ANALYSIS ON THE SELLING PRICE OF USED CARS USING PYTHON

INTRODUCTION:

The used car market is a dynamic and rapidly evolving segment of the automotive industry. With growing consumer interest in pre-owned vehicles, understanding the factors that influence their selling prices has become increasingly important for buyers, sellers, and market analysts alike. This project aims to analyse the selling price of used cars using Python, leveraging data analytics and machine learning techniques to uncover trends and patterns in the market.

The primary objective of this analysis is to identify key factors that drive price variations in used cars, such as vehicle age, mileage, brand, model, fuel type, transmission type, and geographical location. By exploring these variables, we aim to provide actionable insights that can assist stakeholders in making informed decisions.

Using Python's robust data processing and visualization libraries, we will:

Preprocess the Data: Clean and prepare the dataset by handling missing values, outliers, and categorical variables.

Exploratory Data Analysis (EDA): Investigate the dataset to identify patterns, correlations, and trends.

Modeling: Build predictive models to estimate the selling price based on input features.

Insights and Recommendations: Interpret the results to provide meaningful conclusions for buyers and sellers.

This project not only demonstrates the power of Python in data analysis but also sheds light on the complexities of pricing in the used car market, helping participants understand market dynamics and make data-driven decisions.

OBJECTIVE:

The objective of this project is to analyze the factors influencing the selling price of used cars using Python and statistical modeling. By leveraging data analytics techniques, the project aims to:

Identify Key Determinants: Understand how variables such as vehicle age, mileage, brand, model, fuel type, and transmission impact the selling price.

Uncover Market Trends: Detect patterns and trends in the used car market, including pricing differences across regions and time periods.

Build Predictive Models: Develop machine learning models to accurately predict the selling price of used cars based on their features.

Provide Data-Driven Insights: Generate actionable insights that can assist buyers and sellers in making informed decisions.

Libraries used:

1. Pandas
2. Numpy
3. Matplotlib
4. Seaborn
5. Scipy

TASK:

1. IMPORT THE DATASET
2. OBSERVANCE
3. DEFINE HEADERS
4. ANALYSIS ON THE MISSING VALUES
5. CONVERSION
6. DATATYPE CONVERSION
7. NORMALIZING VALUES
8. DESCRIPTIVE ANALYSIS
9. VISUALIZATION
10. GROUPING
11. PIVOT METHOD
12. RESULT

1. IMPORT THE DATASET:
   - We use a CSV file for analysis. The dataset is uploaded to the Jupyter notebook. The CSV file is called out using the pandas library.
   - All the necessary libraries required for analysis and visualization and called out.

```
#Analyze the selling price of Used Cars
```

```
#import the necessary libraries
```

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

```
Matplotlib is building the font cache; this may take a moment.
```

```
df = pd.read_csv("output.csv")
```

2. OBSERVANCE:
   - Let's understand our dataset now. We could possibly call out header function to view the first 5 rows to observe the format of the dataset.

```
df = df.iloc[: , 1:]
df.head()
```

	3	?	alfa-romero	gas	std	two	convertible	rwd	front	88.60	...	130	mpfi	3.47	2.68	9.00	111	5000	21	27	13495
0	3	?	alfa-romero	gas	std	two	convertible	rwd	front	88.6	...	130	mpfi	3.47	2.68	9.0	111	5000	21	27	16500
1	1	?	alfa-romero	gas	std	two	hatchback	rwd	front	94.5	...	152	mpfi	2.68	3.47	9.0	154	5000	19	26	16500
2	2	164	audi	gas	std	four	sedan	fwd	front	99.8	...	109	mpfi	3.19	3.40	10.0	102	5500	24	30	13950
3	2	164	audi	gas	std	four	sedan	4wd	front	99.4	...	136	mpfi	3.19	3.40	8.0	115	5500	18	22	17450
4	2	?	audi	gas	std	two	sedan	fwd	front	99.8	...	136	mpfi	3.19	3.40	8.5	110	5500	19	25	15250

5 rows × 26 columns

   - Our dataset holds 26 columns in total.

3. DEFINE HEADERS
   - We have an option to rename the headers of the columns. On observing the dataset, the column names consists of numbers and short-forms. Let's rename them by columns header function.
   - headers = ["symboling", "normalized-losses", "make",
         "fuel-type", "aspiration","num-of-doors",
          "body-style","drive-wheels", "engine-location",
          "wheel-base","length", "width","height", "curb-weight",
          "engine-type","num-of-cylinders", "engine-size",
          "fuel-system","bore","stroke", "compression-ratio",
          "horsepower", "peak-rpm", "city-mpg", "highway-mpg", "price"]
     df.columns = headers
     df.head()

	symboling	normalized-losses	make	fuel-type	aspiration	num-of-doors	body-style	drive-wheels	engine-location	wheel-base	...	engine-size	fuel-system	bore	stroke	compression-ratio	horsepower	peak-rpm
0	3	?	alfa-romero	gas	std	two	convertible	rwd	front	88.6	...	130	mpfi	3.47	2.68	9.0	111	5000
1	1	?	alfa-romero	gas	std	two	hatchback	rwd	front	94.5	...	152	mpfi	2.68	3.47	9.0	154	5000
2	2	164	audi	gas	std	four	sedan	fwd	front	99.8	...	109	mpfi	3.19	3.40	10.0	102	5500
3	2	164	audi	gas	std	four	sedan	4wd	front	99.4	...	136	mpfi	3.19	3.40	8.0	115	5500
4	2	?	audi	gas	std	two	sedan	fwd	front	99.8	...	136	mpfi	3.19	3.40	8.5	110	5500

5 rows × 26 columns

4. ANALYSIS ON THE MISSING VALUES:
   - Our analysis primarily should include filtering out the missing or null values. These might cause distortion in the results of the analysis. To be more precise, we could possibly filter out such situations.
   - data = df
     data.isna().any()
     data.isnull().any()

```
symboling            False
normalized-losses    False
make                 False
fuel-type            False
aspiration           False
num-of-doors         False
body-style           False
drive-wheels         False
engine-location      False
wheel-base           False
length               False
width                False
height               False
curb-weight          False
engine-type          False
num-of-cylinders     False
engine-size          False
fuel-system          False
bore                 False
stroke               False
compression-ratio    False
horsepower           False
peak-rpm             False
city-mpg             False
highway-mpg          False
price                False
```

5. CONVERSION:
   - We primarily perform conversion is to bring all the values to a common units. This elimination additional calculations and helps us to visualize the results better.

```python
# Converting mpg to L/100 km
data['city-mpg'] = 235 / df['city-mpg']
data.rename(columns = {'city_mpg': "city-L / 100km"}, inplace = True)
print(data.columns)
```

```
Index(['symboling', 'normalized-losses', 'make', 'fuel-type', 'aspiration',
       'num-of-doors', 'body-style', 'drive-wheels', 'engine-location',
       'wheel-base', 'length', 'width', 'height', 'curb-weight', 'engine-type',
       'num-of-cylinders', 'engine-size', 'fuel-system', 'bore', 'stroke',
       'compression-ratio', 'horsepower', 'peak-rpm', 'city-mpg',
       'highway-mpg', 'price'],
      dtype='object')
```

6. DATATYPE CONVERSION:
   - Datatype conversion is required to change the object of certain columns. During visualization, it’s not advised to work with objects over integers or numeric values. We’ll perform the necessary datatype conversions.

```python
# Understanding the datatypes of each colu
data.dtypes
```

```
symboling             int64
normalized-losses    object
make                 object
fuel-type            object
aspiration           object
num-of-doors         object
body-style           object
drive-wheels         object
engine-location      object
wheel-base          float64
length              float64
width               float64
height              float64
curb-weight           int64
engine-type          object
num-of-cylinders     object
engine-size           int64
fuel-system          object
bore                 object
stroke               object
compression-ratio   float64
horsepower           object
peak-rpm             object
city-mpg            float64
highway-mpg           int64
price                object
dtype: object
```

- The price columns should be an integer but here it is object, it’s because of the ‘?’ symbol present in one of the field.
- We’ll remove the ‘?’ symbol as it isn’t required and change the datatype to an integer
  data.price.unique()
  data = data[data.price != '?']
  data['price'] = data['price'].astype('int')

```python
data.dtypes
```

```
symboling              int64
normalized-losses     object
make                  object
fuel-type             object
aspiration            object
num-of-doors          object
body-style            object
drive-wheels          object
engine-location       object
wheel-base           float64
length               float64
width                float64
height               float64
curb-weight            int64
engine-type           object
num-of-cylinders      object
engine-size            int64
fuel-system           object
bore                  object
stroke                object
compression-ratio    float64
horsepower            object
peak-rpm              object
city-mpg             float64
highway-mpg            int64
price                  int32
dtype: object
```

- The datatype of price is now changed to integer.

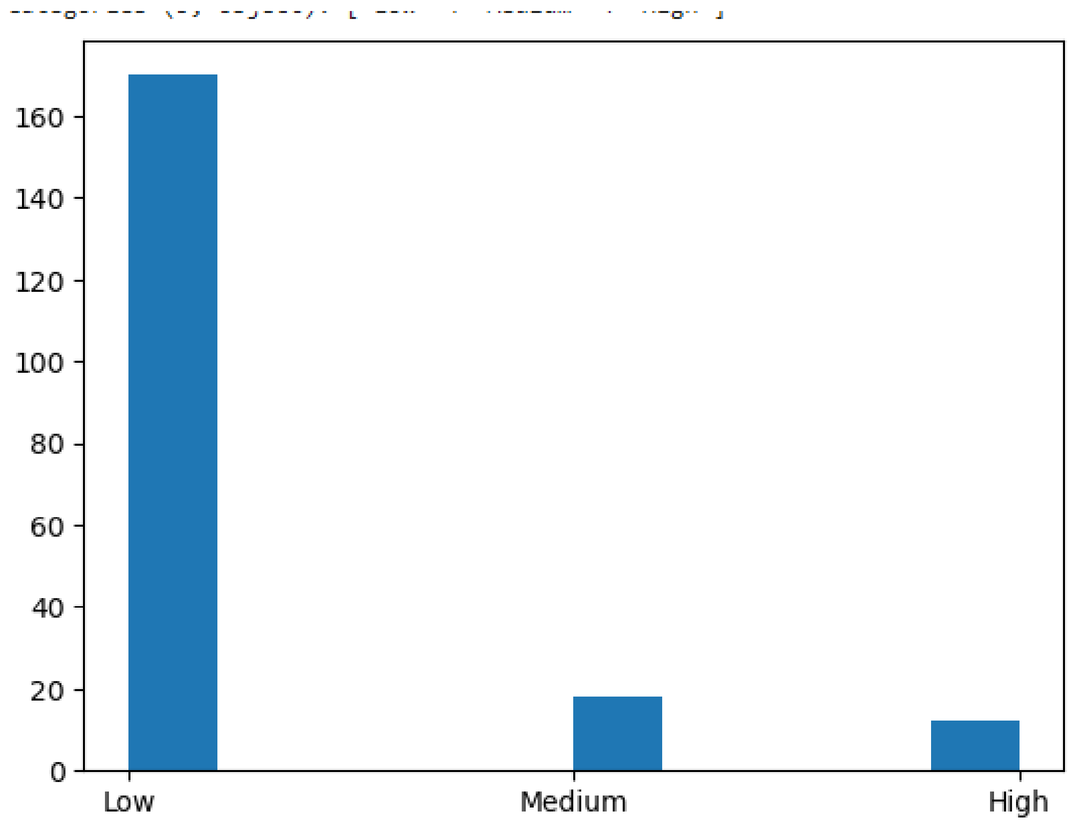
7. NORMALIZING VALUES:
   - Normalizing values by using simple feature scaling method and binning- grouping values
   - Normalization is the process of adjusting data values to fit within a specific range, such as [0, 1]. This is often done using the simple feature scaling method
   - Binning involves dividing data into discrete intervals or "bins." It is used for simplification and categorical analysis, where continuous variables are converted into categories.

```python
data['length'] = data['length']/data['length'].max()
data['width'] = data['width']/data['width'].max()
data['height'] = data['height']/data['height'].max()
```

```python
# binning- grouping values
bins = np.linspace(min(data['price']), max(data['price']), 4)
group_names = ['Low', 'Medium', 'High']
data['price-binned'] = pd.cut(data['price'], bins,
                              labels = group_names,
                              include_lowest = True)

print(data['price-binned'])
plt.hist(data['price-binned'])
plt.show()
```

```
0         Low
1         Low
2         Low
3         Low
4         Low
        ...
199       Low
200    Medium
201    Medium
202    Medium
203    Medium
Name: price-binned, Length: 200, dtype: category
Categories (3, object): ['Low' < 'Medium' < 'High']
```



8. DESCRIPTIVE ANALYSIS:

- Doing descriptive analysis of data categorical to numerical values. By them we could achieve the following:
    1. Facilitates Mathematical Operations
    2. Enhances Compatibility with Models
    3. Improves Data Visualization
    4. Supports Statistical Analysis
    5. Standardization and Consistency
- Benefits for Descriptive Analysis:
    1. Identifying Trends: Numerical data enables better identification of relationships and patterns.
    2. Enabling Advanced Techniques: Makes it possible to apply clustering, regression, and other numerical methods.
    3. Quantifying Relationships: Helps in understanding the contribution or effect of different categories on the target variable.

```
#Descriptive analysis of categorical data to numerical values
pd.get_dummies(data['fuel-type']).head()
data.describe()
```

	symboling	wheel-base	length	width	height	curb-weight	engine-size	compression-ratio	city-mpg	highway-mpg	price
count	200.000000	200.000000	200.000000	200.000000	200.000000	200.000000	200.000000	200.000000	200.000000	200.000000	200.000000
mean	0.830000	98.848000	0.837232	0.915250	0.899523	2555.705000	126.860000	10.170100	9.937914	30.705000	13205.690000
std	1.248557	6.038261	0.059333	0.029207	0.040610	518.594552	41.650501	4.014163	2.539415	6.827227	7966.982558
min	-2.000000	86.600000	0.678039	0.837500	0.799331	1488.000000	61.000000	7.000000	4.795918	16.000000	5118.000000
25%	0.000000	94.500000	0.800937	0.891319	0.869565	2163.000000	97.750000	8.575000	7.833333	25.000000	7775.000000
50%	1.000000	97.000000	0.832292	0.909722	0.904682	2414.000000	119.500000	9.000000	9.791667	30.000000	10270.000000
75%	2.000000	102.400000	0.881788	0.926042	0.928512	2928.250000	142.000000	9.400000	12.368421	34.000000	16500.750000
max	3.000000	120.900000	1.000000	1.000000	1.000000	4066.000000	326.000000	23.000000	18.076923	54.000000	45400.000000

9. VISUALIZATION:

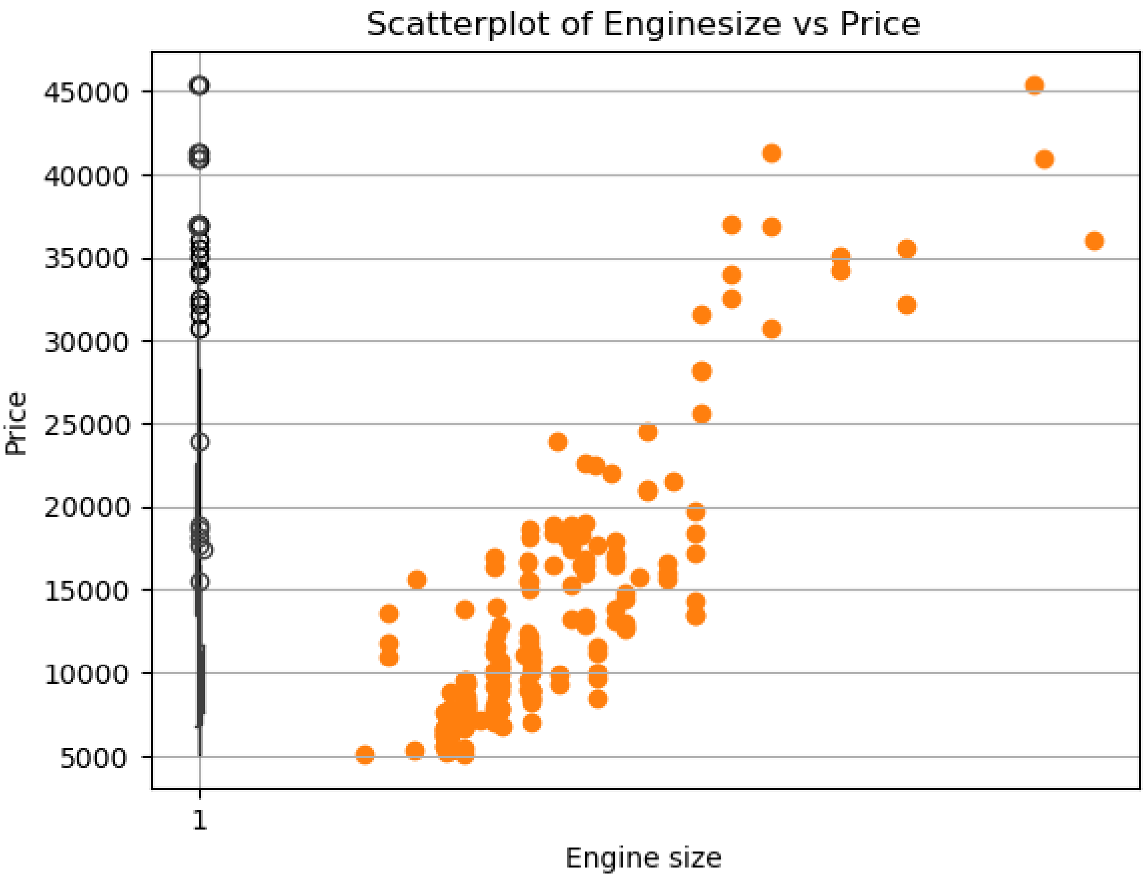
- Visualization provides a quick and intuitive understanding of patterns, trends, and outliers in data.
- Let's plot the data according to the price based on engine size.
- We use scatter plot to achieve the above task.

```
# examples of box plot
plt.boxplot(data['price'])

# by using seaborn
sns.boxplot(x ='drive-wheels', y ='price', data = data)

# Predicting price based on engine size
# Known on x and predictable on y
plt.scatter(data['engine-size'], data['price'])
plt.title('Scatterplot of Enginesize vs Price')
plt.xlabel('Engine size')
```

```
plt.ylabel('Price')
plt.grid()
plt.show()
```



10. GROUPING:

- Grouping enables us to analyze subsets of data by applying aggregate, transformation, or filtering operations to groups based on common characteristics.

```
test = data[['drive-wheels', 'body-style', 'price']]
data_grp = test.groupby(['drive-wheels', 'body-style'],
                        as_index = False).mean()

data_grp
```

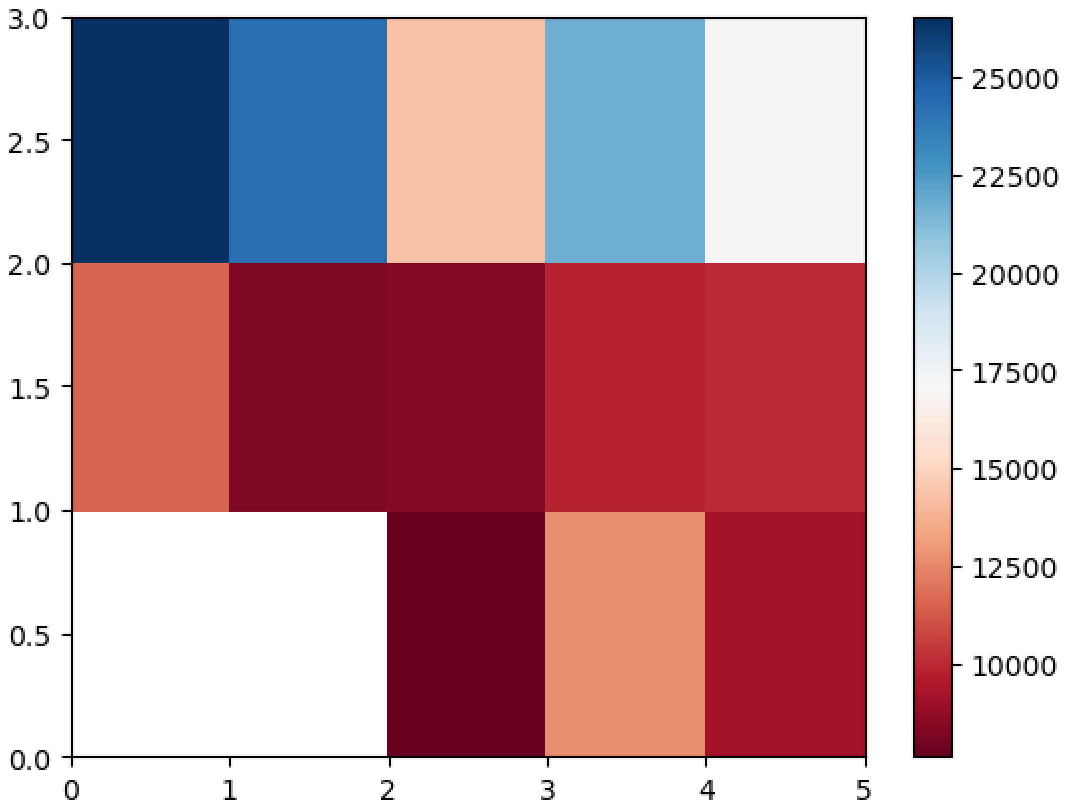
	drive-wheels	body-style	price
0	4wd	hatchback	7603.000000
1	4wd	sedan	12647.333333
2	4wd	wagon	9095.750000
3	fwd	convertible	11595.000000
4	fwd	hardtop	8249.000000
5	fwd	hatchback	8396.387755
6	fwd	sedan	9811.800000
7	fwd	wagon	9997.333333
8	rwd	convertible	26563.250000
9	rwd	hardtop	24202.714286
10	rwd	hatchback	14337.777778
11	rwd	sedan	21711.833333
12	rwd	wagon	16994.222222

11. PIVOT METHOD:

- A Heat map visualizes data with various levels with the intensity using metrics. We derive the values using pivot method for better accuracy.

```
# pivot method
data_pivot = data_grp.pivot(index = 'drive-wheels',
                    columns = 'body-style')
data_pivot

# heatmap for visualizing data
plt.pcolor(data_pivot, cmap ='RdBu')
plt.colorbar()
plt.show()
```



12. RESULT:

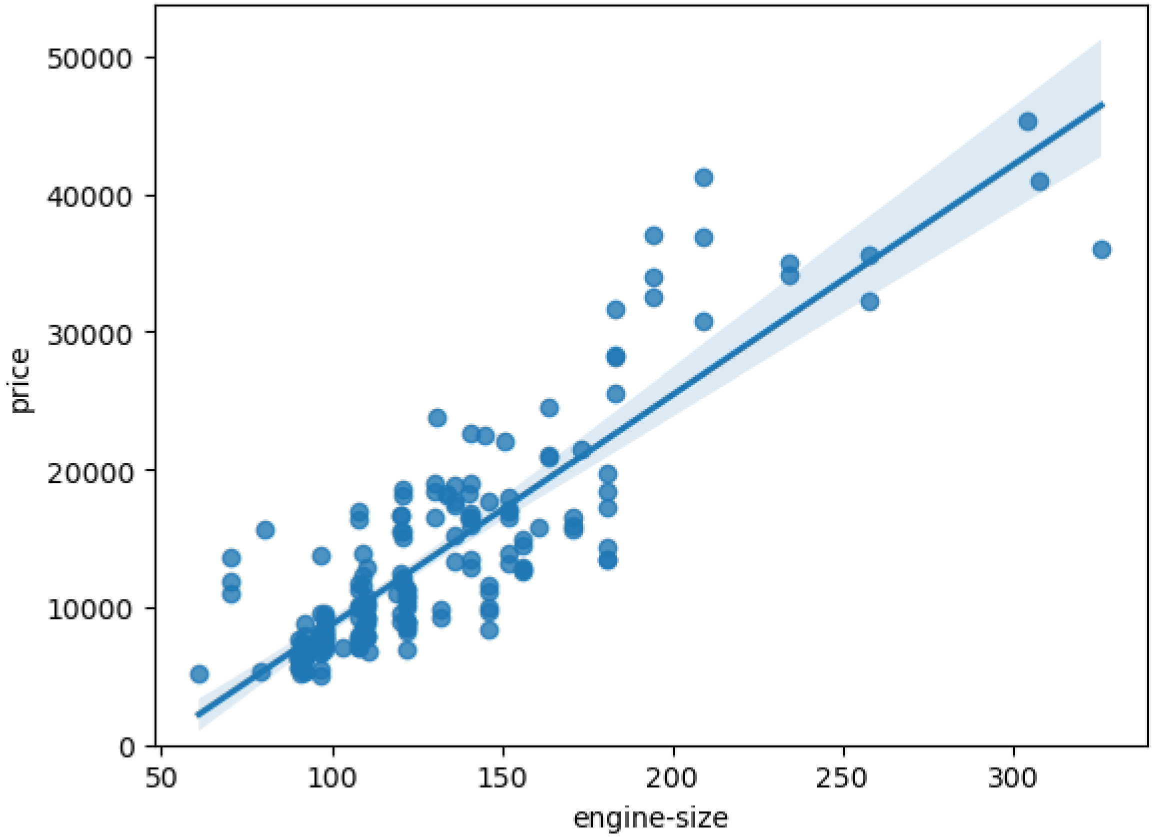
- Obtaining the final result and showing it in the form of a graph. As the slope is increasing in a positive direction, it is a positive linear relationship.
- A positive slope indicates that as the value of the independent variable (x-axis) increases, the value of the dependent variable (y-axis) also increases.
- This linear relationship is often represented by a straight line with a positive gradient in a scatter plot or line graph.

```
# Analysis of Variance- ANOVA
# returns f-test and p-value
# f-test = variance between sample group means divided by
# variation within sample group
# p-value = confidence degree
data_annova = data[['make', 'price']]
grouped_annova = data_annova.groupby(['make'])
annova_results_l = sp.stats.f_oneway(
                    grouped_annova.get_group('honda')['price'],
                    grouped_annova.get_group('subaru')['price']
                        )
print(annova_results_l)

# strong corealtion between a categorical variable
# if annova test gives large f-test and small p-value
```

# Correlation- measures dependency, not causation
sns.regplot(x ='engine-size', y ='price', data = data)
plt.ylim(0,)

```
F_onewayResult(statistic=0.19744030127462606, pvalue=0.6609478240622193)
(0.0, 53743.35218157191)
```



13. CONCLUSION:

In this project, we analyzed the selling price of used cars using Python, employing a combination of data preprocessing, exploratory data analysis, feature engineering. The primary objective was to identify the factors influencing the selling price of used cars and to predict future prices with accuracy.