Kelly Trinh
Dr. Karen Mazidi
CS 4375.004

Read in the data

```
import pandas as pd
import numpy as np
import io
import seaborn as sb
```

For this assignment, I read in the file from the desktop, as seen below

```
from google.colab import files

uploaded = files.upload()
```

Choose Files No file chosen Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable

```
Saving Auto.csv to Auto.csv
```

```
# Read in the file, print the first few rows, and display dimensions of the data frame

df = pd.read_csv(io.BytesIO(uploaded['Auto.csv']))
```

Data Exploration and Cleaning

```
print(df.head())
print('\nDimensions of data frame: ', df.shape)
df[["mpg", "weight", "year"]].describe()
```

```
    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
```

To undo cell deletion use ⌘/Ctrl+M Z or the Undo option in the Edit menu

```
1       1          buick skylark 320
2       1         plymouth satellite
3       1              amc rebel sst
4       1                ford torino

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

	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

The averages for mpg, weight, and year are 23.45, 2977.58, and 76.01 respectively.
The ranges for mpg, weight, and year are 37, 3527, and 12 respectively.

```
print("data types of all columns:\n", df.dtypes)
```

```
# changing dtype using cat.codes
df["cylinders"] = df["cylinders"].astype('category').cat.codes

#changing dtype without using cat.codes
df["cylinders"] = df["cylinders"].astype('category')

print("\nVerifying new data type for cylinders and origin")
print("data type of cylinders: ", df["cylinders"].dtypes)
print("data type of origin: ", df["origin"].dtypes)
```

```
data types of all columns:
 mpg             float64
cylinders          int64
displacement    float64
horsepower         int64
weight             int64
acceleration    float64
year            float64
origin             int64
name              object
dtype: object

Verifying new data type for cylinders and origin
data type of cylinders:  category
data type of origin:  int64
```

Below, I printed the sum of how many NAs there were per attribute. Then, I dropped the NA rows and printed the new dimensions of the data frame.

```
# Print NAs per attribute
print("NAs per attribute:\n", df.isnull().sum())

# Drop NA rows and print new dimensions of data frame
df = df.dropna()
print('\nNew dimensions of data frame after dropping NA columns: ', df.shape)
```

```
NAs per attribute:
 mpg             0
cylinders       0
displacement    0
horsepower      0
weight          0
acceleration    1
year            2
origin          0
name            0
dtype: int64
```



```
(389, 9)
```

Then, I created a new column called "mpg_high" using two categories, 1 and 0. I also dropped the "mpg" and "name" columns and printed the first few rows of the new data frame.

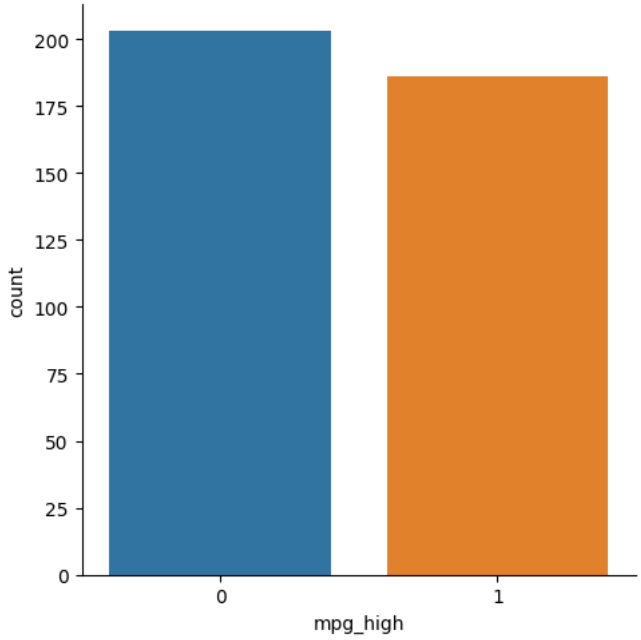
```
# create new mpg_high column and add to data frame
df["mpg_high"] = np.where(df["mpg"] > np.mean(df["mpg"]), 1, 0)

# drop columns
df = df.drop(columns=["mpg", "name"])

df.head()
```

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

Cat Plot

```
print("cat plot with \"mpg_high\"", sb.catplot(x="mpg_high", kind='count', data=df))
```

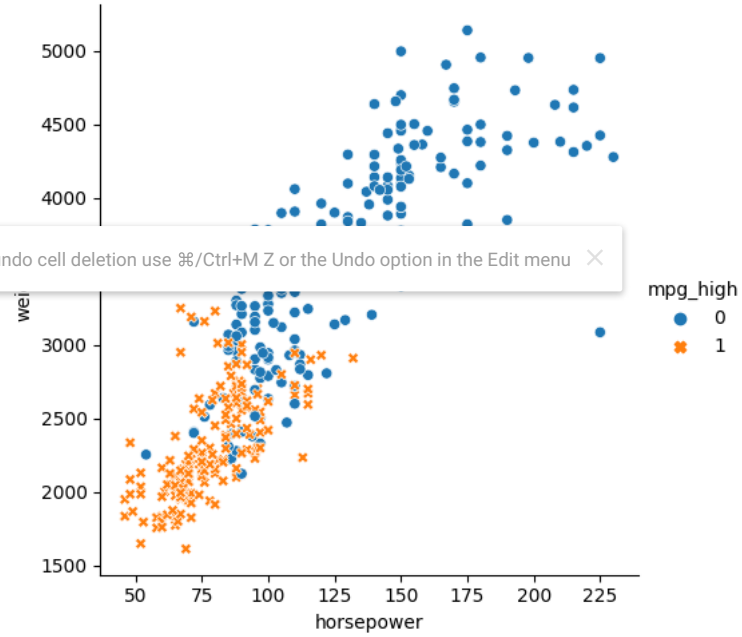
```
cat plot with "mpg_high" <seaborn.axisgrid.FacetGrid object at 0x7f9a6a0ae6d0>
```



Rel Plot

```
print("rel plot with \"horsepower\" on x axis and \"weight\" on y axis and hue/style set to \"mpg_high\"", sb.relplot(x="horsepowe
```

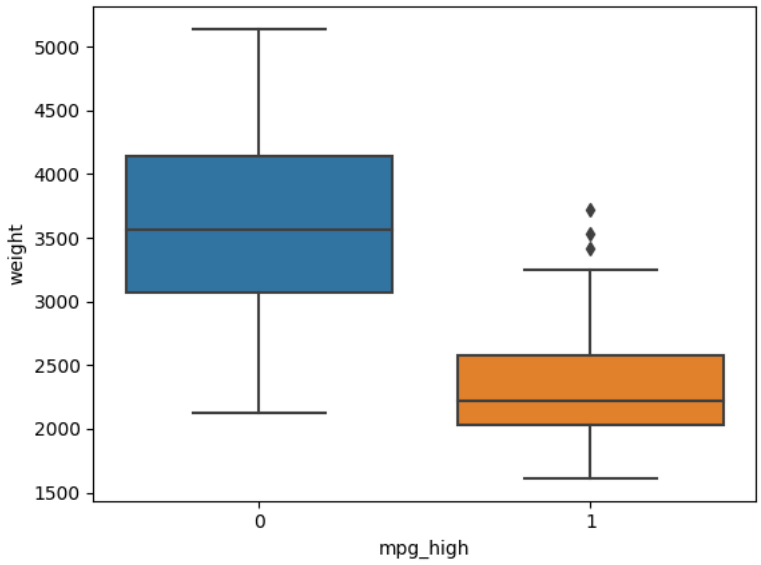
```
rel plot with "horsepower" on x axis and "weight" on y axis and hue/style set to "mpg_high" <seaborn.axisgrid.FacetGrid objec
```



Box Plot

```
print("box plot with \"mpg_high\" on x axis and \"weight\" on y axis", sb.boxplot(x="mpg_high", y="weight", data=df))
```

```
box plot with "mpg_high" on x axis and "weight" on y axis Axes(0.125,0.11;0.775x0.77)
```



Graph Analysis

From the first graph, which is the box plot, we can see that there are more cars that have a lower mpg (lower than the average) in comparison to cars that have a higher mpg (higher than the average).

From the relplot, we can see that cars that have higher mpg also tend to have lower horsepower and are of a lower weight. In contrast, cars that have a lower mpg have higher horsepower and have a greater weight.

Lastly, from the boxplot, we can see that cars with the higher mpg tend to be of lower weight, but we also have to account for the fact that there are more outliers for these types of cars. However, cars with lower mpg are of higher weight, yet they have no outliers to account for.

Splitting the Data Frame

Below, I split the data frame into 80/20 train/test sets. I also scaled the X data and printed the dimensions of the train/test sets

```
from sklearn.model_selection import train_test_split
from sklearn import preprocessing

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=1234)

scaler = preprocessing.StandardScaler().fit(X_train)
X_scaled_train = scaler.transform(X_train)
X_scaled_test = scaler.transform(X_test)

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

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

Logistic Regression

Below, I performed Logistic Regression on the data set. I ouputted the accuracy, classification report, and confusion matrix.

```
# Logistic Regression
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report

logreg = LogisticRegression()

logreg.fit(X_scaled_train, y_train)
print("correlation for train sets: ", logreg.score(X_scaled_train, y_train))

predLR = logreg.predict(X_scaled_test)

print("accuracy score: ", accuracy_score(y_test, predLR))
print(classification_report(y_test, predLR, labels=[0,1]))
print("confusion matrix:\n", confusion_matrix(y_test, predLR))
```

```
correlation for train sets:  0.8971061093247589
accuracy score:  0.8974358974358975
              precision    recall  f1-score   support

           0       1.00      0.84      0.91        50
           1       0.78      1.00      0.88        28

    accuracy                           0.90        78
   macro avg       0.89      0.92      0.89        78
weighted avg       0.92      0.90      0.90        78

confusion matrix:
 [[42  8]
 [ 0 28]]
```

Decision Tree

Below, I created a decision tree using the data set. I outputted the accuracy, classification report, and confusion matrix.

```
from sklearn.tree import DecisionTreeClassifier

dectree = DecisionTreeClassifier()
dectree.fit(X_train, y_train)

predDT = dectree.predict(X_test)


print("accuracy score: ", accuracy_score(y_test, predDT))
```

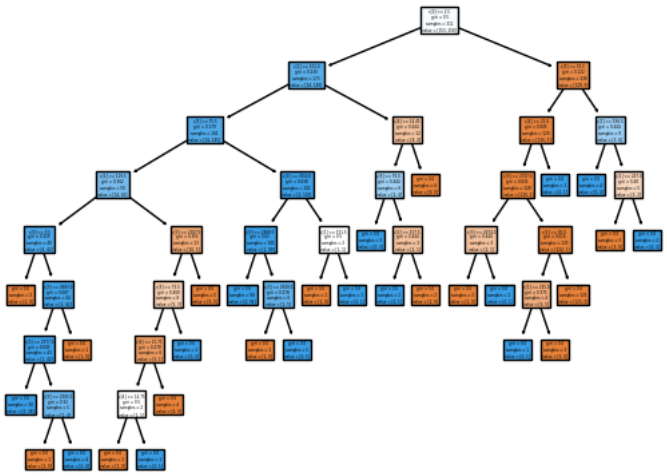


```
))
```

```
accuracy score:  0.9102564102564102
              precision    recall  f1-score   support

           0       0.98      0.88      0.93        50
           1       0.82      0.96      0.89        28

    accuracy                           0.91        78
   macro avg       0.90      0.92      0.91        78
weighted avg       0.92      0.91      0.91        78

confusion matrix:
 [[44  6]
 [ 1 27]]
```

I displayed the tree in the code, but a significant amount of text was also displayed that prevented the print-to-pdf from displaying the tree, so I added what the tree looked like in the text-box below, along with the code that I used to display the tree.

```
from sklearn import tree

tree.plot_tree(dectree)
```



Neural Networks

Below, I created the first neural network. Using the rule of thumb, we know that the number of hidden nodes is between 1-7 due to the number of predictors. For this network, I chose 5 hidden nodes arranged between two layers. I then outputted the accuracy, classification report, and confusion matrix.

```
from sklearn.neural_network import MLPClassifier

# 5 hidden nodes arranged between 2 layers
NN1 = MLPClassifier(solver="lbfgs", hidden_layer_sizes=(3, 2), max_iter=500, random_state=1234)
NN1.fit(X_scaled_train, y_train)

predNN1 = NN1.predict(X_scaled_test)

print("accuracy score: ", accuracy_score(y_test, predNN1))
print(classification_report(y_test, predNN1, labels=[0,1]))
print("confusion matrix:\n", confusion_matrix(y_test, predNN1))
```

```
accuracy score:  0.8846153846153846
              precision    recall  f1-score   support

           1       0.79      0.93      0.85        28

    accuracy                           0.88        78
   macro avg       0.87      0.89      0.88        78
weighted avg       0.90      0.88      0.89        78

confusion matrix:
 [[43  7]
 [ 2 26]]
```


Below, I created the second neural network. Using the same rule of thumb, for this neural network I instead chose 7 hidden nodes arranged between three layers. I then outputted the accuracy, classification report, and confusion matrix.

```
scaler = preprocessing.StandardScaler().fit(X_train)
X_scaled_train = scaler.transform(X_train)
X_scaled_test = scaler.transform(X_test)

# 7 hidden nodes arranged between 3 layers
NN2 = MLPClassifier(solver="lbfgs", hidden_layer_sizes=(2, 2, 3), max_iter=500, random_state=1234)
NN2.fit(X_scaled_train, y_train)

predNN2 = NN2.predict(X_scaled_test)

print("accuracy score: ", accuracy_score(y_test, predNN2))
print(classification_report(y_test, predNN2, labels=[0,1]))
print("confusion matrix:\n", confusion_matrix(y_test, predNN2))
```

```
accuracy score:  0.8974358974358975
              precision    recall  f1-score   support

           0       1.00      0.84      0.91        50
           1       0.78      1.00      0.88        28

    accuracy                           0.90        78
   macro avg       0.89      0.92      0.89        78
weighted avg       0.92      0.90      0.90        78

confusion matrix:
 [[42  8]
 [ 0 28]]
```

After completing both of the Neural Networks, we can see that the second one that I trained had a better accuracy score in comparison to the first neural network. Furthermore, the second neural network was better at predicting cars that had a lower mpg (lower than the average), as the precision was 1.00 in contrast to the first neural network's 0.96 precision. On the other hand, the first neural network was slightly better at predicting cars that had a higher mpg. For the first neural network, I used 5 hidden nodes arranged between 2 layers, and for the second neural network, I chose the toplogy to be 7 hidden nodes arranged between three layers. Since the second neural network performed slightly better, this means that 7 hidden nodes was a better guess than 5 for the network topology.

I think that the performance was different due to the topology that was chosen for each network. With the second neural network, since more layers were used, it took longer to train the network as it took more passes going back and forth through the layers, but there was an extra layer of training, which may allowed it to gain better accuracy in comparison to the first network.

Analysis:

a) Comparing all of the algorithms, they Logistic Regression, Decision Tree, and the second Neural Network all performed relatively similarly, but the first neural network that I trained performed the worst out of the four.

b) Although they all performed relatively similarly in terms of accuracy, their classification reports were somewhat different. Overall, the best precision came from the decision tree, which had a high value of predicting both low and high mpg cars. However, logistic regression was the best at predicting low mpg cars, but did not perform as well when predicting high mpg cars. This is a similar case to the second neural network. The first neural network had a high value for predicting low mpg cars, but a lower value for predicting higher mpg cars. In terms of recall, the first neural network and the decision tree had higher values for predicting low mpg cars, and logistic regression and the second neural network had lower values. However, it was the opposite in terms of predicting cars with high mpgs, meaning that the second neural network and the logistic regression had values of 1.00 for predicting high mpg cars and the first neural network and the decision tree had lower values for it.

c) In regards to the two neural networks, the second one I trained may have performed better due to there being more layers. Since there are ading a small amount of accuracy with a longer training time. Logistic was linearly separable. Based off of the graphs that we saw, there was a good separation between high mpg and low mpg based on the other attributes. Decision Tree might have not done as well because the data set was a bit larger.

d) After completing this assignment, I learned that I have a preference for using R. With Python, there are a lot of libraries to consider and import, and due to this, there are a lot of new and different functions to learn per library. The ones that I found particularly difficult were the mathematical ones, such as numpy. With R, there were a lot of base functions within R-Studio that could be used, and I felt like doing a lot of the math was much easier. This also may be biased because I am currently more familiar with R than I am with Python. With Python, though, I can see how powerful it is because it is much more broad in comparsion to R.