Project Proposal

By Tom Adoni

We will look into the 'Auto MPG' dataset from UC Irvine ML Repository in order to explore trends in the automobile industry between 1970 and 1982.

In [1]:
```
import pandas as pd
df = pd.read_fwf("auto-mpg.data")
```

In [3]:
```
df.head()
```

Out[3]:

	18.0	8	307.0	130.0	3504.	12.0	70	1	"chevrolet chevelle malibu"
0	15.0	8	350.0	165.0	3693.0	11.5	70	1	"buick skylark 320"
1	18.0	8	318.0	150.0	3436.0	11.0	70	1	"plymouth satellite"
2	16.0	8	304.0	150.0	3433.0	12.0	70	1	"amc rebel sst"
3	17.0	8	302.0	140.0	3449.0	10.5	70	1	"ford torino"
4	15.0	8	429.0	198.0	4341.0	10.0	70	1	"ford galaxie 500"

Data Prep

In [4]:
```
df.columns =['displacement', 'mpg', 'cylinders', 'horsepower', 'weight', 'acceleration', 'model_year', 'origin', 'car_name']
```

In [5]:
```
df.head()
```

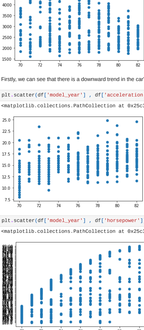
Out[5]:

	displacement	mpg	cylinders	horsepower	weight	acceleration	model_year	origin	car_name
0	15.0	8	350.0	165.0	3693.0	11.5	70	1	"buick skylark 320"
1	18.0	8	318.0	150.0	3436.0	11.0	70	1	"plymouth satellite"
2	16.0	8	304.0	150.0	3433.0	12.0	70	1	"amc rebel sst"
3	17.0	8	302.0	140.0	3449.0	10.5	70	1	"ford torino"
4	15.0	8	429.0	198.0	4341.0	10.0	70	1	"ford galaxie 500"

Data Viz

In [6]:
```
import matplotlib.pyplot as plt
plt.scatter(df['model_year'] , df['weight'])
```

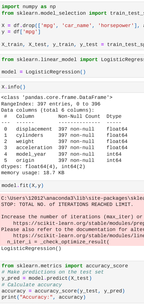
Out[6]:
```
<matplotlib.collections.PathCollection at 0x25c339928e0>
```



Firstly, we can see that there is a downward trend in the car's weight as time goes on.

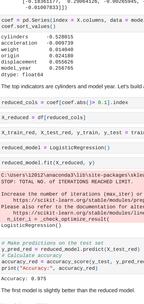
In [7]:
```
plt.scatter(df['model_year'] , df['acceleration'])
```

Out[7]:
```
<matplotlib.collections.PathCollection at 0x25c33a9b9d0>
```



In [8]:
```
plt.scatter(df['model_year'] , df['horsepower'])
```

Out[8]:
```
<matplotlib.collections.PathCollection at 0x25c33b168e0>
```



Next, we can see that there is an upward trend in the car's acceleration and horsepower as time advances.

We will explore more trends later on in the project.

We will evaluate which of these trends should influence a buyer's choice on these cars.

In [9]:
```
import numpy as np
from sklearn.model_selection import train_test_split

X = df.drop(['mpg', 'car_name', 'horsepower'], axis = 1)
y = df['mpg']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

In [10]:
```
from sklearn.linear_model import LogisticRegression

model = LogisticRegression()
```

In [11]:
```
X.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 397 entries, 0 to 396
Data columns (total 6 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   displacement  397 non-null    float64
 1   cylinders     397 non-null    float64
 2   weight        397 non-null    float64
 3   acceleration  397 non-null    float64
 4   model_year    397 non-null    int64
 5   origin        397 non-null    int64
dtypes: float64(4), int64(2)
memory usage: 18.7 KB
```

In [12]:
```
model.fit(X,y)
```

```
C:\Users\12012\anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:814: 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-regression
  n_iter_i = _check_optimize_result(
```

Out[12]:
```
LogisticRegression()
```

In [13]:
```
from sklearn.metrics import accuracy_score
# Make predictions on the test set
y_pred = model.predict(X_test)
# Calculate accuracy
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
```

```
Accuracy: 0.9875
```

In [14]:
```
model.coef_
```

Out[14]:
```
array([[ 0.05582615, -0.52801455,  0.01404901, -0.06973865,  0.26676492,
         0.92413031],
       [-0.14922941, -0.01783396, -0.00627152,  0.19754163,  0.22874187,
        -0.04283349],
       [-0.09976128,  0.63425766, -0.00177016,  0.02858663,  0.04711972,
         0.00148224],
       [-0.12185586,  0.18534167, -0.00333888, -0.07512431, -0.07717701,
         0.02724927],
       [-0.18361177, -0.29064126, -0.00265945, -0.1412653 , -0.4554487 ,
        -0.91007833]])
```

In [15]:
```
coef = pd.Series(index = X.columns, data = model.coef_[0])
coef.sort_values()
```

Out[15]:
```
cylinders      -0.528015
acceleration   -0.069739
weight         -0.014048
origin          0.266492
displacement    0.055826
model_year      0.256765
dtype: float64
```

The top indicators are cylinders and model year. Let's build a reduced model from them.

In [16]:
```
reduced_cols = coef[coef.abs()> 0.1].index
```

In [17]:
```
X_reduced = df[reduced_cols]
```

In [18]:
```
X_train_red, X_test_red, y_train, y_test = train_test_split(X_reduced, y, test_size=0.2, random_state=42)
```

In [19]:
```
reduced_model = LogisticRegression()
```

In [20]:
```
reduced_model.fit(X_reduced, y)
```

```
C:\Users\12012\anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:814: 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-regression
  n_iter_i = _check_optimize_result(
```

Out[20]:
```
LogisticRegression()
```

In [21]:
```
# Make predictions on the test set
y_pred_red = reduced_model.predict(X_test_red)
# Calculate accuracy
accuracy_red = accuracy_score(y_test, y_pred_red)
print("Accuracy", accuracy_red)
```

```
Accuracy: 0.975
```

The first model is slightly better than the reduced model.

Now, let's use KNN

In [22]:
```
from sklearn.neighbors import KNeighborsRegressor
from sklearn.metrics import mean_squared_error, r2_score
# Create a k-NN regressor
k_neighbors = 5  # Define the number of neighbors (you can adjust this)
knn_regressor = KNeighborsRegressor(n_neighbors=k_neighbors)

# Train the regressor on the training data
knn_regressor.fit(X_train, y_train)

# Make predictions on the test data
y_pred = knn_regressor.predict(X_test)

# Calculate evaluation metrics
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print(f"Mean Squared Error (MSE): {mse:.2f}")
print(f"R-squared (R2): {r2:.2f}")
```

```
Mean Squared Error (MSE): 8.22
R-squared (R2): 0.93
```

There is a good MSE and R^2

Now let's evaluate our model using bootstrapping

In [23]:
```
from sklearn.ensemble import RandomForestRegressor
# Number of bootstrap samples
num_bootstrap_samples = 100

# Initialize lists to store evaluation metrics
mse_scores = []
r2_scores = []

for _ in range(num_bootstrap_samples):
    # Create a bootstrap sample by resampling with replacement
    bootstrap_indices = np.random.choice(len(X), len(X), replace=True)
    X_bootstrap = X.iloc[bootstrap_indices]
    y_bootstrap = y.iloc[bootstrap_indices]

    # Split the bootstrap sample into a training and testing set
    X_train, X_test, y_train, y_test = train_test_split(X_bootstrap, y_bootstrap, test_size=0.3, random_state=42)

    # Use your pre-trained model to make predictions
    # Replace 'model' with your actual model
    y_pred = model.predict(X_test)

    # Calculate evaluation metrics
    mse = mean_squared_error(y_test, y_pred)
    r2 = r2_score(y_test, y_pred)

    # Append the scores to the lists
    mse_scores.append(mse)
    r2_scores.append(r2)

# Calculate the average and standard deviation of evaluation metrics
avg_mse = np.mean(mse_scores)
avg_r2 = np.mean(r2_scores)
std_mse = np.std(mse_scores)
std_r2 = np.std(r2_scores)

print(f"Average Mean Squared Error (MSE): {avg_mse:.2f}")
print(f"Average R-squared (R2): {avg_r2:.2f}")
print(f"Standard Deviation of MSE: {std_mse:.2f}")
print(f"Standard Deviation of R2: {std_r2:.2f}")
```

```
Average Mean Squared Error (MSE): 0.09
Average R-squared (R2): 0.97
Standard Deviation of MSE: 0.05
Standard Deviation of R2: 0.02
```

Everything is pointing to good model.

Now, let's try using bagging to enhance our model

In [24]:
```
from sklearn.ensemble import BaggingRegressor
from sklearn.tree import DecisionTreeRegressor

# Create a base Decision Tree regressor
base_model = DecisionTreeRegressor()

# Create a Bagging Regressor with 5 base models
bagging_model = BaggingRegressor(base_model, n_estimators=5, random_state=42)

# Train the Bagging model
bagging_model.fit(X_train, y_train)

# Make predictions on the testing set
y_pred = bagging_model.predict(X_test)

# Evaluate the Bagging model
mse = mean_squared_error(y_test, y_pred)

# Display the results
print(f"Mean Squared Error: {mse:.2f}")
```

```
Mean Squared Error: 0.02
```

As we can see, the MSE is fantastic

Now, let's implement Random Forest

In [25]:
```
from sklearn.ensemble import RandomForestRegressor

# Create a Random Forest Regressor with 100 trees
rf_model = RandomForestRegressor(n_estimators=100, random_state=42)

# Train the Random Forest model
rf_model.fit(X_train, y_train)

# Make predictions on the testing set
y_pred = rf_model.predict(X_test)

# Evaluate the Random Forest model
mse = mean_squared_error(y_test, y_pred)

# Display the results
print(f"Mean Squared Error: {mse:.2f}")
```

```
Mean Squared Error: 0.02
```

The MSE is the same as the bagging one

Now, let's implement boosting tree

In [27]:
```
from sklearn.ensemble import GradientBoostingRegressor

# Create a Gradient Boosting Regressor with 100 trees
gb_model = GradientBoostingRegressor(n_estimators=100, learning_rate=0.1, random_state=42)

# Train the Gradient Boosting model
gb_model.fit(X_train, y_train)

# Make predictions on the testing set
y_pred = gb_model.predict(X_test)

# Evaluate the Gradient Boosting model
mse = mean_squared_error(y_test, y_pred)

# Display the results
print(f"Mean Squared Error: {mse:.2f}")
```

```
Mean Squared Error: 0.02
```

The same MSE for all methods so far

Now, let's implement LASSO

In [28]:
```
from sklearn.linear_model import Lasso

# Create a Lasso model with alpha (regularization strength)
lasso_model = Lasso(alpha=0.01)

# Train the Lasso model
lasso_model.fit(X_train, y_train)

# Make predictions on the testing set
y_pred = lasso_model.predict(X_test)

# Evaluate the Lasso model
mse = mean_squared_error(y_test, y_pred)

# Display the results
print(f"Mean Squared Error: {mse:.2f}")
```

```
Mean Squared Error: 0.25
```

Lasso is the worst method we have used as it has higher MSE than the other three methods combined

Everything about our model points to a fantastic model that will be able to predict mpg effectively

In [ ]:
```

```