```
from google.colab import drive
drive.mount('/content/drive')


###1. Read the Auto 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)
```

```
###2. Data exploration with code
df_2 = df[['mpg','weight','year']]
df_2.describe()

###Range of mpg row is from 9 to 46.6. The average of mpg is 23.45
###Range of weight row is from 1613 to 5140. The average of weight
###is 2977.58
###Range of year row is from 70 to 82. The average of year is 76.01
```

1 to 8 of 8 entries

index	mpg	weight	ye

```
###3. Explore data types
df.dtypes
df.cylinders = df.cylinders.astype('category').cat.codes
df.origin = df.origin.astype('category')
df.dtypes
```

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

```
###4. Deal with NAs
df = df.dropna()
print('\nDimensions of data frame:', df.shape)
```

```
Dimensions of data frame: (389, 9)
```

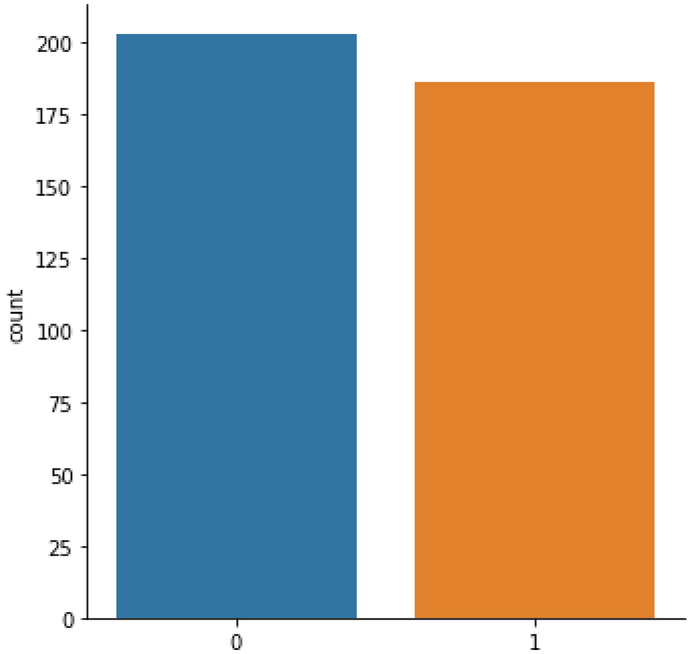
```
###5. Modify columns
import numpy as np

mpg_mean = np.mean(df.mpg)
df['mpg_high'] = df.mpg.apply(lambda x: 1 if x >= mpg_mean else 0)
df.mpg_high = df.mpg_high.astype('category').cat.codes
print(df.dtypes, "\n")
print(df)

df = df.drop(columns=['mpg', 'name'])

print(df.head())
```

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

```
mpg_high           int8
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
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]
   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
```

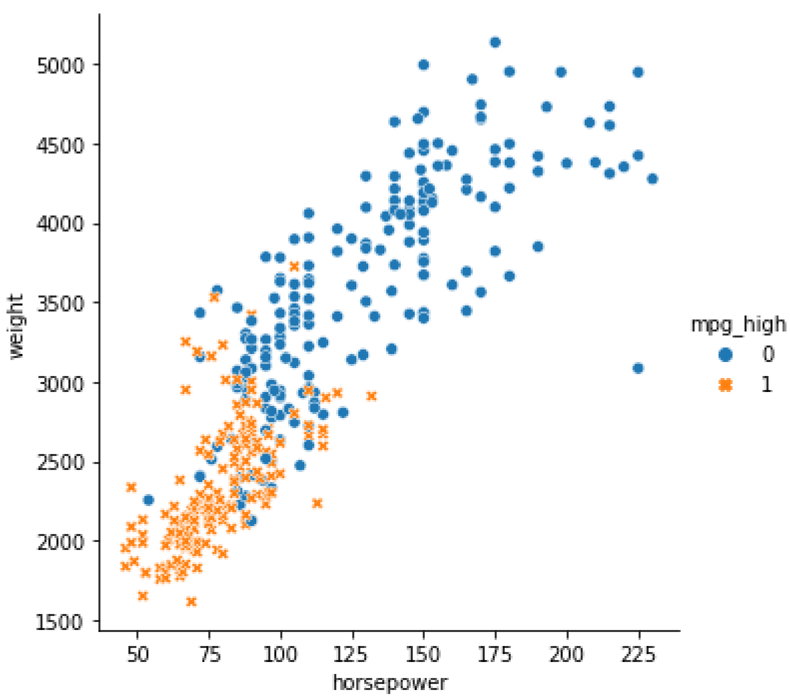
```
###6. Data exporation with graphs
import seaborn as sb
from sklearn import datasets

sb.catplot(x='mpg_high', kind = 'count', data=df)
### The person who has mpg lower than average is more than the person
### who has mpg higher than average.
```

```
<seaborn.axisgrid.FacetGrid at 0x7fc7146af4d0>
```



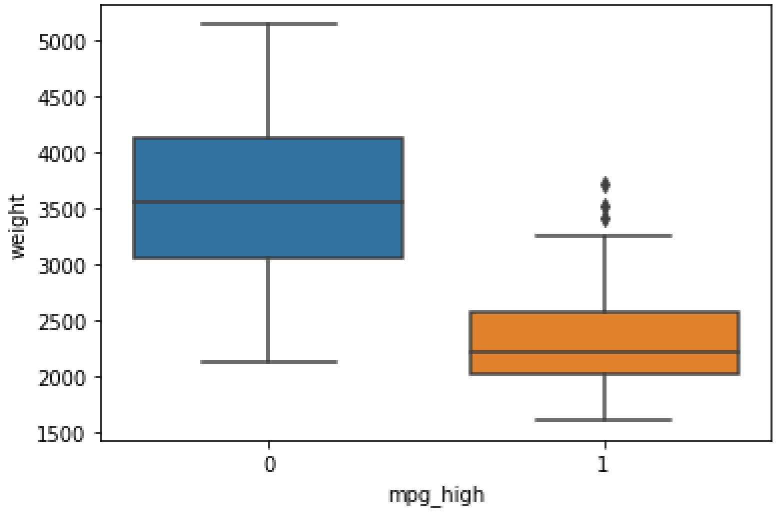

```
sb.relplot(x='horsepower', y='weight', data=df, hue=df.mpg_high,
    style=df.mpg_high)

### The person has mpg higher than average, normally has lower weight
### and horsepower than the person has mpg lower than average.
```

```
<seaborn.axisgrid.FacetGrid at 0x7fc7147f3790>
```




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

### Max and average weight of the person who has mpg lower than average
### is much higher than the person who has mpg higher than average.
```

```
/usr/local/lib/python3.7/dist-packages/seaborn/_decorators.py:43: FutureWarning: Pass th
  FutureWarning
<matplotlib.axes._subplots.AxesSubplot at 0x7fc7147f3a50>
```




```
###7. Train/test split
df1 = df.copy()
df1.displacement = df1.displacement.astype('category').cat.codes
df1.horsepower = df1.horsepower.astype('category').cat.codes
df1.weight = df1.weight.astype('category').cat.codes
df1.acceleration = df1.acceleration.astype('category').cat.codes
df1.year = df1.year.astype('category').cat.codes

from sklearn.model_selection import train_test_split

X = df1.loc[:,['cylinders', 'displacement', 'horsepower', 'weight', 'acceleration', 'year', 'o
y = df1.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)
```

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

clf1 = LogisticRegression( solver="lbfgs")
clf1.fit(X_train, y_train)
clf1.score(X_train, y_train)
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_logistic.py:818: ConvergenceWarning
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
```

```
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG,
0.909967845659164
```

```
pred1 = clf1.predict(X_test)
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score

print('accuracy score: ', accuracy_score(y_test, pred1))
print('precision score: ', precision_score(y_test, pred1))
print('recall score: ', recall_score(y_test, pred1))
print('f1 score: ', f1_score(y_test, pred1))
```

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

```
from sklearn.metrics import confusion_matrix

confusion_matrix(y_test, pred1)
```

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

```
###9. Decision Tree
from sklearn.tree import DecisionTreeClassifier

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

```
DecisionTreeClassifier()
```

```
pred2 = clf2.predict(X_test)
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score

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:  0.8974358974358975
precision score:  0.8571428571428571
recall score:  0.8571428571428571
f1 score:  0.8571428571428571
```

```
from sklearn.metrics import confusion_matrix
```

```
confusion_matrix(y_test, pred2)
```

```
array([[46,  4],
       [ 4, 24]])
```

```
from sklearn.metrics import classification_report
print(classification_report(y_test, pred2))
```

```
              precision    recall  f1-score   support

           0       0.92      0.92      0.92        50
           1       0.86      0.86      0.86        28

    accuracy                           0.90        78
   macro avg       0.89      0.89      0.89        78
weighted avg       0.90      0.90      0.90        78
```

```
from sklearn import tree
tree.plot_tree(clf2)
```

```
[Text(0.6507352941176471, 0.9444444444444444, 'X[0] <= 2.5\ngini = 0.5\nsamples =
311\nvalue = [153, 158]'),
 Text(0.4338235294117647, 0.8333333333333334, 'X[2] <= 45.5\ngini = 0.239\nsamples =
173\nvalue = [24, 149]'),
 Text(0.27941176470588236, 0.7222222222222222, 'X[5] <= 5.5\ngini = 0.179\nsamples =
161\nvalue = [16, 145]'),
 Text(0.14705882352941177, 0.6111111111111112, 'X[1] <= 32.5\ngini = 0.362\nsamples =
59\nvalue = [14, 45]'),
 Text(0.058823529411764705, 0.5, 'X[0] <= 0.5\ngini = 0.159\nsamples = 46\nvalue = [4,
42]'),
 Text(0.029411764705882353, 0.3888888888888889, 'gini = 0.0\nsamples = 2\nvalue = [2,
0]'),
 Text(0.08823529411764706, 0.3888888888888889, 'X[3] <= 144.0\ngini = 0.087\nsamples =
44\nvalue = [2, 42]'),
 Text(0.058823529411764705, 0.2777777777777778, 'X[3] <= 96.5\ngini = 0.045\nsamples =
43\nvalue = [1, 42]'),
 Text(0.029411764705882353, 0.16666666666666666, 'gini = 0.0\nsamples = 38\nvalue = [0,
38]'),
 Text(0.08823529411764706, 0.16666666666666666, 'X[3] <= 98.5\ngini = 0.32\nsamples =
5\nvalue = [1, 4]'),
 Text(0.058823529411764705, 0.05555555555555555, 'gini = 0.0\nsamples = 1\nvalue = [1,
0]'),
 Text(0.11764705882352941, 0.05555555555555555, 'gini = 0.0\nsamples = 4\nvalue = [0,
4]'),
 Text(0.11764705882352941, 0.2777777777777778, 'gini = 0.0\nsamples = 1\nvalue = [1,
0]'),
 Text(0.23529411764705882, 0.5, 'X[4] <= 62.0\ngini = 0.355\nsamples = 13\nvalue = [10,
3]'),
 Text(0.20588235294117646, 0.3888888888888889, 'X[2] <= 27.5\ngini = 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[1] <= 37.5\ngini = 0.278\nsamples =
6\nvalue = [5, 1]'),
 Text(0.20588235294117646, 0.16666666666666666, 'gini = 0.0\nsamples = 4\nvalue = [4,
```

```
###10. Neural Network
from sklearn import preprocessing

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

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

```
 Text(0.4117647058823529, 0.6111111111111112, 'X[3] <= 212.0\ngini = 0.038\nsamples =
```

```
from sklearn.neural_network import MLPClassifier

clf3 = MLPClassifier(solver='lbfgs', hidden_layer_sizes=(5, 2), max_iter=500, random_state=1234)
clf3.fit(X_train_scaled, y_train)
```

```
MLPClassifier(hidden_layer_sizes=(5, 2), max_iter=500, random_state=1234,
              solver='lbfgs')
0]'),
```

```
pred3 = clf3.predict(X_test_scaled)
```

```
print('accuracy = ', accuracy_score(y_test, pred3))

confusion_matrix(y_test, pred3)
```

```
accuracy =  0.8461538461538461
array([[41,  9],
       [ 3, 25]])
```

```
Text(0.5882352941176471, 0.5, 'X[4] <= 20.5\ngini = 0.444\nsamples = 3\nvalue = [2,
1]')
```

```
from sklearn.metrics import classification_report
print(classification_report(y_test, pred3))
```

```
              precision    recall  f1-score   support

           0       0.93      0.82      0.87        50
           1       0.74      0.89      0.81        28

    accuracy                           0.85        78
   macro avg       0.83      0.86      0.84        78
weighted avg       0.86      0.85      0.85        78
```

```
clf4 = MLPClassifier(solver='sgd', hidden_layer_sizes=(3,), max_iter=1500, random_state=1234)
clf4.fit(X_train_scaled, y_train)
```

```
MLPClassifier(hidden_layer_sizes=(3,), max_iter=1500, random_state=1234,
              solver='sgd')
```

```
pred4 = clf4.predict(X_test_scaled)

print('accuracy = ', accuracy_score(y_test, pred4))

confusion_matrix(y_test, pred4)
```

```
accuracy =  0.8333333333333334
array([[40, 10],
       [ 3, 25]])
```

```
print(classification_report(y_test, pred4))
```

```
              precision    recall  f1-score   support

           0       0.93      0.80      0.86        50
           1       0.71      0.89      0.79        28

    accuracy                           0.83        78
   macro avg       0.82      0.85      0.83        78
weighted avg       0.85      0.83      0.84        78
```

```
###The performance of the two models is similar, and the accuracy is almost equal. The 
###so even if the randomness of SGD is large, it can ensure that the objective function
###global optimization. In LBFGS, because the data samples are limited, it will not cause
###to get the optimization
```

```
###11. Analysis
```

Overall, logistic regression is the best solution among all algorithms. The various algorithms of the neural network are inferior to logistic regression and decision trees this time. Compared with logistic regression and decision tree, the accuracy of the two is the same. The accuracy of decision tree is higher than logistic regression, but recall is lower than logistic regression. So in this case only the F1 score can be considered. F1 combines the results of precision and recall, so overall logistic regression is more effective. Maybe because the data features are obvious this time, and there are not too many errors, the obtained samples are linearly separable, and the feature space is not very large, and there is no overfitting. So it is relatively better. Python itself is more inclined towards machine learning, and there are many packages that can further optimize this property. sklearn provides a large number of tools for data mining and analysis, which improves usability. Pandas provides high-performance processing data structures and data analysis tools for Python. There is also the RPy2 library that provides all of R's main functions. For me with some programming background, python is more suitable for me. And python's voice is flexible enough.

Colab 付费产品 - 在此处取消合同

0 秒 完成时间：19:28