Import statements

```
import pandas as pd
import numpy as np
import seaborn as sb
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, confusio
from sklearn.tree import DecisionTreeClassifier
from sklearn import tree
```

1. Read the Auto data

```
# a. use pandas to read the data (available on Piazza)
df = pd.read_csv('Auto.csv')
```

```
# b. print the first few rows
print(df.head())
```

```
    mpg  cylinders  displacement  ...  year  origin                       name
0  18.0          8         307.0  ...  70.0       1  chevrolet chevelle malibu
1  15.0          8         350.0  ...  70.0       1          buick skylark 320
2  18.0          8         318.0  ...  70.0       1         plymouth satellite
3  16.0          8         304.0  ...  70.0       1              amc rebel sst
4  17.0          8         302.0  ...  70.0       1                ford torino

[5 rows x 9 columns]
```

```
# c. print the dimensions of the data
print('dimensions of data frame:', df.shape)
```

```
dimensions of data frame: (392, 9)
```

2. Some data exploration with code

```
# a. use describe() on the mpg, weight, and year columns
print('describe mpg, weight, and year:')
print()
print(df.loc[:, ['mpg', 'weight', 'year']].describe())
```

```
describe mpg, weight, and year:

              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
```

b. write comments indicating the range and average of each column

- mpg:
    - min- 9.0
    - max- 46.6
    - range- 37.6
    - average- 23.445918
- weight:
    - min- 1613.0
    - max- 5140.0
    - range- 3527.0
    - average- 2977.584184
- year
    - min- 70.0
    - max- 82.0
    - range- 12.0
    - average- 76.010256

3. Explore data types

```
# a. check the data types of all columns
print(df.dtypes)
```

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

```
# b. change the cylinders column to categorical (use cat.codes)
df.cylinders = df.cylinders.astype('category').cat.codes
```

```
# c. change the origin column to categorical (don’t use cat.codes)
df.origin = df.origin.astype('category')
```

```
# d. verify the changes with the dtypes attribute
print(df.dtypes)
```

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

4. Deal with NAs

```
# a. delete rows with NAs
df = df.dropna()
```

```
# b. print the new dimensions
print('dimensions of data frame:', df.shape)
```

```
dimensions of data frame: (389, 9)
```

5. Modify columns

```
# a. make a new column, mpg_high, which is categorical:
  # i.  the column == 1 if mpg > average mpg, else == 0
df['mpg_high'] = np.where(df['mpg'] > df['mpg'].mean(), 1, 0)
```

```
# b. delete the mpg and name columns
df = df.drop(columns = ['mpg', 'name'])
```

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

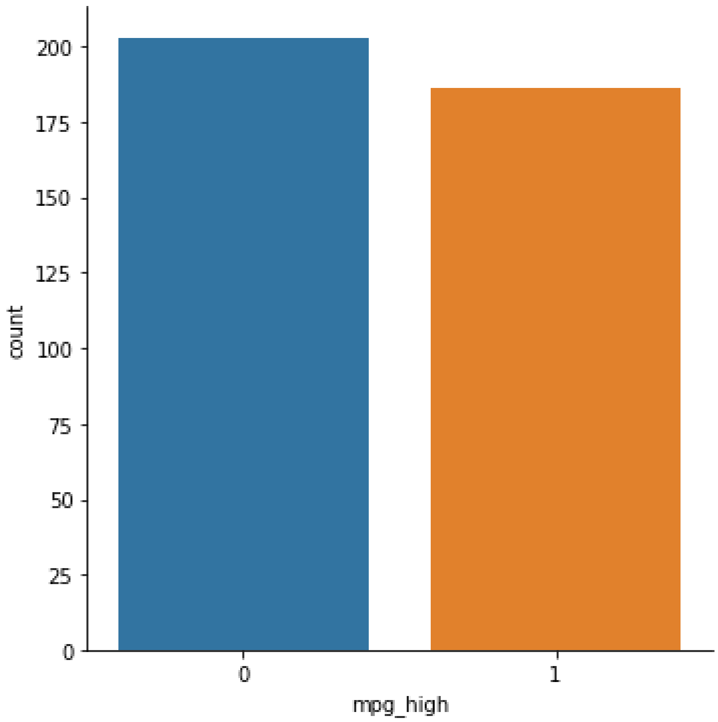
```
   cylinders  displacement  horsepower  ...  year  origin  mpg_high
0          4         307.0         130  ...  70.0       1         0
1          4         350.0         165  ...  70.0       1         0
2          4         318.0         150  ...  70.0       1         0
3          4         304.0         150  ...  70.0       1         0
6          4         454.0         220  ...  70.0       1         0
```

```
[5 rows x 8 columns]
```

6. Data exploration with graphs

```
# a. seaborn catplot on the mpg_high column
sb.catplot(x = 'mpg_high', kind = 'count', data = df)
```

```
<seaborn.axisgrid.FacetGrid at 0x7f21a65c51d0>
```

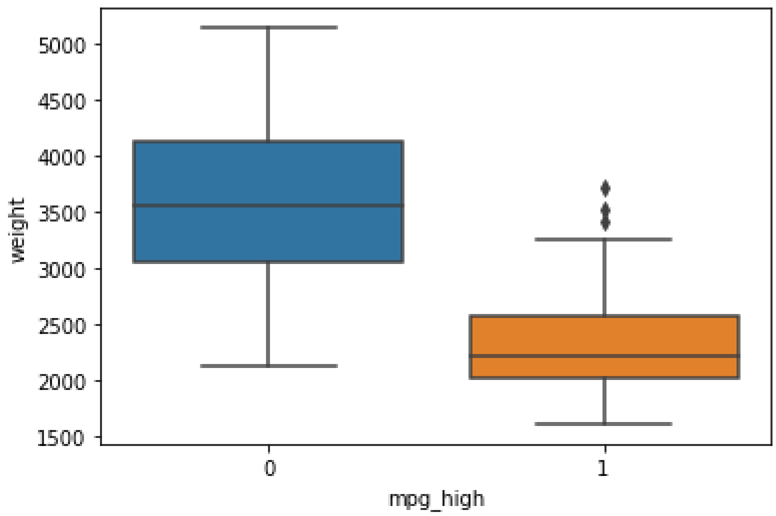



```
# b. seaborn relplot with horsepower on the x axis, weight on the y axis, setting hue or styl
sb.relplot(x = 'horsepower', y = 'weight', data = df, hue = df.mpg_high, style = df.mpg_high)
```

```
<seaborn.axisgrid.FacetGrid at 0x7f21a64d4cd0>
```


```
# c. seaborn boxplot with mpg_high on the x axis and weight on the y axis
sb.boxplot('mpg_high', y = 'weight', data = df)
```

```
/usr/local/lib/python3.7/dist-packages/seaborn/_decorators.py:43: FutureWarning: Pass tl
  FutureWarning
<matplotlib.axes._subplots.AxesSubplot at 0x7f21a6446c10>
```



d. for each graph, write a comment indicating one thing you learned about the data from the graph

- The catplot graph shows that there is a fairly even distribution between the mpg of vehicles that are above and below average in the data set.
- The relplot graph shows that horsepower and weight are positively correlated.
- The boxplot graph shows that there are a couple of outliers in weight for some vehicles with an mpg that is above average.

7. Train/test split

```
# a. 80/20
# b. use seed 1234 so we all get the same results
# c. train /test X data frames consists of all remaining columns except mpg_high
X = df.iloc[:, 0:6]
y = df.iloc[:, 7]
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state = 123
```

```
# d. print the dimensions of train and test
print('train size:', X_train.shape)
print('test size:', X_test.shape)
```

```
print('test size: ', X_test.shape)
```

```
train size: (311, 6)
test size: (78, 6)
```

8. Logistic Regression

```
# a. train a logistic regression model using solver lbfgs
clf = LogisticRegression(solver = 'lbfgs', max_iter = 500000, random_state = 1234)
clf.fit(X_train, y_train)
clf.score(X_train, y_train)
```

```
0.8938906752411575
```

```
# b. test and evaluate
pred = clf.predict(X_test)
print('accuracy score: ', accuracy_score(y_test, pred))
print('precision score: ', precision_score(y_test, pred, average = 'macro'))
print('recall score: ', recall_score(y_test, pred, average = 'macro'))
print('f1 score: ', f1_score(y_test, pred, average = 'macro'))
print()
print('confusion matrix:')
confusion_matrix(y_test, pred)
```

```
accuracy score:  0.8974358974358975
precision score:  0.8856951871657754
recall score:  0.9121428571428571
f1 score:  0.8929306794783802

confusion matrix:
array([[43,  7],
       [ 1, 27]])
```

```
# c. print metrics using the classification report
print(classification_report(y_test, pred))
```

```
              precision    recall  f1-score   support

           0       0.98      0.86      0.91        50
           1       0.79      0.96      0.87        28

    accuracy                           0.90        78
   macro avg       0.89      0.91      0.89        78
weighted avg       0.91      0.90      0.90        78
```

9. Decision Tree

```
# a. train a decision tree
```

```
clf2 = DecisionTreeClassifier(random_state = 1234)
clf2.fit(X_train, y_train)
```

```
DecisionTreeClassifier(ccp_alpha=0.0, class_weight=None, criterion='gini',
                       max_depth=None, max_features=None, max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, presort='deprecated',
                       random_state=1234, splitter='best')
```

```
# b. test and evaluate
pred2 = clf2.predict(X_test)
print('accuracy score: ', accuracy_score(y_test, pred2))
print('precision score: ', precision_score(y_test, pred2, average = 'macro'))
print('recall score: ', recall_score(y_test, pred2, average = 'macro'))
print('f1 score: ', f1_score(y_test, pred2, average = 'macro'))
print()
print('confusion matrix:')
confusion_matrix(y_test, pred2)
```

```
accuracy score:  0.9102564102564102
precision score:  0.8980782429649966
recall score:  0.9142857142857144
f1 score:  0.9045954918748909

confusion matrix:
array([[45,  5],
       [ 2, 26]])
```

```
# c. print the classification report metrics
print(classification_report(y_test, pred2))
```

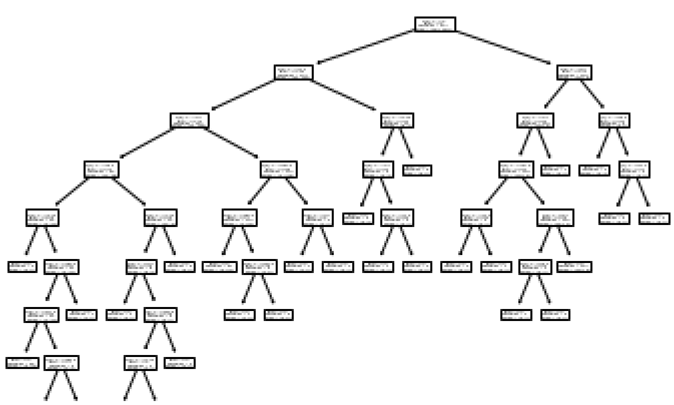
```
              precision    recall  f1-score   support

           0       0.96      0.90      0.93        50
           1       0.84      0.93      0.88        28

    accuracy                           0.91        78
   macro avg       0.90      0.91      0.90        78
weighted avg       0.91      0.91      0.91        78
```

```
# d. plot the tree (optional, see: https://scikit-learn.org/stable/modules/tree.html)
tree.plot_tree(clf2)
```

```
Text(19.694117647058825, 12.079999999999984, 'gini = 0.0\nsamples = 1\nvalue = [1, 0]',
Text(39.38823529411765, 12.079999999999984, 'gini = 0.0\nsamples = 4\nvalue = [0, 4]'),
Text(39.38823529411765, 60.400000000000006, 'gini = 0.0\nsamples = 1\nvalue = [1, 0]'),
Text(78.7764705882353, 108.72, 'X[4] <= 17.75\ngini = 0.355\nsamples = 13\nvalue = [10,
Text(68.92941176470589, 84.56, 'X[2] <= 81.5\ngini = 0.469\nsamples = 8\nvalue = [5, 3
Text(59.082352941176474, 60.400000000000006, 'gini = 0.0\nsamples = 2\nvalue = [0, 2]'
Text(78.7764705882353, 60.400000000000006, 'X[3] <= 2329.5\ngini = 0.278\nsamples = 6\n
Text(68.92941176470589, 36.24000000000001, 'X[4] <= 14.75\ngini = 0.5\nsamples = 2\nva
Text(59.082352941176474, 12.079999999999984, 'gini = 0.0\nsamples = 1\nvalue = [1, 0]'
Text(78.7764705882353, 12.079999999999984, 'gini = 0.0\nsamples = 1\nvalue = [0, 1]'),
Text(88.62352941176471, 36.24000000000001, 'gini = 0.0\nsamples = 4\nvalue = [4, 0]'),
Text(88.62352941176471, 84.56, 'gini = 0.0\nsamples = 5\nvalue = [5, 0]'),
Text(137.85882352941178, 132.88, 'X[3] <= 3250.0\ngini = 0.038\nsamples = 102\nvalue =
Text(118.16470588235295, 108.72, 'X[3] <= 2880.0\ngini = 0.02\nsamples = 100\nvalue =
Text(108.31764705882354, 84.56, 'gini = 0.0\nsamples = 94\nvalue = [0, 94]'),
Text(128.01176470588237, 84.56, 'X[3] <= 2920.0\ngini = 0.278\nsamples = 6\nvalue = [1,
Text(118.16470588235295, 60.400000000000006, 'gini = 0.0\nsamples = 1\nvalue = [1, 0]'
Text(137.85882352941178, 60.400000000000006, 'gini = 0.0\nsamples = 5\nvalue = [0, 5]'
Text(157.5529411764706, 108.72, 'X[5] <= 77.5\ngini = 0.5\nsamples = 2\nvalue = [1, 1]
Text(147.7058823529412, 84.56, 'gini = 0.0\nsamples = 1\nvalue = [1, 0]'),
Text(167.4, 84.56, 'gini = 0.0\nsamples = 1\nvalue = [0, 1]'),
Text(196.94117647058823, 157.04, 'X[4] <= 14.45\ngini = 0.444\nsamples = 12\nvalue = [
Text(187.09411764705882, 132.88, 'X[5] <= 76.0\ngini = 0.444\nsamples = 6\nvalue = [2,
Text(177.24705882352941, 108.72, 'gini = 0.0\nsamples = 3\nvalue = [0, 3]'),
Text(196.94117647058823, 108.72, 'X[2] <= 107.5\ngini = 0.444\nsamples = 3\nvalue = [2,
Text(187.09411764705882, 84.56, 'gini = 0.0\nsamples = 1\nvalue = [0, 1]'),
Text(206.78823529411767, 84.56, 'gini = 0.0\nsamples = 2\nvalue = [2, 0]'),
Text(206.78823529411767, 132.88, 'gini = 0.0\nsamples = 6\nvalue = [6, 0]'),
Text(285.56470588235294, 181.2, 'X[5] <= 79.5\ngini = 0.122\nsamples = 138\nvalue = [1
Text(265.8705882352941, 157.04, 'X[4] <= 21.6\ngini = 0.045\nsamples = 129\nvalue = [1
Text(256.02352941176474, 132.88, 'X[3] <= 2737.0\ngini = 0.031\nsamples = 128\nvalue =
Text(236.3294117647059, 108.72, 'X[2] <= 111.0\ngini = 0.444\nsamples = 3\nvalue = [2,
Text(226.4823529411765, 84.56, 'gini = 0.0\nsamples = 2\nvalue = [2, 0]'),
Text(246.1764705882353, 84.56, 'gini = 0.0\nsamples = 1\nvalue = [0, 1]'),
Text(275.71764705882356, 108.72, 'X[2] <= 83.0\ngini = 0.016\nsamples = 125\nvalue = [
Text(265.8705882352941, 84.56, 'X[1] <= 225.0\ngini = 0.375\nsamples = 4\nvalue = [3,
Text(256.02352941176474, 60.400000000000006, 'gini = 0.0\nsamples = 1\nvalue = [0, 1]'
Text(275.71764705882356, 60.400000000000006, 'gini = 0.0\nsamples = 3\nvalue = [3, 0]'
Text(285.56470588235294, 84.56, 'gini = 0.0\nsamples = 121\nvalue = [121, 0]'),
Text(275.71764705882356, 132.88, 'gini = 0.0\nsamples = 1\nvalue = [0, 1]'),
Text(305.25882352941176, 157.04, 'X[1] <= 196.5\ngini = 0.444\nsamples = 9\nvalue = [3,
Text(295.4117647058824, 132.88, 'gini = 0.0\nsamples = 4\nvalue = [0, 4]'),
Text(315.1058823529412, 132.88, 'X[1] <= 247.0\ngini = 0.48\nsamples = 5\nvalue = [3,
Text(305.25882352941176, 108.72, 'gini = 0.0\nsamples = 3\nvalue = [3, 0]'),
Text(324.95294117647063, 108.72, 'gini = 0.0\nsamples = 2\nvalue = [0, 2]')]
```



10. Analysis

a. which algorithm performed better?

The following were the metrics of each algorithm:

- Logistic Regression:
    - accuracy- 0.8974358974358975
    - precision- 0.8856951871657754
    - recall- 0.9121428571428571
    - f1- 0.8929306794783802
- Decision Tree:
    - accuracy- 0.9102564102564102
    - precision- 0.8980782429649966
    - recall- 0.9142857142857144
    - f1- 0.9045954918748909

Based on the metrics above , I believe the Decision Tree algorithm performed better than the Logistic Regression algorithm.

b. compare accuracy, recall, and precision metrics by class

The following are the metrics based on class for each algorithm:

- Logistic Regression:

```
                   precision    recall  f1-score   support

     0             0.98      0.86      0.91        50
     1             0.79      0.96      0.87        28

     accuracy                          0.90        78
     macro avg     0.89      0.91      0.89        78
     weighted avg  0.91      0.90      0.90        78
```

- Decision Tree:

```
                  precision    recall  f1-score   support

           0       0.96      0.90      0.93        50
           1       0.84      0.93      0.88        28

    accuracy                           0.91        78
   macro avg       0.90      0.91      0.90        78
weighted avg       0.91      0.91      0.91        78
```

We see that the Decision Tree algorithm and Logistic Regression algorithm both have a precision of 0.98 for the negative class, while Decision Tree has a higher precision in the positive class, with a precision of 0.84 against Logistic Regression's precision of 0.79. Logistic Regression's recall in the positive class is 0.96 is greater than Decision Tree's 0.93, while Decision Tree's recall in the negative class of 0.90 is greater than Logistic Regression's 0.86. Lastly, Decision Tree has a higher f1-score in the positive class of 0.88 compared to Logistic Regression's at 0.87, and Decision Tree also has a higher f1-score in the negative class of 0.93 compared to Logistic Regression's at 0.91. In most metrics of both classes, Decision Tree performs either better or the same as Logistic Regression, which is further reiterated by Decision Tree's higher accuracy than Logistic Regression's.

c. give your analysis of why the better-performing algorithm might have outperformed the other

Decision Trees perform better than Logistic Regression when the data is not linearly separable, as well as when there is a lot of categorical data. Our dataset used a couple of categorical predictors such as cylinders (a numerical category) and origin, which could have been a reason why the Decision Tree algorithm performed better. Furthermore, the relplot from our data exploration section showed some possibility for non-linear separatability, which also could be an advantage for Decision Trees and explain why it outperformed Logistic Regression.