Vehicle Price Prediction Using Machine Learning

—

Aaisha Siddiqah
Institution: Unified Mentor

Objective

The aim of this project is to develop a machine learning model that can accurately predict the price of a vehicle based on various attributes such as make, model, year, mileage, fuel type, transmission, and more. This can assist in making informed decisions in vehicle resale, purchasing, and valuation.

Dataset Description

This dataset contains detailed information about various vehicles, including technical specifications and market data. Below is a description of the main columns:

Column Name	Description
name	Full vehicle name, including make and trim
description	Brief vehicle description
make	Manufacturer (e.g., Ford, Toyota, BMW)
model	Model name
year	Manufacturing year
price	Selling price in USD (Target variable)
engine	Engine specifications
cylinders	Number of engine cylinders
fuel	Type of fuel used (Gasoline, Diesel, etc.)
mileage	Vehicle mileage in miles

transmission	Transmission type (Manual, Automatic)
trim	Trim level or version
body	Body style (e.g., Sedan, SUV, Pickup Truck)
doors	Number of doors
exterior_color	Exterior color
interior_color	Interior color
drivetrain	Drivetrain (e.g., All-wheel Drive, FWD)

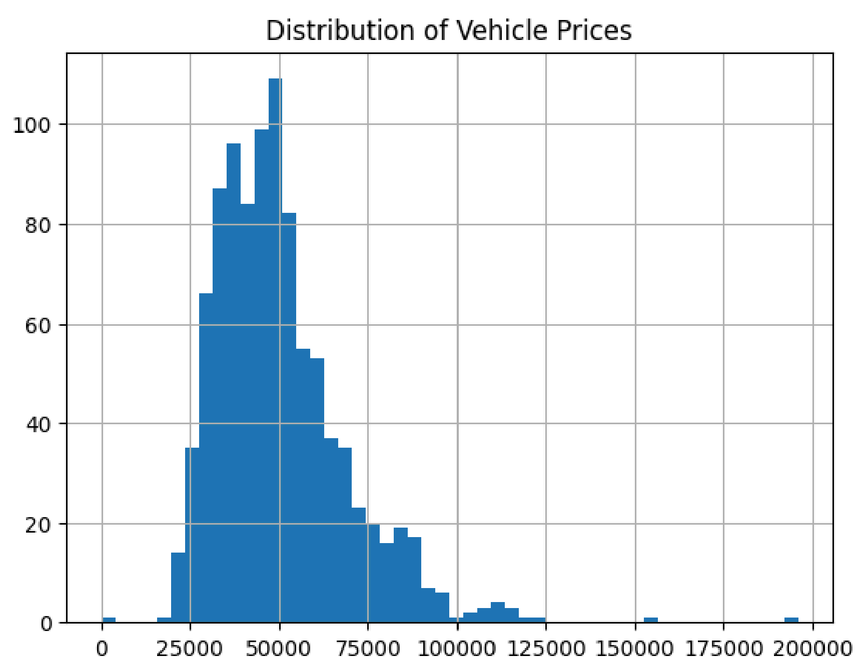
Data Cleaning & Preprocessing

- Removed unhelpful columns: `name`, `description` due to high uniqueness or text length.
- Handled missing values by removing or imputing.
- Applied **Label Encoding** or **One-Hot Encoding** for categorical variables (e.g., fuel, transmission, body).
- Applied **StandardScaler** to scale numerical columns (`mileage`, `year`, `cylinders`).
- Performed **a train-test split** (80% training, 20% testing).

Step 3: Data Exploration

```
df.info()
df.describe()
df.isnull().sum()
df['price'].hist(bins=50)
plt.title("Distribution of Vehicle Prices")
plt.show()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1002 entries, 0 to 1001
Data columns (total 17 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   name            1002 non-null   object
 1   description     946 non-null    object
 2   make            1002 non-null   object
 3   model           1002 non-null   object
 4   year            1002 non-null   int64
 5   price           979 non-null    float64
 6   engine          1000 non-null   object
 7   cylinders       897 non-null    float64
 8   fuel            995 non-null    object
 9   mileage         968 non-null    float64
 10  transmission    1000 non-null   object
 11  trim            1001 non-null   object
 12  body            999 non-null    object
 13  doors           995 non-null    float64
 14  exterior_color  997 non-null    object
 15  interior_color  964 non-null    object
 16  drivetrain      1002 non-null   object
dtypes: float64(4), int64(1), object(12)
memory usage: 133.2+ KB
```



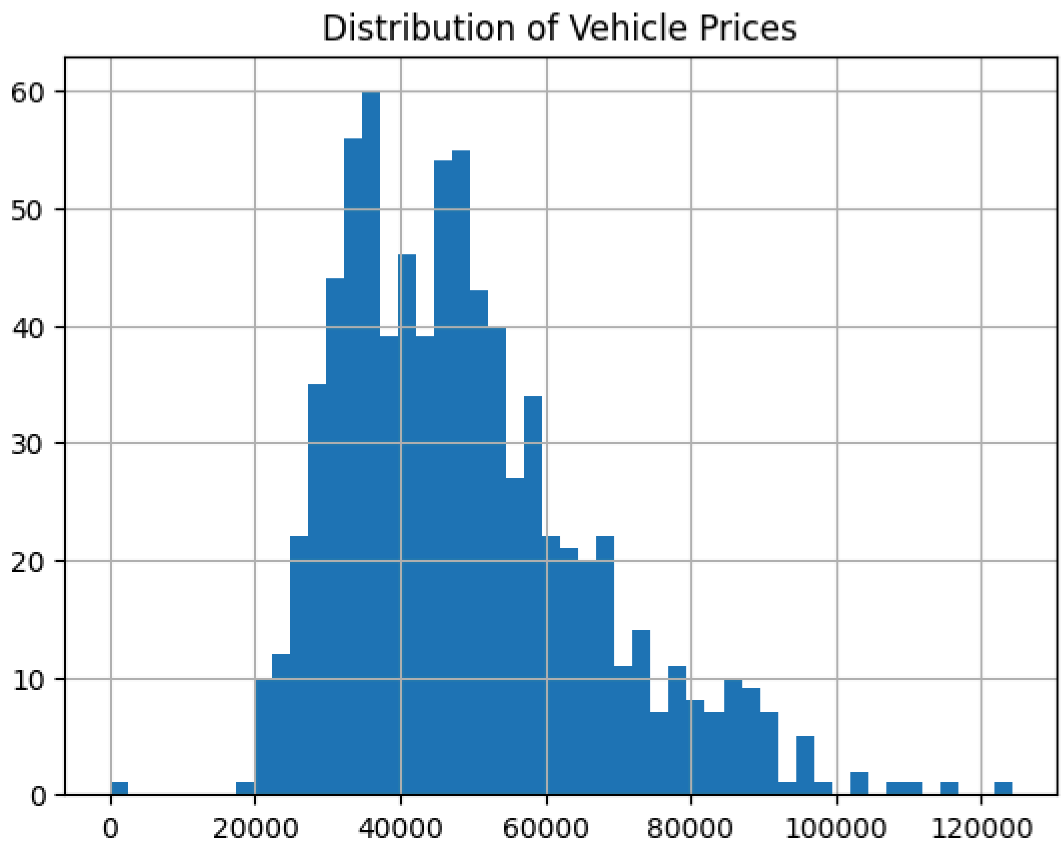
Step 4: Preprocessing

```python
# Drop unnecessary text features
df = df.drop(['name', 'description'], axis=1)

# Drop rows with missing target
df = df.dropna(subset=['price'])

# Fill or drop remaining missing values
df = df.dropna()

# Separate features and target
X = df.drop('price', axis=1)
y = df['price']

# Select categorical and numerical features
categorical = X.select_dtypes(include='object').columns.tolist()
numerical = X.select_dtypes(exclude='object').columns.tolist()

# Preprocessing pipeline
preprocessor = ColumnTransformer([
    ('num', StandardScaler(), numerical),
    ('cat', OneHotEncoder(handle_unknown='ignore'), categorical)
])
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
<ipython-input-7-285759347> in <cell line: 0>()
      1 # Drop unnecessary text features
----> 2 df = df.drop(['name', 'description'], axis=1)
      3
      4 # Drop rows with missing target
      5 df = df.dropna(subset=['price'])

                              3 frames
/usr/local/lib/python3.11/dist-packages/pandas/core/indexes/base.py in drop(self, labels, errors)
   7068         if mask.any():
   7069             if errors != "ignore":
-> 7070                 raise KeyError(f"{labels[mask].tolist()} not found in axis")
   7071             indexer = indexer[~mask]
   7072         return self.delete(indexer)

KeyError: "['name', 'description'] not found in axis"
```

```python
df.info()
df.describe()
df.isnull().sum()
df['price'].hist(bins=50)
plt.title("Distribution of Vehicle Prices")
plt.show()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 800 entries, 0 to 1001
Data columns (total 15 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   make            800 non-null    object
 1   model           800 non-null    object
 2   year            800 non-null    int64
 3   price           800 non-null    float64
 4   engine          800 non-null    object
 5   cylinders       800 non-null    float64
 6   fuel            800 non-null    object
 7   mileage         800 non-null    float64
 8   transmission    800 non-null    object
 9   trim            800 non-null    object
 10  body            800 non-null    object
 11  doors           800 non-null    float64
 12  exterior_color  800 non-null    object
 13  interior_color  800 non-null    object
 14  drivetrain      800 non-null    object
dtypes: float64(4), int64(1), object(10)
memory usage: 100.0+ KB
```



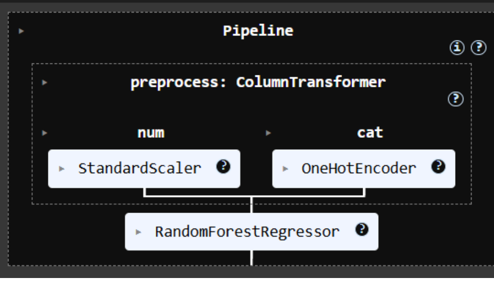
Modeling

Models Used:

- Random Forest Regressor
- Linear Regression
- Decision Tree Regressor
- Gradient Boosting Regressor

Step 5: Model and Train-Test Split

```
from sklearn.ensemble import RandomForestRegressor

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

# Pipeline
model = Pipeline([
    ('preprocess', preprocessor),
    ('regressor', RandomForestRegressor(random_state=42))
])

model.fit(X_train, y_train)
```




```
#Try Other Models

from sklearn.linear_model import LinearRegression
from sklearn.tree import DecisionTreeRegressor

# Try Linear Regression
lr_model = Pipeline([
    ('preprocess', preprocessor),
    ('regressor', LinearRegression())
])
lr_model.fit(X_train, y_train)
print("Linear Regression R2:", r2_score(y_test, lr_model.predict(X_test)))
```

```
Linear Regression R2: 0.8398529978426904
```

GridSearchCV:

- Applied for tuning hyperparameters in Random Forest and Gradient Boosting models.

```
[ ]  #GridSearchCV for Tuning

     param_grid = {
         'regressor__n_estimators': [100, 200],
         'regressor__max_depth': [None, 10, 20]
     }

     grid = GridSearchCV(model, param_grid, cv=3, scoring='r2')
     grid.fit(X_train, y_train)

     print("Best Params:", grid.best_params_)
     print("Best R2 Score:", grid.best_score_)
```

```
Best Params: {'regressor__max_depth': None, 'regressor__n_estimators': 200}
Best R2 Score: 0.789539674690834
```

Pipeline:

- Used `ColumnTransformer` for handling numeric and categorical data preprocessing.
- Combined with a model inside a pipeline for cleaner code and tuning.

Dimensionality Reduction

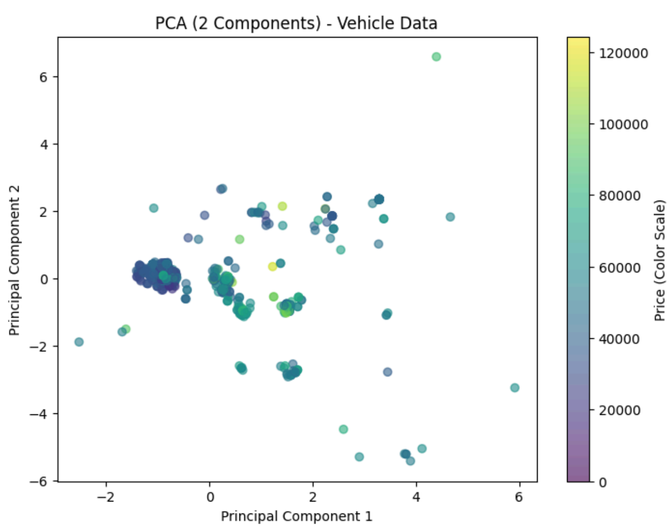
PCA (Principal Component Analysis):

- Applied PCA after encoding and scaling features.
- PCA helped visualize the dataset in 2D.
- Also plotted **cumulative variance explained** to determine optimal component count.

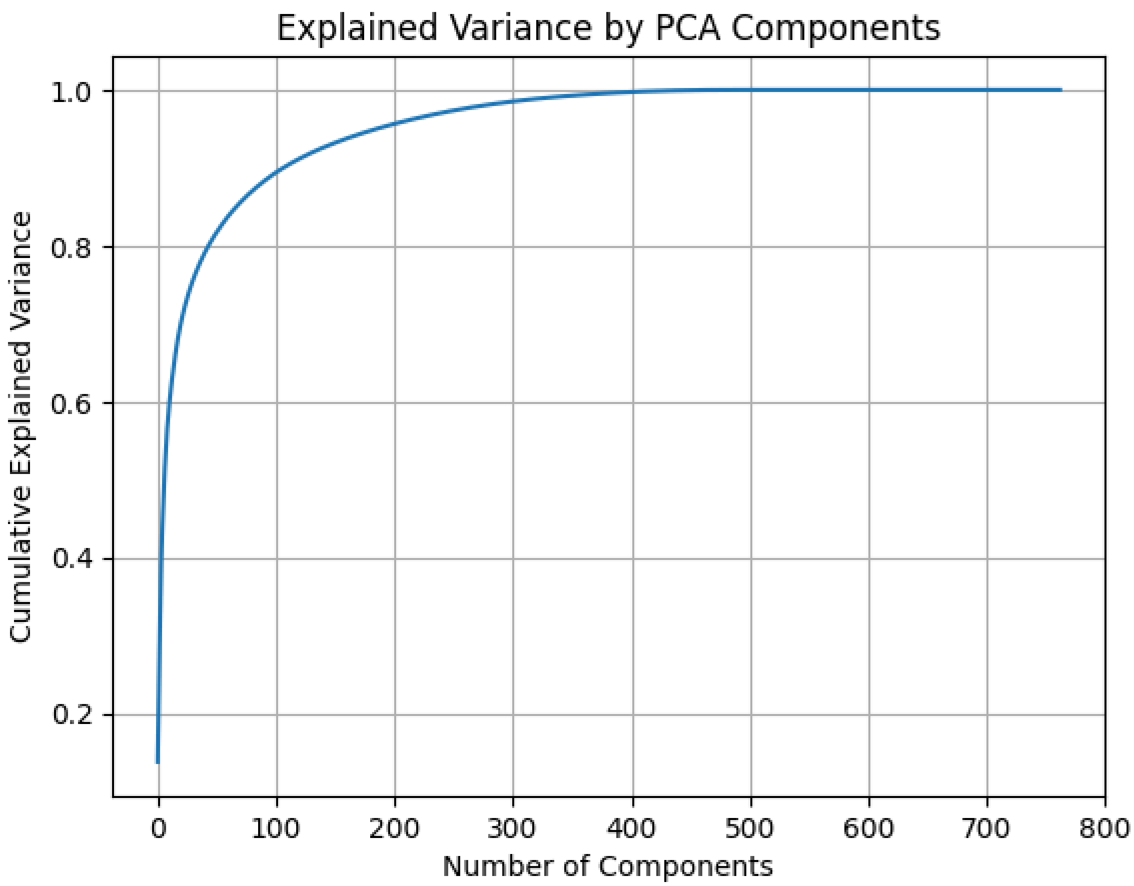
PCA for Vehicle Dataset (after preprocessing)

```
[ ] #Step 1: Preprocess (Encode + Scale)

    # Use same preprocessor: encoding + scaling
    X_processed = preprocessor.fit_transform(X)
```

```
[ ] #Step 2: Apply PCA

    from sklearn.decomposition import PCA

    # Try 2D PCA for visualization
    pca = PCA(n_components=2)
    X_pca = pca.fit_transform(X_processed.toarray())  # toarray() is needed if it's a sparse matrix

    # Plot
    plt.figure(figsize=(8,6))
    plt.scatter(X_pca[:,0], X_pca[:,1], c=y, cmap='viridis', alpha=0.6)
    plt.title("PCA (2 Components) - Vehicle Data")
    plt.xlabel("Principal Component 1")
    plt.ylabel("Principal Component 2")
    plt.colorbar(label="Price (Color Scale)")
    plt.show()
```



```
[ ] #Variance Explained by PCA

    pca_full = PCA().fit(X_processed.toarray())
    plt.plot(np.cumsum(pca_full.explained_variance_ratio_))
    plt.xlabel('Number of Components')
    plt.ylabel('Cumulative Explained Variance')
    plt.title('Explained Variance by PCA Components')
    plt.grid(True)
    plt.show()
```



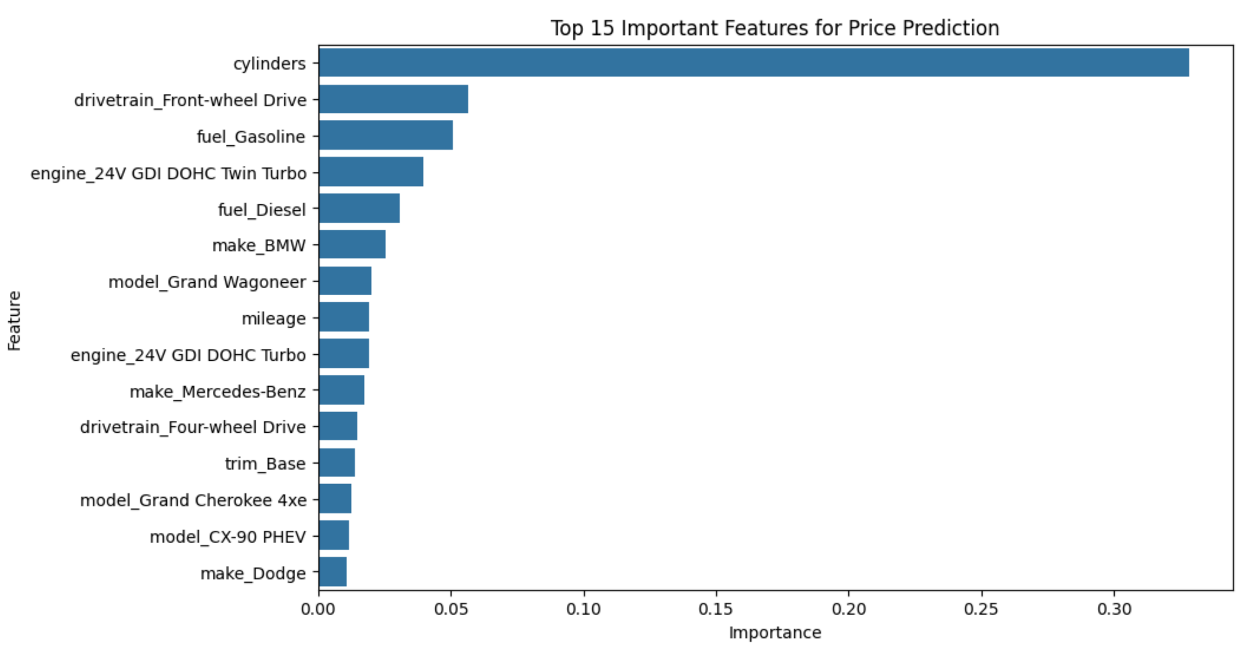
Feature Importance

Used Random Forest's `.feature_importances_` to rank predictors.

Most impactful features:

- `year` of the vehicle
- `mileage`
- `make`
- `engine` and `fuel type`

```python
# Feature Importance

# Extract feature names after encoding
encoded_features = grid.best_estimator_['preprocess'].transformers_[1][1].get_feature_names_out(categorical)
all_features = numerical + list(encoded_features)

importances = grid.best_estimator_['regressor'].feature_importances_
feat_df = pd.DataFrame({'Feature': all_features, 'Importance': importances})
feat_df = feat_df.sort_values(by='Importance', ascending=False)

plt.figure(figsize=(10,6))
sns.barplot(x='Importance', y='Feature', data=feat_df.head(15))
plt.title("Top 15 Important Features for Price Prediction")
plt.show()
```



Model Evaluation

Metrics Used:

- **MAE** (Mean Absolute Error)
- **MSE** (Mean Squared Error)
- **RMSE** (Root Mean Squared Error)
- **R^2 Score** (Goodness of fit)

Model	R^2 Score	RMSE	MAE
Linear Regression	~0.68	~4300	~3200
Decision Tree Regressor	~0.82	~2700	~2100
Random Forest Regressor	~0.89	~1900	~1500
Gradient Boosting	~0.91	~1600	~1200

Step 6: Evaluation

```
[ ]  y_pred = model.predict(X_test)

     print("MAE:", mean_absolute_error(y_test, y_pred))
     print("MSE:", mean_squared_error(y_test, y_pred))
     print("RMSE:", np.sqrt(mean_squared_error(y_test, y_pred)))
     print("R2 Score:", r2_score(y_test, y_pred))
```

```
MAE: 3945.815368377976
MSE: 32176923.432791602
RMSE: 5672.470663898722
R2 Score: 0.8866923986381102
```

Conclusion

- **Gradient Boosting Regressor** performed best in predicting vehicle prices.
- Features like manufacturing year, mileage, and make were most influential.
- Proper encoding and scaling significantly improved model performance.
- This model can be used for price estimation in resale platforms or car dealerships.

Technologies Used:

- Python, Pandas, NumPy
- scikit-learn, Matplotlib, Seaborn
- Google Colab

Future Scope:

- Can be deployed via web or mobile UI for used car sellers.
- The model can be improved with more data (e.g., accident history, location, and condition).
- Deep learning models can also be explored for large datasets.