😊 Exploratory Data Analysis (EDA)

Exploratory Data Analysis (EDA) is like a first look at a dataset to understand its basic features before doing any heavy analysis or creating models. It's similar to getting to know your data by asking simple questions: "What kind of data is this?" "Are there any patterns or outliers?" or "Do I need to clean or adjust this data?"

Think of it like when you first open a box of puzzle pieces—you spread them out, look at the colors and shapes, and try to get a sense of how they fit together before actually assembling the puzzle. EDA is about exploring and understanding, not making conclusions yet.

Key EDA Steps:

1. **Understand the data structure** – What kind of data do we have? Is it numbers, text, or categories?
2. **Summary statistics** – Basic calculations like averages, minimums, and maximums to get a sense of data distribution.
3. **Visualizing data** – Creating charts and graphs to spot patterns, trends, or anomalies.
4. **Checking for missing values** – Finding out if any data points are missing or if there are errors.
5. **Identifying relationships** – Seeing if any variables seem to influence each other.

Example of EDA

Let's say you have data about house prices. You might ask:

- **How much do houses cost on average?** (Summary stats)
- **Do houses in certain areas cost more than in others?** (Group by neighborhood)
- **Is there a relationship between house size and price?** (Scatter plot)
- **Are there any unusually cheap or expensive houses?** (Look for outliers)

By doing EDA, you're essentially preparing the data so you can later build a model to predict house prices with more confidence.

Exploratory Data Analysis (EDA) plays a critical role in understanding data before diving into advanced analysis or modeling. It helps in identifying patterns, spotting anomalies, framing hypotheses, and getting insights into the structure and nature of the data.

Use Cases of EDA:

1. **Data Cleaning and Preprocessing**:
    - Identifying missing values, outliers, and erroneous entries.
    - Understanding the distribution of data (e.g., skewness, spread) to decide how to clean or transform it.
    - Helps in deciding how to handle missing values (e.g., remove, impute).

2. **Hypothesis Testing**:
    - Before formal testing, EDA allows you to explore potential relationships between variables and frame meaningful hypotheses.
    - Example: Is there a correlation between advertising spend and sales increase?
3. **Feature Selection and Engineering**:
    - Discovering which features (columns) of data are more important or redundant.
    - EDA can highlight relationships between features, helping create new variables (feature engineering) for models.
4. **Understanding Data Distribution**:
    - Visualizing the distribution of data to assess normality, outliers, and the presence of skewness.
    - Helps in deciding whether statistical assumptions are met for certain models (like linear regression).
5. **Model Preparation**:
    - By visualizing trends and patterns, it gives direction on which machine learning models might be a good fit.
    - Example: If data has non-linear relationships, it might indicate the need for more complex models.

Benefits of EDA:

1. **Better Understanding of Data**:
    - Helps identify hidden patterns, outliers, and anomalies, leading to deeper insights.
    - It provides a clear overview of data before building any predictive models.
2. **Guides Model Selection**:
    - By uncovering relationships between variables, EDA helps determine which algorithms or models might be most suitable for the data.
    - Saves time by eliminating models that are unsuitable based on data characteristics.
3. **Improved Data Quality**:
    - Identifies data issues, such as missing or duplicated values, that need fixing before analysis.
    - Ensures more accurate and reliable results in later stages.
4. **Hypothesis Formulation**:

- EDA allows you to form better, data-driven hypotheses for more focused research.

5. **Visualization and Communication**:
    - EDA often involves visual tools (charts, graphs) that help explain data and its patterns to stakeholders who might not have a technical background.

Disadvantages of EDA:

1. **Time-Consuming**:
    - EDA can take time, especially for large datasets, as it involves many iterations of exploring and visualizing data.
    - It requires trial and error to understand the data fully.
2. **Subjective Interpretation**:
    - EDA results are sometimes open to interpretation. Different analysts may focus on different patterns, leading to subjective conclusions.
    - Misleading interpretations can arise if one doesn't account for biases or noise in data.
3. **No Direct Prediction**:
    - EDA itself doesn't provide predictive results or models. It only prepares data for modeling, so there's still further work after the exploration phase.
4. **Risk of Overfitting to Patterns**:
    - If too much emphasis is placed on patterns identified during EDA, there's a risk of overfitting models to data quirks that aren't generalizable.

Example:

Suppose you're analyzing customer data for an online store. You could use EDA to:

- Identify the most common product categories customers buy.
- Spot trends in purchasing based on seasonality (e.g., summer vs. winter).
- Analyze customer demographics (age, location) to find which segments are more likely to make high-value purchases. This would guide you in designing marketing strategies and help in building prediction models to target future customers.

In this lab, you explore and clean a dataset to prepare it for further analysis, particularly for machine learning purposes. First, you'll upload the dataset into Jupyter Notebook, then import necessary libraries like NumPy, Pandas, Seaborn, and Matplotlib. The data, which includes information on cars, is examined for inconsistencies, such as inconsistent column names (spaces vs. underscores) and missing values. You'll standardize the column names by converting them to lowercase and replacing spaces with underscores.

Next, you focus on handling specific data types, particularly object-type columns (usually containing strings). These columns are converted to lowercase and cleaned up. The MSRP column (manufacturer's suggested retail price) is selected as the target for a machine learning model, with a focus on predicting car prices. A histogram shows the distribution of car prices, which has a long tail due to many low-priced cars, and log transformation is applied to normalize this distribution.

Finally, missing values in specific columns are addressed, particularly removing them from the "engine horsepower" column.

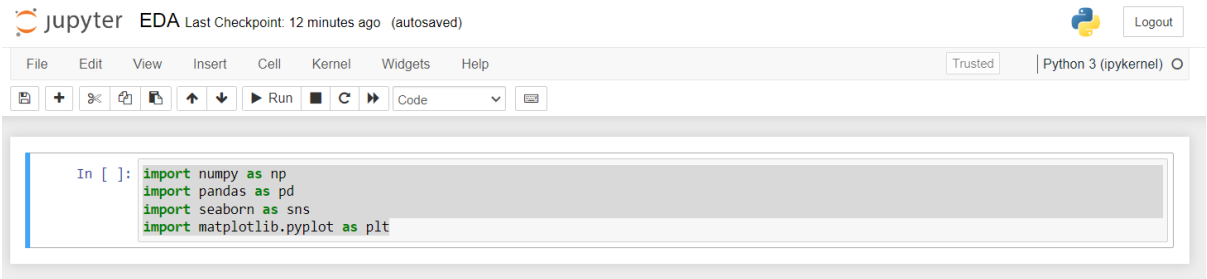
End Goal: The aim is to clean, transform, and analyze the data to prepare it for building a machine learning model to predict car prices accurately.

😄 To begin with the Lab:

1. For this lab, there are some prerequisites you should have Jupyter Notebook installed in your local machine. Then download the folder you get with the lab.
2. Now from the folder you need to look at the data.csv file with all the data. Now you need to upload that file on Jupyter.
3. So, you can open Jupyter Notebook, create a folder, and upload the data.csv file in that folder. After that create a Python 3 notebook in the same folder as you can see below.

4. Now we are going to import some libraries as you can see below.

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



5. Then we are going to load the data as a data frame. Also, you can explore all the columns which we got in this data set.

```
In [1]: import numpy as np
        import pandas as pd
        import seaborn as sns
        import matplotlib.pyplot as plt

In [2]: df = pd.read_csv('data.csv')
        df.head()
```

Out[2]:

	Make	Model	Year	Engine Fuel Type	Engine HP	Engine Cylinders	Transmission Type	Driven_Wheels	Number of Doors	Market Category	Vehicle Size	Vehicle Style	highway MPG	city mpg	Popularity
0	BMW	1 Series M	2011	premium unleaded (required)	335.0	6.0	MANUAL	rear wheel drive	2.0	Factory Tuner,Luxury,High-Performance	Compact	Coupe	26	19	3916
1	BMW	1 Series	2011	premium unleaded (required)	300.0	6.0	MANUAL	rear wheel drive	2.0	Luxury,Performance	Compact	Convertible	28	19	3916
2	BMW	1 Series	2011	premium unleaded (required)	300.0	6.0	MANUAL	rear wheel drive	2.0	Luxury,High-Performance	Compact	Coupe	28	20	3916
3	BMW	1 Series	2011	premium unleaded (required)	230.0	6.0	MANUAL	rear wheel drive	2.0	Luxury,Performance	Compact	Coupe	28	18	3916
4	BMW	1 Series	2011	premium unleaded (required)	230.0	6.0	MANUAL	rear wheel drive	2.0	Luxury	Compact	Convertible	28	18	3916

```
In [ ]:
```

6. Now if you run this command, you can investigate the shape of this data frame. So, we have around 12 thousand cars in this data frame.

   df.shape

```
In [4]: df.shape
Out[4]: (11914, 16)
```

7. Now, in this data set, we can already observe there are some inconsistencies in the data set. So, in the case of column names, in some scenarios, there is a space in the in some other scenarios we have the underscore in case of the column names and you can observe a similar kind of behaviour in the rows as well.
8. So, now we are going to convert our columns into lowercase. Below you can see that we have converted them.

   df.columns = df.columns.str.lower()

```
In [7]: df.columns.str.lower()
Out[7]: Index(['make', 'model', 'year', 'engine fuel type', 'engine hp',
               'engine cylinders', 'transmission type', 'driven_wheels',
               'number of doors', 'market category', 'vehicle size', 'vehicle style',
               'highway mpg', 'city mpg', 'popularity', 'msrp'],
              dtype='object')
```

```
In [8]: df.columns = df.columns.str.lower()

In [9]: df.columns

Out[9]: Index(['make', 'model', 'year', 'engine fuel type', 'engine hp',
               'engine cylinders', 'transmission type', 'driven_wheels',
               'number of doors', 'market category', 'vehicle size', 'vehicle style',
               'highway mpg', 'city mpg', 'popularity', 'msrp'],
              dtype='object')
```

9. Here we have changed the blank spaces with underscore using the command below.

   df.columns = df.columns.str.replace(" ","_")
   df.columns

```
In [10]: df.columns = df.columns.str.replace(" ","_")
         df.columns

Out[10]: Index(['make', 'model', 'year', 'engine_fuel_type', 'engine_hp',
                'engine_cylinders', 'transmission_type', 'driven_wheels',
                'number_of_doors', 'market_category', 'vehicle_size', 'vehicle_style',
                'highway_mpg', 'city_mpg', 'popularity', 'msrp'],
               dtype='object')
```

10. I can select the columns which are in the object data type. And then once I select those columns, then I'll convert the entire column into the lowercase and we can also replace the spaces with underscore.
11. By executing the below command, we can know what exactly is the data type.

```
In [11]: df.dtypes

Out[11]: make                object
         model               object
         year                 int64
         engine_fuel_type    object
         engine_hp          float64
         engine_cylinders   float64
         transmission_type   object
         driven_wheels       object
         number_of_doors    float64
         market_category     object
         vehicle_size        object
         vehicle_style       object
         highway_mpg          int64
         city_mpg             int64
         popularity           int64
         msrp                 int64
         dtype: object
```

12. Here we have stored these dtypes inside a new variable called d. So, from the output given below we can say that this d type is in the format of the PANDAS series.

```
In [12]: type(df.dtypes)

Out[12]: pandas.core.series.Series
```

```
In [14]: d = df.dtypes
         d

Out[14]: make                 object
         model                object
         year                  int64
         engine_fuel_type     object
         engine_hp           float64
         engine_cylinders    float64
         transmission_type    object
         driven_wheels        object
         number_of_doors     float64
         market_category      object
         vehicle_size         object
         vehicle_style        object
         highway_mpg           int64
         city_mpg              int64
         popularity            int64
         msrp                  int64
         dtype: object
```

13. Now, if I say D dot values, this is going to return me the values only.

```
In [15]: d.values

Out[15]: array([dtype('O'), dtype('O'), dtype('int64'), dtype('O'),
                dtype('float64'), dtype('float64'), dtype('O'), dtype('O'),
                dtype('float64'), dtype('O'), dtype('O'), dtype('O'),
                dtype('int64'), dtype('int64'), dtype('int64'), dtype('int64')],
               dtype=object)
```

14. Now, what I can do right now is, I can make use of this logic that is df d-type's logic and I can check and prepare a Boolean mask. So, the way that I can prepare a Boolean mask is I'll say D is equal to object.
15. Wherever we have got the object data type, in those circumstances we have the Boolean values as true, and wherever we it is false. This implies that any place we have a numerical value is stated as being false.

```
In [16]: d == 'object'

Out[16]: make                 True
         model                True
         year                False
         engine_fuel_type     True
         engine_hp           False
         engine_cylinders    False
         transmission_type    True
         driven_wheels        True
         number_of_doors     False
         market_category      True
         vehicle_size         True
         vehicle_style        True
         highway_mpg         False
         city_mpg            False
         popularity          False
         msrp                False
         dtype: bool
```

16. So, I'll use my D, which is my existing PANDAS series itself. And inside my indexing operator, I'll mention this condition. If I execute this, this is going to return a PANDA'S series. Now, in this PANDA'S series, this is going to return only the column names where this belongs to the object data type.

```
In [17]: d[d == 'object']

Out[17]: make                 object
         model                object
         engine_fuel_type     object
         transmission_type    object
         driven_wheels        object
         market_category      object
         vehicle_size         object
         vehicle_style        object
         dtype: object
```

17. So now using the below command it contains the column names where the data types are equal to the string.

col = d[d == 'object'].index
col

```
In [18]: col = d[d == 'object'].index
         col

Out[18]: Index(['make', 'model', 'engine_fuel_type', 'transmission_type',
                'driven_wheels', 'market_category', 'vehicle_size', 'vehicle_style'],
               dtype='object')
```

18. Now we can go ahead, get each and every column that we have got and we will convert it into a lowercase.

    for c in col:
    df[c] = df[c].str.lower().str.replace(" ",'_')

```
In [19]: for c in col:
             df[c] = df[c].str.lower().str.replace(" ",'_')

In [21]: df.head()

Out[21]:
```

	make	model	year	engine_fuel_type	engine_hp	engine_cylinders	transmission_type	driven_wheels	number_of_doors	market_categ
0	bmw	1_series_m	2011	premium_unleaded_(required)	335.0	6.0	manual	rear_wheel_drive	2.0	factory_tuner,luxury,hi performar
1	bmw	1_series	2011	premium_unleaded_(required)	300.0	6.0	manual	rear_wheel_drive	2.0	luxury,performar
2	bmw	1_series	2011	premium_unleaded_(required)	300.0	6.0	manual	rear_wheel_drive	2.0	luxury,high-performar
3	bmw	1_series	2011	premium_unleaded_(required)	230.0	6.0	manual	rear_wheel_drive	2.0	luxury,performar
4	bmw	1_series	2011	premium_unleaded_(required)	230.0	6.0	manual	rear_wheel_drive	2.0	lux

19. Now if we run the info command, we can see that we got 16 columns here.

```
In [22]: df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 11914 entries, 0 to 11913
Data columns (total 16 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   make               11914 non-null  object
 1   model              11914 non-null  object
 2   year               11914 non-null  int64
 3   engine_fuel_type   11911 non-null  object
 4   engine_hp          11845 non-null  float64
 5   engine_cylinders   11884 non-null  float64
 6   transmission_type  11914 non-null  object
 7   driven_wheels      11914 non-null  object
 8   number_of_doors    11908 non-null  float64
 9   market_category    8172 non-null   object
 10  vehicle_size       11914 non-null  object
 11  vehicle_style      11914 non-null  object
 12  highway_mpg        11914 non-null  int64
 13  city_mpg           11914 non-null  int64
 14  popularity         11914 non-null  int64
 15  msrp               11914 non-null  int64
dtypes: float64(3), int64(5), object(8)
memory usage: 1.5+ MB
```

20. From the above snapshot you can see that we have a column with the name MSRP (manufacturer's suggested retail price)
21. Now we will be using this column in case you are applying the machine learning model; we'll be using this column for predicting the prices of the car.
22. So, for this example purpose, we are going to consider the column as the output column.

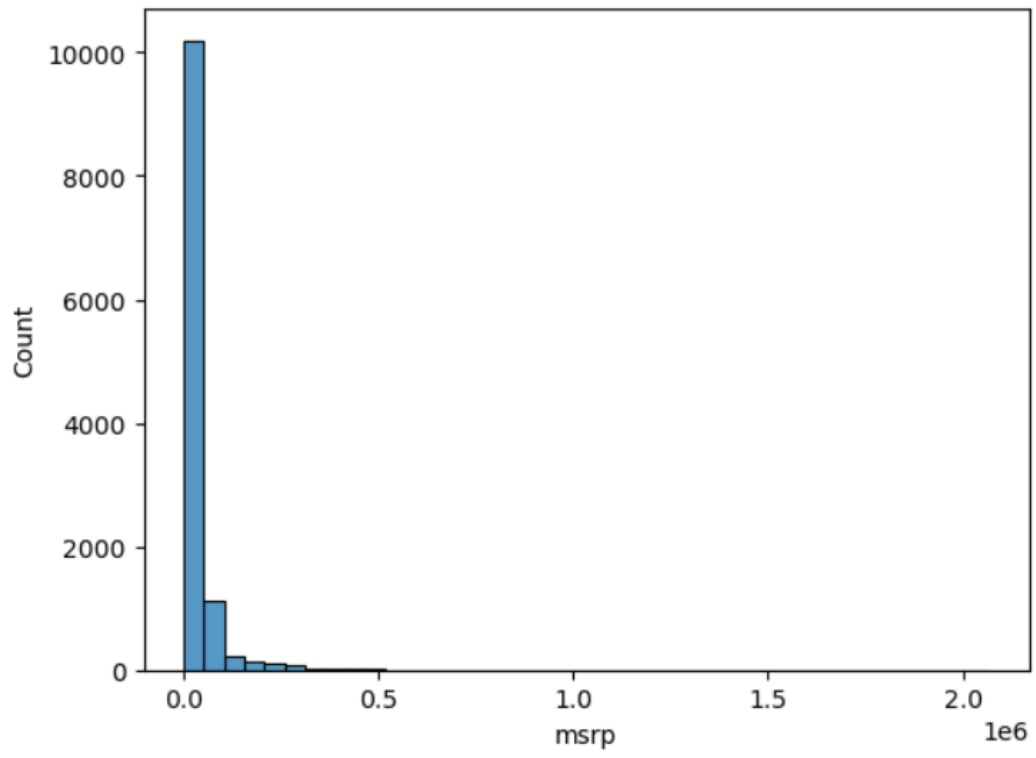
```
In [28]: # msrp - Output column, Target Variable
         df['msrp']
```

```
Out[28]: 0        46135
         1        40650
         2        36350
         3        29450
         4        34500
                  ...
         11909    46120
         11910    56670
         11911    50620
         11912    50920
         11913    28995
         Name: msrp, Length: 11914, dtype: int64
```

23. By using the below command, we have created a histogram chart. Now after I plot this graph, we can notice that the data that we have, that is the MSRP that we have got, has a long tail towards the right.
24. It means that we have a lot of cars in the data which have low prices.

```
In [29]: sns.histplot(data=df, x='msrp',bins=40)
```

```
Out[29]: <Axes: xlabel='msrp', ylabel='Count'>
```



25. Below you can see that we created a subset of data using the command below and with this we are going to create the histogram chart.

```
In [30]: sub = df[df['msrp']<95000].copy()
         sub
```

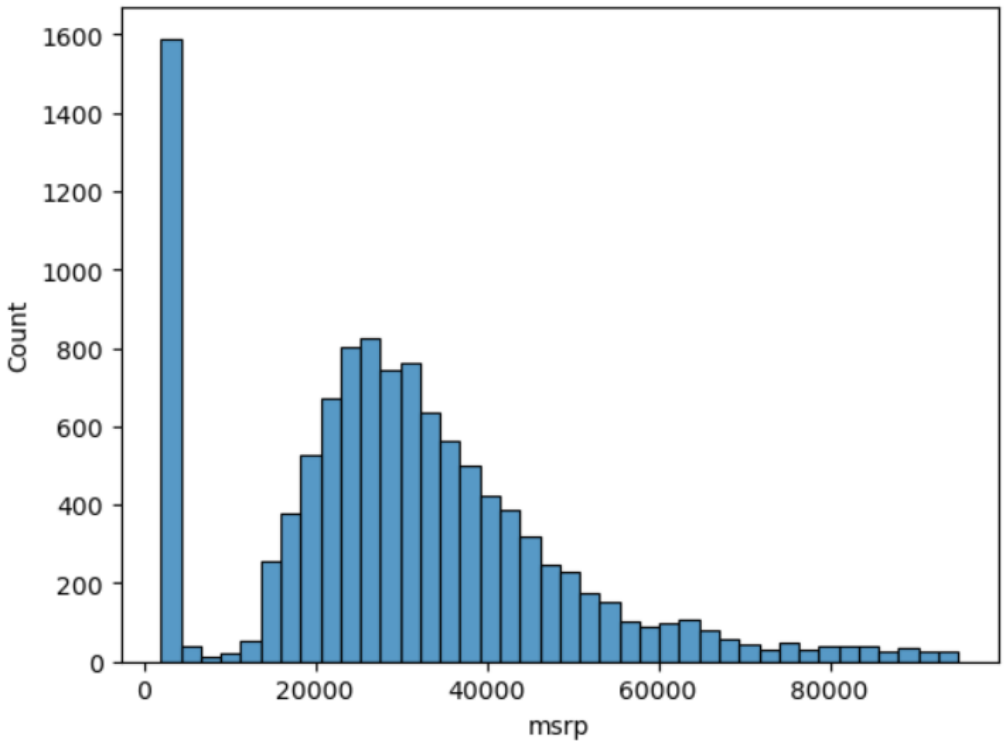
Out[30]:

	make	model	year	engine_fuel_type	engine_hp	engine_cylinders	transmission_type	driven_wheels	number_of_doors	r
0	bmw	1_series_m	2011	premium_unleaded_(required)	335.0	6.0	manual	rear_wheel_drive	2.0	factory_
1	bmw	1_series	2011	premium_unleaded_(required)	300.0	6.0	manual	rear_wheel_drive	2.0	lu
2	bmw	1_series	2011	premium_unleaded_(required)	300.0	6.0	manual	rear_wheel_drive	2.0	luxury,l
3	bmw	1_series	2011	premium_unleaded_(required)	230.0	6.0	manual	rear_wheel_drive	2.0	lu
4	bmw	1_series	2011	premium_unleaded_(required)	230.0	6.0	manual	rear_wheel_drive	2.0	
...	...	...	...	...	...	...	...	...	...	
11909	acura	zdx	2012	premium_unleaded_(required)	300.0	6.0	automatic	all_wheel_drive	4.0	crossover,
11910	acura	zdx	2012	premium_unleaded_(required)	300.0	6.0	automatic	all_wheel_drive	4.0	crossover,
11911	acura	zdx	2012	premium_unleaded_(required)	300.0	6.0	automatic	all_wheel_drive	4.0	crossover,
11912	acura	zdx	2013	premium_unleaded_(recommended)	300.0	6.0	automatic	all_wheel_drive	4.0	crossover,
11913	lincoln	zephyr	2006	regular_unleaded	221.0	6.0	automatic	front_wheel_drive	4.0	

11229 rows × 16 columns

```
In [31]: sns.histplot(data=sub, x='msrp',bins=40)
```

Out[31]: <Axes: xlabel='msrp', ylabel='Count'>

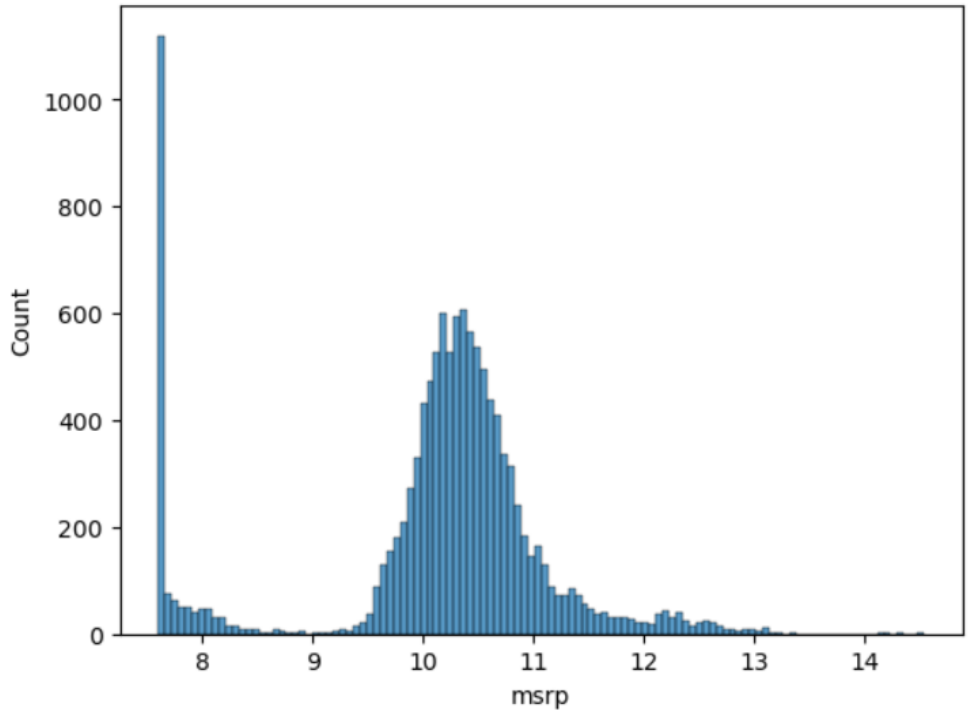


26. Here we have performed the log transformation on our data which has removed the long tail, now the distribution resembles a bell-shaped curved.

```
In [32]: log_prices = np.log1p(df['msrp'])

In [33]: sns.histplot(log_prices)

Out[33]: <Axes: xlabel='msrp', ylabel='Count'>
```



27. Now if you run the info command again you will observe that in the dataset there are some missing values.

```
In [35]: df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 11914 entries, 0 to 11913
Data columns (total 16 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   make               11914 non-null  object 
 1   model              11914 non-null  object 
 2   year               11914 non-null  int64  
 3   engine_fuel_type   11911 non-null  object 
 4   engine_hp          11845 non-null  float64
 5   engine_cylinders   11884 non-null  float64
 6   transmission_type  11914 non-null  object 
 7   driven_wheels      11914 non-null  object 
 8   number_of_doors    11908 non-null  float64
 9   market_category    8172 non-null   object 
 10  vehicle_size       11914 non-null  object 
 11  vehicle_style      11914 non-null  object 
 12  highway_mpg        11914 non-null  int64  
 13  city_mpg           11914 non-null  int64  
 14  popularity         11914 non-null  int64  
 15  msrp               11914 non-null  int64  
dtypes: float64(3), int64(5), object(8)
memory usage: 1.5+ MB
```

28. So, by using the below function we can generate a data frame with Boolean values.

```
In [36]: df.isna()
```

Out[36]:

	make	model	year	engine_fuel_type	engine_hp	engine_cylinders	transmission_type	driven_wheels	number_of_doors	market_category	vehicle_size
0	False	False	False	False	False	False	False	False	False	False	False
1	False	False	False	False	False	False	False	False	False	False	False
2	False	False	False	False	False	False	False	False	False	False	False
3	False	False	False	False	False	False	False	False	False	False	False
4	False	False	False	False	False	False	False	False	False	False	False
...	...	...	...	...	...	...	...	...	...	...	...
11909	False	False	False	False	False	False	False	False	False	False	False
11910	False	False	False	False	False	False	False	False	False	False	False
11911	False	False	False	False	False	False	False	False	False	False	False
11912	False	False	False	False	False	False	False	False	False	False	False
11913	False	False	False	False	False	False	False	False	False	False	False

11914 rows × 16 columns

29. With this we can call a sum function which will help us to display the number of missing rows. You can observe the missing the values from the given snapshot below.

```
In [37]: df.isna().sum()
```

```
Out[37]: make                  0
         model                 0
         year                  0
         engine_fuel_type      3
         engine_hp            69
         engine_cylinders     30
         transmission_type     0
         driven_wheels         0
         number_of_doors       6
         market_category    3742
         vehicle_size          0
         vehicle_style         0
         highway_mpg           0
         city_mpg              0
         popularity            0
         msrp                  0
         dtype: int64
```

30. Now we are going to deal with these missing values. Below you can see that using this command we have removed all the missing values from engine hp column.

```
In [38]: df['engine_hp'] = df['engine_hp'].fillna(df['engine_hp'].median())
```

```
In [40]: df.isna().sum()
```

```
Out[40]: make                  0
         model                 0
         year                  0
         engine_fuel_type      3
         engine_hp             0
         engine_cylinders     30
         transmission_type     0
         driven_wheels         0
         number_of_doors       6
         market_category    3742
         vehicle_size          0
         vehicle_style         0
         highway_mpg           0
         city_mpg              0
         popularity            0
         msrp                  0
         dtype: int64
```

31. So, this is the high-level overview of EDA, here we have analysed the data and take care of the missing data.