```
import pandas as pd
import seaborn as sb
from sklearn import datasets
#1
#importing data
url = 'https://raw.githubusercontent.com/rjg190002/Machine_Learning/main/Auto.csv'
auto = pd.read_csv(url)

#Printing first few rows of data
first_five_rows = auto.head(5)
print("First 5 rows of the data: ")
print(first_five_rows)

#Printing dimensions of the data
size = auto.size
shape = auto.shape
print("Size of data: ")
print(size)
print("Rows: ")
print(shape[0])
print("Columns: ")
print(shape[1])
```

```
First 5 rows of the data:
    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
Size of data:
3528
Rows:
392
Columns:
9
```

```
#2
#Using describe
print("Describe on mpg, weight, and year")
print(auto.mpg.describe())
#Range of mpg comes out to 37, and the average is 22.75
print("")
```

```
print(auto.weight.describe())
#Range of weight comes out to 3527, and the average is 2803.5
print("")

print(auto.year.describe())
#Range of year comes out to 12, and the average is 76
print("")
```

```
Describe on mpg, weight, and year
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

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

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
```

```
#3
#Finding data types
print("Data types: ")
print(auto.dtypes)

#Changing data type to categorical with cat.codes
auto_copy = auto.copy()
auto_copy.cylinders = auto_copy.cylinders.astype('category').cat.codes
print("\nData set with cylinders changed to be categorical with cat.codes")
print(auto_copy.dtypes)

#Changing data type to categorical without cat.codes
```

```
auto_copy1 = auto.copy()
auto_copy1.origin = auto_copy1.origin.astype('category')
print("\nData set with origin changed to be categorical without cat.codes")
print(auto_copy1.dtypes)
```

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

Data set with cylinders changed to be categorical with cat.codes
mpg             float64
cylinders          int8
displacement    float64
horsepower        int64
weight            int64
acceleration    float64
year            float64
origin            int64
name             object
dtype: object

Data set with origin changed to be categorical without cat.codes
mpg              float64
cylinders          int64
displacement     float64
horsepower         int64
weight             int64
acceleration     float64
year             float64
origin          category
name              object
dtype: object
```

```
#4
#Delete rows with NA's
print("\nDimensions before dropping NA's: ", auto.shape)
auto = auto.dropna()
print("New dimensions after dropping NA's: ", auto.shape)
```

```
Dimensions before dropping NA's:  (392, 9)
New dimensions after dropping NA's:  (389, 9)
```

```
#5
#Add a column mpg_high
```

```
auto1 = auto.copy()
#Create an array of the new mpg_high variables to be inserted into dataframe
mpg_high = []
for row in auto['mpg']:
  if row > 22.75:
    mpg_high.append(1)
  else:
    mpg_high.append(0)
#Insert into data frame and delte the rows
auto1.insert(1, "mpg_high", mpg_high, True)
auto1 = auto1.drop(columns=["mpg", "name"])
print("\nFirst 5 rows of modified list with deleted/adjusted rows: ")
print(auto1.head(5))
```

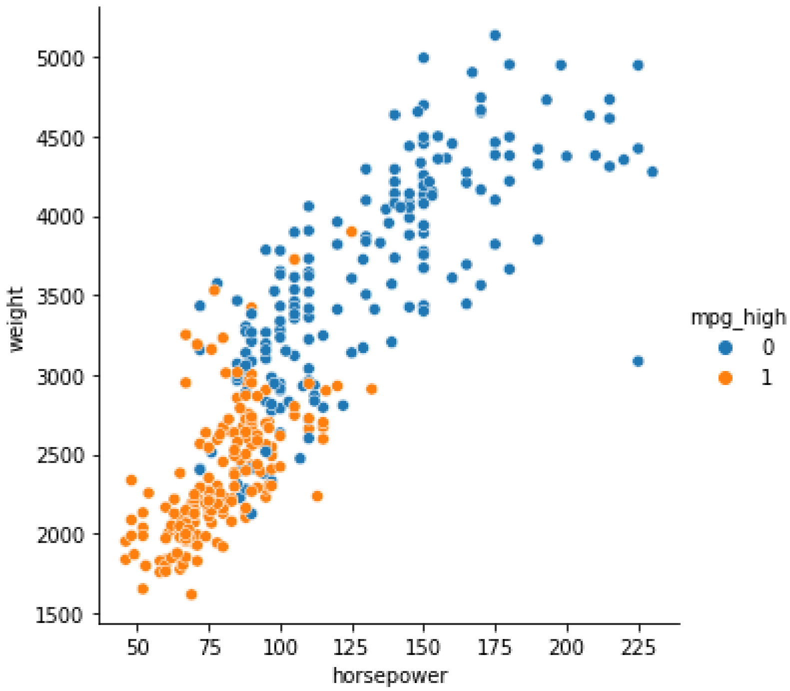
```
First 5 rows of modified list with deleted/adjusted rows: 
   mpg_high  cylinders  displacement  horsepower  weight  acceleration  year  \
0         0          8         307.0         130    3504          12.0  70.0   
1         0          8         350.0         165    3693          11.5  70.0   
2         0          8         318.0         150    3436          11.0  70.0   
3         0          8         304.0         150    3433          12.0  70.0   
6         0          8         454.0         220    4354           9.0  70.0   

   origin  
0       1  
1       1  
2       1  
3       1  
6       1  
```

```
#6
#Creating a catplot for the mpg_high column
sb.catplot(x="mpg_high", kind='count', data = auto1)
#There seemse to be a very even amount of both high and low mileage cars
```

<seaborn.axisgrid.FacetGrid at 0x7ff4448b1fd0>

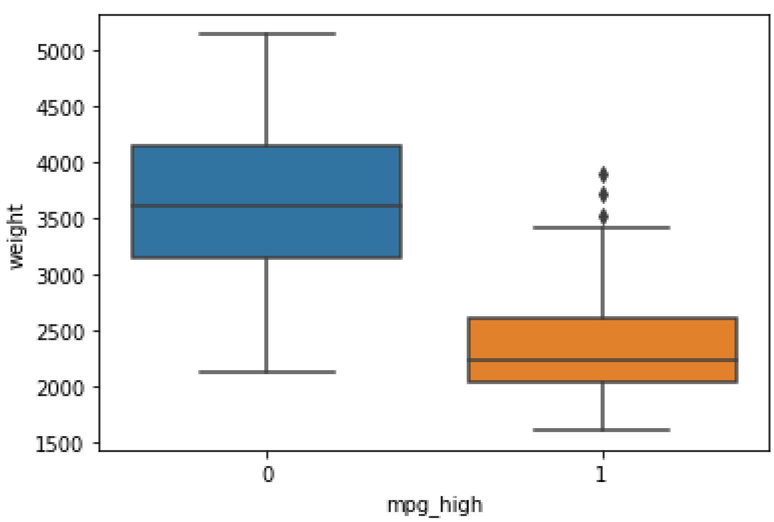

```
#6
#Creating relplot
sb.relplot(x="horsepower", y="weight", hue="mpg_high", data=auto1)
#This graph shows that higher horsepower/weight is correlated with low mpg
```

<seaborn.axisgrid.FacetGrid at 0x7ff4448b6690>




```
#Creating boxplot
sb.boxplot(x="mpg_high", y="weight", data=auto1)
#This plot further shows that the higher weight of a car generally means that the cars mpg wi
```

<matplotlib.axes._subplots.AxesSubplot at 0x7ff44475aed0>



```
#7
#Splitting the train and test data
from sklearn.model_selection import train_test_split
x = auto1.iloc[:, 1:7]
y = auto1.mpg_high
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_state=1234)

print("Train size: ", x_train.shape)
print("Test size: ", x_test.shape)
```

```
Train size:  (311, 6)
Test size:  (78, 6)
```

```
#8
from sklearn.linear_model import LogisticRegression
#Form model
logreg = LogisticRegression()
logreg.fit(x_train, y_train)
print("Logistic Regression Score for train: ")
logreg.score(x_train, y_train)
```

```
Logistic Regression Score for train:
0.9196141479099679
```

```
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score

pred = logreg.predict(x_test)
from sklearn.metrics import classification_report
print(classification_report(y_test, pred))

print("Accuracy: ", accuracy_score(y_test, pred))
print("Precision: ", precision_score(y_test, pred))
print("Recall Score: ", recall_score(y_test, pred))
print("F1 Score: ", f1_score(y_test, pred))
```

```
              precision    recall  f1-score   support

           0       0.97      0.83      0.90        47
           1       0.79      0.97      0.87        31

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

Accuracy:  0.8846153846153846
Precision:  0.7894736842105263
Recall Score:  0.967741935483871
F1 Score:  0.8695652173913043
```

```
#9
```

```
from sklearn.tree import DecisionTreeClassifier
#creating model and predictions
dtc = DecisionTreeClassifier()
dtc.fit(x_train, y_train)
pred_tree = dtc.predict(x_test)

from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
#Comparing predictions and seeing details of the results
from sklearn.metrics import classification_report
print(classification_report(y_test, pred_tree))

print("Accuracy: ", accuracy_score(y_test, pred_tree))
print("Precision: ", precision_score(y_test, pred_tree))
print("Recall Score: ", recall_score(y_test, pred_tree))
print("F1 Score: ", f1_score(y_test, pred_tree))

#Creating tree
#from sklearn.datasets import load_iris
#from sklearn import tree
#iris = load_iris()
#X, y = iris.data, iris.target
#tree.plot_tree(dtc)
```

```
              precision    recall  f1-score   support

           0       0.96      0.91      0.93        47
           1       0.88      0.94      0.91        31

    accuracy                           0.92        78
   macro avg       0.92      0.93      0.92        78
weighted avg       0.93      0.92      0.92        78

Accuracy:  0.9230769230769231
Precision:  0.8787878787878788
Recall Score:  0.9354838709677419
F1 Score:  0.90625
```

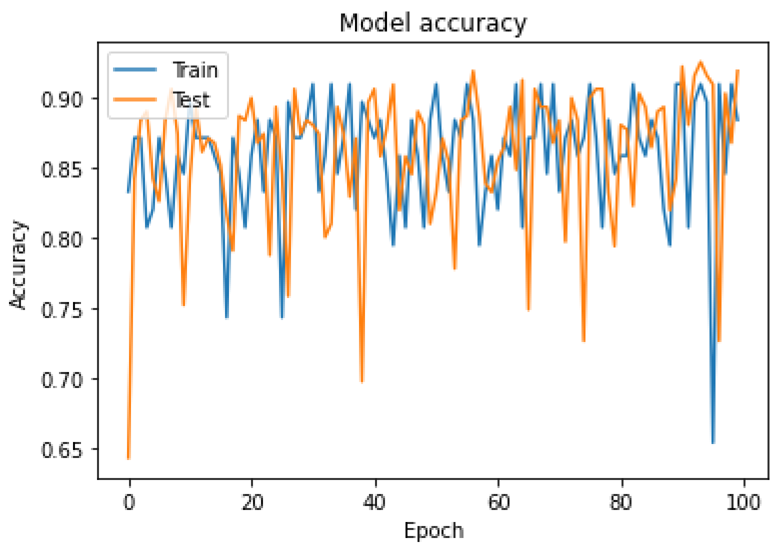
```
#10
from sklearn import preprocessing
#Scaling the data
scaler = preprocessing.StandardScaler().fit(x_train)
x_train_scaled = scaler.transform(x_train)
x_test_scaled = scaler.transform(x_test)


from sklearn.neural_network import MLPClassifier

clf = MLPClassifier(solver='lbfgs', hidden_layer_sizes=(5, 2), max_iter=500, random_state=123
clf.fit(x_train_scaled, y_train)

pred_nn = clf.predict(x_test_scaled)
from sklearn.metrics import classification_report
```

```
print(classification_report(y_test, pred_nn))
print("Accuracy: ", accuracy_score(y_test, pred_nn))
```

```
              precision    recall  f1-score   support

           0       0.91      0.85      0.88        47
           1       0.79      0.87      0.83        31

    accuracy                           0.86        78
   macro avg       0.85      0.86      0.85        78
weighted avg       0.86      0.86      0.86        78

Accuracy:  0.8589743589743589
```

```
#Second neural network
import keras
from keras.models import Sequential
from keras.layers import Dense, Dropout
from keras.optimizers import RMSprop

y_train_nn = keras.utils.to_categorical(y_train, 2)
y_test_nn = keras.utils.to_categorical(y_test, 2)

batch_size = 128
epochs = 100
#Creating the parameters for the model that will be implemented
model = Sequential()
model.add(Dense(512, activation='relu', input_shape=(6,)))
model.add(Dropout(0.2))
model.add(Dense(2, activation='sigmoid'))


model.compile(loss='binary_crossentropy',
              optimizer=RMSprop(),
              metrics=['accuracy'])

history = model.fit(x_train, y_train_nn,
                    batch_size=batch_size,
                    epochs=epochs,
                    verbose=1,
                    validation_data=(x_test, y_test_nn))


import matplotlib.pyplot as plt

#Plot the graph of the NN
plt.plot(history.history['val_accuracy'])
plt.plot(history.history['accuracy'])
plt.title('Model accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
```

```
plt.legend(['Train', 'Test'], loc='upper left')
plt.show()
```




```
score = model.evaluate(x_test, y_test_nn, verbose=0)
print('Test loss:', score[0])
print('Test accuracy:', score[1])
```

```
Test loss: 0.3301587998867035
Test accuracy: 0.8846153616905212
```

Comparing the two NN models.

The two NN models were both relativley good, however the second one outperformed the first likely due to the fact that over the course of 100 epochs, the second model had a better chance to find, fit, and predict the data as compared to the first model. This resulted in the second model getting an accuracy of .88 compared to the first model's accuracy of .85.

Analysis

The tree algorithm performed the best out of the four differnt models made on the same data with an accuracy of .92 and being both very precise and having a good recall score. Behind the tree algorithm are the epoch neural network and the linear regression algorithms. Both of them got around .88 accuracy and with similar precision and recall score as well. Last place was the first Neural Network model with a .86, which is not far behind the other two. The likely reason for this difference in the accuracy and scores is that the data is better fit with something like a tree diagram with predictors highly correlating to the target, meaning that the tree diagram can analyze each different predictor and use it to precit the target.

In my opinion there is not much of a difference between using R and sklearn because both of them will get you the same results. However I feel that while R is very useful for handling large data and analyzing it on a mass scale, learning python is a skill that is useful in more broader applications. I think that both are useful but that R is much more focused where python is not.