1. Read the Auto Data using Pandas

```
import pandas as pd
df = pd.read_csv("Auto.csv")

print(df.head(5))
print(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
(392, 9)
```

2. Data Exploration

```
df[["mpg","weight","year"]].describe()
# mpg -> Avg: 23.45 and Range: 37.6
# weight -> Avg: 2977.58 and Range: 3527
# year -> Avg: 76.01 and Range: 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

Saved successfully!

```
print('Before the changes:\n', df.dtypes)

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

print('\nAfter the changes:\n', df.dtypes)
```

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

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

4. Delete Na's

```
df = df.dropna()
print('New Dimensions after NA drops: ', df.shape)
```

```
New Dimensions after NA drops:  (389, 9)
```

5. Modify columns

```
avg_mpg = df.mpg.mean()

df['mpg_high'] = [1 if mpg > avg_mpg else 0 for mpg in df.mpg]
```

Saved successfully!

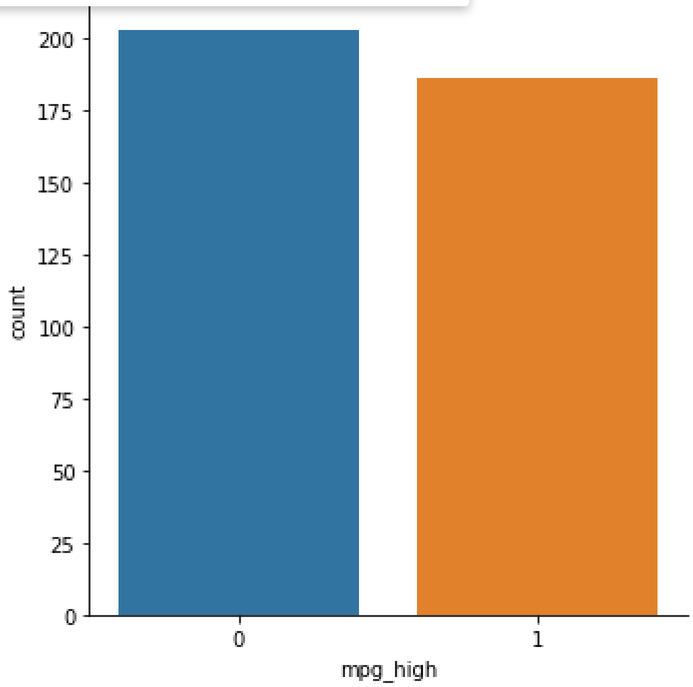
```
nplace=True)
print(df.head(5))
```

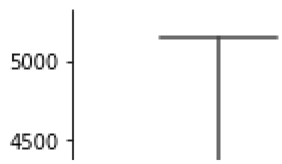
```
   cylinders  displacement  horsepower  weight  acceleration  year origin  \
0          4         307.0         130    3504          12.0  70.0      1
1          4         350.0         165    3693          11.5  70.0      1
2          4         318.0         150    3436          11.0  70.0      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
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:3: SettingWithCopyWa
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stab
  This is separate from the ipykernel package so we can avoid doing imports unti
/usr/local/lib/python3.7/dist-packages/pandas/core/frame.py:4913: SettingWithCop
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stab
  errors=errors,
```

6. Data exploration with graphs

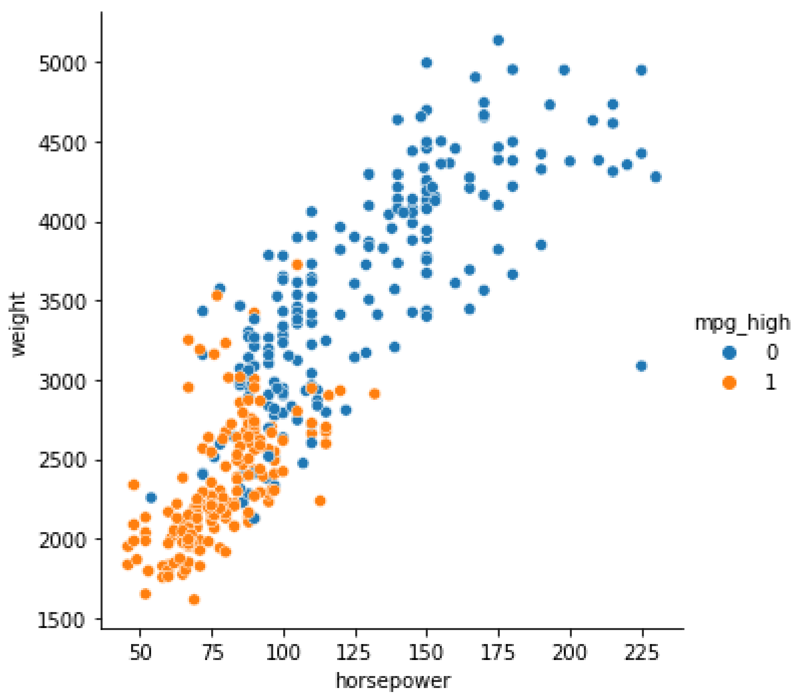
```
import seaborn as sns

# with this graph, I learned that there fewer vehicles with a mpg higher than the aver
g1 = sns.catplot(x='mpg_high', kind='count', data=df)
```

Saved successfully!



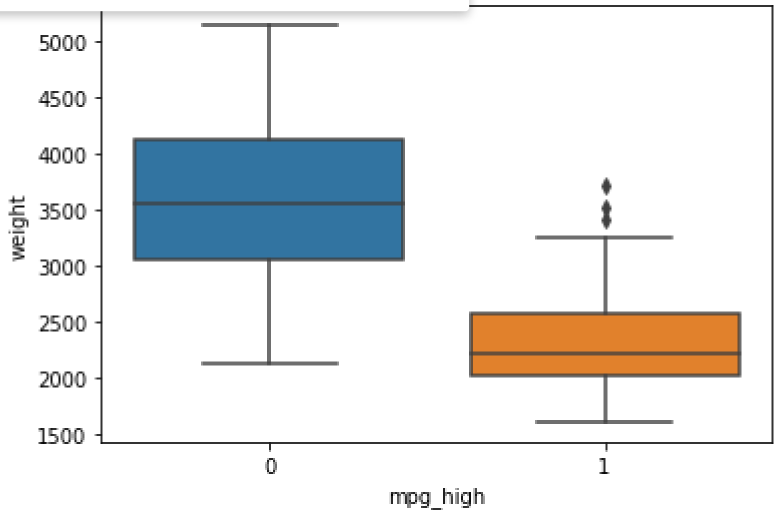



```
# this graphs shows that there is a direct correlation between weight and horsepower,
# this graphs also shows that the more weight/more horsepower has the more likely it i
g2 = sns.relplot(x='horsepower', y='weight', hue='mpg_high', data=df)
```




```
# this graph shows that cars that don't have high mileage tend to be much heavier thar
g3 = sns.boxplot(x='mpg_high', y='weight', data=df)
```

Saved successfully!



7. Train/Test Split

```
from sklearn.model_selection import train_test_split

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

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


print('train size:', X_train.shape)
print('test size:', X_test.shape)
```

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

8. Logistic Regression

```
from sklearn.linear_model import LogisticRegression

lrm = LogisticRegression(max_iter=500, solver='lbfgs')

lrm.fit(X_train, y_train)
pred = lrm.predict(X_test)

# print metrics
from sklearn.metrics import classification_report

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

```
              precision    recall  f1-score   support

                              0.84      0.91        50
```

Saved successfully!

```
                                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
```

9. Decision Tree

```
from sklearn import tree

dtm = tree.DecisionTreeClassifier()
dtm.fit(X_train, y_train)
pred = dtm.predict(X_test)

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

```
              precision    recall  f1-score   support

           0       0.96      0.92      0.94        50
           1       0.87      0.93      0.90        28

    accuracy                           0.92        78
   macro avg       0.91      0.92      0.92        78
weighted avg       0.93      0.92      0.92        78
```

10. Neural Network

```
from sklearn import preprocessing
from sklearn.neural_network import MLPClassifier

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

X_train_scaled = scaler.transform(X_train)
X_test_scaled = scaler.transform(X_test)

# train, test, evaluate first model
nnm = MLPClassifier(solver='lbfgs', hidden_layer_sizes=(5, 2), max_iter=500, random_st
nnm.fit(X_train_scaled, y_train)
pred = nnm.predict(X_test)

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

# train, test, evaluate second model
gd', hidden_layer_sizes=(3,), max_iter=1500, random_s
```

Saved successfully!

```
in)
pred = nnm_two.predict(X_test)

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

# Both models are the same, I think since our dataset is so small not only are neural
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/base.py:444: UserWarning: X has f
  f"X has feature names, but {self.__class__.__name__} was fitted without"
/usr/local/lib/python3.7/dist-packages/sklearn/metrics/_classification.py:1318: 
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.7/dist-packages/sklearn/metrics/_classification.py:1318: 
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.7/dist-packages/sklearn/metrics/_classification.py:1318: 
  _warn_prf(average, modifier, msg_start, len(result))
              precision    recall  f1-score   support

           0       0.64      1.00      0.78        50
           1       0.00      0.00      0.00        28

    accuracy                           0.64        78
   macro avg       0.32      0.50      0.39        78
weighted avg       0.41      0.64      0.50        78

              precision    recall  f1-score   support

           0       0.64      1.00      0.78        50
           1       0.00      0.00      0.00        28

    accuracy                           0.64        78
   macro avg       0.32      0.50      0.39        78
weighted avg       0.41      0.64      0.50        78

/usr/local/lib/python3.7/dist-packages/sklearn/base.py:444: UserWarning: X has f
  f"X has feature names, but {self.__class__.__name__} was fitted without"
/usr/local/lib/python3.7/dist-packages/sklearn/metrics/_classification.py:1318: 
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.7/dist-packages/sklearn/metrics/_classification.py:1318: 
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.7/dist-packages/sklearn/metrics/_classification.py:1318: 
  _warn_prf(average, modifier, msg_start, len(result))
```

```
11. Analysis

a. The Decision Tree algorithm performed t

b.
```

11. Analysis

a. The Decision Tree algorithm performed the best.

b.

Saved successfully!

```
        1. DT
        2. LR
        3. Both NN
Recall

        1. Both NN
        2. DT
        3. LR
Precision

        1. LR
        2. DT
        3. Both NN

c. Decision Trees and Logistic Regression
comparable to each other and both performe
better than either Neural Networks. I thin
reason this is because since the dataset i
the Neural Networks just have a huge disad
are probably overfitting or learning from
than the other algorithms.
d. Honestly, they're similar and each have
pros and cons. On one hand R has very intu
strict rules as to how everything needs to
There's only way to do it but it makes sen
find out what it is. Python is very differ
way because there are so many ways to acco
same task. I think I prefer Python, simply
there is more documentation and online hel
than there is for R.
```


Accuracy

1. DT
2. LR
3. Both NN

Recall

1. Both NN
2. DT
3. LR

Precision

1. LR
2. DT
3. Both NN

c. Decision Trees and Logistic Regression are very comparable to each other and both performed so much better than either Neural Networks. I think the reason this is because since the dataset is so small the Neural Networks just have a huge disadvantage and are probably overfitting or learning from more noise than the other algorithms. d. Honestly, they're similar and each have their own pros and cons. On one hand R has very intuitive and strict rules as to how everything needs to be done. There's only way to do it but it makes sense once you find out what it is. Python is very different in that way because there are so many ways to accomplish the same task. I think I prefer Python, simply because there is more documentation and online help for it than there is for R.

Saved successfully!

Colab paid products - Cancel contracts here

Saved successfully!