Car Price Prediction

SATHVIK YADAV

Objective

1.Analyze and Visualize Used Car Prices:

•Perform descriptive statistics and visualize data trends.

•Identify patterns and relationships between features.

2.Predict Car Prices Using Regression Models:

•**Linear Regression:** Predict car prices and evaluate with R-squared.

•**Ridge Regression:** Handle multicollinearity and prevent overfitting.

•**Lasso Regression:** Feature selection by penalizing coefficients.

•**ElasticNet Regression:** Combine ridge and lasso for balanced regularization.

Dataset Description

Source: Kaggle (Vehicle dataset from Cardekho)

Link: [Vehicle Dataset from Cardekho]

- **Dataset Size:**
- 301 records
- 9 features

- **Features:**

- **Car_Name:** The name of the car.
- **Year:** The year in which the car was bought.
- **Selling_Price:** The price at which the owner wants to sell the car.
- **Present_Price:** The current ex-showroom price of the car.
- **Kms_Driven:** The distance completed by the car in kilometers.
- **Fuel_Type:** The type of fuel the car uses (e.g., Diesel, Petrol, CNG).
- **Seller_Type:** Defines whether the seller is a dealer or an individual.
- **Transmission:** Defines whether the car is manual or automatic.
- **Owner:** The number of previous owners the car has had.

Data Loading

Importing Libraries and the dataset.

Inspect the dataset for missing values and clean the data as necessary

Importing Libraries

In [1]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

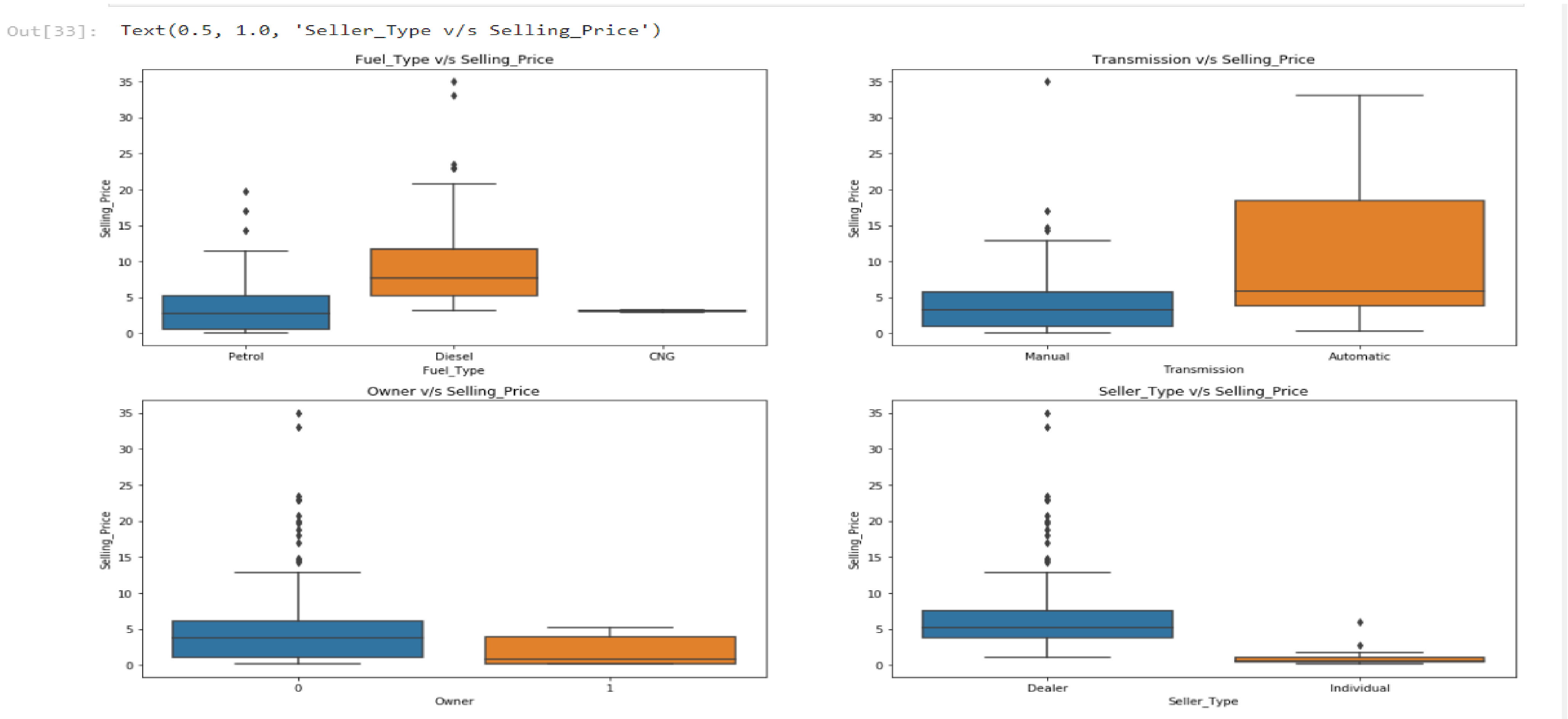
Importing the dataset

In [2]:

```
cars = pd.read_csv("car data.csv")
```

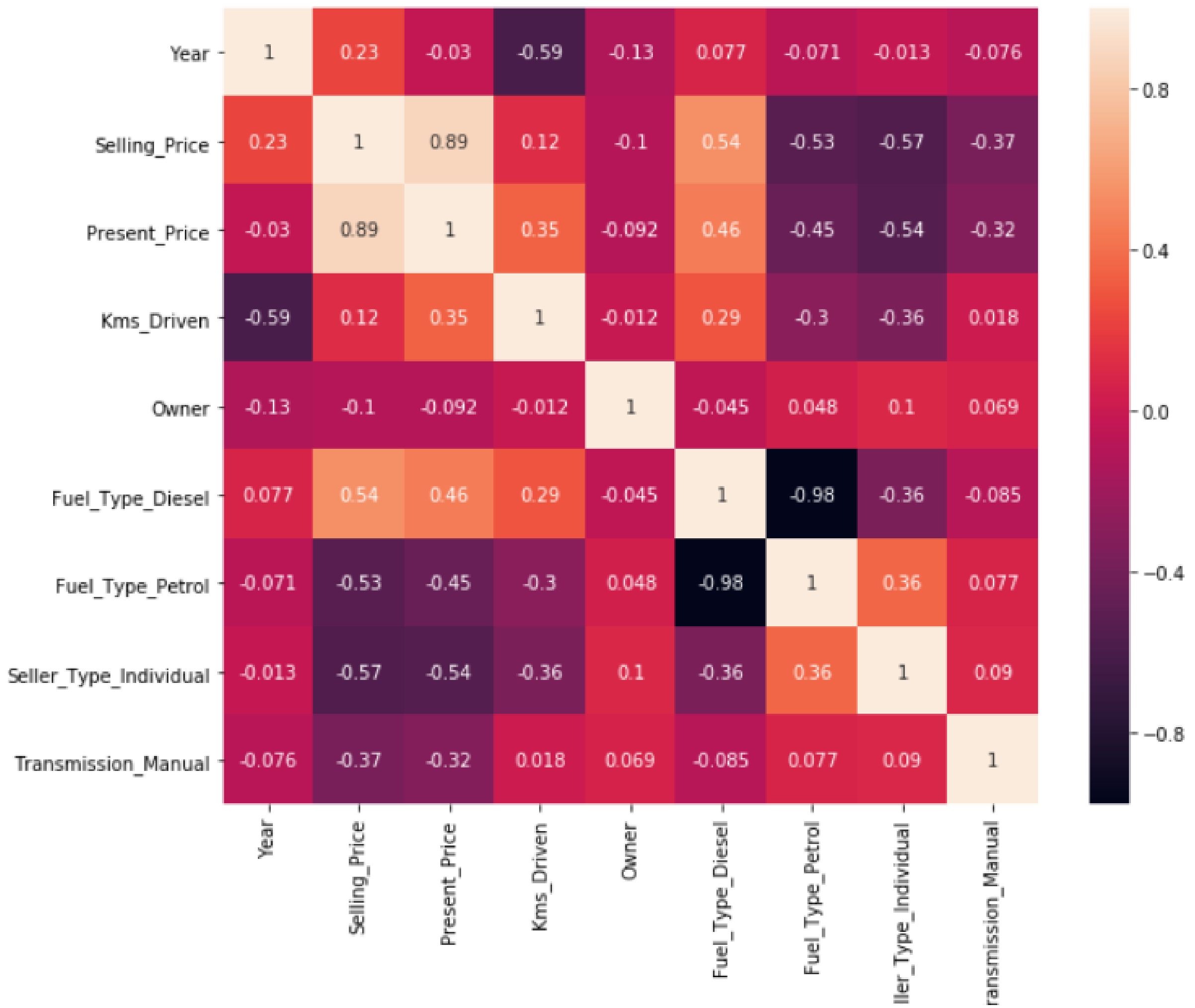
Data Exploration (EDA)

- **Goals:** Understand the distribution of data.
- Identify relationships between different features.



Heatmap to show the correlation between various variables of the dataset

```
plt.figure(figsize=(10, 8))
cor = cars.corr()
ax = sns.heatmap(cor,annot=True)
bottom, top = ax.get_ylim()
ax.set_ylim(bottom + 0.5, top - 0.5)
plt.show()
```



Feature Engineering

Converting categorical variables to dummy variables

In [36]:

```
#Fuel_Type

cars.Fuel_Type.value_counts()
```

Out[36]:

```
Petrol    234
Diesel     57
CNG         2
Name: Fuel_Type, dtype: int64
```

In [37]:

```
cars.Seller_Type.value_counts()
```

Out[37]:

```
Dealer        192
Individual    101
Name: Seller_Type, dtype: int64
```

In [38]:

```
cars.Transmission.value_counts()
```

Out[38]:

```
Manual       258
Automatic     35
Name: Transmission, dtype: int64
```

In [39]:

```
cars = pd.get_dummies(cars,columns=['Fuel_Type','Seller_Type','Transmission'],drop_first=True)
```

Model Building

Linear Regression Model

The simplest form of regression is the linear regression, which assumes that the predictors have a linear relationship with the target variable.

The linear regression equation can be expressed in the following form:

y = a1x1 + a2x2 + a3x3 + ... + anxn + b

- y is the target variable.
- x1, x2, x3,...xn are the features.
- a1, a2, a3,...., an are the coefficients.
- b is the parameter of the model.

In [44]:

```
y = cars['Selling_Price']
X = cars.drop(['Selling_Price'],axis=1)
```

In [49]:

```
#Splitting the data into train and test

from sklearn.model_selection import train_test_split

X_train , X_test , y_train , y_test = train_test_split(X,y,test_size = 0.30 , random_state = 1)

print(X_train.shape)
print(X_test.shape)
print(y_test.shape)
```

```
(205, 8)
(88, 8)
(88,)
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:          Selling_Price   R-squared:                       0.895
Model:                            OLS   Adj. R-squared:                  0.892
Method:                 Least Squares   F-statistic:                     301.8
Date:                Sat, 01 Aug 2020   Prob (F-statistic):          5.17e-134
Time:                        12:15:37   Log-Likelihood:                -559.84
No. Observations:                 293   AIC:                             1138.
Df Residuals:                     284   BIC:                             1171.
Df Model:                           8
Covariance Type:            nonrobust
==========================================================================================
                             coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------------------
const                   -542.8984     99.248     -5.470      0.000    -738.254    -347.543
Year                       0.2713      0.049      5.513      0.000       0.174       0.368
Present_Price              0.4556      0.016     29.261      0.000       0.425       0.486
Kms_Driven             -3.593e-05   6.73e-06     -5.340      0.000   -4.92e-05   -2.27e-05
Owner                      0.3906      0.544      0.718      0.473      -0.680       1.461
Fuel_Type_Diesel           2.5594      1.204      2.126      0.034       0.189       4.929
Fuel_Type_Petrol           0.4161      1.185      0.351      0.726      -1.916       2.749
Seller_Type_Individual    -1.4694      0.257     -5.719      0.000      -1.975      -0.964
Transmission_Manual       -1.4271      0.320     -4.455      0.000      -2.058      -0.797
==============================================================================
Omnibus:                       74.778   Durbin-Watson:                   1.907
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              609.812
Skew:                           0.767   Prob(JB):                    3.81e-133
Kurtosis:                       9.899   Cond. No.                     4.00e+07
==============================================================================
```

Model Implementation

Other Models:

1. **Ridge Regression:** Apply ridge regression to address multicollinearity.
2. **Lasso Regression:** Use lasso regression for feature selection.
3. **ElasticNet Regression:** Combine ridge and lasso penalties with elasticnet.

Ridge Regression

In [117…

```
from sklearn.linear_model import RidgeCV,Ridge

alphas = 10**np.linspace(10,-2,100)*0.5

ridgecv = RidgeCV(alphas = alphas,normalize = True)
ridgecv.fit(X_train, y_train)
ridgecv.alpha_
```

Out[117…

```
0.08148754173103201
```

The value of alpha that results in the smallest cross-validation error is 0.0814.

In [119…

```
rr = Ridge(alpha = ridgecv.alpha_, normalize = True)
rr.fit(X_train, y_train)
```

Out[119…

```
Ridge(alpha=0.08148754173103201, normalize=True)
```

In [120…

```
print('Root Mean Squared Error:', np.sqrt(metrics.mean_squared_error(y_test, rr.predict(X_test))))

print('r2_score:', metrics.r2_score(y_test, rr.predict(X_test)))
```

```
Root Mean Squared Error: 2.04354646231782
r2_score: 0.7564859864371551
```

Lasso Regression

In [123…

```
from sklearn.linear_model import LassoCV,Lasso

lasso = Lasso(max_iter = 10000, normalize = True)
coefs = []

for a in alphas:
    lasso.set_params(alpha=a)
    lasso.fit(X_train, y_train)
    coefs.append(lasso.coef_)
```

We now perform 10-fold cross-validation to choose the best alpha, refit the model, and compute the associated score:

In [129…

```
lassocv = LassoCV(alphas = None, cv = 10, max_iter = 100000, normalize = True)
lassocv.fit(X_train, y_train)

lasso.set_params(alpha=lassocv.alpha_)
lasso.fit(X_train, y_train)
```

Out[129…

```
Lasso(alpha=0.00033290967622077165, max_iter=10000, normalize=True)
```

The value of alpha that results in the smallest cross-validation error is 0.000332.

In [127…

```
print('Root Mean Squared Error:', np.sqrt(metrics.mean_squared_error(y_test, lasso.predict(X_test))))

print('r2_score:', metrics.r2_score(y_test, lasso.predict(X_test)))
```

```
Root Mean Squared Error: 2.151902612277087
r2_score: 0.7299773770623945
```

3. ElasticNet Regression

ElasticNet combines the properties of both Ridge and Lasso regression. It works by penalizing the model using both the l2-norm and the l1-norm.

In [137…

```
# Let's perform a cross-validation to find the best combination of alpha and l1_ratio
from sklearn.linear_model import ElasticNetCV, ElasticNet

# how much importance should be given to l1 regularization
cv_model = ElasticNetCV(l1_ratio=[.1, .5, .7, .9, .95, .99, .995, 1], eps=0.001, n_alphas=100, fit_intercept=True,
                        normalize=True, precompute='auto', max_iter=2000, tol=0.0001, cv=5,
                        copy_X=True, verbose=0, n_jobs=-1, positive=False, random_state=None, selection='cyclic')
```

In [138…

```
cv_model.fit(X_train, y_train)
```

Out[138…

```
ElasticNetCV(cv=5, l1_ratio=[0.1, 0.5, 0.7, 0.9, 0.95, 0.99, 0.995, 1],
             max_iter=2000, n_jobs=-1, normalize=True)
```

In [139…

```
print('Optimal alpha: %.8f'%cv_model.alpha_)
#The amount of penalization chosen by cross validation

print('Optimal l1_ratio: %.3f'%cv_model.l1_ratio_)
#The compromise between l1 and l2 penalization chosen by cross validation

print('Number of iterations %d'%cv_model.n_iter_)
#number of iterations run by the coordinate descent solver to reach the specified tolerance for the optimal alpha.
```

```
Optimal alpha: 0.00108512
Optimal l1_ratio: 0.500
Number of iterations 25
```

In [140…

```
# train model with best parameters from CV
elastic = ElasticNet(l1_ratio=cv_model.l1_ratio_, alpha = cv_model.alpha_, max_iter=cv_model.n_iter_, fit_intercept=True
elastic.fit(X_train, y_train)
```

Out[140…

```
ElasticNet(alpha=0.0010851196453481816, max_iter=25, normalize=True)
```

In [141…

```
print('Root Mean Squared Error:', np.sqrt(metrics.mean_squared_error(y_test, elastic.predict(X_test))))

print('r2_score:', metrics.r2_score(y_test, elastic.predict(X_test)))
```

```
Root Mean Squared Error: 2.005172333877088
r2_score: 0.7655456290276874
```

Conclusion

- **Key Takeaways:**
- **OLS Regression:** Achieved the highest R-squared value of 0.952, indicating the best fit for this dataset.
- **Regularized Models:** Ridge, lasso, and elasticnet showed lower R-squared values, demonstrating their role in preventing overfitting but not necessarily improving prediction accuracy in this case.
- **Model Suitability:** While regularized models are beneficial for complex datasets with multicollinearity, OLS performed better for this dataset.