Ved Nigam Introduction to ML in Python with NumPy, Pandas, Seaborn, an SKLearn

In [ ]:

```
import pandas as pd
import numpy as np

# 1. Reading in data
# a.
df = pd.read_csv('Auto.csv')

# b.
print(df.head())

#c.
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)
```

In [ ]:

```
# 2. Data Exploration
# a. b.
df_mpg = df[['mpg']]
print(df_mpg.describe())
# average mpg is 23.445918 and the range for mpg in the data is 37

df_weight = df[['weight']]
print(df_weight.describe())
# average weight is 2977.584184 and the range for weight is 3527

df_year = df[["year"]]
print(df_year.describe())
# average year is 76.010256 and the range is 12
```

```
              mpg
count  392.000000
mean    23.445918
std      7.805007
min      9.000000
25%     17.000000
50%     22.750000
75%     29.000000
max     46.600000
            weight
count   392.000000
mean   2977.584184
std     849.402560
min    1613.000000
25%    2225.250000
50%    2803.500000
75%    3614.750000
max    5140.000000
             year
count  390.000000
mean    76.010256
std      3.668093
min     70.000000
25%     73.000000
50%     76.000000
75%     79.000000
max     82.000000
```

In [ ]:

```
# 3.
# a.
print(df.dtypes)

# b.
df.cylinders = df.cylinders.astype('category').cat.codes

# c.
df['origin'] = pd.Categorical(df.origin)

# d.
print(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
```

In [ ]:

```
# 4.
# a.
df = df.dropna()

# b.
print(df.shape)
```

```
(389, 9)
```

In [ ]:

```
# 5.
# a.
df['mpg_high'] = np.where(df.mpg > np.mean(df.mpg), 1, 0)
del df['mpg']
del df['name']

# b.
df.head
```

Out[ ]:

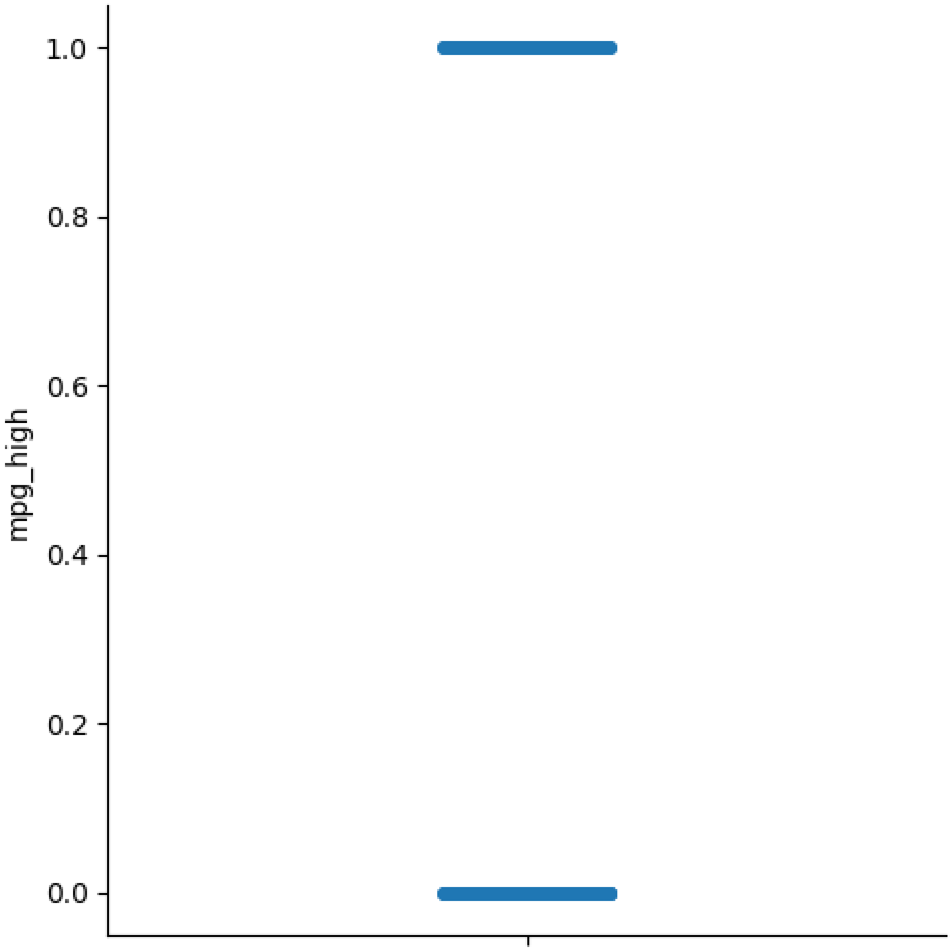
```
<bound method NDFrame.head of      cylinders  displacement  horsepower  wei
ght  acceleration  year origin
0            4         307.0         130    3504          12.0  70.0      1
\
1            4         350.0         165    3693          11.5  70.0      1
2            4         318.0         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
..         ...           ...         ...     ...           ...   ...    ...
387          1         140.0          86    2790          15.6  82.0      1
388          1          97.0          52    2130          24.6  82.0      2
389          1         135.0          84    2295          11.6  82.0      1
390          1         120.0          79    2625          18.6  82.0      1
391          1         119.0          82    2720          19.4  82.0      1

     mpg_high
0           0
1           0
2           0
3           0
6           0
..        ...
387         1
388         1
389         1
390         1
391         1

[389 rows x 8 columns]>
```

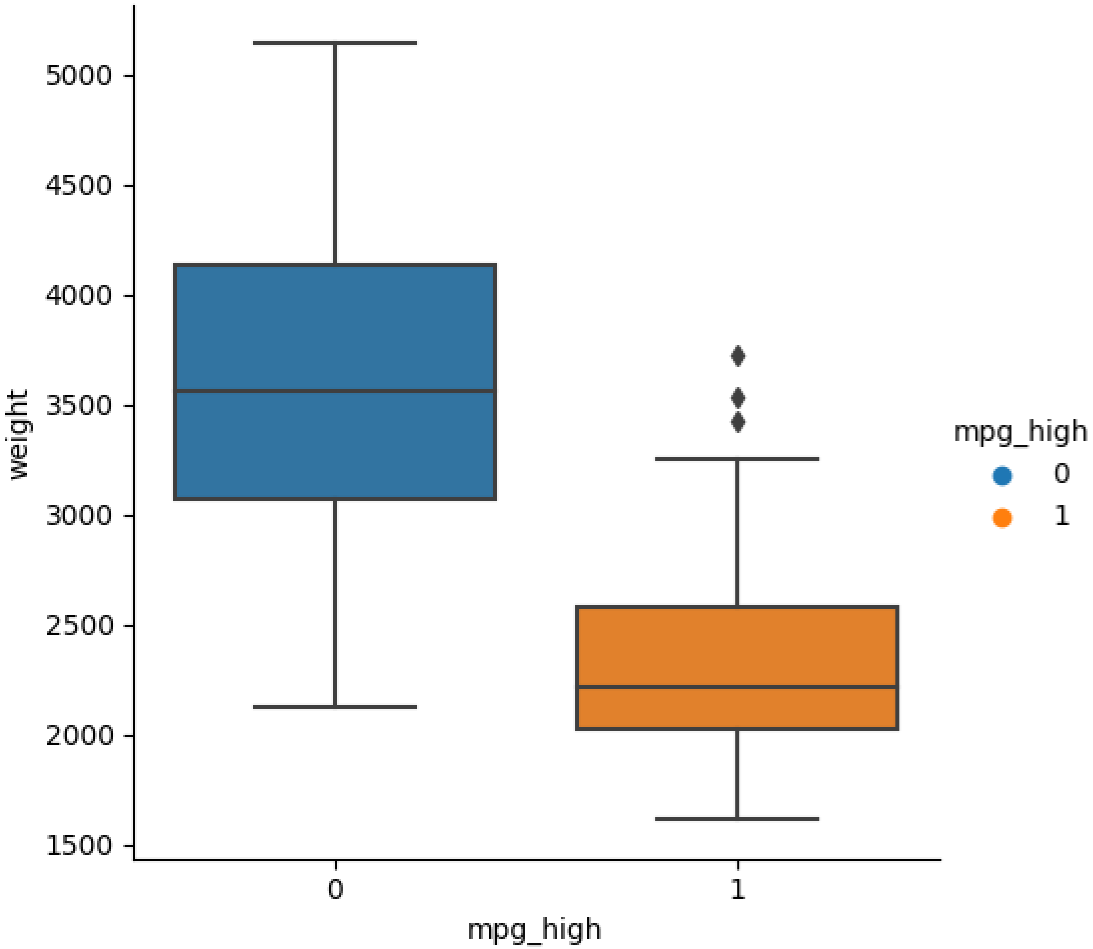
In [ ]:

```python
import seaborn as sb

# 6.
# a.
sb.catplot(y = 'mpg_high', data = df)
# Data is either 0 or 1

# b.
sb.relplot(x = 'horsepower', y = 'weight', hue = 'mpg_high', data = df)
# Higher horsepower cars are less fuel efficient

# c.
sb.boxplot(x = 'mpg_high', y = 'weight', data = df)
# The fuel efficient cars are lighter
```

Out[ ]:

```
<Axes: xlabel='mpg_high', ylabel='weight'>
```


1.0
0.8
0.6
0.4
0.2
0.0
mpg_high



In [ ]:

```
from sklearn.model_selection import train_test_split

X = df.iloc[:, 0:6]
y = df.iloc[:, 7]

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

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

print(X_train)
```

```
train size: (311, 6)
test size: (78, 6)
     cylinders  displacement  horsepower  weight  acceleration  year
184          1         101.0          83    2202          15.3  76.0
355          3         145.0          76    3160          19.6  81.0
57           1          97.5          80    2126          17.0  72.0
170          1          90.0          71    2223          16.5  75.0
210          4         350.0         180    4380          12.1  76.0
..         ...           ...         ...     ...           ...   ...
207          1         120.0          88    3270          21.9  76.0
56           1         113.0          95    2278          15.5  72.0
297          1         141.0          71    3190          24.8  79.0
214          1          98.0          68    2045          18.5  77.0
306          1         151.0          90    2556          13.2  79.0

[311 rows x 6 columns]
```

In [ ]:

```
# 8
from sklearn.linear_model import LogisticRegression

clf = LogisticRegression()
clf.fit(X_train, y_train)
print(clf.score(X_train, y_train))

pred = clf.predict(X_test)

from sklearn.metrics import accuracy_score, precision_score, recall_score, f
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))
```

```
0.9035369774919614
accuracy score:  0.8589743589743589
precision score:  0.7297297297297297
recall score:  0.9642857142857143
f1 score:  0.8307692307692307
```

```
/Library/Frameworks/Python.framework/Versions/3.11/lib/python3.11/site-packag
es/sklearn/linear_model/_logistic.py:458: ConvergenceWarning: lbfgs failed to
converge (status=1):
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-regres
sion
  n_iter_i = _check_optimize_result(
```

In [ ]:

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

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

pred_tree = clf.predict(X_test)
```

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

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

```
accuracy score:  0.8589743589743589
precision score:  0.7297297297297297
recall score:  0.9642857142857143
f1 score:  0.8307692307692307
```

In [ ]:

```
# 10.
from sklearn import preprocessing

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,
clf.fit(X_train_scaled, y_train)

pred = clf.predict(X_test_scaled)

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

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

clf_sgd = MLPClassifier(solver='sgd', hidden_layer_sizes=(5, 2), max_iter=50
clf_sgd.fit(X_train_scaled, y_train)

pred_sgd = clf.predict(X_test_scaled)

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

print(classification_report(y_test, pred_sgd))
```

```
accuracy =  0.8846153846153846
              precision    recall  f1-score   support

           0       0.94      0.88      0.91        50
           1       0.81      0.89      0.85        28

    accuracy                           0.88        78
   macro avg       0.87      0.89      0.88        78
weighted avg       0.89      0.88      0.89        78

accuracy =  0.8846153846153846
              precision    recall  f1-score   support

           0       0.94      0.88      0.91        50
           1       0.81      0.89      0.85        28

    accuracy                           0.88        78
   macro avg       0.87      0.89      0.88        78
weighted avg       0.89      0.88      0.89        78
```

```
/Library/Frameworks/Python.framework/Versions/3.11/lib/python3.11/site-packag
es/sklearn/neural_network/_multilayer_perceptron.py:686: ConvergenceWarning:
Stochastic Optimizer: Maximum iterations (500) reached and the optimization h
asn't converged yet.
  warnings.warn(
```

11.

The 2 neural networks got the same response. In terms of efficiency in algorithms, the logistic regression using Sci kit learn was the fastest, next to decision trees and the neural networks. Maybe with a larger dataset, the difference will be more prevalant. Decision Trees are known to be a greedy algorithm because of how they work by splitting at every single node. With datasets larger than normal, this can get highly inefficient. I was expecting the Neural Networks to be the fastest, maybe I did something wrong. I wish I had more time to investigate, but it is definitely something I will be observant of as I continue to experiment with algorithms vs Neural Networks in later projects.

I don't see a significant difference between R and Python so far because it is all just knowing a bunch of keywords for me. Having coded in Python, this is not coding in Python, but a good example of syntactically powerful the languge us. At the stage we're at of ML in Python, I would prefer R because I like to be able to see the data/datatypes/workspace in R Studios. Although that is an IDE difference, that is really the only difference I've noticed. There is more documentation online for Python, which is definitely a plus point.