Jonathan Ho CS 4375

ML with sklearn using Auto dataset

1. Reading in the Auto data

(a. and b.) Using pandas to read the data and output the first few rows

In [128…

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

Out[128…

	mpg	cylinders	displacement	horsepower	weight	acceleration	year	origin	name
0	18.0	8	307.0	130	3504	12.0	70.0	1	chevrolet chevelle malibu
1	15.0	8	350.0	165	3693	11.5	70.0	1	buick skylark 320
2	18.0	8	318.0	150	3436	11.0	70.0	1	plymouth satellite
3	16.0	8	304.0	150	3433	12.0	70.0	1	amc rebel sst
4	17.0	8	302.0	140	3449	NaN	70.0	1	ford torino

(c.) Output dimensions of the data

In [129…

```
print("Dimensions of data frame: ", df.shape)
```

```
Dimensions of data frame:  (392, 9)
```

The first number is how much data there is, and the second is how many attribute columns there are

2. Data exploration with code

(a. and b.) Using describe on the mpg, weight, and year columns then noting the range and mean of each

In [130…

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

Out[130…

	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

	mpg	weight	year
max	46.600000	5140.000000	82.000000

The mean and range of each attribute described are as follows:

1. mpg
   - mean = 23.445918
   - range = {9, 46.6} or 37.6
2. weight
   - mean = 2977.584184
   - range = {1613, 5140} or 3527
3. year
   - mean = 76.010256
   - range = {70, 82} or 12

3. Explore data types

a. Check data types of all columns

In [131…

```
print("Column data types: ")
print(df.dtypes)
```

```
Column data types:
mpg             float64
cylinders         int64
displacement    float64
horsepower        int64
weight            int64
acceleration    float64
year            float64
origin            int64
name             object
dtype: object
```

(b. and d.) Changing the cylinders column to categorical (with cat.codes) and verified.

In [132…

```
df['cylinders'] = df.cylinders.astype('category').cat.codes
print(df.dtypes)
```

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

(c. and d.) Changing the origin column to categorical (without cat.codes) and verified

In [133…

```
df = df.astype({"origin":'category'})
print(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

a. Delete rows with NAs

In [134…

```
print(df.isnull().any())
```

```
mpg             False
cylinders       False
displacement    False
horsepower      False
weight          False
acceleration     True
year             True
origin          False
name            False
dtype: bool
```

Drop all rows that have an NaN

In [135…

```
df = df.dropna()
print(df.isnull().any())
df.head()
```

```
mpg             False
cylinders       False
displacement    False
horsepower      False
weight          False
acceleration    False
year            False
origin          False
name            False
dtype: bool
```

Out[135…

	mpg	cylinders	displacement	horsepower	weight	acceleration	year	origin	name
0	18.0	4	307.0	130	3504	12.0	70.0	1	chevrolet chevelle malibu
1	15.0	4	350.0	165	3693	11.5	70.0	1	buick skylark 320
2	18.0	4	318.0	150	3436	11.0	70.0	1	plymouth satellite
3	16.0	4	304.0	150	3433	12.0	70.0	1	amc rebel sst
6	14.0	4	454.0	220	4354	9.0	70.0	1	chevrolet impala

Renumber index of data after NaNs are dropped

In [136...

```
df = df.reset_index(drop=True) # Reset the index numbering
df.head()
```

Out[136...

	mpg	cylinders	displacement	horsepower	weight	acceleration	year	origin	name
0	18.0	4	307.0	130	3504	12.0	70.0	1	chevrolet chevelle malibu
1	15.0	4	350.0	165	3693	11.5	70.0	1	buick skylark 320
2	18.0	4	318.0	150	3436	11.0	70.0	1	plymouth satellite
3	16.0	4	304.0	150	3433	12.0	70.0	1	amc rebel sst
4	14.0	4	454.0	220	4354	9.0	70.0	1	chevrolet impala

b. Output new dimensions

In [137...

```
print(df.shape)
```

```
(389, 9)
```

5. Modify columns

a. Make a new column, mpg_high, and make categorical with 1 being mpg > average mpg and 0 otherwise

In [138...

```
avg_mpg = df['mpg'].mean()
print(avg_mpg)
```

```
23.490488431876607
```

In [139...

```
# Import numpy
import numpy as np

mpg_t_f = [(df['mpg'] > avg_mpg), (df['mpg'] <= avg_mpg)]
value = [1, 0]
df['mpg_high'] = np.select(mpg_t_f, value)
df
```

Out[139...

	mpg	cylinders	displacement	horsepower	weight	acceleration	year	origin	name	mpg_hig
0	18.0	4	307.0	130	3504	12.0	70.0	1	chevrolet chevelle malibu	
1	15.0	4	350.0	165	3693	11.5	70.0	1	buick skylark 320	
2	18.0	4	318.0	150	3436	11.0	70.0	1	plymouth satellite	
3	16.0	4	304.0	150	3433	12.0	70.0	1	amc rebel sst	
4	14.0	4	454.0	220	4354	9.0	70.0	1	chevrolet impala	

	mpg	cylinders	displacement	horsepower	weight	acceleration	year	origin	name	mpg_hig
...	...	...	...	...	...	...	...	...	...	
384	27.0	1	140.0	86	2790	15.6	82.0	1	ford mustang gl	
385	44.0	1	97.0	52	2130	24.6	82.0	2	vw pickup	
386	32.0	1	135.0	84	2295	11.6	82.0	1	dodge rampage	
387	28.0	1	120.0	79	2625	18.6	82.0	1	ford ranger	
388	31.0	1	119.0	82	2720	19.4	82.0	1	chevy s-10	

389 rows × 10 columns

(b. and c.) Delete mpg and name columns and output the first few rows

In [140…

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

Out[140…

	cylinders	displacement	horsepower	weight	acceleration	year	origin	mpg_high
0	4	307.0	130	3504	12.0	70.0	1	0
1	4	350.0	165	3693	11.5	70.0	1	0
2	4	318.0	150	3436	11.0	70.0	1	0
3	4	304.0	150	3433	12.0	70.0	1	0
4	4	454.0	220	4354	9.0	70.0	1	0

6. Data exploration with graphs

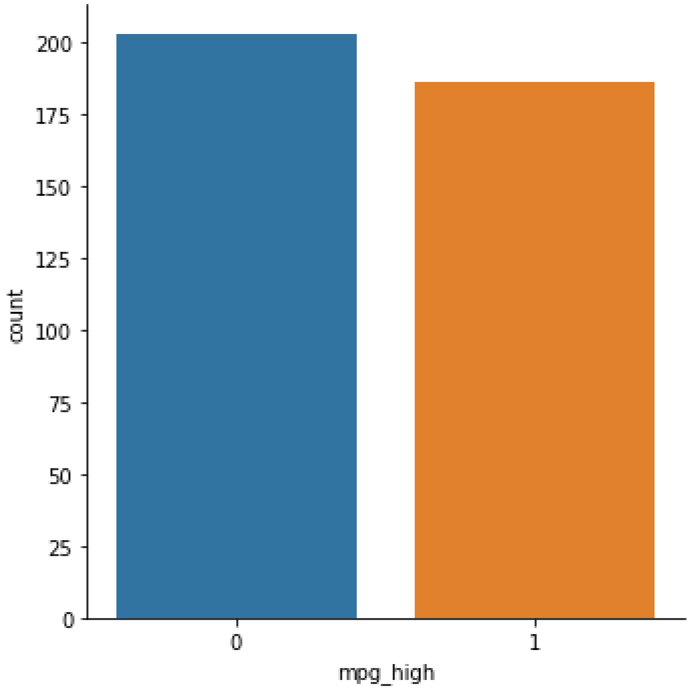
(a. and d.) Seaborn catplot on the mpg_high column and takeaway.

In [141…

```
import seaborn as sb

sb.catplot(x="mpg_high", kind='count', data=df)
```

Out[141…

```
<seaborn.axisgrid.FacetGrid at 0x2324f2ae130>
```



This plotting of categorical data tells me how many are mpg_high and how many are not.

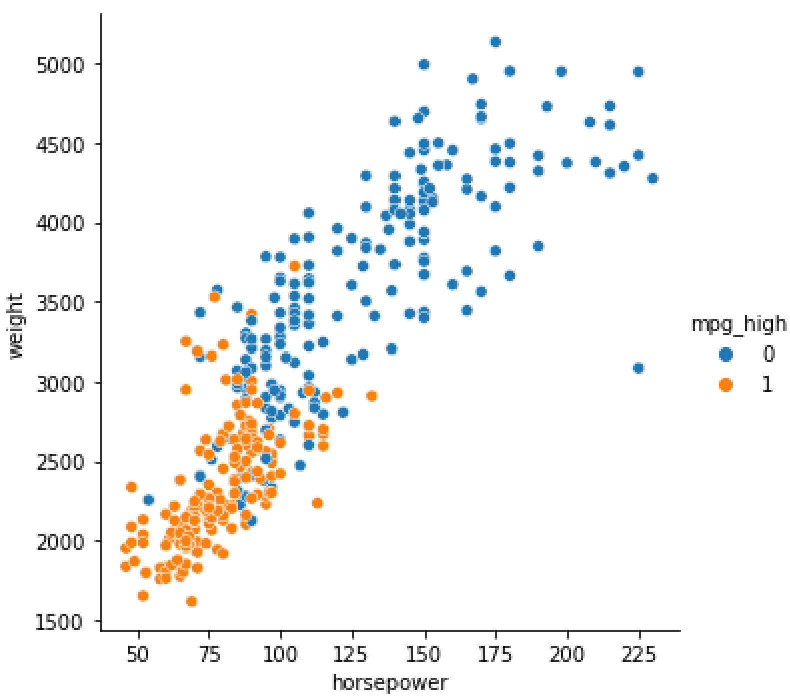
(b. and d.) Seaborn relplot with horsepower on x axis, weight on y axis, and setting hue to mpg_high and takeaway.

In [142…

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

Out[142…

```
<seaborn.axisgrid.FacetGrid at 0x2324fca1d00>
```

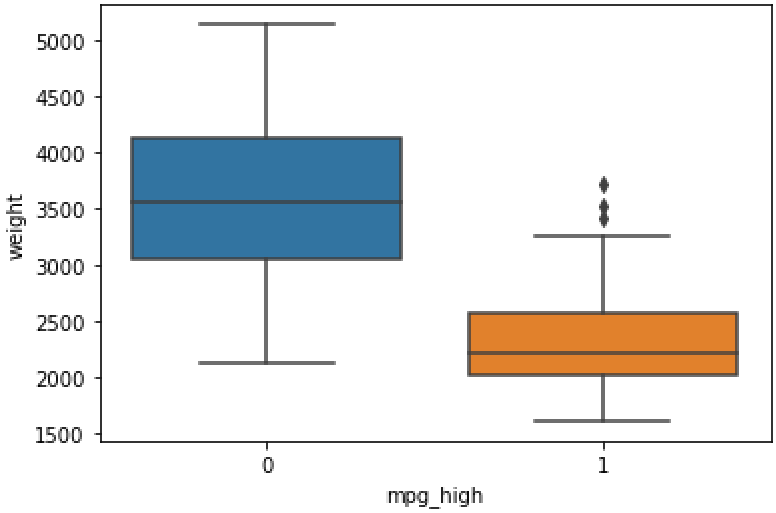


The relation plot shows a relation between weight, horsepower, and mpg_high. A vehicle is normally mpg_high if it has a lower weight and horsepower.

(c. and d.) Seaborn boxplot with mpg_high on the x axis and weight on the y axis and takeaway.

In [143…
```
sb.boxplot(x='mpg_high', y='weight', data=df)
```

Out[143…
```
<AxesSubplot:xlabel='mpg_high', ylabel='weight'>
```



The takeaway from the box plot is that on average, mpg_high vehicles are around the 2000 - 2500 weight range, while not mpg_high vehicles are around the 3100 - 4200 weight range. However, it is seen that there are outliers where there are cars in the 3400 - 3800 weight range that is mpg_high.

7. Train/test split

(a. - d.) 80/20 split with seed 1234. X has train and test of all attributes except mpg_high, which is stored in a different variable. Dimensions of X train and test are output.

In [144…
```
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)
```

8. Logistic Regression

a. Train a logistic regression model. The default solver method is lbfgs and the data comes from the train/test split. Max iterations had to be increased from the default 100 to 110.

In [145…
```
from sklearn.linear_model import LogisticRegression

clf = LogisticRegression(max_iter=110)
clf.fit(X_train, Y_train)
clf.score(X_train, Y_train)
```

Out[145…
```
0.9067524115755627
```

b Test and evaluate the metrics as well as tthe confusion matrix.

In [146…

```
# Make predictions given the test dataset
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))
print('recall score: ', recall_score(Y_test, pred))
print('f1 score: ', f1_score(Y_test, pred))
```

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

Confusion matrix metric. Astype function used at the end of the confusion matrix function to get rid of 'dtype=int64'.

In [147…

```
# Confusion matrix
from sklearn.metrics import confusion_matrix

confusion_matrix(Y_test, pred).astype('int')
```

Out[147…

```
array([[40, 10],
       [ 1, 27]])
```

c. Print out the classification report metrics

In [148…

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

```
              precision    recall  f1-score   support

           0       0.98      0.80      0.88        50
           1       0.73      0.96      0.83        28

    accuracy                           0.86        78
   macro avg       0.85      0.88      0.85        78
weighted avg       0.89      0.86      0.86        78
```

9. Decision Tree

a. Train a decision tree (using the train and test data from section 7.)

In [149…

```
from sklearn.tree import DecisionTreeClassifier

clf_dt = DecisionTreeClassifier(random_state=1234)
clf_dt.fit(X_train, Y_train)
```

Out[149…

```
DecisionTreeClassifier(random_state=1234)
```

b Test and evaluate the metrics as well as the confusion matrix. Astype function used at the end of the confusion matrix function to get rid of 'dtype=int64'.

In [150…

```
# Make predictions given the test dataset
pred_dt = clf_dt.predict(X_test)

# Evaluate
print('accuracy score: ', accuracy_score(Y_test, pred_dt))
print('precision score: ', precision_score(Y_test, pred_dt))
print('recall score: ', recall_score(Y_test, pred_dt))
print('f1 score: ', f1_score(Y_test, pred_dt))

# Confusion matrix
confusion_matrix(Y_test, pred_dt).astype('int')
```

```
accuracy score:  0.9230769230769231
precision score:  0.8666666666666667
recall score:  0.9285714285714286
f1 score:  0.896551724137931
```

Out[150…

```
array([[46,  4],
       [ 2, 26]])
```

c. Print out the classification report metrics

In [151…

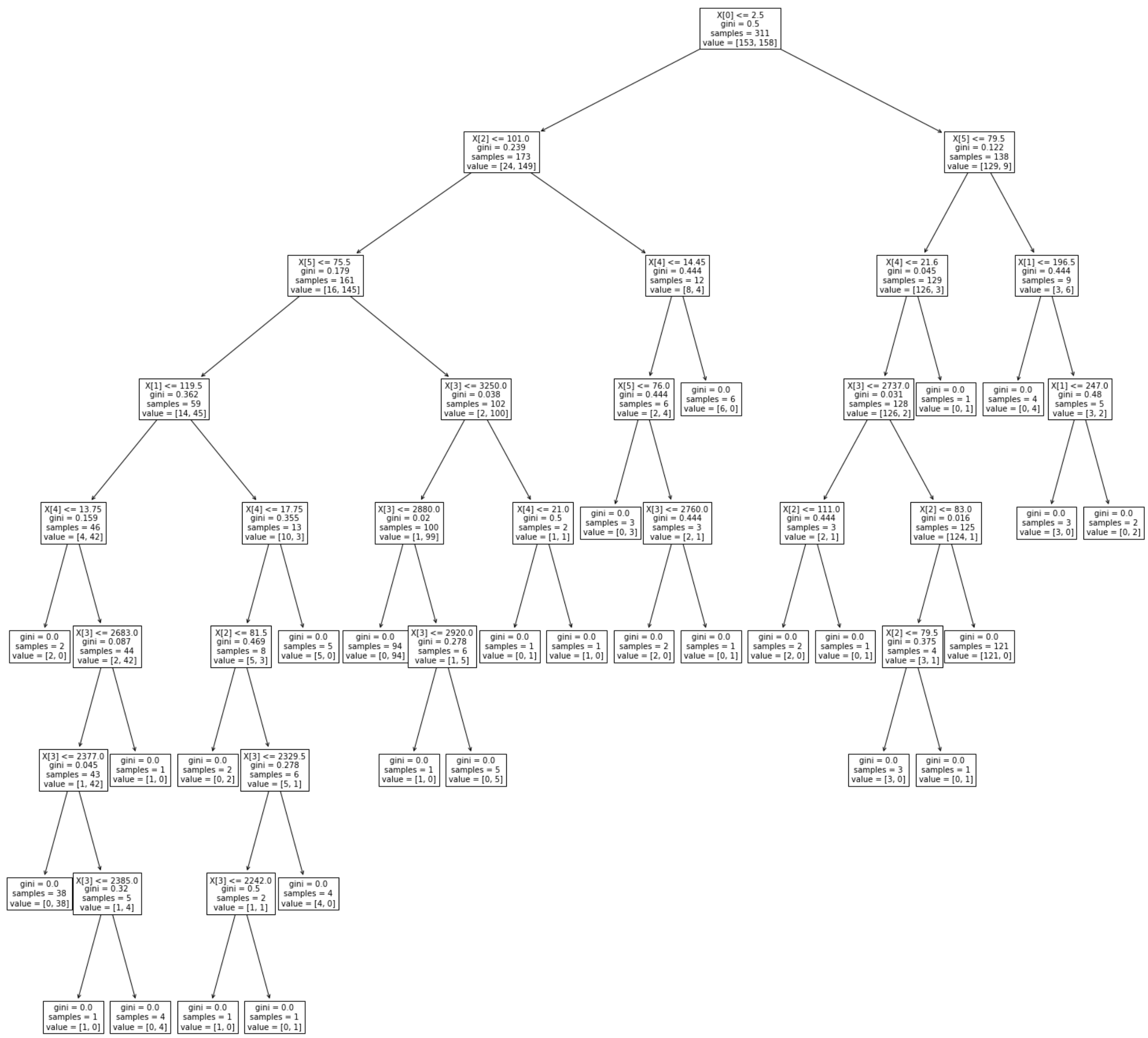
```
print(classification_report(Y_test, pred_dt))
```

```
              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
```

d. Plot the tree. I believe it is comparing all the different attributes and comparing them to a specific given value.

In [152…

```
from sklearn import tree
import matplotlib.pyplot as plt

plt.figure(figsize=(26, 26))
tree.plot_tree(clf_dt, fontsize=10)
plt.show()
```



10. Neural Network

a. Neural network training using the multi-layer perceptron topology for classification.

Recall the metrics using logistical regression was the following:

In [153…

```
print('accuracy score: ', accuracy_score(Y_test, pred))
print('precision score: ', precision_score(Y_test, pred))
print('recall score: ', recall_score(Y_test, pred))
confusion_matrix(Y_test, pred).astype('int')
```

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

Out[153…

```
array([[40, 10],
       [ 1, 27]])
```

Carrying out a neural network, data must be normalized and then trained.

In [154…

```
# Normalizing data
```

```
from sklearn import preprocessing

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

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

# Training
from sklearn.neural_network import MLPClassifier

clf_nn1 = MLPClassifier(solver='lbfgs', hidden_layer_sizes=(10, 8, 3),
                        max_iter=500, random_state=1234)
clf_nn1.fit(X_train_scaled, Y_train)
```

Out[154…

```
MLPClassifier(hidden_layer_sizes=(10, 8, 3), max_iter=500, random_state=1234,
              solver='lbfgs')
```

b. Make a prediction with the test data and evaluate.

In [155…

```
# Prediction
pred_nn1 = clf_nn1.predict(X_test_scaled)

# Output results
print('accuracy of NN1 = ', accuracy_score(Y_test, pred_nn1))
print('precision score: ', precision_score(Y_test, pred_nn1))
print('recall score: ', recall_score(Y_test, pred_nn1))
confusion_matrix(Y_test, pred_nn1).astype('int')
```

```
accuracy of NN1 =  0.8974358974358975
precision score:  0.7941176470588235
recall score:  0.9642857142857143
```

Out[155…

```
array([[43,  7],
       [ 1, 27]])
```

c. Second neural network training using a different topology and settings (Keras).

In [189…

```
# Renaming train and test data for Keras
from sklearn.model_selection import train_test_split

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

X_train_keras, X_test_keras, Y_train_keras, Y_test_keras = train_test_split(X, Y,
                                                            test_size=0.2,
                                                            random_state=1234)
```

In [181…

```
# Normalizing data
scaler_keras = preprocessing.StandardScaler().fit(X_train_keras)

X_train_keras = scaler_keras.transform(X_train_keras)
X_test_keras = scaler_keras.transform(X_test_keras)

# Converting class vectors to binary class matrices
import keras
tf.keras.utils.set_random_seed(1234)
```

```
Y_train_keras = keras.utils.to_categorical(Y_train_keras, 2)
Y_test_keras = keras.utils.to_categorical(Y_test_keras, 2)
```

In [182…

```
from __future__ import print_function

from keras.models import Sequential
from keras.layers import Dense, Dropout
from keras.optimizers import RMSprop

batch_size = 128
epochs = 15

model = Sequential()
model.add(Dense(512, activation='relu', input_shape=(7,)))
model.add(Dropout(0.2))
model.add(Dense(2, activation='sigmoid'))
```

d. Make a prediction with the test data and evaluate

In [183…

```
model.compile(loss='binary_crossentropy',
              optimizer=RMSprop(),
              metrics=['accuracy'])

history = model.fit(X_train_keras, Y_train_keras,
                    batch_size=batch_size,
                    epochs=epochs,
                    verbose=1,
                    validation_data=(X_test_keras, Y_test_keras))
```

```
Epoch 1/15
3/3 [==============================] - 2s 245ms/step - loss: 0.6880 - accuracy: 0.5370 -
val_loss: 0.5618 - val_accuracy: 0.8462
Epoch 2/15
3/3 [==============================] - 0s 42ms/step - loss: 0.5290 - accuracy: 0.8714 -
val_loss: 0.4744 - val_accuracy: 0.8590
Epoch 3/15
3/3 [==============================] - 0s 45ms/step - loss: 0.4552 - accuracy: 0.8810 -
val_loss: 0.4234 - val_accuracy: 0.8590
Epoch 4/15
3/3 [==============================] - 0s 42ms/step - loss: 0.4036 - accuracy: 0.8939 -
val_loss: 0.3870 - val_accuracy: 0.8590
Epoch 5/15
3/3 [==============================] - 0s 44ms/step - loss: 0.3676 - accuracy: 0.8971 -
val_loss: 0.3636 - val_accuracy: 0.8590
Epoch 6/15
3/3 [==============================] - 0s 40ms/step - loss: 0.3392 - accuracy: 0.8939 -
val_loss: 0.3467 - val_accuracy: 0.8590
Epoch 7/15
3/3 [==============================] - 0s 44ms/step - loss: 0.3198 - accuracy: 0.8971 -
val_loss: 0.3320 - val_accuracy: 0.8590
Epoch 8/15
3/3 [==============================] - 0s 44ms/step - loss: 0.2996 - accuracy: 0.8971 -
val_loss: 0.3194 - val_accuracy: 0.8590
Epoch 9/15
3/3 [==============================] - 0s 43ms/step - loss: 0.2856 - accuracy: 0.8971 -
val_loss: 0.3094 - val_accuracy: 0.8590
Epoch 10/15
```

```
3/3 [==============================] - 0s 37ms/step - loss: 0.2754 - accuracy: 0.9003 -
val_loss: 0.3050 - val_accuracy: 0.8590
Epoch 11/15
3/3 [==============================] - 0s 45ms/step - loss: 0.2655 - accuracy: 0.9035 -
val_loss: 0.2952 - val_accuracy: 0.8590
Epoch 12/15
3/3 [==============================] - 0s 47ms/step - loss: 0.2575 - accuracy: 0.9003 -
val_loss: 0.2881 - val_accuracy: 0.8590
Epoch 13/15
3/3 [==============================] - 0s 48ms/step - loss: 0.2489 - accuracy: 0.9035 -
val_loss: 0.2876 - val_accuracy: 0.8590
Epoch 14/15
3/3 [==============================] - 0s 47ms/step - loss: 0.2441 - accuracy: 0.9003 -
val_loss: 0.2796 - val_accuracy: 0.8590
Epoch 15/15
3/3 [==============================] - 0s 45ms/step - loss: 0.2373 - accuracy: 0.9003 -
val_loss: 0.2717 - val_accuracy: 0.8590
```

e. Analysis between the two models.

Comparing between Multi-Layer Perceptron (MLP) and Keras neural networks, it appears that the MLP obtains a higher accuracy than Keras. Both however still obtain higher accuracies than the control of logisitc regression. One reason the accuracies could be different is the parameters I used, specifically with the neural network layers themselves. In MLP, the first hidden layer I used is of size 10, which is following the suggestion of trying to find the hidden layer sizes (the one that it is less than double the number of attributes). There is also only three hidden layers. For Keras, only an input layer is taken in, which is just a size less than the number of attributes (7,). The hidden layers are the Dense and Dropout I believe. Another reason for accuracy loss could be either MLP overfitting or Keras underfitting. My guess on which model is doing so, probably MLP since Keras usually overfits data more often. It also could be for MLP, my hidden layers setup may not be correct, but I cannot be sure.

11. Analysis

(a. and b.) Which algorithm performed better? Compare accuracy, recall, and precision metrics by class.

These were the classification reports of each algorithm by class:

In [160…

```python
print("Logicistic Regression:")
print(classification_report(Y_test, pred))

print("Decision Tree:")
print(classification_report(Y_test, pred_dt))

print("Multi-Layer Perceptron")
print(classification_report(Y_test, pred_nn1))
```

```
Logicistic Regression:
              precision    recall  f1-score   support

           0       0.98      0.80      0.88        50
           1       0.73      0.96      0.83        28

    accuracy                           0.86        78
   macro avg       0.85      0.88      0.85        78
```

```
weighted avg       0.89      0.86      0.86        78

Decision Tree:
              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

Multi-Layer Perceptron
              precision    recall  f1-score   support

           0       0.98      0.86      0.91        50
           1       0.79      0.96      0.87        28

    accuracy                           0.90        78
   macro avg       0.89      0.91      0.89        78
weighted avg       0.91      0.90      0.90        78
```

It seems that precision and recall were usually better for cars that were not mpg_high rather than cars that were. This could be due to more cars that are not mpg_high than those that are. Also there seems to be a couple of outliers in mpg_high cars based off the boxplot, so that could alter the metrics.

Based off the results, it seems that the decision tree algorithm performed the best in both accuracy and precision, but both logistic regression and MLP had the highest recalls. Overall, I think decision trees performed the best on the data. There is a chance that decision trees are overfitting since I did not prune the tree, but I am going to assume here that it is fine.

c. Why did the best performing algorithm outperform the other two?

Decision trees outperformed logistic regression potentially because the data itself may not be linearly separable. Decision trees are non-linear classifiers while logistic regression are linear classifiers. If the plotted data has trouble separating data linearly, then it is possible the metrics for the algorithm may be lower. In this case, I believe that the data may not be entirely separatable by a linear function, so a decision tree did better. As for neural networks, decision tree may be better because it is a better fit for smaller datasets. The Auto.csv dataset only has, after changes, about 8 columns with approximately 300 tuples. This is a relatively small data set versus medium or big ones. Since it is so simple, a complex algorithm like neural networks may be working too hard to try to work, which in turn ends up hurting the accuracy. That is not to say that decision trees cannot overfit data either, but assuming data did not overfit, decision tree did out perform MLP.

d. Experience of R vs. Sklearn.

Having experience now using both R and Sklearn, I am glad I was exposed to both. However, I think I would prefer to use sklearn over R in more cases than not. Sklearn feels very streamlined with minimal need to do any formatting. It also is in the python language which I am more familiar with. R is also pretty simple too, but just me having less exposure to the syntax versus Python makes it slightly more difficult to use. However, I do like that in RStudio, you are able to see all the objects stored within memory easily, such as the test and train data tables. That, and formatting of the notebook when using knit looks nicer than JupyterLab using sklearn, although I am guessing there is a LaTex option for Jupyter notebooks. So overall, both feel pretty user friendly, but I prefer sklearn more just due to more familiarity of using Python.