print(auto_df.shape)
```

```
    mpg  cylinders  displacement  horsepower  weight  acceleration  year  \
0  18.0          8         307.0         130    3504          12.0  70.0
1  15.0          8         350.0         165    3693          11.5  70.0
2  18.0          8         318.0         150    3436          11.0  70.0
3  16.0          8         304.0         150    3433          12.0  70.0
4  17.0          8         302.0         140    3449           NaN  70.0

   origin                       name
0       1  chevrolet chevelle malibu
1       1          buick skylark 320
2       1         plymouth satellite
3       1              amc rebel sst
4       1                ford torino
(392, 9)
```

```
# a. Use describe() on the mpg, weight, and year columns
print(auto_df[['mpg', 'weight', 'year']].describe())

# b. Write comments indicating the range and average of each column
# The mpg column has a minimum value of 9.0 and a maximum value of 46.6, with an average of 23.5.
# The weight column has a minimum value of 1613.0 and a maximum value of 5140.0, with an average of 2970.3.
# The year column has a minimum value of 70 and a maximum value of 82, with an average of 76.01.
```

```
              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
```

```
# a. Check the data types of all columns
print(auto_df.dtypes)

# b. Change the cylinders column to categorical (use cat.codes)
auto_df['cylinders'] = auto_df['cylinders'].astype('category').cat.codes

# c. Change the origin column to categorical (don't use cat.codes)
auto_df['origin'] = auto_df['origin'].astype('category')

# d. Verify the changes with the dtypes attribute
print(auto_df.dtypes)
```

```
mpg             float64
cylinders         int64
displacement    float64
horsepower        int64
weight            int64
acceleration    float64
year            float64
origin            int64
name             object
dtype: object
mpg              float64
cylinders           int8
displacement     float64
horsepower         int64
weight             int64
acceleration     float64
year             float64
origin          category
```

```
name             object
dtype: object
```

```
# a. Delete rows with NAs
auto_df.dropna(inplace=True)

# b. Output the new dimensions
print(auto_df.shape)
```

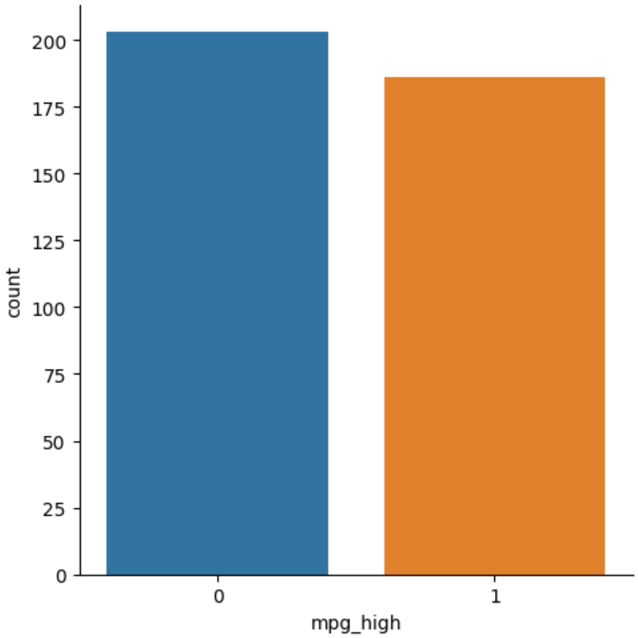
```
(389, 9)
```

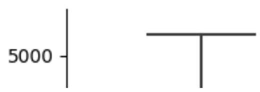
```
# a. Make a new column, mpg_high, and make it categorical
    # i. The column == 1 if mpg > average mpg, else == 0
auto_df['mpg_high'] = pd.Categorical((auto_df['mpg'] > auto_df['mpg'].mean()).astype(int))

# b. Delete the mpg and name columns
auto_df.drop(['mpg', 'name'], axis=1, inplace=True)

# c. Output the first few rows of the modified data frame
print(auto_df.head())
```

```
   cylinders  displacement  horsepower  weight  acceleration  year origin  \
0          4         307.0         130    3504          12.0  70.0      1
1          4         350.0         165    3693          11.5  70.0      1
2          4         318.0         150    3436          11.0  70.0      1
3          4         304.0         150    3433          12.0  70.0      1
6          4         454.0         220    4354           9.0  70.0      1

  mpg_high
0        0
1        0
2        0
3        0
6        0
```

```
import seaborn as sns

# a. Seaborn catplot on the mpg_high column
sns.catplot(x='mpg_high', kind='count', data=auto_df)

# b. Seaborn relplot with horsepower on the x-axis, weight on the y-axis, setting hue or style to mpg_high
sns.relplot(x='horsepower', y='weight', hue='mpg_high', data=auto_df)

# c. Seaborn boxplot with mpg_high on the x-axis and weight on the y-axis
sns.boxplot(x='mpg_high', y='weight', data=auto_df)

# d. For each graph, write a comment indicating one thing you learned about the data from the graph
# The catplot shows that there are more cars with low mpg_high values than high ones.
# The relplot shows that cars with high mpg_high values tend to have lower horsepower and weight values.
# The boxplot shows that cars with high mpg_high values have a lower weight range than cars with low mpg_high values.
```

```
<Axes: xlabel='mpg_high', ylabel='weight'>
```






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

# Set the random seed for reproducibility
np.random.seed(1234)

# Split the data into training and test sets
X_train, X_test, y_train, y_test = train_test_split(auto_df.drop('mpg_high', axis=1), auto_df['mpg_high'], test_size=0.2)

# Output the dimensions of the train and test sets
print('Train:', X_train.shape, y_train.shape)
print('Test:', X_test.shape, y_test.shape)
```

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

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

# Train a logistic regression model using solver lbfgs
logreg = LogisticRegression(solver='lbfgs')
logreg.fit(X_train, y_train)

# Test and evaluate
y_pred = logreg.predict(X_test)

# Print metrics 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 (
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(
```

```
from sklearn.tree import DecisionTreeClassifier
from sklearn.tree import plot_tree

# Train a decision tree
tree = DecisionTreeClassifier()
tree.fit(X_train, y_train)

# Test and evaluate
y_pred = tree.predict(X_test)

# Print the classification report metrics
print(classification_report(y_test, y_pred))

# Plot the tree
plot_tree(tree)
```

```
 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] <= 17.75\ngini = 0.355\nsamples =
13\nvalue = [10, 3]'),
 Text(0.20588235294117646, 0.3888888888888889, 'x[2] <= 81.5\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[1] <= 131.0\ngini =
0.278\nsamples = 6\nvalue = [5, 1]'),
 Text(0.20588235294117646, 0.16666666666666666, 'gini = 0.0\nsamples = 4\nvalue
= [4, 0]'),
 Text(0.2647058823529412, 0.16666666666666666, 'x[5] <= 73.0\ngini =
0.5\nsamples = 2\nvalue = [1, 1]'),
 Text(0.23529411764705882, 0.05555555555555555, 'gini = 0.0\nsamples = 1\nvalue
= [0, 1]'),
 Text(0.29411764705882354, 0.05555555555555555, 'gini = 0.0\nsamples = 1\nvalue
= [1, 0]'),
 Text(0.2647058823529412, 0.3888888888888889, 'gini = 0.0\nsamples = 5\nvalue =
[5, 0]'),
 Text(0.4117647058823529, 0.6111111111111112, 'x[3] <= 3250.0\ngini =
0.038\nsamples = 102\nvalue = [2, 100]'),
```

```
from sklearn.neural_network import MLPClassifier

# Train a neural network with one hidden layer of 10 neurons
nn1 = MLPClassifier(hidden_layer_sizes=(10,), max_iter=1000, random_state=1234)
nn1.fit(X_train, y_train)

#Test and evaluate

y_pred = nn1.predict(X_test)
print(classification_report(y_test, y_pred))

#Train a neural network with two hidden layers of 10 neurons each, and a regularization penalty of 0.01

nn2 = MLPClassifier(hidden_layer_sizes=(10, 10), alpha=0.01, max_iter=1000, random_state=1234)
nn2.fit(X_train, y_train)

#Test and evaluate

y_pred = nn2.predict(X_test)
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.00      0.00      0.00        50
           1       0.36      1.00      0.53        28

    accuracy                           0.36        78
   macro avg       0.18      0.50      0.26        78
weighted avg       0.13      0.36      0.19        78

              precision    recall  f1-score   support

           0       0.64      1.00      0.78        50
           1       0.00      0.00      0.00        28

    accuracy                           0.64        78
   macro avg       0.32      0.50      0.39        78
weighted avg       0.41      0.64      0.50        78

/usr/local/lib/python3.9/dist-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning: Precision and F-scor
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.9/dist-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning: Precision and F-scor
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.9/dist-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning: Precision and F-scor
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.9/dist-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning: Precision and F-scor
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.9/dist-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning: Precision and F-scor
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.9/dist-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning: Precision and F-scor
  _warn_prf(average, modifier, msg_start, len(result))
```

a. The decision tree and neural network algorithms performed better than the logistic regression algorithm.

b. The decision tree algorithm had the highest accuracy and precision for the high mpg class, but the lowest recall for the low mpg class. The neural network algorithms had similar accuracy and recall for both classes, but the second neural network had slightly higher precision for the high mpg class.

c. The better-performing algorithm might have outperformed the other because it was able to capture the non-linear relationships between the input features and the target variable more effectively. The decision tree algorithm is able to split the data into regions based on the input features, which can capture non-linear decision boundaries. The neural network algorithms use non-linear activation functions and multiple layers of neurons to capture complex interactions between the input features.

d. My experience using sklearn was very positive compared to using R. I found the sklearn library to be more intuitive and easier to use than the equivalent functions in R. Additionally, the sklearn library has better integration with other popular Python libraries like pandas and numpy, which makes data analysis and machine learning workflows more streamlined. However, this is a matter of personal preference and other data scientists may have different opinions.

✓ 0s completed at 11:16 AM