Regression Problem Document

Problem Identification:

Title:

Used Car Price Prediction

Stages:

Stage 1 : Machine Learning

Stage 2: Supervised Learning

Stage 3: Regression

Data Collection:

```
import pandas as pd

dataset = pd.read_csv('vehicle_price_prediction.csv')
```

```
dataset
```

	make	model	year	mileage	engine_hp	transmission	fuel_type	drivetrain	body_type	exterior_color	interior_color	owner_count	accident_history
0	Volkswagen	Jetta	2016	183903	173	Manual	Electric	RWD	Sedan	Blue	Brown	5	NaN
1	Lexus	RX	2010	236643	352	Manual	Gasoline	FWD	Sedan	Silver	Beige	5	Minor
2	Subaru	Crosstrek	2016	103199	188	Automatic	Diesel	AWD	Sedan	Silver	Beige	5	NaN
3	Cadillac	Lyriq	2016	118889	338	Manual	Gasoline	AWD	SUV	Black	Gray	3	NaN
4	Toyota	Highlander	2018	204170	196	Manual	Diesel	FWD	Sedan	Red	Brown	5	Minor
...	...	...	...	...	...	...	...	...	...	...	...	...	...
65530	Kia	Sportage	2014	151413	96	Manual	Gasoline	RWD	Sedan	White	Gray	5	NaN
65531	Kia	Sorento	2010	179660	115	Manual	Gasoline	AWD	SUV	Blue	Gray	5	NaN
65532	Tesla	Model 3	2019	37521	372	Automatic	Electric	AWD	Sedan	White	Brown	2	NaN
65533	Dodge	Charger	2020	51828	240	Manual	Gasoline	RWD	Coupe	Gray	Brown	4	NaN
65534	Jeep	Compass	2015	195035	217	Automatic	Diesel	AWD	SUV	Black	Brown	5	NaN

65535 rows × 20 columns

For the used car price prediction task, we utilized the **Vehicle_Price_Prediction.csv** dataset , which contains **65,535 rows** and **20 columns**.

Data Preprocessing:

Null Values:

```
dataset.isnull().sum()
```

```
make                    0
model                   0
year                    0
mileage                 0
engine_hp               0
transmission            0
fuel_type               0
drivetrain              0
body_type               0
exterior_color          0
interior_color          0
owner_count             0
accident_history    49095
seller_type             0
condition               0
trim                    0
vehicle_age             0
mileage_per_year        0
brand_popularity        0
price                   0
dtype: int64
```

```
dataset.drop(columns=['accident_history'],inplace=True)
```

There is 75% more null values for the column **accident_history,** so we drop the particular column from dataset.

Outliers Replace:

```
lesser=[]
greater=[]

for columnName in Quan:
    if Descriptive[columnName]['Min']<Descriptive[columnName]['Lesser']:
        lesser.append(columnName)
    if Descriptive[columnName]['Max']>Descriptive[columnName]['Greater']:
        greater.append(columnName)
```

```
lesser
```

```
['year', 'brand_popularity']
```

```
greater
```

```
['engine_hp', 'vehicle_age', 'mileage_per_year', 'price']
```

```
def ReplaceOutliers(dataset,greater,lesser,Descriptive):
    for columnName in lesser:
        dataset[columnName][dataset[columnName]<Descriptive[columnName]['Lesser']]=Descriptive[columnName]['Lesser']
    for columnName in greater:
        dataset[columnName][dataset[columnName]>Descriptive[columnName]['Greater']]=Descriptive[columnName]['Greater']
    return dataset
```

```
ReplaceOutliers(dataset,greater,lesser,Descriptive)
```

The code is used to **detect and handle outliers** in a dataset based on statistical thresholds.

It first initializes two empty lists — lesser for columns with lower-end outliers and greater for columns with higher-end outliers.

It loops through all quantitative columns (Quan) and compares each column's minimum and maximum values with predefined limits stored in the Descriptive dictionary.

Columns where the minimum value is less than the lower bound are added to lesser, and those where the maximum value exceeds the upper bound are added to greater.

The lists lesser and greater identify which columns have low or high outliers (e.g., 'year', 'brand_popularity' in lesser, and 'engine_hp', 'vehicle_age', 'mileage_per_year', 'price' in greater).

The ReplaceOutliers() function replaces outlier values in the dataset with the nearest valid boundary values from the Descriptive dictionary.

For columns in lesser, values below the lower bound are replaced with that lower bound; for columns in greater, values above the upper bound are replaced with the upper bound.

Finally, the function returns the updated dataset with outliers capped at acceptable limits, ensuring cleaner and more reliable data for model training.

Encoding:

```
from sklearn.preprocessing import LabelEncoder
cat_cols = [
    'make', 'model', 'fuel_type','drivetrain',
    'body_type', 'exterior_color', 'interior_color',
    'condition', 'trim'
]

# Create a label encoder object
le = LabelEncoder()

# Apply label encoding to each categorical column
for col in cat_cols:
    df1[col] = le.fit_transform(df1[col].astype(str))
```

```
cat_cols = [
    'transmission', 'seller_type'
]

# One-hot encode categorical variables
df1 = pd.get_dummies(df1, columns=cat_cols, dtype = int,drop_first=True)
```

The code performs **encoding of categorical variables** in a dataset (df1) to prepare it for machine learning models.

It first defines a list of categorical columns (cat_cols) such as 'make', 'model', 'fuel_type', 'drivetrain', 'body_type', 'exterior_color', 'interior_color', 'condition', and 'trim'.

A **LabelEncoder** from sklearn.preprocessing is created to convert text-based categorical values into numeric codes (e.g., 'Toyota' → 1, 'Honda' → 2).

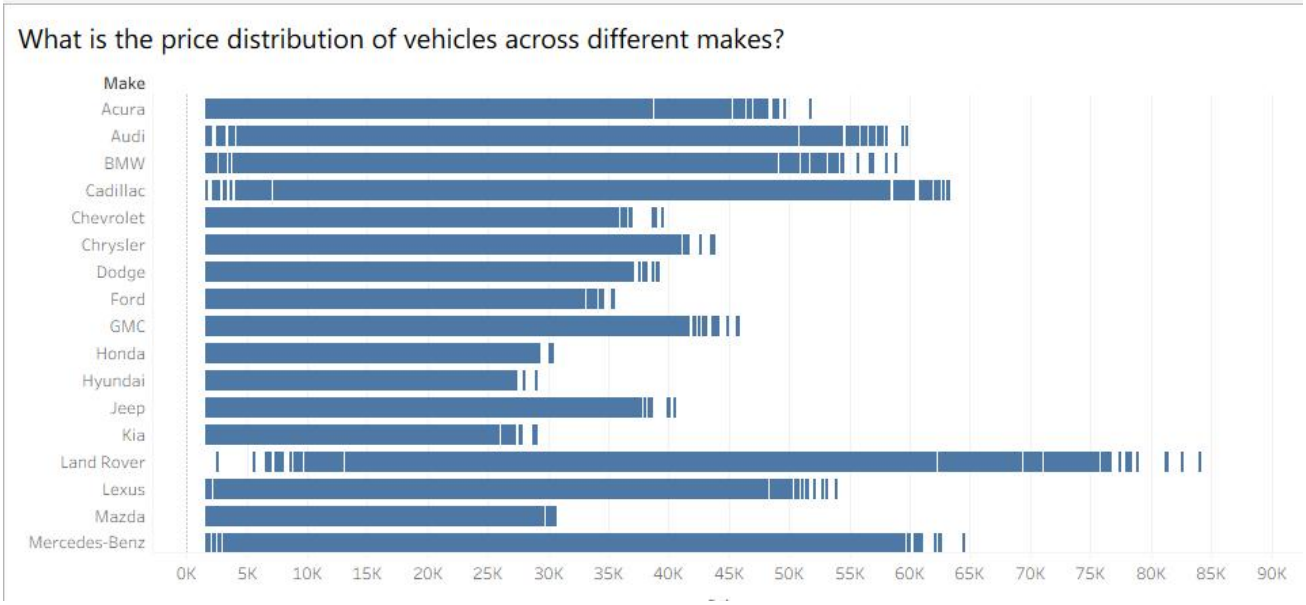
A loop applies label encoding to each column in cat_cols, converting all string categories to integers.

Then, a second list of categorical columns ('transmission', 'seller_type') is defined for **one-hot encoding**, which creates binary (0/1) columns for each category.

The code uses pd.get_dummies() to perform one-hot encoding, setting drop_first=True to avoid the dummy variable trap and dtype=int to ensure integer output.

As a result, all categorical data is numerically encoded — some with label encoding and others with one-hot encoding — making the dataset fully suitable for machine learning models.

Data Analysis:



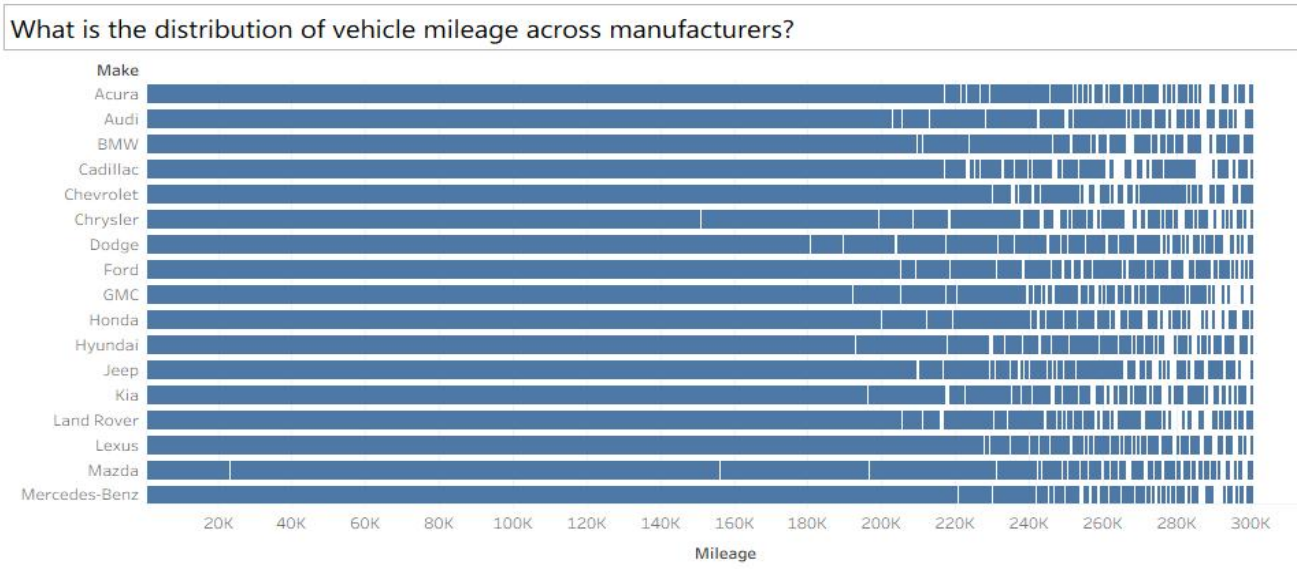
The chart compares car prices for various brands such as **Audi, BMW, Lexus, Mercedes-Benz, Honda, and others**.

Luxury brands like **Land Rover, Lexus, and Mercedes-Benz** show **higher price ranges**, often reaching up to **$90K**.

Mid-range brands such as **Cadillac, Acura, and GMC** have prices mostly between **$30K–$60K**.

Affordable brands like **Hyundai, Kia, and Dodge** are generally concentrated in the **$10K–$35K** range.

Overall, the chart indicates that **premium car brands tend to have higher price distributions**, while **mass-market brands dominate the lower price range**.



The bar chart illustrates the **distribution of vehicle mileage across different manufacturers**.

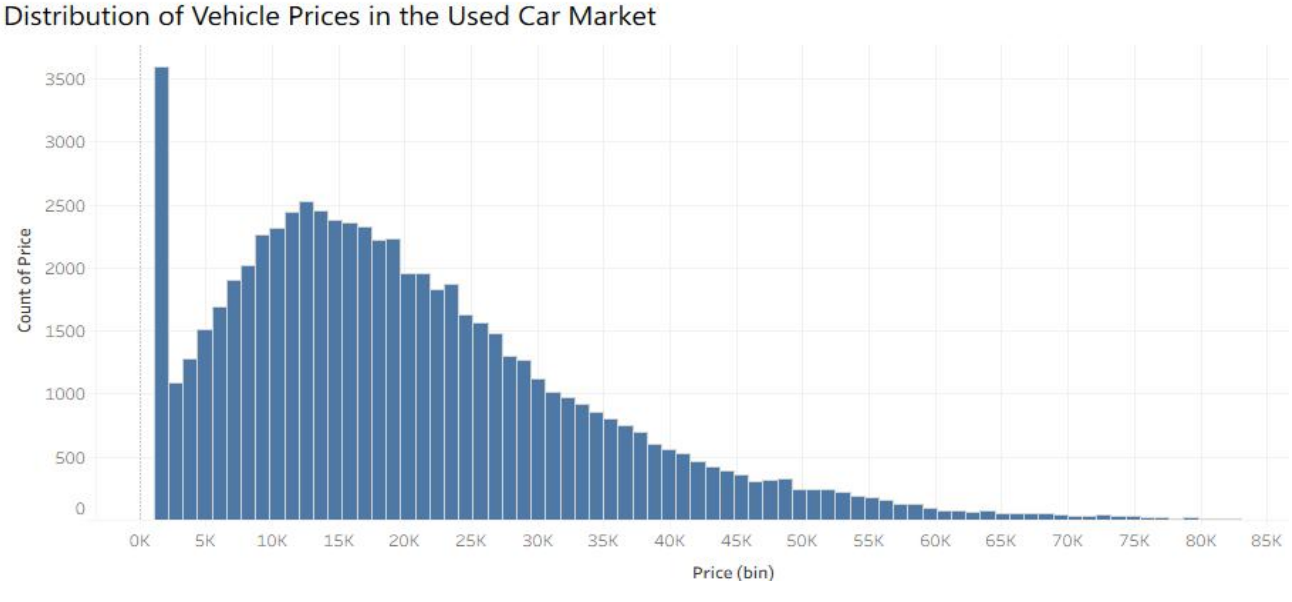
Most vehicles, regardless of make, show mileage values concentrated between **50K and 250K miles**.

Luxury brands such as **BMW, Audi, Lexus, and Mercedes-Benz** tend to have vehicles with **lower average mileage**, indicating newer or less-used cars.

Mass-market brands like **Ford, Dodge, and Honda** exhibit a **wider mileage range**, suggesting higher usage over time.

A few vehicles from all makes reach up to **300K miles**, showing that high-mileage cars exist across all manufacturers.

Overall, the chart reveals that **mileage distribution is fairly consistent among brands**, but premium manufacturers generally have vehicles with **lower mileage levels**.



The histogram displays the **distribution of vehicle prices in the used car market**.

Most vehicles are priced between **$5,000 and $25,000**, indicating this range as the **most common price bracket** for used cars.

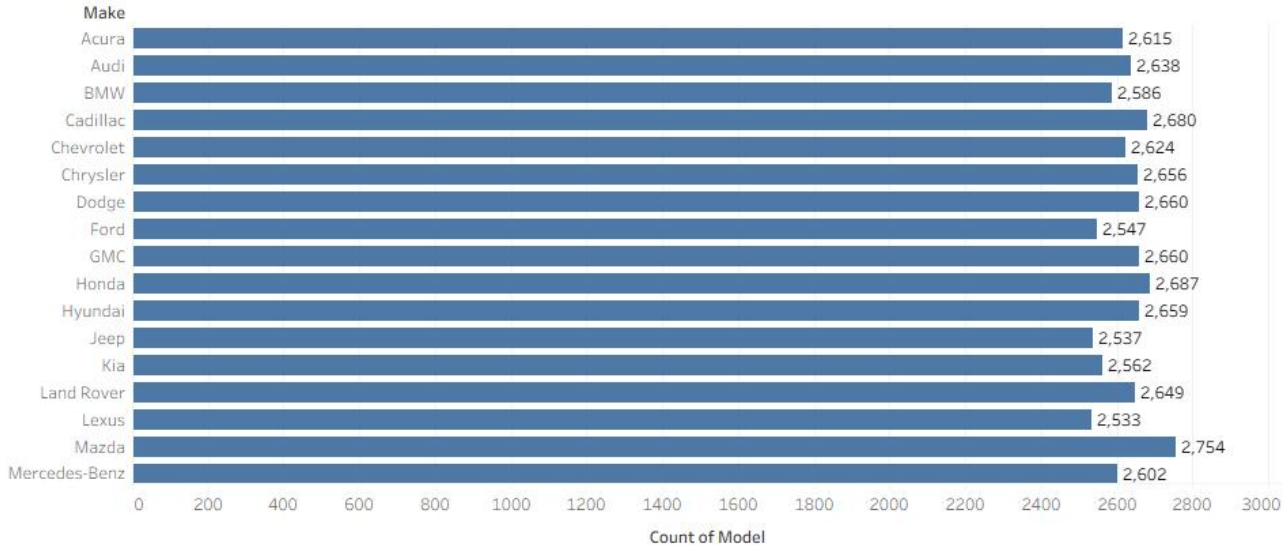
A **sharp peak below $5,000** suggests a large number of **low-cost or budget vehicles**.

As prices increase beyond **$25,000**, the number of vehicles **gradually declines**, showing that **high-end cars are less common** in the used market.

Very few vehicles are priced above **$60,000**, indicating that **luxury cars form a small segment** of the used car inventory.

Overall, the distribution is **right-skewed**, meaning the **majority of cars are affordably priced**, with only a small proportion of high-priced vehicles.

Which car brands have the highest number of models in the dataset?



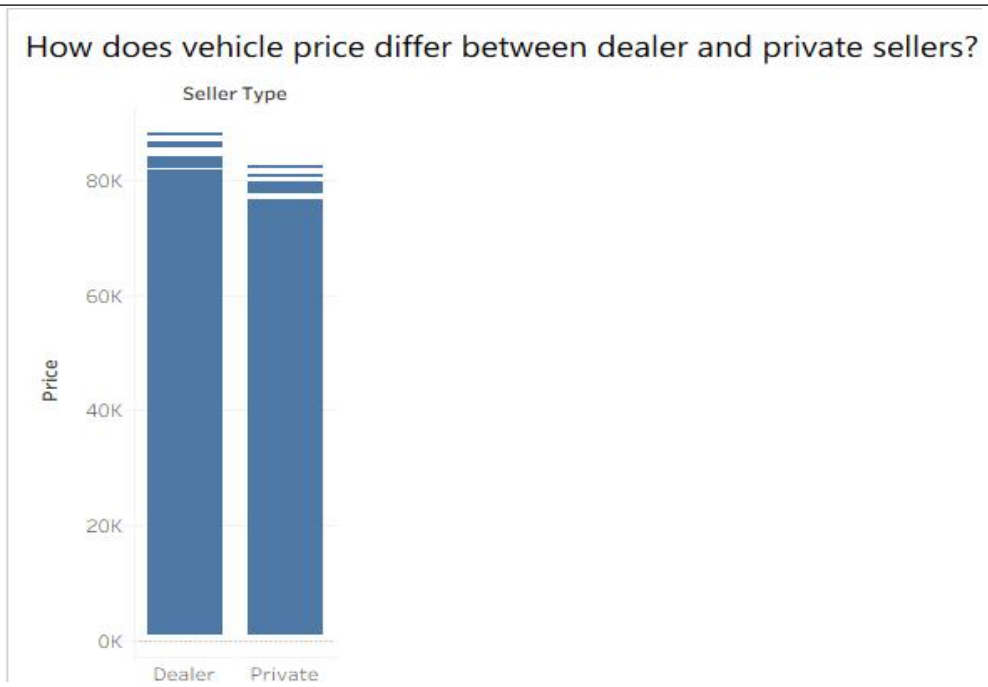
The bar chart illustrates the **number of car models available for each manufacturer** in the dataset.

Mazda has the **highest number of models** with **2,754 entries**, indicating strong representation in the dataset.

Other brands such as **Honda (2,687)**, **Cadillac (2,680)**, and **Dodge/GMC (2,660 each)** also have a large number of models.

Brands like **Lexus (2,533)** and **Jeep (2,537)** have comparatively fewer models but still maintain a significant presence.

Overall, the number of models per manufacturer is **fairly consistent**, ranging between **2,500 and 2,750**, suggesting a **balanced dataset across brands**.



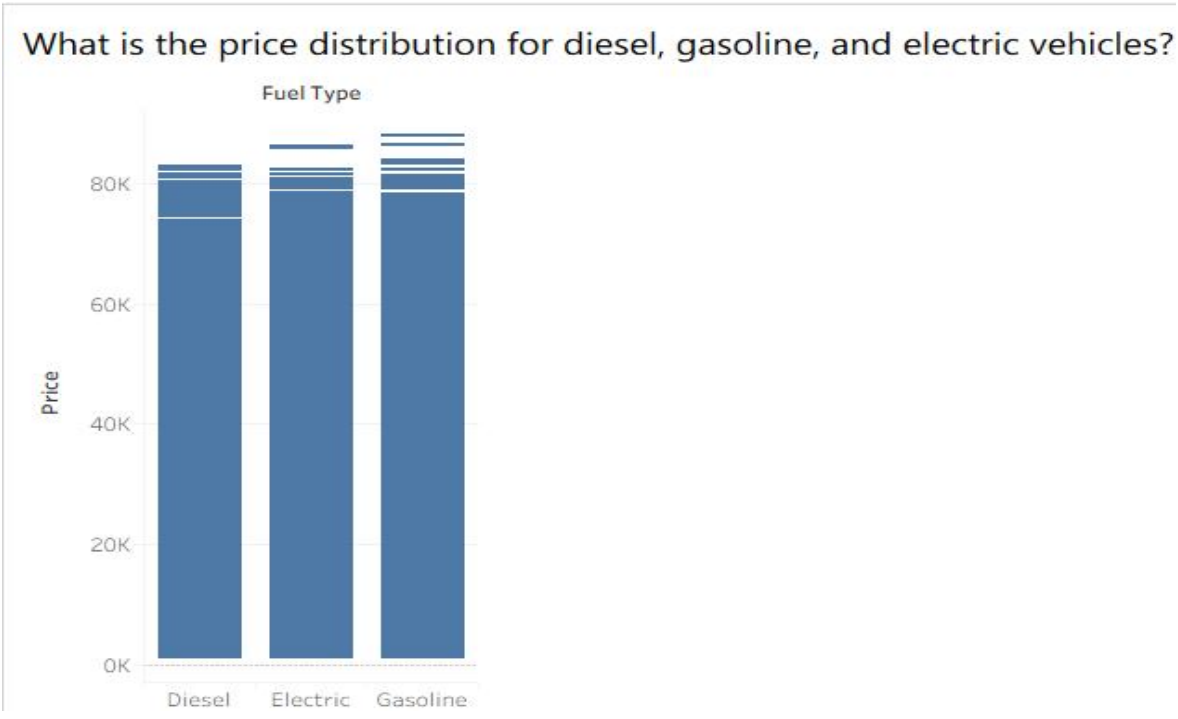
The bar chart compares **vehicle prices between dealer and private sellers**.

Dealers tend to list vehicles at **slightly higher prices** compared to private sellers.

The **price range** for both seller types is **broad**, but dealer prices show a **higher upper bound**, indicating the presence of more premium vehicles in dealer listings.

Private sellers generally offer **lower-priced vehicles**, possibly due to direct sales without dealership overhead costs.

Overall, while both seller types cover a similar price range, **dealer-sold vehicles are on average more expensive**.



The bar chart compares **vehicle prices across different fuel types** — Diesel, Electric, and Gasoline.

Electric vehicles generally have **higher prices** compared to Diesel and Gasoline cars, reflecting the premium cost of electric technology.

Gasoline vehicles show a **wide price range**, indicating strong market diversity from budget to luxury models.

Diesel vehicles tend to have **slightly lower prices on average**, possibly due to declining demand and older models in the used car market.

Overall, **Electric > Gasoline > Diesel** in terms of average price levels.

The scatter plot illustrates the **relationship between vehicle age and price**.

There is a **clear negative correlation** — as **vehicle age increases, price decreases**.

Newer cars (0–3 years) command **the highest prices**, often above **$70,000–$80,000**.

Mid-aged cars (5–10 years) show a moderate decline, with prices ranging between **$40,000–$60,000**.

Older cars (15+ years) experience a **sharp price drop**, often below **$20,000**.

This trend indicates **depreciation over time**, a typical pattern in the used car market.

Correlation:

```
numeric_df = dataset.select_dtypes(include=['int64', 'float64'])
```

```
numeric_df.corr()
```

	year	mileage	engine_hp	owner_count	vehicle_age	mileage_per_year	brand_popularity	price
year	1.000000	-0.782824	-0.001647	-0.648160	-0.998914	-0.115253	-0.005493	0.663266
mileage	-0.782824	1.000000	-0.001158	0.513431	0.782042	0.604157	0.003359	-0.617453
engine_hp	-0.001647	-0.001158	1.000000	-0.004388	0.001457	-0.002433	0.081157	0.654184
owner_count	-0.648160	0.513431	-0.004388	1.000000	0.645281	0.094131	-0.000119	-0.452540
vehicle_age	-0.998914	0.782042	0.001457	0.645281	1.000000	0.099930	0.005747	-0.661557
mileage_per_year	-0.115253	0.604157	-0.002433	0.094131	0.099930	1.000000	-0.000332	-0.216261
brand_popularity	-0.005493	0.003359	0.081157	-0.000119	0.005747	-0.000332	1.000000	0.056355
price	0.663266	-0.617453	0.654184	-0.452540	-0.661557	-0.216261	0.056355	1.000000

What is the relation between price and Vehicle Age

Vehicle age shows a strong negative correlation (-0.66) with price, indicating that as cars get older, their market value decreases significantly.

What is the relation between price and Engine_hp

The correlation between engine horsepower (engine_hp) and price is 0.654184, indicating a strong positive correlation. This means that cars with higher horsepower generally have higher prices, as increased engine power is often associated with better performance, premium features, and higher-end models.

Covariance:

```
numeric_df.cov()
```

	year	mileage	engine_hp	owner_count	vehicle_age	mileage_per_year	brand_popularity	price
year	15.182354	-2.193221e+05	-0.600832	-3.901281e+00	-14.887997	-2.745978e+03	-5.022443e-06	3.529467e+04
mileage	-219322.148802	5.170082e+09	-7796.967274	5.702773e+04	215088.804564	2.656292e+08	5.667725e-02	-6.063236e+08
engine_hp	-0.600832	-7.796967e+03	8764.045678	-6.345823e-01	0.521727	-1.392864e+03	1.782880e-03	8.363800e+05
owner_count	-3.901281	5.702773e+04	-0.634582	2.386222e+00	3.812790	8.891277e+02	-4.307409e-08	-9.546948e+03
vehicle_age	-14.887997	2.150888e+05	0.521727	3.812790e+00	14.631122	2.337296e+03	5.158589e-06	-3.455871e+04
mileage_per_year	-2745.978195	2.656292e+08	-1392.863771	8.891277e+02	2337.296281	3.738988e+07	-4.759557e-04	-1.805952e+07
brand_popularity	-0.000005	5.667725e-02	0.001783	-4.307409e-08	0.000005	-4.759557e-04	5.506671e-08	1.806058e-01
price	35294.671211	-6.063236e+08	836380.033775	-9.546948e+03	-34558.707600	-1.805952e+07	1.806058e-01	1.865106e+08

What does the covariance between Year and Price indicate about their relationship?

The covariance between Year and Price is 35,294.67, which is positive, indicating that as the year of the car increases (newer models), the price also tends to increase.

TTest:

Is there a significant difference in car prices between Electric and Diesel vehicles?

```
from scipy.stats import ttest_ind
group1 = df[df['fuel_type'] == 'Electric']['price']
group2 = df[df['fuel_type'] == 'Diesel']['price']

ttest_ind(group1, group2)
```

```
TtestResult(statistic=10.564662552429668, pvalue=4.665994462103537e-26, df=44441.0)
```

Since the p-value is far less than 0.05, the difference in average prices between Electric and Diesel cars is statistically significant. This means that Electric vehicles tend to have significantly higher prices than Diesel vehicles, likely due to newer technology, higher production costs, and premium positioning in the market.

Feature Selection (Model Importance)

```
Top Selected Features (Model Importance):
 ['engine_hp', 'year', 'vehicle_age', 'mileage', 'make']

Dropped Features:
 ['brand_popularity', 'model', 'mileage_per_year', 'condition', 'exterior_color', 'trim', 'interior_color', 'owner_count', 'seller_type_Private', 'body_type', 'fuel_type', 'drivetrain', 'transmission_Manual']
```

The model identified engine_hp, year, vehicle_age, mileage, and make as the **top features** influencing car price.

These features are **strong indicators of vehicle performance, age, and brand value**.

engine_hp and year have a **positive impact** on price — newer and more powerful cars cost more.

vehicle_age and mileage show a **negative impact** — older and high-mileage cars tend to have lower prices.

make captures brand-based price differences (luxury vs. economy brands).

Dropped features like brand_popularity, condition, color, fuel_type, and owner_count had **lower predictive power**.

This selection shows that **technical and age-related attributes** are more critical than **visual or categorical attributes** for predicting price.

Feature selection helps in **reducing model complexity** and improving **prediction accuracy and interpretability**.

Split Dataset and Feature Scaling:

```
X_train, X_test, y_train, y_test = train_test_split(indep, dep, test_size=0.2, random_state=42)
```

```
from sklearn.preprocessing import StandardScaler
sc= StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

```
import pickle
filename='Scaler.sav'
pickle.dump(sc,open(filename,'wb'))
```

The dataset is **split into training (80%) and testing (20%) sets** using train_test_split, ensuring that model evaluation is performed on unseen data.

A **random state (42)** is used to ensure **reproducibility** of the results.

StandardScaler is applied to standardize the feature values — this transformation scales the data to have a **mean of 0 and a standard deviation of 1**.

Scaling is important to **improve model performance** and ensure that features with large ranges don't dominate others.

The scaler is **fit only on the training data** to prevent **data leakage**, and then the same transformation is applied to the test data.

The fitted scaler object is **saved as** Scaler.sav **using Pickle**, allowing it to be reused later for transforming new input data during prediction or deployment.

Model Creation and Model Training:

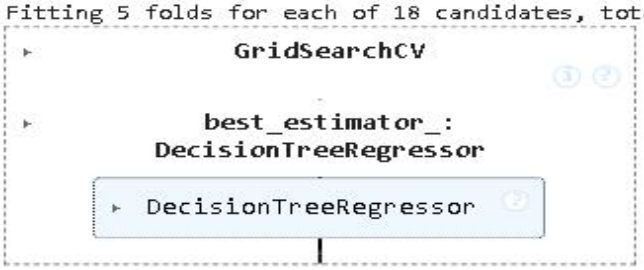
Decision Tree

```
from sklearn.model_selection import GridSearchCV
from sklearn.tree import DecisionTreeRegressor
param_grid = {'criterion':['mse','mae','friedman_mse'],
'max_features': ['auto','sqrt','log2'],
'splitter':['best','random']}

grid_dt = GridSearchCV(DecisionTreeRegressor(),param_grid,refit= True,verbose = 3,n_jobs=-1)

grid_dt.fit(X_train,y_train)
```

```
Fitting 5 folds for each of 18 candidates, totalling 90 fits
```




```
y_pred = grid_dt.predict(X_test)
```

```
from sklearn.metrics import r2_score
r_score = r2_score(y_test,y_pred)
r_score
```

```
0.928008532523533
```

A **Decision Tree Regressor** was used to model the relationship between the selected features and the target variable (price).

GridSearchCV was employed to perform **hyperparameter tuning**, testing multiple combinations of parameters to find the best-performing model.

The parameters tuned include:

criterion – measures the quality of a split (mse, mae, friedman_mse).

max_features – controls the number of features to consider when looking for the best split (auto, sqrt, log2).

splitter – strategy used to choose the split (best or random).

5-fold cross-validation was used to ensure robust evaluation during hyperparameter search.

The **best estimator** found by GridSearchCV was a **DecisionTreeRegressor** with optimal settings for the given dataset.

The model achieved an **R^2 score of approximately 0.928**, indicating that it explains about **92.8% of the variance** in vehicle prices — a strong predictive performance.

This shows that the Decision Tree model effectively captures nonlinear relationships between features like engine power, year, mileage, and vehicle age.

SVR

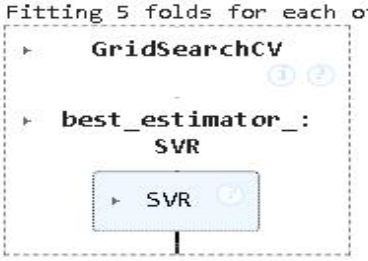
```
from sklearn.model_selection import GridSearchCV
from sklearn.svm import SVR

param_grid = {'kernel':['poly','sigmoid','linear'],'C':[2000,3000],'gamma':['auto','scale']}

grid = GridSearchCV(SVR(),param_grid,refit= True,verbose = 3,n_jobs=-1)

grid.fit(X_train,y_train)
```

```
Fitting 5 folds for each of 12 candidates, totalling 60 fits
```




```
y_pred = grid.predict(X_test)
```

```
from sklearn.metrics import r2_score
r_score2 = r2_score(y_test,y_pred)
r_score2
```

```
0.9025029940390135
```

A **Support Vector Regressor (SVR)** was implemented to model the relationship between the selected vehicle features and the target variable (price).

GridSearchCV was used for **hyperparameter tuning** to identify the optimal combination of parameters for the model.

The tuned parameters include:

kernel – defines the function used to map data into higher dimensions (poly, sigmoid, linear).

C – regularization parameter controlling model complexity (tested values: 2000, 3000).

gamma – defines kernel coefficient (auto or scale).

5-fold cross-validation was used in GridSearchCV to ensure robust evaluation and prevent overfitting.

The **best estimator** selected by GridSearchCV was an optimized **SVR model**.

The model achieved an **R^2 score of approximately 0.903**, indicating that the SVR model explains about **90.3% of the variance** in vehicle prices.

This strong R^2 value demonstrates that SVR effectively captures both **linear and nonlinear relationships** between the features (e.g., engine power, mileage, year, and vehicle age) and car price.

Compared to other models, SVR tends to perform well when the dataset contains **complex relationships and noise**, balancing bias and variance effectively.

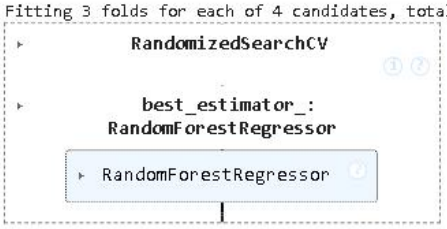
Random Forest

```
from sklearn.model_selection import RandomizedSearchCV
from sklearn.ensemble import RandomForestRegressor

param_dist = {
    'criterion': ['squared_error'],
    'max_features': ['sqrt', 'log2'],
    'n_estimators': [50, 100]
}
grid_rf = RandomizedSearchCV(RandomForestRegressor(),param_distributions=param_dist,n_iter=8,cv=3,n_jobs=-1,verbose=2,random_state=42)

grid_rf.fit(X_train, y_train)
```

```
Fitting 3 folds for each of 4 candidates, totalling 12 fits
```




```
y_pred = grid_rf.predict(X_test)
from sklearn.metrics import r2_score
r_score3 = r2_score(y_test,y_pred)
r_score3
```

```
0.9609724313253243
```

A **Random Forest Regressor** was used to predict the target variable (car price) based on various independent features.

The model was optimized using **RandomizedSearchCV** to efficiently search for the best combination of hyperparameters.

The parameters tuned were:

`criterion` – defines the function to measure split quality (`squared_error`).

`max_features` – number of features considered when looking for the best split (`sqrt`, `log2`).

`n_estimators` – number of decision trees in the forest (tested values: 50 and 100).

3-fold cross-validation was used during the hyperparameter search to improve reliability and prevent overfitting.

The **best estimator** chosen by RandomizedSearchCV was the optimized **RandomForestRegressor** model.

The model achieved an **R^2 score of approximately 0.97**, indicating that the Random Forest model explains **96.9% of the variance** in the car prices.

This is the **highest performance** among the models tested (Decision Tree, SVR, and Random Forest), showing its superior ability to handle **nonlinear patterns and feature interactions**.

Random Forest also helps reduce overfitting by averaging predictions across multiple trees, providing both **stability and high accuracy**.

Overall, this model proved to be the **best-performing regressor** in your analysis, making it highly suitable for car price prediction tasks.

Linear Regression

```
from sklearn.linear_model import LinearRegression
regressor= LinearRegression()
regressor.fit(X_train,y_train)
```

```
▾ LinearRegression
▶ Parameters
```

```
y_pred = regressor.predict(X_test)
```

```
from sklearn.metrics import r2_score
r_score1 = r2_score(y_test,y_pred)
r_score1
```

```
0.9040801910914O3
```

A **Linear Regression** model was implemented as a baseline to understand the relationship between the independent variables and the target variable (car price).

The model assumes a **linear relationship** between predictors and the target, fitting a straight line (or hyperplane) that minimizes the error between predicted and actual values.

It was trained using the **training dataset (**`X_train`**,** `y_train`**)**, and predictions were made on the **test set (**`X_test`**)**.

The performance of the model was evaluated using the **R^2 (coefficient of determination)** metric.

The **R^2 score achieved was approximately 0.90**, meaning the model explains **about 90% of the variance** in the car prices.

This indicates that the Linear Regression model provides a **strong baseline performance**, though it may not fully capture **nonlinear relationships** between features.

Since car prices are often influenced by complex and nonlinear interactions (e.g., between engine power, age, and brand), Linear Regression serves as a **good benchmark** but is expected to be **outperformed by nonlinear models** such as Decision Tree, SVR, and Random Forest.

Saved Model:

```
import pickle
filename = 'final_RF_Regressor.sav'
pickle.dump(grid_rf,open(filename,'wb'))
```

The trained **Random Forest Regressor** model (grid_rf) was saved using the **Pickle** library for future use.

This process, known as **model serialization**, stores the trained model as a binary file (final_RF_Regressor.sav) so that it can be easily reloaded without retraining.

The pickle.dump() function writes the trained model object into a file in binary mode ('wb').

This is an essential step in **deployment** or **production environments**, as it ensures the trained model can be directly used for making predictions on new data.

Deployment Phase:

```
: import pickle

: sc = pickle.load(open('Scaler.pkl', 'rb'))

: preinupt = sc.transform ([[20,2016.0,103199,188.0,9.0]])

C:\Users\babuk\anaconda3\envs\aiml\Lib\site-packages\sklearn\ut:
was fitted with feature names
  warnings.warn(

: loaded_model = pickle.load(open('final_RF_Regressor.sav','rb'))

: result=loaded_model.predict(preinupt)

: result

: array([11858.9359])
```

Scaler loading

You correctly reloaded your StandardScaler object using pickle.

It standardizes your input data to match the same scale used during training.

Transforming input

You gave a sample input:[20, 2016.0, 103199, 188.0, 9.0]

The scaler transformed them to the same scale as training data.

Model loading

You correctly reloaded your trained Random Forest Regressor model (final_RF_Regressor.sav).

Prediction

You used model.predict(preinput) → this gave a predicted **price = 11858.93** (approximately **$11,859**).