Import pandas package and read the sample csv data from Github

```
import pandas as pd
df = pd.read_csv('https://raw.githubusercontent.com/neshuynh/Sample_Portfolio/main/sklearn/Au
```

Check the first few rows and the dimensions of the data

```
print('First 5 rows: \n', df.head())
print('\nDimensions: ', df.shape)
```

```
First 5 rows:
     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:  (392, 9)
```

Trying the describe function on the mpg, weight, and year columns

```
df[["mpg","weight","year"]].describe(include="all")
```

	mpg	weight	year

mpg range (min-max): (9-46.6), 37.6

weight range (min-max): (1613-5140), 3527

year range (min-max): (70-82), 12

mpg average: 23.445918

weight average: 2977.584184

year average: 76.010256

max	46.600000	5140.000000	82.000000

Check datatypes of the columns

```
df.dtypes
```

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

Change the 'cylinders' and 'origin' columns to categorical

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

Verify by checking the datatypes of the altered columns

```
df[['cylinders', 'origin']].dtypes
```

```
cylinders        int8
origin       category
dtype: object
```

Data Cleaning

Delete the rows that contain NA's and check the new dimensions

```
print('\nDimensions before dropping NAs: ', df.shape)
df = df.dropna()
print('\nDimensions after dropping NAs: ', df.shape)
```

```
Dimensions before dropping NAs:  (392, 9)

Dimensions after dropping NAs:  (389, 9)
```

Create a new column

It is a categorical column created from the 'mpg' column where it equals 1 if mpg > average, else it is 0

```
df['mpg_high'] = pd.cut(df['mpg'], bins=[0, df["mpg"].mean(), float('Inf')], labels=[0, 1])
```

Delete 'mpg' and 'name' columns

Then view the head of the dataframe

```
df.drop(columns=['mpg', 'name'], inplace=True)
print('First 5 rows: \n', df.head())
```

```
First 5 rows:
    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:

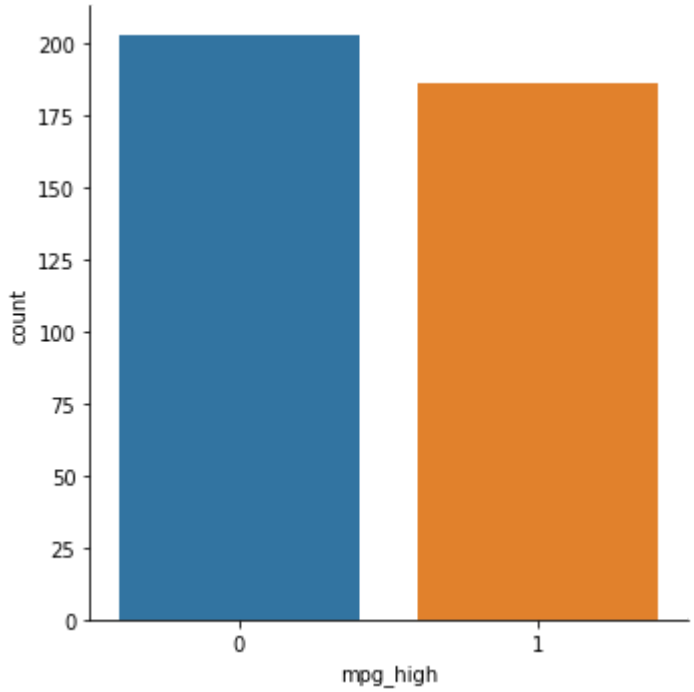
Import seaborn package

```
import seaborn as sb
```

Seaborn catplot on 'mpg_high'

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

```
<seaborn.axisgrid.FacetGrid at 0x7f203f15f990>
```



There are slightly more vehicles with below average 'mpg' compared to vehicles with above average 'mpg'

Seaborn relplot ('weight' vs. 'horsepower')

```
sb.relplot(x='horsepower', y='weight', data=df, hue=df['mpg_high'], style=df['mpg_high'])
```

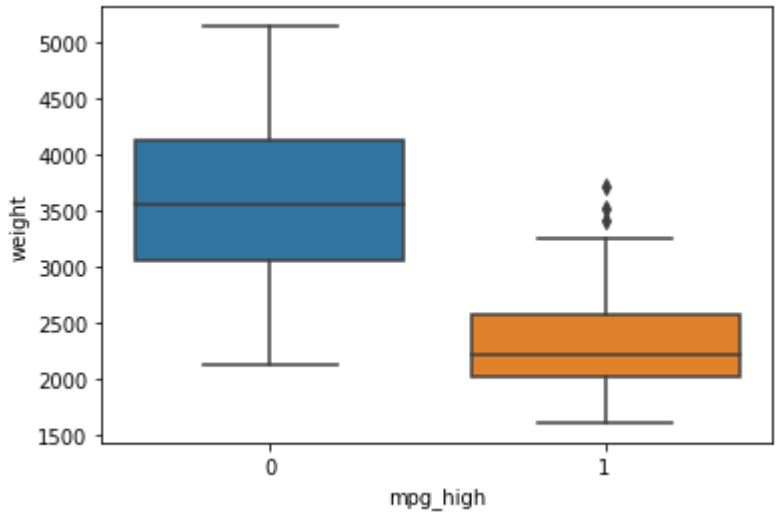
```
<seaborn.axisgrid.FacetGrid at 0x7f204cb66050>
```

Vehicles with higher than average 'mpg' tend to be on the lighter side and have lower horsepower than those with the lower than average 'mpg'

Seaborn boxplot ('weight' vs. 'mpg_high')


```
sb.boxplot('mpg_high', y='weight', data=df)
```

```
/usr/local/lib/python3.7/dist-packages/seaborn/_decorators.py:43: FutureWarning: Pass th
  FutureWarning
<matplotlib.axes._subplots.AxesSubplot at 0x7f203bb58190>
```



Again, we can see that vehicles with higher than average 'mpg' tend to be lighter

Train/test split

Preparation for the next set of tests

```
# import the necessary package
from sklearn.model_selection import train_test_split
```

```
# split the x and y dataframes into their respective columns
x = df.drop(columns=['mpg_high'])
y = df['mpg_high']


# use given seed 1234 for consistency
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_state=1234)

# check dimensions of both x_train and x_test
print('train size:', x_train.shape)
print('test size:', x_test.shape)
```

```
train size: (311, 7)
test size: (78, 7)
```

Logistic Regression

```
# import the necessary package
from sklearn.linear_model import LogisticRegression

# train the model
clf = LogisticRegression(max_iter=420)
clf.fit(x_train, y_train)
clf.score(x_train, y_train)

# make predictions
pred = clf.predict(x_test)

# evaluate
# import metrics
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score

print('accuracy score: ', accuracy_score(y_test, pred))
print('precision score: ', precision_score(y_test, pred))
print('recall score: ', recall_score(y_test, pred))
print('f1 score: ', f1_score(y_test, pred))

# confusion matrix
# import the matrix
from sklearn.metrics import confusion_matrix

confusion_matrix(y_test, pred)
```

```
accuracy score:  0.8974358974358975
precision score:  0.7777777777777778
recall score:  1.0
f1 score:  0.8750000000000001
```

```
array([[42,  8],
       [ 0, 28]])
```

Decision Tree

```
# import the necessary package
from sklearn.tree import DecisionTreeClassifier

# create a decision tree
dt = DecisionTreeClassifier()
dt.fit(x_train, y_train)
DecisionTreeClassifier()

# make predictions
pred2 = dt.predict(x_test)

# evaluate
print('accuracy score: ', accuracy_score(y_test, pred2))
print('precision score: ', precision_score(y_test, pred2))
print('recall score: ', recall_score(y_test, pred2))
print('f1 score: ', f1_score(y_test, pred2))
```

```
accuracy score:  0.8846153846153846
precision score:  0.8518518518518519
recall score:  0.8214285714285714
f1 score:  0.8363636363636364
```

Neural Network

```
# import the necessary package to normalize data
from sklearn import preprocessing
# import the classification report
from sklearn.metrics import classification_report

scaler = preprocessing.StandardScaler().fit(x_train)
x_train_scaled = scaler.transform(x_train)
x_test_scaled = scaler.transform(x_test)

# train
from sklearn.neural_network import MLPClassifier
# use seed 1234
neural = MLPClassifier(solver='lbfgs', hidden_layer_sizes=(4, 2), max_iter=690, random_state=
neural.fit(x_train_scaled, y_train)

# make predictions
```

```
pred3 = neural.predict(x_test_scaled)

# evaluate
print('accuracy = ', accuracy_score(y_test, pred3))
# confusion matrix
print('confusion matrix: \n', confusion_matrix(y_test, pred3))
# classification report
print(classification_report(y_test, pred3))

# train with a different network topology and settings
neural2 = MLPClassifier(solver='sgd', hidden_layer_sizes=(6, 3), max_iter=720, random_state=1
neural2.fit(x_train_scaled, y_train)

# make predictions
pred4 = neural2.predict(x_test_scaled)

# evaluate
print('accuracy = ', accuracy_score(y_test, pred4))
# confusion matrix
print('confusion matrix: \n', confusion_matrix(y_test, pred4))
# classification report
print(classification_report(y_test, pred4))
```

```
accuracy =  0.8846153846153846
confusion matrix:
 [[43  7]
 [ 2 26]]
              precision    recall  f1-score   support

           0       0.96      0.86      0.91        50
           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

accuracy =  0.8717948717948718
confusion matrix:
 [[40 10]
 [ 0 28]]
              precision    recall  f1-score   support

           0       1.00      0.80      0.89        50
           1       0.74      1.00      0.85        28

    accuracy                           0.87        78
   macro avg       0.87      0.90      0.87        78
weighted avg       0.91      0.87      0.87        78
```

The performance was pretty similar which I believe is due to the large number of iterations, granting both models ample time to learn and become accurate and precise

Analysis

1. Logistic Regression performed best in these tests
2. It had the greatest accuracy, recall, and f1 score, but its precision was lower than most other tests
3. Logistic regression performed best, probably because the data is relatively simple and the other algorithms may have overfitted the data slightly
4. I personally prefer Python over R as I am more familiar with Python and its syntax.