Using Machine Learning to Predict Car Value by Eric Williams
D606 Task 2

## A. Research Question

Given a dataset of used car data, my research question is: What factors influence the value of used cars, and can our Random Forests model accurately predict car prices based on their features? In this analysis, I will use car brand, model, year, fuel type, engine type, accident record, color, and title status to predict price. I will then test the accuracy of the model to measure the efficacy and usefulness of the model from a business standpoint. My hypothesis is that by analyzing the variables in a Random Forest regression, a model can be created to predict the value of used cars. The null hypothesis in this scenario is that there is no significant relationship between the features of a car and its market value.

## B. Data Collection

The data for this analysis was provided by WGU. One potential **advantage** of this is that this dataset has likely been inspected before and will hopefully be free from errors that might prevent the analysis from proper function. One **disadvantage** of this is that I don't know the source of this data and I do not know for certain that this data is representative of real life data. For example, I do not know the reason for missing values, which would give context to how to deal with missing data.

## C. Data Extraction and Preparation

First I looked for missing data

```
#Looking for missing data
print(data.isnull().sum())
```

```
brand             0
model             0
model_year        0
milage            0
fuel_type       170
engine            0
transmission      0
ext_col           0
int_col           0
accident        113
clean_title     596
price             0
dtype: int64
```

In the indicated columns it is possible that the missing data could be indicative of a problem with a car. For example, every title is either listed as clean or blank. It could very well be that every rebuilt title is not listed for a common reason. In short, there might be a common thread between the blank titles types that might cause the car to be worth less. Because of this, I changed the missing data to "unknown". One **advantage** of this is that if there is a common thread for missing title type--such as cars with repaired titles not being listed--then the car value should reflect that. However, one **disadvantage** of that is that I do not know the reason for the missing

data--perhaps all the titles are clean and I have no way of knowing why the data points are missing. Regardless, I thought it would be helpful for the model to understand the missing data as its own category:

```
#Filling unknown data with 'unknown'
df['fuel_type'].fillna("Unknown", inplace=True)
df['accident'].fillna("Unknown", inplace=True)
df['clean_title'].fillna("Unknown", inplace=True)
print(df.isnull().sum())
```

```
brand           0
model           0
model_year      0
milage          0
fuel_type       0
engine          0
transmission    0
ext_col         0
int_col         0
accident        0
clean_title     0
price           0
dtype: int64
```

Next, I found that the mileage column had extra letters and punctuation that had to be removed in order to be processed numerically. This is the code I used to clean the column and leave only the numerical data:

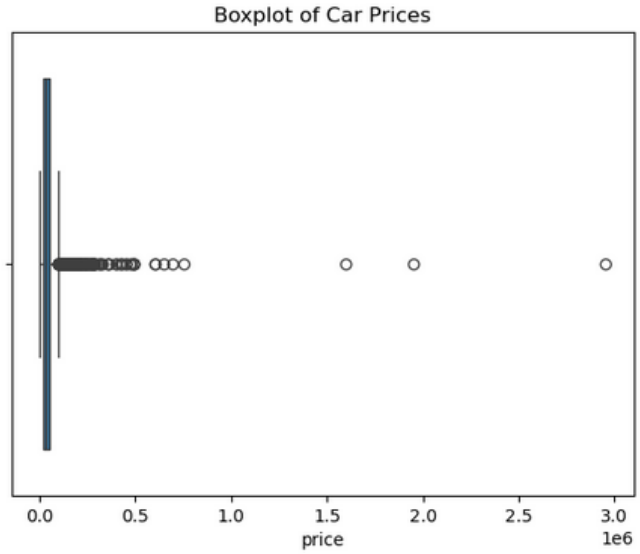
```
#Removing non-numeric characters from milage and keep only the numbers
df['milage'] = df['milage'].str.replace(r'[^\d]', '', regex=True)

# Convert the column to integers for further analysis (optional)
df['milage'] = df['milage'].astype(int)
```

```
print(max((df['milage'])))
```

```
405000
```

There aren't really any advantages or disadvantages here--the data just needed to be slightly altered to be able to be processed.

Next, I created a boxplot of the prices to look for outliers.




Notice that there are a small handful of cars outside of the normal distribution. These data points could possibly skew our data despite not being very helpful to predict. Since most people do not buy used cars for over $100,000, I decided to remove the most expensive cars from my analysis. In all practicality, the most expensive cars would probably need to be looked at closely by a human to predict value. In short, a model that predicts prices for a very small, niche set of expensive cars is not very useful and those data points might skew our model--so I decided to exclude them. These expensive cars made up about 5% of the dataset. To summarize, one **advantage** of this exclusion is that it prevents the rest of the model from being skewed by rare, expensive cars. One **disadvantage**, however, is that the model will not be able to accurately predict the value of expensive cars.

```
# Filter the DataFrame to include only rows where price <= 100,000
df_filtered = df[df['price'] <= 100000]

# Check price for outliers
print(df_filtered['price'].describe())
```

```
count      3775.000000
mean      33694.054570
std       21611.948944
min        2000.000000
25%       16500.000000
50%       29798.000000
75%       45880.000000
max      100000.000000
Name: price, dtype: float64
```

At this point, the data was prepared and ready to be analyzed.

**D. Analysis**

I then encoded the categorical variables and converted them to integers as necessary for the algorithm. An **advantage** of using one-hot encoding is that it can convert many categorical variables to binary inputs all at once. However, a **disadvantage** is that if the variables are ordinal, it might be more reasonable to "rank" certain kinds of car preferences above others. However, because I do not know enough about the specifics of car specifications and how they correlate to value, I decided to use one-hot encoding.

```
#One-hot encoding for categorical variables
encoded_df = pd.get_dummies(df_filtered, columns=['brand', 'fuel_type', 'transmission', 'ext_col', 'int_col', 'accident', 'engine', 'model', 'clean_title'

print(encoded_df)
```

```
#Identifying boolean columns and convert them to integer
boolean_columns = encoded_df.select_dtypes(include='bool').columns
encoded_df[boolean_columns] = encoded_df[boolean_columns].astype(int)

# Check the result
print(encoded_df)
```

```
      model_year  milage  price  brand_Alfa  brand_Aston  brand_Audi  \
0           2013   51000  10300           0            0           0
1           2021   34742  38005           0            0           0
2           2022   22372  54598           0            0           0
3           2015   88900  15500           0            0           0
4           2021    9835  34999           0            0           1
...          ...     ...    ...         ...          ...         ...
4003        2018   53705  25900           0            0           0
4005        2022   10900  53900           0            0           1
4006        2022    2116  90998           0            0           0
4007        2020   33000  62999           0            0           0
4008        2020   43000  40000           0            0           0

      brand_BMW  brand_Bentley  brand_Buick  brand_Cadillac  ...  \
0             0              0            0               0  ...
1             0              0            0               0  ...
2             0              0            0               0  ...
3             0              0            0               0  ...
4             0              0            0               0  ...
...         ...            ...          ...             ...  ...
4003          0              0            0               0  ...
4005          0              0            0               0  ...
4006          0              0            0               0  ...
4007          0              0            0               0  ...
4008          1              0            0               0  ...
```

Next, I ran the random forest regression.

```
#Splitting the data and training it
X = encoded_df.drop('price', axis=1)
y = encoded_df['price']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.4, random_state=2)

#Running the Random Forest Regressor
rf_model = RandomForestRegressor(n_estimators=100, random_state=2)

#Training the model and making predictions
rf_model.fit(X_train, y_train)
y_test_pred = rf_model.predict(X_test)

#Evaluating accuracy
print("Test Mean Absolute Error:", mean_absolute_error(y_test, y_test_pred))
print("Test Mean Squared Error:", mean_squared_error(y_test, y_test_pred))
print("Test R-squared:", r2_score(y_test, y_test_pred))
```

```
Test Mean Absolute Error: 8182.452099337748
Test Mean Squared Error: 137224314.26837134
Test R-squared: 0.6893123410173398
```

One **advantage** of Random Forest regressor is that it is able to make use of the many variables by creating random trees to make predictions. In short, the tool I am using is great for making predictions while analyzing many variables. However, one **disadvantage** of this tool is that if we have a lot of unique makes and models of cars, the model will be less accurate. In short, it is a disadvantage to use this tool if the dataset is not big enough to include many types of the same cars to accurately train the model to make predictions for those specific kinds of cars.

**E. Data Summary and Implications**

As shown above, the mean absolute error of the model was $8,182.45. This means the model can predict car prices given the previously stated variables within about $8,200. Also the R-squared value of 68.9 means that the model can explain 69% of the variance in price. This answers the research question of whether or not we can predict prices given certain variables and defines the accuracy of the model.

**One limitation** of my analysis is that the model is not very good at predicting car values that are over $100,000 because they were rare outliers and were excluded from the model.

**Two directions** or approaches for future study of interest:

1. I propose to expand the selection of the dataset to include more cars to ensure greater training. Because some cars and specifications were very rare, the model was less accurate because it would learn how to predict a certain kind of car, then never see the car again. With a larger dataset, the training would be better and with more repeated car makes and models, the model would predict them better the more times it saw those specifications.

2. Next, I would consider grouping the cars by value before running this analysis. Having a dataset with cars that range in value from $2,000 to $2,954,083 is a problem. If you want to be able to predict the value of very common cars, certain rare/expensive makes and models should be excluded from the analysis. However, if you want to predict the cost of those expensive cars, you would need a dataset and training model designed specifically for those kinds of cars.

In short, this model is great for predicting generally if a car is valuable or not, but if most cars are not valued over $10,000, a model that only predicts car prices within $8,000 or so is not yet accurate enough to predict the value of a random given car. I recommend the above two steps be taken to improve the model.

## Sources

No sources were used besides official WGU materials.