1. Read the Auto data

```
import numpy as np
import sklearn as sk
import pandas as pd
import seaborn as sb

# Loading csv file to data frame

df = pd.read_csv("sample_data/Auto.csv")
print(df.head())
print("\nDimensions of data frame:", 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

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

2. Data exploration with code

```
# Describing Columns

print("\n\nMPG\n", df['mpg'].describe(), "\n")
print("\n\nWeight\n", df['weight'].describe(), "\n")
print("\n\nYear\n", df['year'].describe(), "\n")

# Showing range and average of columns

print("        MPG      Weight       Year\n",

 "Range: 37       3527         12\n",

 "  Avg: 23.446   2977.584     76.010")
```

```
MPG
 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



Weight
 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



Year
 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

        MPG      Weight       Year
 Range: 37       3527         12
   Avg: 23.446   2977.584     76.010
```

3. Explore data types

```
# Changing Cylinders and Origin to a categorical type

print(df.info())

df["cylinders"] = df["cylinders"].astype('category').cat.codes

df["origin"] = df["origin"].astype('category')

print("\nCylinders:", df["cylinders"].dtypes,
    "\nOrigin:", df["origin"].dtypes)
```

```
<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
None

Cylinders: int8
Origin: category
```

4. Deal with NAs

```
# Remove NAs

print(df.isnull().sum())

df = df.dropna()

print("\nDimensions of data frame:", df.shape)
```

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

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

5. Modify columns

```
# Adding mpg_high

df["mpg_high"] = np.where(df.mpg > np.mean(df.mpg), 1, 0)
```

```
df["mpg_high"] = np.where(df.mpg > np.mean(df.mpg), 1, 0)

print(df)

# Dropping columns

del df["mpg"]

del df["name"]

print(df[:5])
```

```
      mpg  cylinders  displacement  horsepower  weight  acceleration  year  \
0    18.0          4         307.0         130    3504          12.0  70.0
1    15.0          4         350.0         165    3693          11.5  70.0
2    18.0          4         318.0         150    3436          11.0  70.0
3    16.0          4         304.0         150    3433          12.0  70.0
6    14.0          4         454.0         220    4354           9.0  70.0
..    ...        ...           ...         ...     ...           ...   ...
387  27.0          1         140.0          86    2790          15.6  82.0
388  44.0          1          97.0          52    2130          24.6  82.0
389  32.0          1         135.0          84    2295          11.6  82.0
390  28.0          1         120.0          79    2625          18.6  82.0
391  31.0          1         119.0          82    2720          19.4  82.0

    origin                       name  mpg_high
0        1  chevrolet chevelle malibu         0
1        1          buick skylark 320         0
2        1         plymouth satellite         0
3        1              amc rebel sst         0
6        1           chevrolet impala         0
..     ...                        ...       ...
387      1            ford mustang gl         1
388      2                  vw pickup         1
389      1              dodge rampage         1
390      1                ford ranger         1
391      1                 chevy s-10         1

[389 rows x 10 columns]
   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
```

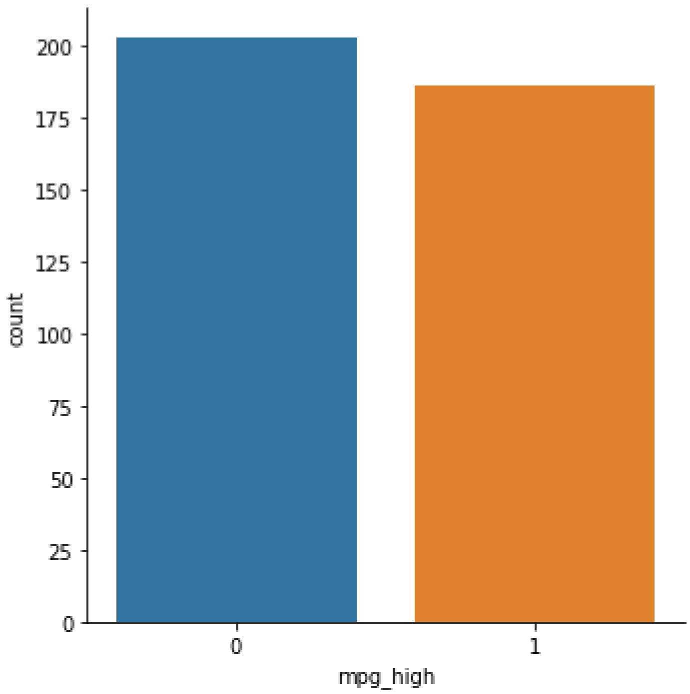
6. Data exploration with graphs

```
# Catplot

sb.catplot(x = "mpg_high", kind = "count", data = df)

print("\nThere are more cars below the mpg mean than above it.\n")
```

```
There are more cars below the mpg mean than above it.
```




```
# Relplot

sb.relplot(x = "horsepower", y = "weight", hue = "mpg_high", data = df)

print("\nYou need more horsepower to move heavier cars.\n")
```

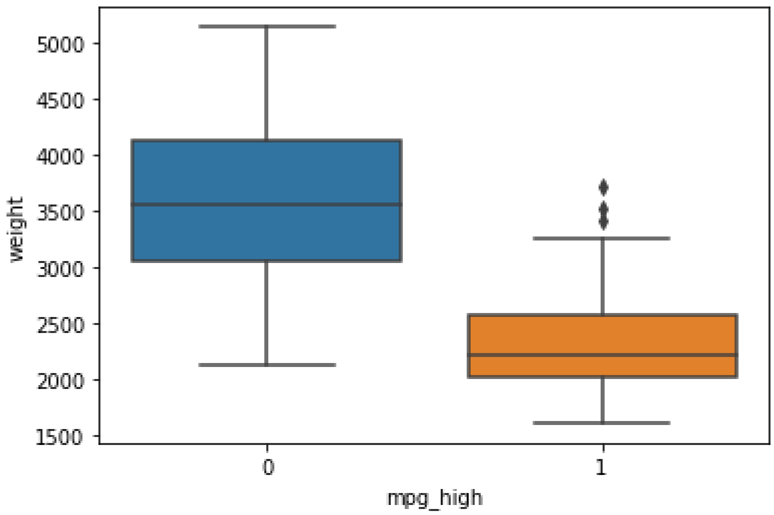
```
You need more horsepower to move heavier cars.
```


```
# Boxplot

sb.boxplot(x = "mpg_high", y = "weight", data = df)

print("\nCars with lower fuel mileage are heavier.\n")
```

```
Cars with lower fuel mileage are heavier.
```



7. Train / test split

```
# Splitting dataframe 80 / 20

x = df1 = df.loc[:, df.columns != "mpg_high"]
y = df.mpg_high

from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size = 0.2, random_state = 123

print("\nDimensions of train:", x_train.shape)
print("\nDimensions of test:", x_test.shape)
```

```
Dimensions of train: (311, 7)

Dimensions of test: (78, 7)
```

8. Logistic Regression

```
# Train model

from sklearn.linear_model import LogisticRegression

logR = LogisticRegression(solver = "lbfgs")

logR.fit(x_train, y_train)
logR.score(x_train, y_train)
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_logistic.py:818: Convergenc
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
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG,
0.9067524115755627
```

```
# Predicitons

pred = logR.predict(x_test)

# Evaluation

from sklearn.metrics import classification_report, confusion_matrix

print(classification_report(y_test, pred), "\n")

confusion_matrix(y_test, 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


array([[40, 10],
       [ 1, 27]])
```

9. Decision Tree

```
# Train tree

from sklearn.tree import DecisionTreeClassifier

dtc = DecisionTreeClassifier()
dtc.fit(x_train, y_train)
```

```
DecisionTreeClassifier()
```

```
# Prediciton

pred = dtc.predict(x_test)

# Evaluation

print(classification_report(y_test, pred))
```

```
              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
```

10. Neural Network

```
# Normalizing data

from sklearn import preprocessing

scaler = preprocessing.StandardScaler().fit(x_train)

x_train_scaled = scaler.transform(x_train)
x_test_scaled = scaler.transform(x_test)
```

```
# Train LBFGS

from sklearn.neural_network import MLPClassifier

nn = MLPClassifier(solver='lbfgs', hidden_layer_sizes=(5, 2), max_iter=500, random_state=1234
nn.fit(x_train_scaled, y_train)
```

```
MLPClassifier(hidden_layer_sizes=(5, 2), max_iter=500, random_state=1234,
              solver='lbfgs')
```

```
# Prediction

pred = nn.predict(x_test_scaled)

# Results

print(classification_report(y_test, pred), "\n")

confusion_matrix(y_test, pred)
```

```
              precision    recall  f1-score   support

           0       0.93      0.86      0.90        50
           1       0.78      0.89      0.83        28

    accuracy                           0.87        78
   macro avg       0.86      0.88      0.86        78
weighted avg       0.88      0.87      0.87        78


array([[43,  7],
       [ 3, 25]])
```

```
# Train SGD

nn1 = MLPClassifier(solver='sgd', hidden_layer_sizes=(5, 2), max_iter=500, random_state=1234)
nn1.fit(x_train_scaled, y_train)
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/neural_network/_multilayer_perceptron.py
  ConvergenceWarning,
MLPClassifier(hidden_layer_sizes=(5, 2), max_iter=500, random_state=1234,
              solver='sgd')
```

```
# Prediction

pred1 = nn1.predict(x_test_scaled)

# Results

print(classification_report(y_test, pred1), "\n")

confusion_matrix(y_test, pred1)
```

```
              precision    recall  f1-score   support

           0       0.93      0.84      0.88        50
           1       0.76      0.89      0.82        28

    accuracy                           0.86        78
   macro avg       0.85      0.87      0.85        78
```

```
weighted avg       0.87      0.86      0.86        78


array([[42,  8],
       [ 3, 25]])
```

```
print(
"The difference is very marginal, but the LBFGS performed better than SGD.\n",
"The accuracy was slightly better at 0.87 to SGD's 0.86. The confusion matrices\n",
"are also very similar, SGD was only off by one data point. But I honestly think\n",
"since the dataset is so small, the difference between the two methods is\n",
"negligible. I would like to see how a larger dataset would perform to compare\n",
"to this one, then I'd definitively say which is better.")
```

```
The difference is very marginal, but the LBFGS performed better than SGD.
 The accuracy was slightly better at 0.87 to SGD's 0.86. The confusion matrices
 are also very similar, SGD was only off by one data point. But I honestly think
 since the dataset is so small, the difference between the two methods is
 negligible. I would like to see how a larger dataset would perform to compare
 to this one, then I'd definitively say which is better.
```

+ Code + Text

11. Analysis

```
print(
"The decision tree seemed to perform the best, it had an accuracy of 0.88, which\n",
"is higher than the 0.87 from the neural network and the 0.86 from the logistic\n",
"regression. It also has the least difference in precision and recall for both 0\n",
"and 1 on the classification report. Typically speaking smaller datasets perform\n",
"well with decision trees so it's not much of a surprise that it did better than\n",
"the other two. The difference between all three of them is very marginal though,\n",
"again I would love to see how all three of the algorithms would perform with a\n",
"larger dataset.\n\n"
"I very much prefer to use R versus Python, primarily because of the fact that I\n",
"can run just one line of code at a time when I'm debugging, and don't have to run\n",
"to the top of the file and run the whole thing. I also have more experience with\n",
"R ouside of this class, so I'm definitely biased! :)")
```

```
The decision tree seemed to perform the best, it had an accuracy of 0.88, which
 is higher than the 0.87 from the neural network and the 0.86 from the logistic
 regression. It also has the least difference in precision and recall for both 0
 and 1 on the classification report. Typically speaking smaller datasets perform
 well with decision trees so it's not much of a surprise that it did better than
 the other two. The difference between all three of them is very marginal though,
 again I would love to see how all three of the algorithms would perform with a
 larger dataset.

I very much prefer to use R versus Python, primarily because of the fact that I
 can run just one line of code at a time when I'm debugging, and don't have to run
 to the top of the file and run the whole thing. I also have more experience with
 R ouside of this class, so I'm definitely biased! :)
```