Dmitrii Obideiko

1. Read the Auto data

```
import pandas as pd

# Use pandas to read the data
df = pd.read_csv('Auto.csv')

# Output the first few rows
print(df.head())

# Output the dimensions of the data
print('Dimensions of the data: ', 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 DataFrame:  (392, 9)
```

2. Data exploration with code

```
# Use describe() on the mpg, weight, and year columns
print(df[['mpg', 'weight', 'year']].describe())

# Find range and average of the "mpg" column
mpg_min = df['mpg'].min()
mpg_max = df['mpg'].max()
mpg_range = mpg_max - mpg_min
mpg_mean = df['mpg'].mean()
print("Range of 'mpg' column:", mpg_range)
print("Average of 'mpg' column:", mpg_mean)
```

```
# Find range and average of the "weight" column
weight_min = df['weight'].min()
weight_max = df['weight'].max()
weight_range = weight_max - weight_min
weight_mean = df['weight'].mean()
print("Range of 'weight' column:", weight_range)
print("Average of 'weight' column:", weight_mean)

# Find range and average of the "year" column
year_min = df['year'].min()
year_max = df['year'].max()
year_range = year_max - year_min
year_mean = df['year'].mean()
print("Range of 'year' column:", year_range)
print("Average of 'year' column:", year_mean)
```

```
              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' column: 37.6
Average of 'mpg' column: 23.445918367346938
Range of 'weight' column: 3527
Average of 'weight' column: 2977.5841836734694
Range of 'year' column: 12.0
Average of 'year' column: 76.01025641025642
```

3. Explore data types

```
# Check the data types of all columns
print(df.dtypes)

# Change the "cylinders" column to categorical
df['cylinders'] = df['cylinders'].astype('category')
df['cylinders'] = df['cylinders'].cat.codes

# Change the origin column to categorical
df['origin'] = df['origin'].astype('category')

# Verify the changes with the dtypes attribute
print('\n')
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
```

4. Deal with NAs

```
# Drop rows with NAs
df.dropna(inplace=True)

# Output the new dimensions
print("New dimensions:", df.shape)
```

```
New dimensions: (389, 9)
```

5. Modify columns

```
## Make a new column, mpg_high, and make it categorical
# Calculate the average of mpg
avg_mpg = df['mpg'].mean()
df['mpg_high'] = (df['mpg'] > avg_mpg).astype('category')
df['mpg_high'].cat.rename_categories({False: '0', True: '1'}, inplace=True)

# Delete the mpg and name columns (delete mpg so the algorithm doesn’t just learn to
df.drop(['mpg', 'name'], axis=1, inplace=True)

# Output the first few rows of the modified data frame
print(df.head())
```

```
   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
<ipython-input-7-4d234214e67b>:6: FutureWarning: The `inplace` parameter in pand
  df['mpg_high'].cat.rename_categories({False: '0', True: '1'}, inplace=True)
```

6. Data exploration with graphs

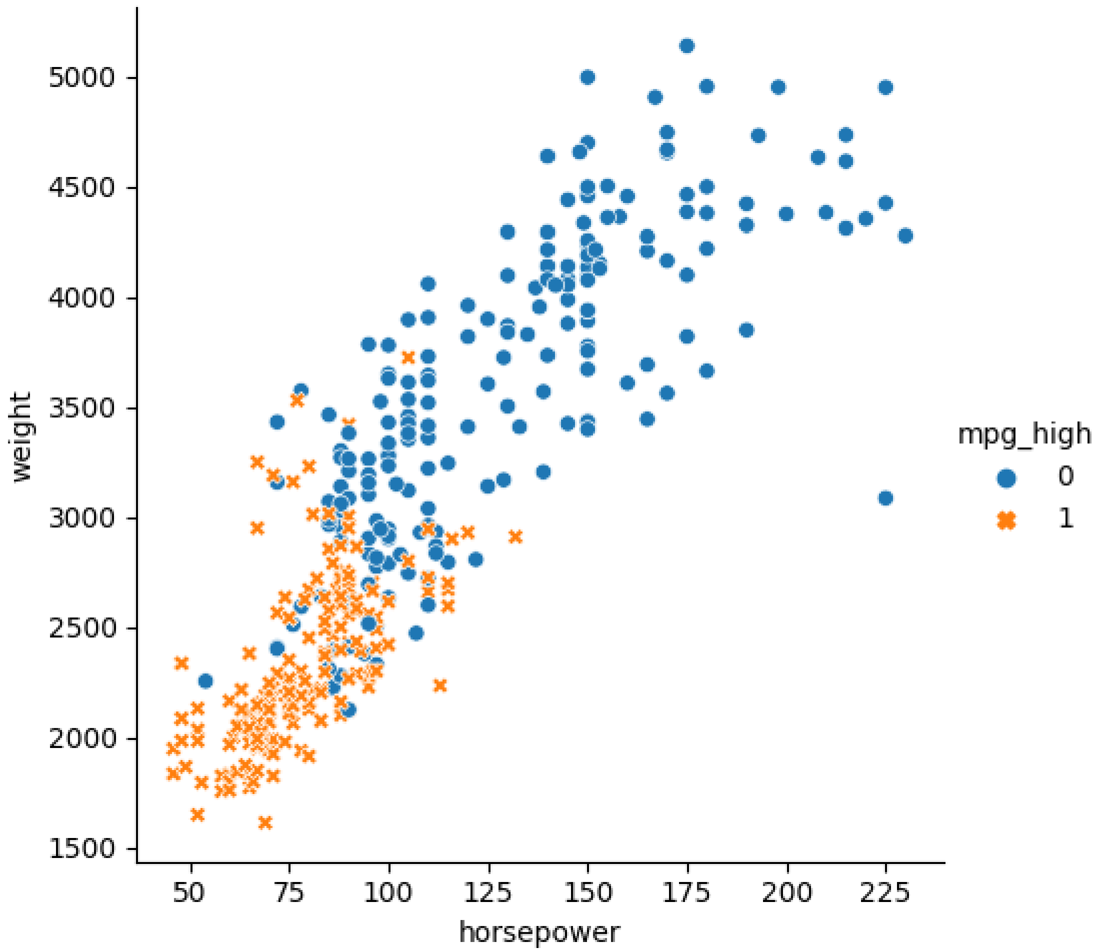
```
import seaborn as sns

# Create a Seaborn catplot on the mpg_high column
sns.catplot(data=df, x='mpg_high', kind='count')
```

It looks like when mpg_high is 0, the count is higher. Although the difference is not significant.

```
# Create a Seaborn relplot with horsepower on the x axis, weight on the y axis, setti
sns.relplot(data=df, x='horsepower', y='weight', hue='mpg_high', style='mpg_high')
```

```
<seaborn.axisgrid.FacetGrid at 0x7f5ddd573790>
```

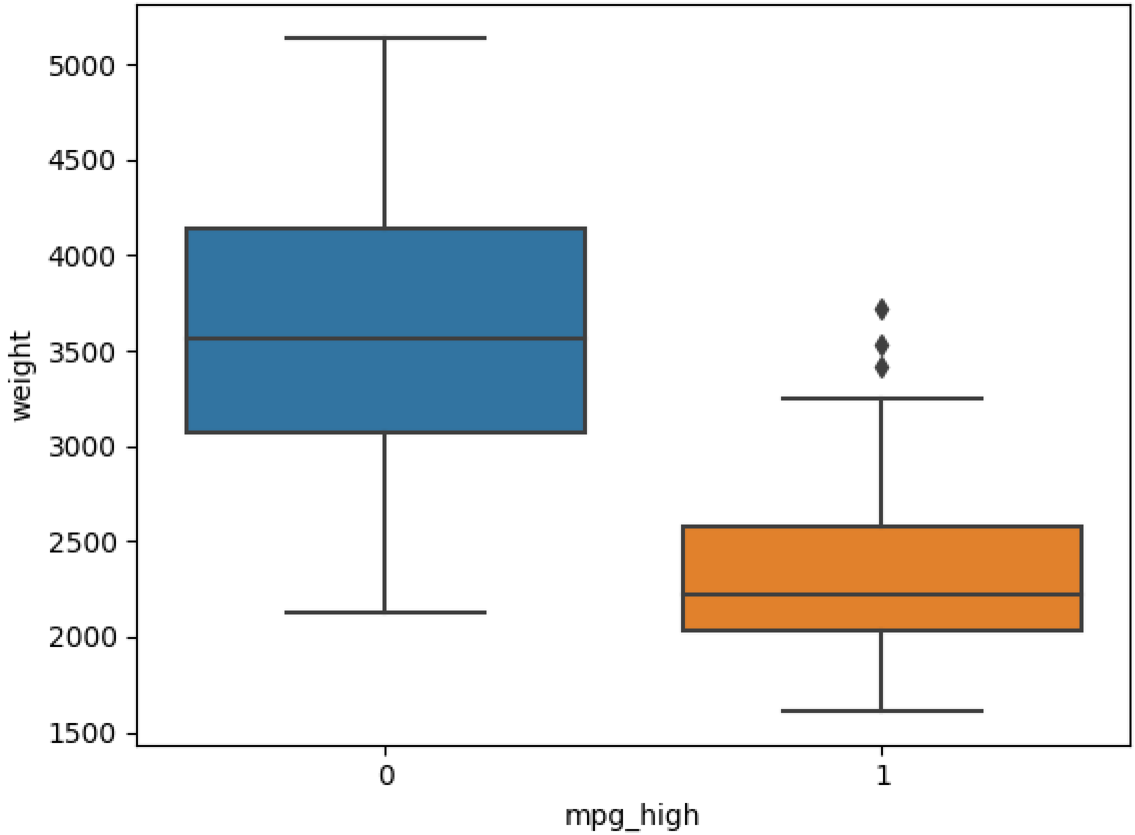


It looks like mpg = 1 more often when the horsepower is between 0 and 100 and weight is beween 1500 and 3000.

It looks like mpg = 0 more often when teh horsepower is between 125 adn 225 and weight is between 3000 and 4500.

```
# Seaborn boxplot with mpg_high on the x axis and weight on the y axis
sns.boxplot(data=df, x='mpg_high', y='weight')
```

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



It looks like the medium weight for when mpg_high = 0 is 3500 and the medium weight for when mpg_high = 1 is 2250.

7. Train/test split

```
from sklearn.model_selection import train_test_split
# Split the data into 80% training and 20% testing sets
X = df.drop('mpg_high', axis=1)
y = df['mpg_high']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_sta

# Output the dimensions of the training and testing sets
print('Training data dimensions:', X_train.shape)
print('Testing data dimensions:', X_test.shape)
```

```
Training data dimensions: (311, 7)
Testing data dimensions: (78, 7)
```

8. Logistic Regression

```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report

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

# Test and evaluate
y_pred = model.predict(X_test)

# Print metrics using the classification report
print(classification_report(y_test, y_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

/usr/local/lib/python3.9/dist-packages/sklearn/linear_model/_logistic.py:458: Cc
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-regressic
  n_iter_i = _check_optimize_result(
```

9. Decision Tree

```
from sklearn.tree import DecisionTreeClassifier, plot_tree
from sklearn.metrics import classification_report

# Train a neural network
model = DecisionTreeClassifier(random_state=1234)
model.fit(X_train, y_train)

# Test and evaluate
y_pred = model.predict(X_test)
```

```
# Print the classification report metrics
print(classification_report(y_test, y_pred))

# Plot the tree
plot_tree(model, feature_names=X_train.columns)
```

```
              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

[Text(0.6433823529411765, 0.9444444444444444, 'cylinders <= 2.5\ngini = 0.5\nsam
 Text(0.4338235294117647, 0.8333333333333334, 'horsepower <= 101.0\ngini = 0.239
 Text(0.27941176470588236, 0.7222222222222222, 'year <= 75.5\ngini = 0.179\nsamp
 Text(0.14705882352941177, 0.6111111111111112, 'displacement <= 119.5\ngini = 0.
 Text(0.058823529411764705, 0.5, 'acceleration <= 13.75\ngini = 0.159\nsamples =
 Text(0.029411764705882353, 0.3888888888888889, 'gini = 0.0\nsamples = 2\nvalue
 Text(0.08823529411764706, 0.3888888888888889, 'weight <= 2683.0\ngini = 0.087\n
 Text(0.058823529411764705, 0.2777777777777778, 'weight <= 2377.0\ngini = 0.045\
 Text(0.029411764705882353, 0.16666666666666666, 'gini = 0.0\nsamples = 38\nvalu
 Text(0.08823529411764706, 0.16666666666666666, 'weight <= 2385.0\ngini = 0.32\n
 Text(0.058823529411764705, 0.05555555555555555, 'gini = 0.0\nsamples = 1\nvalue
 Text(0.11764705882352941, 0.05555555555555555, 'gini = 0.0\nsamples = 4\nvalue
 Text(0.11764705882352941, 0.2777777777777778, 'gini = 0.0\nsamples = 1\nvalue =
 Text(0.23529411764705882, 0.5, 'acceleration <= 17.75\ngini = 0.355\nsamples =
 Text(0.20588235294117646, 0.3888888888888889, 'horsepower <= 81.5\ngini = 0.469
 Text(0.17647058823529413, 0.2777777777777778, 'gini = 0.0\nsamples = 2\nvalue =
 Text(0.23529411764705882, 0.2777777777777778, 'weight <= 2329.5\ngini = 0.278\n
 Text(0.20588235294117646, 0.16666666666666666, 'weight <= 2242.0\ngini = 0.5\ns
 Text(0.17647058823529413, 0.05555555555555555, 'gini = 0.0\nsamples = 1\nvalue
 Text(0.23529411764705882, 0.05555555555555555, 'gini = 0.0\nsamples = 1\nvalue
 Text(0.2647058823529412, 0.16666666666666666, 'gini = 0.0\nsamples = 4\nvalue =
 Text(0.2647058823529412, 0.3888888888888889, 'gini = 0.0\nsamples = 5\nvalue =
 Text(0.4117647058823529, 0.6111111111111112, 'weight <= 3250.0\ngini = 0.038\ns
 Text(0.35294117647058826, 0.5, 'weight <= 2880.0\ngini = 0.02\nsamples = 100\nv
 Text(0.3235294117647059, 0.3888888888888889, 'gini = 0.0\nsamples = 94\nvalue =
 Text(0.38235294117647056, 0.3888888888888889, 'weight <= 2920.0\ngini = 0.278\n
 Text(0.35294117647058826, 0.2777777777777778, 'gini = 0.0\nsamples = 1\nvalue =
 Text(0.4117647058823529, 0.2777777777777778, 'gini = 0.0\nsamples = 5\nvalue =
 Text(0.47058823529411764, 0.5, 'acceleration <= 21.0\ngini = 0.5\nsamples = 2\n
 Text(0.4411764705882353, 0.3888888888888889, 'gini = 0.0\nsamples = 1\nvalue =
 Text(0.5, 0.3888888888888889, 'gini = 0.0\nsamples = 1\nvalue = [1, 0]'),
 Text(0.5882352941176471, 0.7222222222222222, 'acceleration <= 14.45\ngini = 0.4
```

10. Neural Network

```
 Text(0.6176470588235294, 0.3888888888888889, 'gini = 0.0\nsamples = 1\nvalue =
```

```
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import classification_report

# Train a neural network
model1 = MLPClassifier(hidden_layer_sizes=(5, 2), random_state=1234)
model1.fit(X_train, y_train)

# Test and evaluate
y_pred1 = model1.predict(X_test)
```

```
print("Model 1")
print(classification_report(y_test, y_pred1))

# Train a second network with a different topology and different settings
model2 = MLPClassifier(hidden_layer_sizes=(10, 5, 2), max_iter=500, random_state=1234
model2.fit(X_train, y_train)

# Test and evaluate
y_pred2 = model2.predict(X_test)
print("Model 2")
print(classification_report(y_test, y_pred2))
```

```
Model 1
              precision    recall  f1-score   support

           0       1.00      0.02      0.04        50
           1       0.36      1.00      0.53        28

    accuracy                           0.37        78
   macro avg       0.68      0.51      0.29        78
weighted avg       0.77      0.37      0.22        78

Model 2
              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/python3.9/dist-packages/sklearn/metrics/_classification.py:1344:
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.9/dist-packages/sklearn/metrics/_classification.py:1344:
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.9/dist-packages/sklearn/metrics/_classification.py:1344:
  _warn_prf(average, modifier, msg_start, len(result))
```

Let's compare Model 1 and Model 2

It looks like Model 1 is better at predicting instances of class 1 as it has a higher recall for class 1 (1.00) while Model 2's recall for class 1 is 0.00. However, it looks like model 2 is better at predicting true negatives for class 0. This might be because of the differences in their topologies and settings in terms of the number of hidden layers and neurons as well as the number of iterations during training.

Analysis

It looks like the decision tree performed the best as it achieved the highest accruacy, which is 0.92. Logical regression, on the other hand, received a score of 0.92 and the two neural network models received accuracies of 0.37 and 0.64.

For class 0, model 2 of neural networks received the worst precision (0.64). Model 1 of neural networks received the best accuracy (1.00). The decision trees model and the logistic regression model received the accuracy of 0.96 and 0.98. When it comes to recall, model 2 received the highest score (1.0) The worst recall score was 0.02 and was received from neural networks (model 2). The decision tree model and the logistic regression model received scores of 0.92 and 0.80.

For class 1, decision trees received the higehst precision score (0.87). The lowest accuracy was acheived by Model 2 of Neural Networks (0.00). Logistic Regression and Model 1 of Neural Networks received a score of 0.73 and 0.36. Model 1 of Neural Networks received the highest recall score of 1.00. Model 2 of Neural Networks received the lowest recall score of 0.00. Logistic Regression and Decision trees received a recall score of 0.96 and 0.93.

The decision trees algorithsm outperformed all other alorithms that were used in this project based on acuracy scoers and racall scores. The decision tree model probably performed better because it can capture complex patterns in data as well as work with both numerical and categorical data. The decision tree model is more prone to overfitting if were were to compare it to neural netowrks, for example.

I would say that I still feel more confident using sklearn becasuse it's a python lirbary and I just feel more confident myself in python as I use it most of the time. I would say that it takes less lines to write something in R compared to python and for non computer science majors, R is probably a better langauge for them. At the end of the day, everything comes to prefereces, and the fact that I prfer python over R doesn't mean in any way that Python is better than R for machine learning.