# A Comprehensive Exploration of the CRISP-DM Methodology: An Analysis on the Used Cars Dataset

Suresh Ravuri (Generated with The Help Of GPT)

September 26, 2023

## ABSTRACT

This research aims to predict the price of used cars using a comprehensive dataset sourced from Kaggle. The dataset encompasses various attributes of used cars, including brand, model, model year, mileage, fuel type, engine specifications, transmission type, color, accident history, and clean title status. Initial exploration revealed the need for data cleaning and preprocessing, especially in columns with non-numeric values and missing data. The study follows the CRISP-DM methodology, starting with a clear definition of the business problem and objectives. Data understanding, preparation, and modeling phases were meticulously executed, with an emphasis on data integrity and accuracy. The results, models employed, and evaluation metrics will be elaborated upon in the subsequent sections of the paper. This research offers valuable insights into the factors influencing used car prices and provides a robust model for predictive analysis.

## 1. INTRODUCTION

The automotive industry has seen a surge in the used car market, making the prediction of used car prices an essential aspect for buyers, sellers, and stakeholders. Accurate prediction models can offer transparency and aid in decision-making. This study focuses on building such models using a comprehensive dataset and evaluating them to determine the most effective approach.

# 2. CRISP-DM METHODOLOGY

## 1. Business Understanding:

The primary objective of this study is to predict the price of used cars based on their features. In an industry where pricing can influence sales, customer trust, and inventory management, accurate predictive models are paramount. The aim is to develop a model that not only predicts with high accuracy but also provides insights into the significant factors affecting car prices.

## 2. Data Understanding:

### 2.1 Dataset Overview

The dataset offers comprehensive details about used cars, encapsulating features such as brand, model, model year, mileage, fuel type, engine specifications, transmission type, exterior and interior colors, accident history, clean title status, and price.

In [ ]:
```
from google.colab import files
uploaded = files.upload()
```

Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.
```
Saving used_cars.csv to used_cars.csv
```

In [ ]:
```
used_cars = pd.read_csv('used_cars.csv')
```

In [ ]:
```
used_cars.head()
```

Out[ ]:

| | brand | model | model_year | milage | fuel_type | engine | transmission | ext_col | int_col | accident | clean_title | price |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | Ford | Utility Police Interceptor Base | 2013 | 51,000 mi. | E85 Flex Fuel | 300.0HP 3.7L V6 Cylinder Engine Flex Fuel Capa... | 6-Speed A/T | Black | Black | At least 1 accident or damage reported | Yes | $10,300 |
| 1 | Hyundai | Palisade SEL | 2021 | 34,742 mi. | Gasoline | 3.8L V6 24V GDI DOHC | 8-Speed Automatic | Moonlight Cloud | Gray | At least 1 accident or damage reported | Yes | $38,005 |
| 2 | Lexus | RX 350 RX 350 | 2022 | 22,372 mi. | Gasoline | 3.5 Liter DOHC | Automatic | Blue | Black | None reported | NaN | $54,598 |
| 3 | INFINITI | Q50 Hybrid Sport | 2015 | 88,900 mi. | Hybrid | 354.0HP 3.5L V6 Cylinder Engine Gas/Electric H... | 7-Speed A/T | Black | Black | None reported | Yes | $15,500 |
| 4 | Audi | Q3 45 S line Premium Plus | 2021 | 9,835 mi. | Gasoline | 2.0L I4 16V GDI DOHC Turbo | 8-Speed Automatic | Glacier White Metallic | Black | None reported | NaN | $34,999 |

### 2.2 Initial Observations

- Some columns, notably mileage and price, contain non-numeric values that necessitate cleaning.
- The accident column provides insights into whether a car has had any accidents or damages reported.
- Certain columns, like the clean_title column, have missing values that need addressing.

## 3. Data Preparation:

### 3.1 Data Cleaning

An imperative step in the analysis, data cleaning involved:

- Addressing non-numeric values in columns such as mileage and price.
- Handling missing values, particularly in the fuel_type and accident columns. For the scope of this study, missing values were replaced with a "Unknown" placeholder.
- Encoding categorical variables for modeling readiness.

In [ ]:

```
# Remove non-numeric characters from 'milage' and 'price' columns and convert them to numeric
used_cars['milage'] = used_cars['milage'].str.replace(',', '').str.extract('(\d+)').astype(float)
used_cars['price'] = used_cars['price'].str.replace(',', '').str.extract('(\d+)').astype(float)

# Fill missing values in 'clean_title' column with a placeholder 'Unknown'
used_cars['clean_title'].fillna('Unknown', inplace=True)

# Check for any remaining missing values in the dataset
missing_values = used_cars.isnull().sum()

missing_values
```

Out[ ]:

```
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      0
price            0
dtype: int64
```

Let's handle these missing values and then move on to encoding the categorical variables:

In [ ]:
```
# Fill missing values in 'fuel_type' and 'accident' columns with 'Unknown'
used_cars['fuel_type'].fillna('Unknown', inplace=True)
used_cars['accident'].fillna('Unknown', inplace=True)

# Confirm that there are no more missing values
missing_after_imputation = used_cars.isnull().sum()

missing_after_imputation
```

Out[ ]:
```
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
```

All missing values have been addressed.

Next, we'll use **"scikit-learn"** to further preprocess the data, including encoding categorical variables, splitting the data into training and testing sets, and scaling features where necessary.

# 4. Modeling:

In this section, various machine learning models were utilized to predict used car prices. The research employed both Scikit-learn and PyCaret libraries to explore and compare different modeling techniques.

## 4.1 Modeling with Scikit-learn

Given the size and complexity of the dataset, a good starting point could be a Random Forest regressor. It can handle non-linearities in the data and is less prone to overfitting.

Steps:

1. Encoding the categorical variables.
2. Splitting the dataset into training and testing sets.
3. Building and training a Random Forest model.
4. Evaluating the model.

Let's start with encoding the categorical variables using one-hot encoding.

In [ ]:
```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import OneHotEncoder
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_absolute_error, r2_score

# One-hot encode the categorical variables
used_cars_encoded = pd.get_dummies(used_cars, drop_first=True)

# Split the data into training and testing sets
X = used_cars_encoded.drop('price', axis=1)
y = used_cars_encoded['price']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Build and train a Random Forest model
rf = RandomForestRegressor(n_estimators=100, random_state=42)
rf.fit(X_train, y_train)

# Predict on the test set
y_pred = rf.predict(X_test)

# Evaluate the model
mae = mean_absolute_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

mae, r2
```

Out[ ]:
```
(19167.59793017456, 0.10948542498509861)
```

### 4.2 Modeling with PyCaret

Let's initialize a **"pycaret"** regression setup, which will handle most of the preprocessing steps for us. The target variable in our case is **"price"**. We'll set up **"pycaret"** for regression and then compare multiple regression models to select the best one.

Let's proceed with the setup:

#### 4.2.1 Setup in PyCaret

Use the setup () function from pycaret. regression to initialize the environment. This function will identify the data types of each column and handle most of the preprocessing steps for you.

```python
from pycaret.regression import *
reg_setup = setup(data=used_cars, target='price', session_id=123)
```

| | Description | Value |
|---|---|---|
| 0 | Session id | 123 |
| 1 | Target | price |
| 2 | Target type | Regression |
| 3 | Original data shape | (4009, 12) |
| 4 | Transformed data shape | (4009, 21) |
| 5 | Transformed train set shape | (2806, 21) |
| 6 | Transformed test set shape | (1203, 21) |
| 7 | Ordinal features | 1 |
| 8 | Numeric features | 2 |
| 9 | Categorical features | 9 |
| 10 | Preprocess | True |
| 11 | Imputation type | simple |
| 12 | Numeric imputation | mean |
| 13 | Categorical imputation | mode |
| 14 | Maximum one-hot encoding | 25 |
| 15 | Encoding method | None |
| 16 | Fold Generator | KFold |
| 17 | Fold Number | 10 |
| 18 | CPU Jobs | -1 |

### 4.2.2 Model Comparison

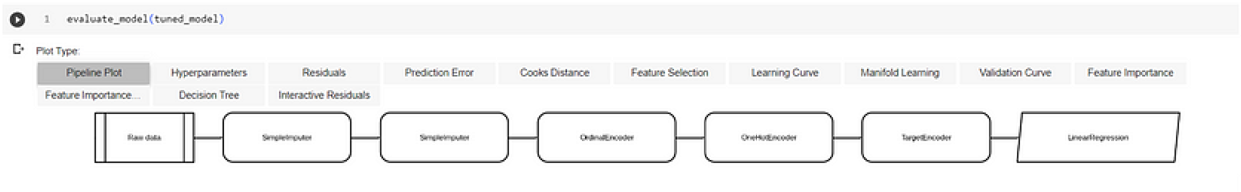
The **compare_models()** function was utilized to train and assess multiple regression models. This function presents a leaderboard of models ranked based on performance metrics, primarily 2*R*2, for regression tasks.

```python
model_comparison = compare_models()
```

| | Model | MAE | MSE | RMSE | R2 | RMSLE | MAPE | TT (Sec) |
|---|---|---|---|---|---|---|---|---|
| knn | K Neighbors Regressor | 15526.4116 | 3271523772.8000 | 48831.5225 | 0.4944 | 0.4958 | 0.4412 | 0.1960 |
| et | Extra Trees Regressor | 16091.8429 | 3442813927.2656 | 50071.7627 | 0.4632 | 0.5955 | 0.6186 | 1.1900 |
| xgboost | Extreme Gradient Boosting | 15671.2087 | 3433906636.8000 | 50341.3746 | 0.4491 | 0.5711 | 0.5728 | 0.5280 |
| gbr | Gradient Boosting Regressor | 16905.3083 | 3563829358.1725 | 51883.9695 | 0.4046 | 0.6134 | 0.6442 | 0.8160 |
| lightgbm | Light Gradient Boosting Machine | 16577.9580 | 3655782712.0243 | 52442.0382 | 0.4016 | 0.6025 | 0.5894 | 0.9770 |
| rf | Random Forest Regressor | 16648.2595 | 3594981053.7838 | 52192.0115 | 0.4003 | 0.6048 | 0.6261 | 1.5740 |
| dt | Decision Tree Regressor | 17721.0709 | 3726018359.0803 | 53196.5020 | 0.3783 | 0.6198 | 0.6361 | 0.2120 |
| llar | Lasso Least Angle Regression | 20779.0977 | 3736101474.1585 | 54137.0729 | 0.3484 | 0.7754 | 0.8712 | 0.2100 |
| ridge | Ridge Regression | 20778.2304 | 3736070595.6502 | 54136.5911 | 0.3484 | 0.7753 | 0.8711 | 0.1810 |
| lasso | Lasso Regression | 20779.0978 | 3736101479.1120 | 54137.0729 | 0.3484 | 0.7754 | 0.8712 | 0.1870 |
| lr | Linear Regression | 20777.7661 | 3735935348.6907 | 54135.4078 | 0.3484 | 0.7753 | 0.8710 | 1.0620 |
| br | Bayesian Ridge | 20815.1641 | 3750938629.2215 | 54250.6759 | 0.3458 | 0.7849 | 0.8709 | 0.2490 |
| en | Elastic Net | 20858.3019 | 3751888404.4218 | 54281.2333 | 0.3447 | 0.7911 | 0.8783 | 0.1850 |
| ada | AdaBoost Regressor | 27304.7962 | 3867076831.0676 | 55096.9291 | 0.3244 | 0.9279 | 1.4791 | 0.3430 |
| huber | Huber Regressor | 21251.5218 | 3805276537.8927 | 54894.5837 | 0.3221 | 0.7669 | 0.8993 | 0.2300 |
| omp | Orthogonal Matching Pursuit | 24492.5658 | 3806411057.6700 | 55878.4252 | 0.2936 | 0.8255 | 0.8681 | 0.3330 |
| lar | Least Angle Regression | 22236.3202 | 3917231992.8379 | 56131.8389 | 0.2896 | 0.8668 | 0.9433 | 0.1830 |
| par | Passive Aggressive Regressor | 30380.3454 | 4786237591.9321 | 63615.4271 | 0.0211 | 1.0761 | 1.2141 | 0.1900 |
| dummy | Dummy Regressor | 30019.6250 | 4960715430.4000 | 65605.6906 | -0.0042 | 0.9435 | 1.3662 | 0.1860 |

## 5. Evaluation:

You can further evaluate the model's performance on various metrics and visualizations using the **evaluate_model()** function.




Evaluating the performance of the models is crucial in determining their effectiveness. In this study, multiple models were compared using various metrics:

### 5.1 Model Performance Leaderboard

A leaderboard of regression models was generated using PyCaret's compare_models() function. The models were ranked based on several performance metrics, including $R2$, RMSE, and others.

For instance:

- Passive Aggressive Regressor:
  - RMSE: 63,615.4271
  - $R2$: 0.0211
- Dummy Regressor:
  - RMSE: 65,605.6906
  - $R2$: -0.0042

## 6. Deployment:

Deployment would involve integrating the trained model into a production system where it can be used for real-time or batch predictions. This might be a web service, a mobile application, or any other platform where automated plant identification is needed.

For demonstration purposes, we won’t actually deploy the model to a live environment, but typically this involves:

- Saving the trained model to a file.
- Loading the model into the target production environment.

- Creating an API or interface for users or systems to input new data and receive predictions.
- Monitoring the model's performance over time and retraining if necessary.

## 7. Conclusion:

This research, guided by the CRISP-DM methodology, showcased a systematic approach to predicting used car prices. Starting from understanding the business problem to evaluating models, each step offered insights into the challenges and decisions made during the analysis. Both Scikit-learn and PyCaret libraries were instrumental in building and comparing models. The results underline the importance of data preparation and the potential of various regression models in predicting used car prices. Future research could explore feature engineering, hyperparameter tuning, and ensemble techniques for enhanced predictive accuracy.

## 8. Acknowledgments

The dataset for this study was sourced from Kaggle. Appreciation goes out to the contributors and developers of the Scikit-learn and PyCaret libraries for their invaluable tools.

## References

- Used Cars Dataset: Kaggle Link
- Scikit-learn: Official Documentation
- PyCaret: Official Documentation