```
import pandas as pd

#Load our data
from google.colab import files
uploaded = files.upload()


df = pd.read_csv('Auto.csv')
print(df.head())
print('\nDimensions of data frame:', df.shape)
```

Choose Files Auto.csv

- **Auto.csv**(text/csv) - 17859 bytes, last modified: 11/4/2022 - 100% done

```
Saving Auto.csv to Auto (5).csv
    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)
```

```
#Use describe on mpg, weight, and year columns

print(df[['mpg', 'weight', 'year']].describe())

#Range of mpg is: 37.6
print('\nRange of mpg is: ')
print(df[['mpg']].max() - df[['mpg']].min())

#Range of weight is: 3527
print('\nRange of weight is: ')
print(df[['weight']].max() - df[['weight']].min())

#Range of year is: 12.0
print('\nRange of year is: ')
print(df[['year']].max() - df[['year']].min())
```

```
              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

Range of mpg is:
mpg    37.6
dtype: float64

Range of weight is:
weight    3527
dtype: int64

Range of year is:
year    12.0
dtype: float64
```

```
#Explore the data types

#check data types of all columns:
print(df.dtypes)

#b. change the cylinders column to categorical (use cat.codes)
#df1 = df.copy()
df.cylinders = df.cylinders.astype('category').cat.codes


#c. change the origin column to categorical (don’t use cat.codes)
#df = df.copy()

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


#d. verify the changes with the dtypes attribute
print("\n\n", df.dtypes)
```

```
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
```

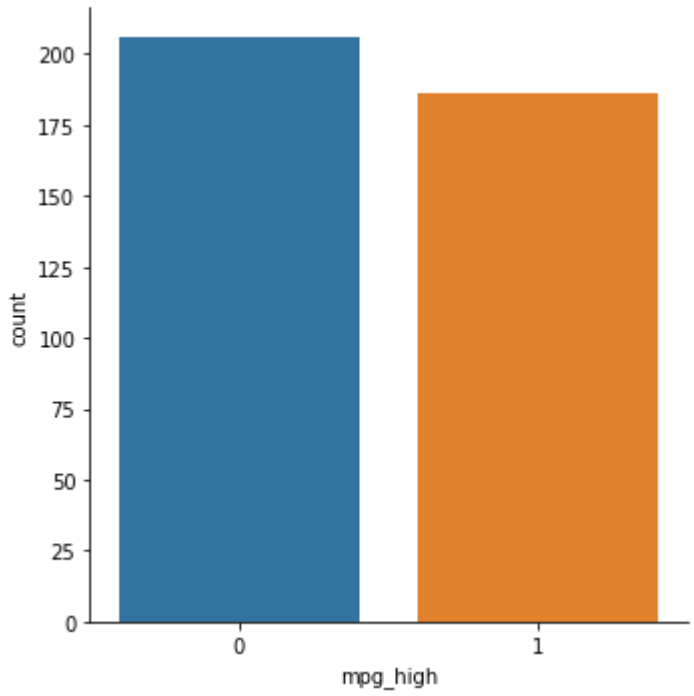
```
horsepower         int64
weight             int64
acceleration     float64
year             float64
origin          category
name              object
dtype: object
```

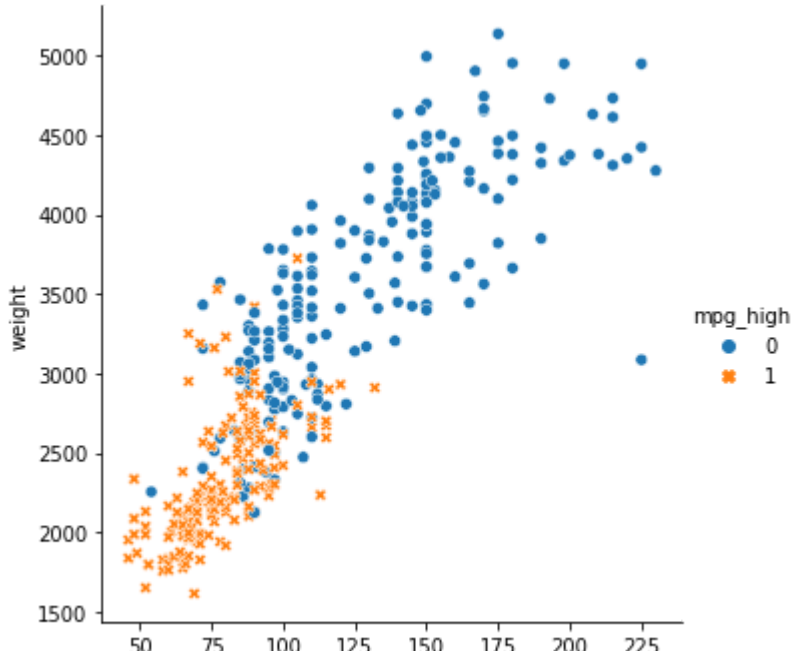
```
#Deal with NAs
import numpy as np
df1 = df.copy()

#a. make a new column, mpg_high, and make it categorical:
df['mpg_high'] = np.where(df.mpg > np.mean(df.mpg), 1, 0)
df.mpg_high = df.mpg_high.astype('category')

#b. delete the mpg and name columns (delete mpg so the algorithm doesn’t just learn to pre
df = df.drop(columns=['mpg', 'name'])


#c. output the first few rows of the modified data frame
df.head()
```

	cylinders	displacement	horsepower	weight	acceleration	year	origin	mpg_
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	
4	4	302.0	140	3449	NaN	70.0	1	

```
#Data exploration with graphs
import seaborn as sb

#a. seaborn catplot on the mpg_high column
sb.catplot(x="mpg_high", kind='count', data=df)
#I learning that the mph_high is mostly, but that there is slight more mpg below the avera


#b. seaborn relplot with horsepower on the x axis, weight on the y axis, setting hue or st
sb.relplot(x='horsepower', y='weight', data=df, hue="mpg_high", style="mpg_high")
#I learned that as horsepower increases, mpg decreases. Meaning higher horsepower leads to


#d. for each graph, write a comment indicating one thing you learned about the data from t
```

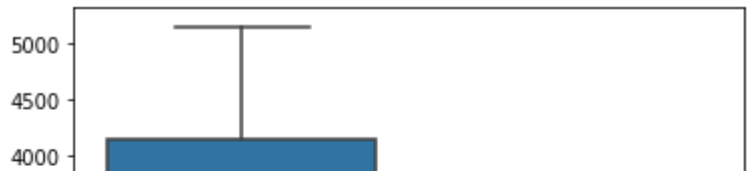
```
<seaborn.axisgrid.FacetGrid at 0x7f10881fa790>
```






```
#c. seaborn boxplot with mpg_high on the x axis and weight on the y axis
sb.boxplot(x="mpg_high", y='weight', data=df)
# I learned that the heavier vehicles generally have lower miles per gallon. (more strain
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f10880d1f10>
```




```
#Train/test split
from sklearn.model_selection import train_test_split

print(df.isnull().sum())
acceleration_mean = np.mean(df.acceleration)
df.acceleration.fillna(acceleration_mean, inplace=True)
year_mean = np.mean(df.year)
df.year.fillna(year_mean, inplace=True)
df.head()
#a. 80/20
#b. use seed 1234 so we all get the same results
#c. train /test X data frames consists of all remaining columns except mpg_high

X = df.loc[:, ['cylinders', 'displacement', 'horsepower', 'weight', 'acceleration', 'year'
y = df.mpg_high

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=123

#d. output the dimensions of train and test
print('train size:', X_train.shape)
print('test size:', X_test.shape)
```

```
cylinders       0
displacement    0
horsepower      0
weight          0
acceleration    0
year            0
origin          0
mpg_high        0
dtype: int64
train size: (313, 7)
test size: (79, 7)
```

```
#Logistic Regression
from sklearn.linear_model import LogisticRegression

#a. train a logistic regression model using solver lbfgs
clf = LogisticRegression()
clf.fit(X_train, y_train)

#b. test and evaluate
clf.score(X_train, y_train)

#predict:
```

```
pred = clf.predict(X_test)

#Evaluate:
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score

#c. print the classification report metrics

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))
```

```
accuracy score:  0.8860759493670886
precision score:  0.9
recall score:  0.8780487804878049
f1 score:  0.888888888888889
```

```
#Decision Tree
#We are using the test/train from the previous Logistic Regression
#a. train a decision tree
from sklearn.tree import DecisionTreeClassifier

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

#b. test and evaluate
#predictions
pred = clf.predict(X_test)


#c. print the classification report metrics
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))

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

```
accuracy score:  0.9620253164556962
precision score:  0.975
recall score:  0.9512195121951219
f1 score:  0.9629629629629629
              precision    recall  f1-score   support

           0       0.95      0.97      0.96        38
           1       0.97      0.95      0.96        41

    accuracy                           0.96        79
   macro avg       0.96      0.96      0.96        79
```

```
       weighted avg       0.96      0.96      0.96         79
```

```
#d. plot the tree (optional)
from sklearn import tree
print("\n\n\n")
tree.plot_tree(clf)
```

```
[Text(0.57, 0.96875, 'X[0] <= 2.5\ngini = 0.497\nsamples = 313\nvalue =
[168, 145]'),
 Text(0.34, 0.90625, 'X[2] <= 84.5\ngini = 0.267\nsamples = 164\nvalue =
[26, 138]'),
 Text(0.16, 0.84375, 'X[5] <= 72.5\ngini = 0.079\nsamples = 97\nvalue = [4,
93]'),
 Text(0.08, 0.78125, 'X[3] <= 2209.0\ngini = 0.42\nsamples = 10\nvalue = [3,
7]'),
 Text(0.04, 0.71875, 'gini = 0.0\nsamples = 7\nvalue = [0, 7]'),
 Text(0.12, 0.71875, 'gini = 0.0\nsamples = 3\nvalue = [3, 0]'),
 Text(0.24, 0.78125, 'X[1] <= 137.5\ngini = 0.023\nsamples = 87\nvalue = [1,
86]'),
 Text(0.2, 0.71875, 'gini = 0.0\nsamples = 79\nvalue = [0, 79]'),
 Text(0.28, 0.71875, 'X[5] <= 75.5\ngini = 0.219\nsamples = 8\nvalue = [1,
7]'),
 Text(0.24, 0.65625, 'X[2] <= 76.5\ngini = 0.5\nsamples = 2\nvalue = [1,
1]'),
 Text(0.2, 0.59375, 'gini = 0.0\nsamples = 1\nvalue = [0, 1]'),
 Text(0.28, 0.59375, 'gini = 0.0\nsamples = 1\nvalue = [1, 0]'),
 Text(0.32, 0.65625, 'gini = 0.0\nsamples = 6\nvalue = [0, 6]'),
 Text(0.52, 0.84375, 'X[5] <= 79.5\ngini = 0.441\nsamples = 67\nvalue = [22,
45]'),
 Text(0.48, 0.78125, 'X[3] <= 2861.5\ngini = 0.495\nsamples = 49\nvalue =
[22, 27]'),
 Text(0.44, 0.71875, 'X[4] <= 18.05\ngini = 0.459\nsamples = 42\nvalue =
[15, 27]'),
 Text(0.4, 0.65625, 'X[1] <= 88.5\ngini = 0.439\nsamples = 40\nvalue = [13,
27]'),
 Text(0.36, 0.59375, 'gini = 0.0\nsamples = 2\nvalue = [2, 0]'),
 Text(0.44, 0.59375, 'X[1] <= 113.5\ngini = 0.411\nsamples = 38\nvalue =
[11, 27]'),
```

```
#Neural Network
#We are using the test/train from the previous Logistic Regression
#a. train a neural network, choosing a network topology of your choice
from sklearn import preprocessing

#Normalizing data
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

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

#b. test and evaluate
```

```
#predict
pred = clf.predict(X_test_scaled)

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


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

```
accuracy =  0.8734177215189873
              precision    recall  f1-score   support

           0       0.82      0.95      0.88        38
           1       0.94      0.80      0.87        41

    accuracy                           0.87        79
   macro avg       0.88      0.88      0.87        79
weighted avg       0.88      0.87      0.87        79

Text(0.68, 0.78125, 'X[2] <= 83.0\ngini = 0.028\nsamples = 140\nvalue =
```

```
#c. train a second network with a different topology and different settings
clf = MLPClassifier(solver='sgd', hidden_layer_sizes=(3,), max_iter=1500, random_state=123
clf.fit(X_train_scaled, y_train)

#d. test and evaluate
pred = clf.predict(X_test_scaled)
print('accuracy = ', accuracy_score(y_test, pred))
print(classification_report(y_test, pred))

#e. compare the two models and why you think the performance was same/different
#The first model had lower accuracy in every way. I think that the performance is mostly t
# on the lower accuracy model was higher, as it sacrificed accuracy for speed, while the p
```

```
accuracy =  0.8860759493670886
              precision    recall  f1-score   support

           0       0.91      0.84      0.88        38
           1       0.86      0.93      0.89        41

    accuracy                           0.89        79
   macro avg       0.89      0.88      0.89        79
weighted avg       0.89      0.89      0.89        79
```

```
#Analysis

#b. compare accuracy, recall and precision metrics by class
#Accuracies:
#Logistic - .886
#Decision tREE - .962
#Neural Network - .873
#Neural network Model 2 - 0.886
```

```
#Neural network Model 2 - 0.880

#The Decision Tree Algorithm performed the best with a drastically higher accuracy

#Precisions
#Logistic - .9
#Decision Tree - .975
#Neural Network - .88
#Neural network Model 2 - 0.89

#The Decision Tree Algorithm performed the best with higher precision values

#Recall values:
#Logistic - .878
#Decision Tree - .951
#Neural Network - .88
#Neural network Model 2 - 0.88

#The Decision Tree Algorithm performed the best with a higher recall value

#a. which algorithm performed better?
#Overall The Decision Tree Algorithm performed much better than every other algorithm

#c. give your analysis of why the better-performing algorithm might have outperformed the
#Suprisingly, The decision tree algorithm, which is known for being interpretable but not
# I think this is because of how the algorithm works. The algorithm divides observations r
# This dividing sometimes appears arbitrary when the observation are disimilar, I believe
# to partition were already in similar groups, meaning it had to do less work, and less ca


#d. write a couple of sentences comparing your experiences using R versus sklearn. Feel fr
# I felt like R and sklearn were pretty similar in how they operated. However, I felt like
# I was able to logical operations to fill missing data. I prefer to use sklearn, however
```

Colab paid products - Cancel contracts here

✓ 1s completed at 5:09 PM