```
# Reads the Auto data
import pandas as pd

# use pandas to read the data
auto_df = pd.read_csv("Auto.csv")

# output the first few rows
print(auto_df.head())

# 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)
```

```
# Data exploration with code
# describe() method on the 'mpg' column with comments indicating the range and average of each column
mpg_desc = auto_df['mpg'].describe()
print("MPG column description:")
print(mpg_desc)
print(f"MPG column range: {mpg_desc['min']} to {mpg_desc['max']}")
print(f"Average MPG: {mpg_desc['mean']}")

# describe() method on the 'weight' column  with comments indicating the range and average of each column
weight_desc = auto_df['weight'].describe()
print("\nWeight column description:")
print(weight_desc)
print(f"Weight column range: {weight_desc['min']} to {weight_desc['max']}")
print(f"Average weight: {weight_desc['mean']}")

# describe() method on the 'year' column with comments indicating the range and average of each column
year_desc = auto_df['year'].describe()
print("\nYear column description:")
print(year_desc)
print(f"Year column range: {year_desc['min']} to {year_desc['max']}")
print(f"Average year: {year_desc['mean']}")
```

```
MPG column description:
count    392.000000
mean      23.445918
std        7.805007
min        9.000000
25%       17.000000
50%       22.750000
75%       29.000000
max       46.600000
Name: mpg, dtype: float64
------------------------------
MPG column range: 9.0 to 46.6
Average MPG: 23.445918367346938

Weight column description:
count     392.000000
mean     2977.584184
std       849.402560
min      1613.000000
25%      2225.250000
50%      2803.500000
75%      3614.750000
```

```
max      5140.000000
Name: weight, dtype: float64
------------------------------
Weight column range: 1613.0 to 5140.0
Average weight: 2977.5841836734694

Year column description:
count    390.000000
mean      76.010256
std        3.668093
min       70.000000
25%       73.000000
50%       76.000000
75%       79.000000
max       82.000000
Name: year, dtype: float64
------------------------------
Year column range: 70.0 to 82.0
Average year: 76.01025641025642
```

```
# Explore data types
# check the data types of all columns
print("Data types before conversion:")
print(auto_df.dtypes)

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

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

# verify the changes with the dtypes attribute
print("\nData types after conversion:")
print(auto_df.dtypes)
```

```
Data types before conversion:
mpg             float64
cylinders         int64
displacement    float64
horsepower        int64
weight            int64
acceleration    float64
year            float64
origin            int64
name             object
dtype: object

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

```
# Deal with NAs
# delete rows with NAs
auto_df.dropna(inplace=True)

# output the new dimensions
print("New dimensions:")
print(auto_df.shape)
```

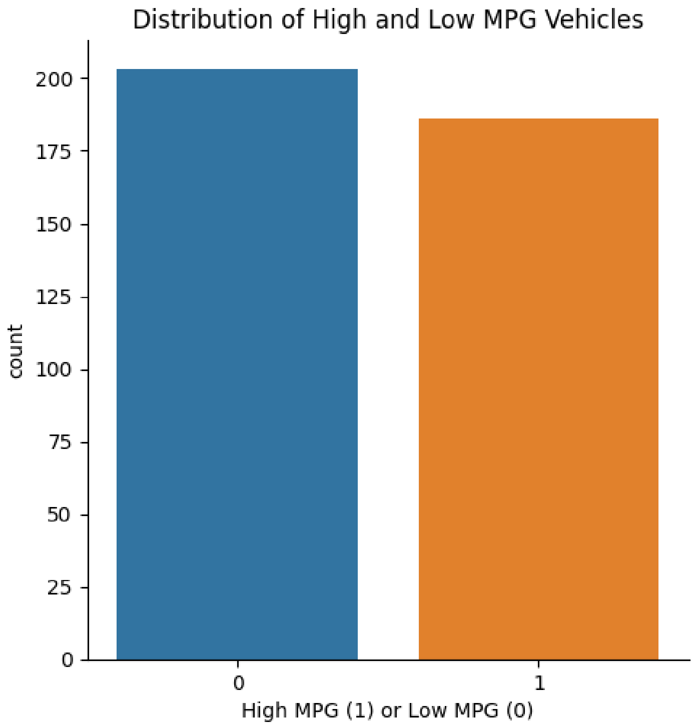
```
New dimensions:
(389, 9)
```

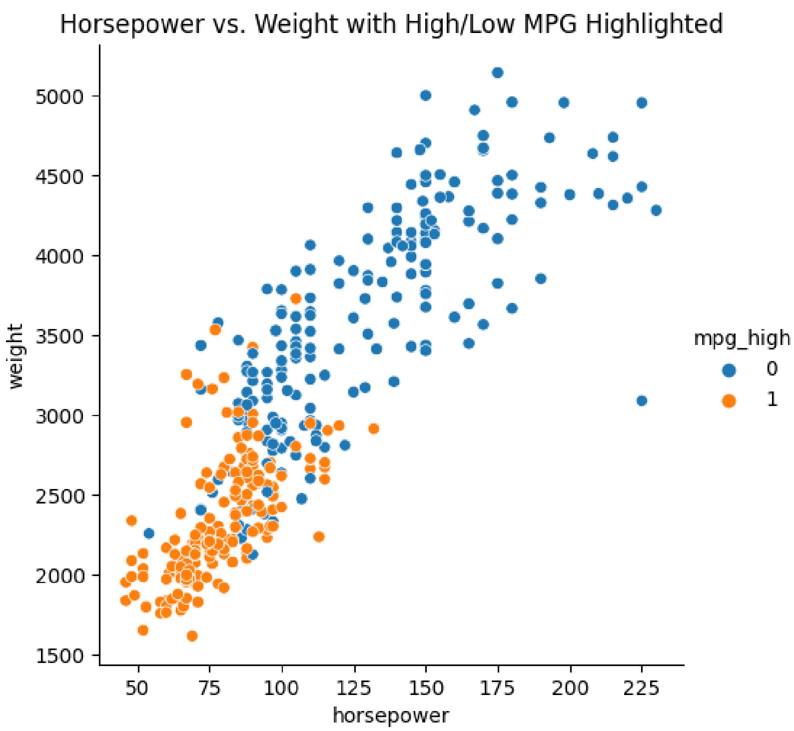
```
# Modify columns
# make a new column, mpg_high, and make it categorical:
auto_df['mpg_high'] = (auto_df['mpg'] > auto_df['mpg'].mean()).astype(int)
```

```
auto_df['mpg_high'] = auto_df['mpg_high'].astype('category')

# delete the mpg and name columns (delete mpg so the algorithm doesn't just learn to predict mpg_high from mpg)
auto_df.drop(['mpg', 'name'], axis=1, inplace=True)

# 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
```

```
# Data exploration with graphs
import seaborn as sns
import matplotlib.pyplot as plt

# seaborn catplot on the mpg_high column
sns.catplot(data=auto_df, x='mpg_high', kind='count')
plt.title('Distribution of High and Low MPG Vehicles')
plt.xlabel('High MPG (1) or Low MPG (0)')
plt.show()

# seaborn relplot with horsepower on the x axis, weight on the y axis, setting hue or style to mpg_high
sns.relplot(data=auto_df, x='horsepower', y='weight', hue='mpg_high')
plt.title('Horsepower vs. Weight with High/Low MPG Highlighted')
plt.show()

# seaborn boxplot with mpg_high on the x axis and weight on the y axis
sns.boxplot(data=auto_df, x='mpg_high', y='weight')
plt.title('Distribution of Vehicle Weights by High/Low MPG')
plt.xlabel('High MPG (1) or Low MPG (0)')
plt.ylabel('Vehicle Weight')
plt.show()

# for each graph, write a comment indicating one thing you learned about the data from the graph
# Graph 1 - There are about the same amount of distribution with slightly more low than high
# Graph 2 - The distribution is linear with both weight and horse power.
# Graph 3 - In terms of distribution, the low has a higher average and range.
```





Distribution of Vehicle Weights by High/Low MPG

```
# Train/test split
from sklearn.model_selection import train_test_split

# 80/20
# use seed 1234 so we all get the same results
# train /test X data frames consists of all remaining columns except mpg_high
X = auto_df.drop(['mpg_high'], axis=1)
y = auto_df['mpg_high']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=1234)

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

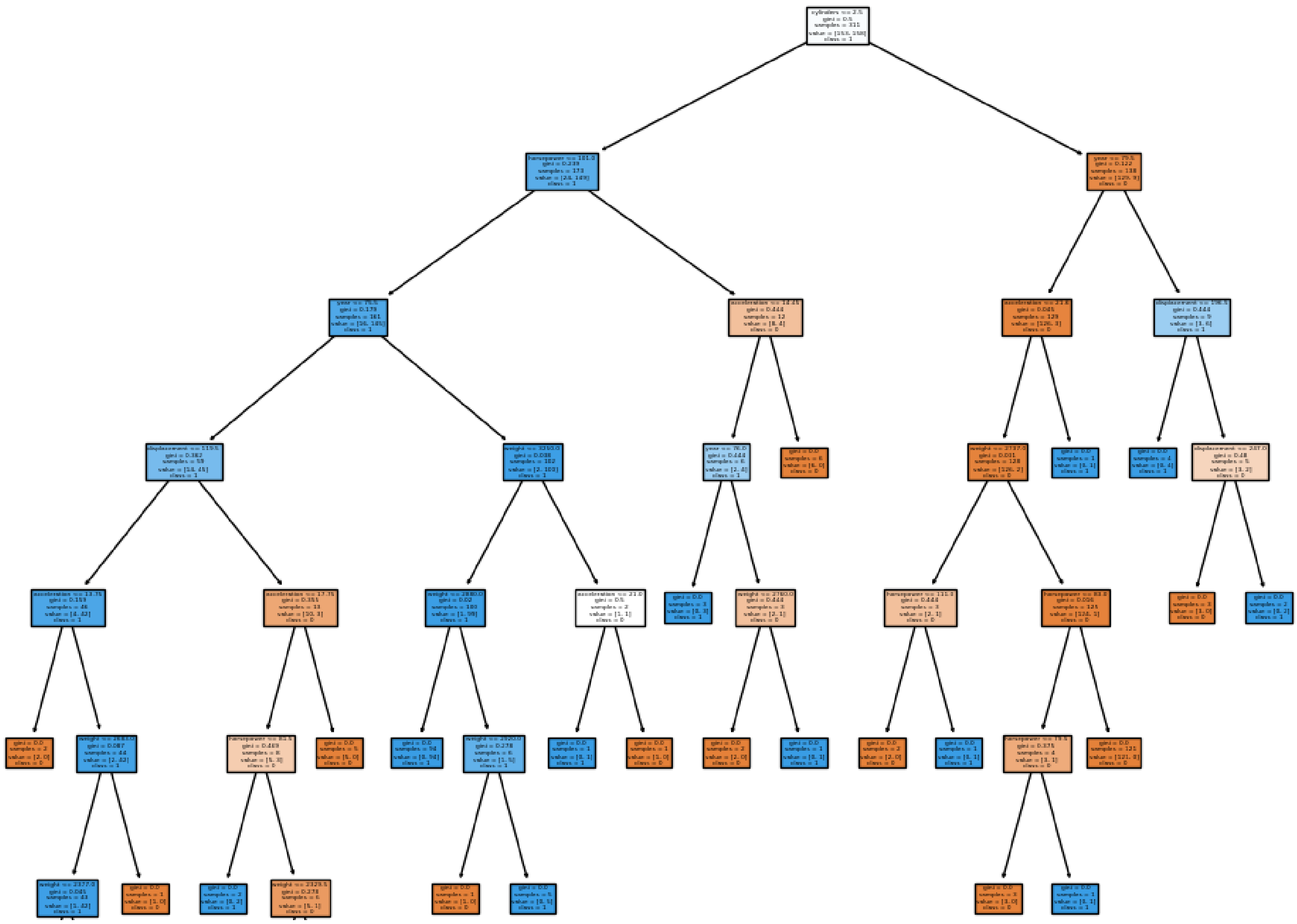
```
Train dimensions: (311, 7) (311,)
Test dimensions: (78, 7) (78,)
```

```
# Logistic Regression
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
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(
```

```
# Decision Tree
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import classification_report
from sklearn.tree import plot_tree
import matplotlib.pyplot as plt

# train a decision tree
tree = DecisionTreeClassifier(random_state=1234)
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
fig, ax = plt.subplots(figsize=(12, 12))
plot_tree(tree, filled=True, feature_names=X.columns, class_names=['0', '1'], ax=ax)
plt.show()
```

```
              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
```



```
# Neural Network (15 points)
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import classification_report

# train a neural network, choosing a network topology of your choice
nn1 = MLPClassifier(hidden_layer_sizes=(10, 10), activation='relu', solver='adam', max_iter=1000, random_state=1234)
nn1.fit(X_train, y_train)

# test and evaluate
y_pred1 = nn1.predict(X_test)

# Evaluate model
print("First Neural Network:")
print(classification_report(y_test, y_pred1))

# train a second network with a different topology and different settings
nn2 = MLPClassifier(hidden_layer_sizes=(20, 20), activation='logistic', solver='lbfgs', max_iter=5000, random_state=12
nn2.fit(X_train, y_train)

# test and evaluate
y_pred2 = nn2.predict(X_test)

# Evaluate model
print("Second Neural Network:")
print(classification_report(y_test, y_pred2))
```

```
# compare the two models and why you think the performance was same/different
# there are different iterrations showing and trying different methods and therefore, they differ in results.
```

```
First Neural Network:
              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

Second Neural Network:
              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

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

```
#                               Accuracy    Recall (0)  Recall (1)  Precision (0)   Precision (1)
#Logistic Regression               0.86      0.80          0.96        0.98          0.73
#Decision Tree                   0.92        0.92          0.93        0.96          0.87
#Neural Network (ReLU)           0.64        1.00          0.00        0.64          0.00
#Neural Network (logistic)       0.36        0.00          1.00        0.00          0.36
# From the graph we can see that the decision tree had the best acuracy, followed by the logistic regression,
# then the neural network RELU, and finally the Neural network logistic.
#
# Fortunately for us, decision tree didn't over fit the data causing performance, therefore it was the best.
# The data is linear, and therefore the logistic regression represents that.
# As for Neural networks, I believe with more tuning it would perform better as they often do.

# I think I might be bias on which I prefer, but I strongly prefer skit-learning. This is becuase at my
# previous internship, I was taught to work with it. I was also taught to work with the libraries used in the project.
# That being said, I do still like R, as I find it easier to do things, but skit-learning, is by far my favorite.
```