

this is a heading

```
from google.colab import files
uploaded = files.upload()
```

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

```
### load the data
import pandas as pd
```

```
df = pd.read_csv('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)

```
df.loc[:, ['mpg', 'weight', 'year']].describe()
#the average of mpg is 23.446, average of weight is 2977.584,
#average of year is 76.010.
```

```
#Range of mpg is 37.6, Range of weight is 3,527, Range of year is 12
```

	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

```
# check data types
```

```
print(df.dtypes, "\n")
```

```
df.origin = df.origin.astype('category')
```

```
# convert cylinders to categorical data type with numeric factor codes
```

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

```
# convert origin to categorical data type
```

```
df.origin = df.origin.astype('category')
```

```
print(df.dtypes, "\n")
```

```
print(df.head())
```

mpg	float64
cylinders	int64
displacement	float64
horsepower	int64
weight	int64

```

acceleration    float64
year            float64
origin          int64
name            object
dtype: object

```

```

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

```

	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	
4	17.0	4	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

```
# check for NAs
```

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

```
# drop remaining rows with NAs
```

```
df = df.dropna()
```

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

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

```
import numpy as np
```

```
df.loc[df['mpg'] > df['mpg'].mean(), 'mpg_high'] = '1'
df.loc[df['mpg'] < df['mpg'].mean(), 'mpg_high'] = '0'
df.mpg_high = df.mpg_high.astype('category')
```

```
print(df)
print(df.dtypes, "\n")
```

```
##delete cols mpg and name
```

```
df = df.drop('mpg', axis=1)
df = df.drop('name', axis=1)
print(df.head())
```

	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]

```

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

```

	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

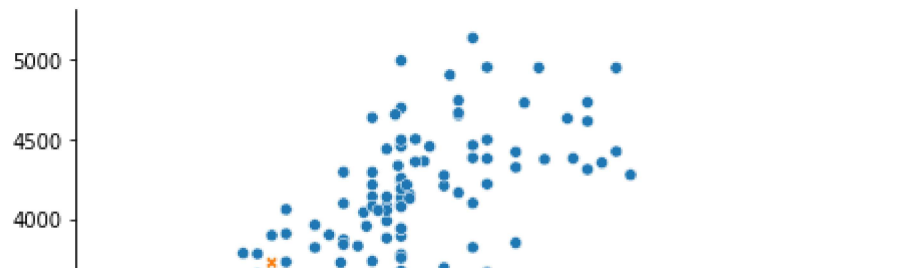
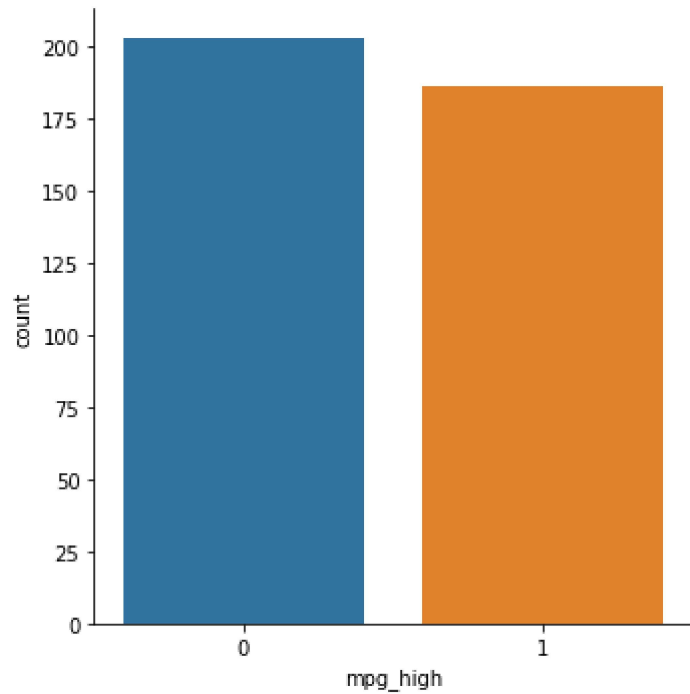
	mpg_high
0	0
1	0
2	0
3	0
6	0

```
import seaborn as sb
from sklearn import datasets
```

```
## catplot,
# from the graph we can tell that there are more 0's than 1's.
# There are 200 '0's and there are around 190 '1's
sb.catplot(x="mpg_high", kind='count', data=df)
```

```
## relplot
# from the relplot we can tell that mpg_high of '0' has a higher weight than
# mpg_high of '1'.
sb.relplot(x='horsepower', y='weight', data=df, hue="mpg_high",
           style="mpg_high")
```

<seaborn.axisgrid.FacetGrid at 0x7fea19be6d50>



boxplot

From the graph we can tell that for '0' median is little above 3500

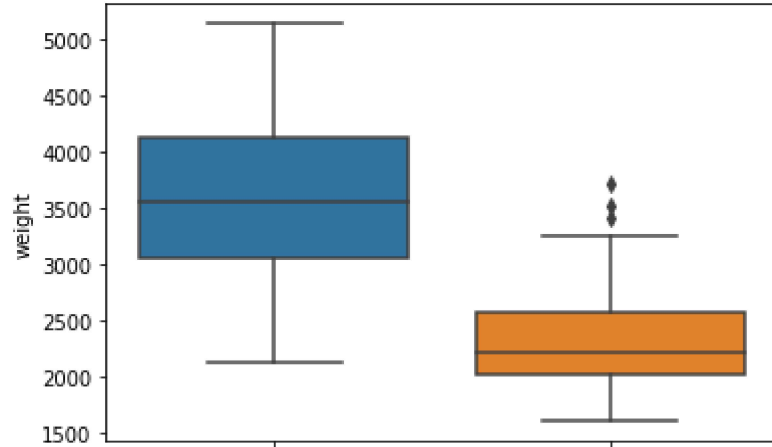
whereas for '1' it is around 2150. Weight min value of '0' is 2100

whereas for '1' it is 1600. Weight max value of '0' is 5250

whereas for '1' it is 3250. The outliers of '1' are between 3500 and 4000.

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

```
/usr/local/lib/python3.7/dist-packages/seaborn/_decorators.py:43: FutureWarning: P
FutureWarning
<matplotlib.axes._subplots.AxesSubplot at 0x7fea19c47ad0>
```



```
# train test split
from sklearn.model_selection import train_test_split

X = df.loc[:, ['cylinders', 'displacement', 'horsepower', 'weight',
               'acceleration', 'year', 'origin']]
y = df.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, 7)
test size: (78, 7)
```

```
## Logistic Regression
```

```
import numpy as np
```

```
from sklearn.linear_model import LogisticRegression
```



```

clf = LogisticRegression(max_iter=300)
clf.fit(X_train, y_train)
clf.score(X_train, y_train)

# make predictions
pred = clf.predict(X_test)

# evaluate
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, average="binary",
                                           pos_label="1"))
print('recall score: ', recall_score(y_test, pred, average="binary",
                                     pos_label="1"))
print('f1 score: ', f1_score(y_test, pred, average="binary", pos_label="1"))

# classification report
from sklearn.metrics import classification_report
print(classification_report(y_test, pred, target_names=None))

# confusion 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

```

	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

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

```
# Decision Tree
```

```
from sklearn.tree import DecisionTreeClassifier  
from sklearn import tree
```

```
clf2 = DecisionTreeClassifier()  
clf2.fit(X_train, y_train)  
clf2.score(X_train, y_train)
```

```
# make predictions  
pred = clf2.predict(X_test)
```

```
# evaluate  
print('accuracy score: ', accuracy_score(y_test, pred))  
print('precision score: ', precision_score(y_test, pred, average="binary",  
                                           pos_label="1"))  
print('recall score: ', recall_score(y_test, pred, average="binary",  
                                     pos_label="1"))  
print('f1 score: ', f1_score(y_test, pred, average="binary", pos_label="1"))
```

```
# confusion matrix  
confusion_matrix(y_test, pred)
```

```
#classification report  
print(classification_report(y_test, pred, target_names=None))
```

```
#plot  
tree.plot_tree(clf2)
```

accuracy score: 0.9102564102564102
precision score: 0.8387096774193549
recall score: 0.9285714285714286
f1 score: 0.8813559322033899

	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

```
[Text(0.6433823529411765, 0.9444444444444444, 'X[0] <= 2.5\nngini = 0.5\nnsamples = 311\nnvalue = [153, 158]'),  
Text(0.4338235294117647, 0.8333333333333334, 'X[2] <= 101.0\nngini = 0.239\nnsamples = 173\nnvalue = [24, 149]'),  
Text(0.27941176470588236, 0.7222222222222222, 'X[5] <= 75.5\nngini = 0.179\nnsamples = 161\nnvalue = [16, 145]'),  
Text(0.14705882352941177, 0.6111111111111112, 'X[1] <= 119.5\nngini = 0.362\nnsamples = 59\nnvalue = [14, 45]'),  
Text(0.058823529411764705, 0.5, 'X[0] <= 0.5\nngini = 0.159\nnsamples = 46\nnvalue = [4, 42]'),  
Text(0.029411764705882353, 0.3888888888888889, 'gini = 0.0\nnsamples = 2\nnvalue = [2, 0]'),  
Text(0.08823529411764706, 0.3888888888888889, 'X[3] <= 2683.0\nngini = 0.087\nnsamples = 44\nnvalue = [2, 42]'),  
Text(0.058823529411764705, 0.2777777777777778, 'X[3] <= 2377.0\nngini = 0.045\nnsamples = 43\nnvalue = [1, 42]'),  
Text(0.029411764705882353, 0.1666666666666666, 'gini = 0.0\nnsamples = 38\nnvalue = [0, 38]'),  
Text(0.08823529411764706, 0.1666666666666666, 'X[3] <= 2385.0\nngini = 0.32\nnsamples = 5\nnvalue = [1, 4]'),  
Text(0.058823529411764705, 0.0555555555555555, 'gini = 0.0\nnsamples = 1\nnvalue = [1, 0]'),  
Text(0.11764705882352941, 0.0555555555555555, 'gini = 0.0\nnsamples = 4\nnvalue = [0, 4]'),  
Text(0.11764705882352941, 0.2777777777777778, 'gini = 0.0\nnsamples = 1\nnvalue = [1, 0]'),  
Text(0.23529411764705882, 0.5, 'X[4] <= 17.75\nngini = 0.355\nnsamples = 13\nnvalue = [10, 3]'),  
Text(0.20588235294117646, 0.3888888888888889, 'X[2] <= 81.5\nngini =
```

```

0.469\nsamples = 8\nvalue = [5, 3]'),
Text(0.17647058823529413, 0.2777777777777778, 'gini = 0.0\nsamples = 2\nvalue
= [0, 2]'),
Text(0.23529411764705882, 0.2777777777777778, 'X[5] <= 71.5\ngini =
0.278\nsamples = 6\nvalue = [5, 1]'),
Text(0.20588235294117646, 0.16666666666666666, 'X[3] <= 2242.0\ngini =
0.5\nsamples = 2\nvalue = [1, 1]'),
Text(0.17647058823529413, 0.05555555555555555, 'gini = 0.0\nsamples = 1\nvalue
= [1, 0]'),
Text(0.23529411764705882, 0.05555555555555555, 'gini = 0.0\nsamples = 1\nvalue
= [0, 1]'),
Text(0.2647058823529413, 0.16666666666666666, 'gini = 0.0\nsamples = 4\nvalue

```

```

# Neural Network
# Using classification
# normalize the data
from sklearn import preprocessing
#from sklearn.metrics import plot_confusion_matrix

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
from sklearn.metrics import plot_confusion_matrix

clf = MLPClassifier(solver='lbfgs', hidden_layer_sizes=(5, 2), max_iter=500,
                    random_state=1234)
clf.fit(X_train_scaled, y_train)

# make predictions
pred = clf.predict(X_test_scaled)

# output results

print('accuracy = ', accuracy_score(y_test, pred))

print(confusion_matrix(y_test, pred))

```

```
from sklearn.metrics import classification_report
print(classification_report(y_test, pred))

# try different settings

clf = MLPClassifier(solver='sgd', hidden_layer_sizes=(3,), max_iter=1500,
                    random_state=1234)
clf.fit(X_train_scaled, y_train)

# make predictions
pred = clf.predict(X_test_scaled)

print('accuracy = ', accuracy_score(y_test, pred))

# confusion matrix
print(confusion_matrix(y_test, pred))

print(classification_report(y_test, pred))

"""First neural network using solver lbfgs slightly performed better
in accuracy and precision compared to the solver sgd. Recall is about the same.
Performance could be different due to number of iterations and more complex
topology"""
```

```

accuracy = 0.8717948717948718
[[43  7]
 [ 3 25]]

```

	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

"""11.a.

Logistic Regression perform the best because it has the highest accuracy of 0.898.

b.

In Logistic Regression, class 0 precision has 1.00, but class 1 precision is 0.78.

And class 0 recall score is 0.84 but class 1 recall score is 1.

And accuracy is 0.898.

In Decision Tree, class 0 precision has 0.92, but class 1 precision is 0.80.

And class 0 recall score is 0.88 but class 1 recall score is 0.86.

And accuracy is 0.871.

In Neural Network Classifier solver lbfgs, class 0 precision has 0.93, but class 1 is 0.78. And class 0 recall score is 0.86 but class 1 recall score is 0.89. And accuracy is 0.871.

In Neural Network Classifier solver sgd, class 0 precision has 0.93, but class 1 is 0.71. And class 0 recall score is 0.80 but class 1 recall score is 0.89. And accuracy is 0.833.

11.c.

Logistic regression might have performed the best because the dataset used is relatively small and there were some NA's which affects the predictors.

11.d.

I found R to be harder to learn than sklearn, as it was simpler to use the

inbuilt libraries. It was also a bit faster than sklearn. For data visualization sklearn uses seaborn where as R use ggplot, and graphs in R is more statistic related. I may have preferred sklearn over R due to fact that I was new to ML when doing R, but I'm now more familiar which might have made it easier to use sklearn.

"""

[Colab paid products](#) - [Cancel contracts here](#)

