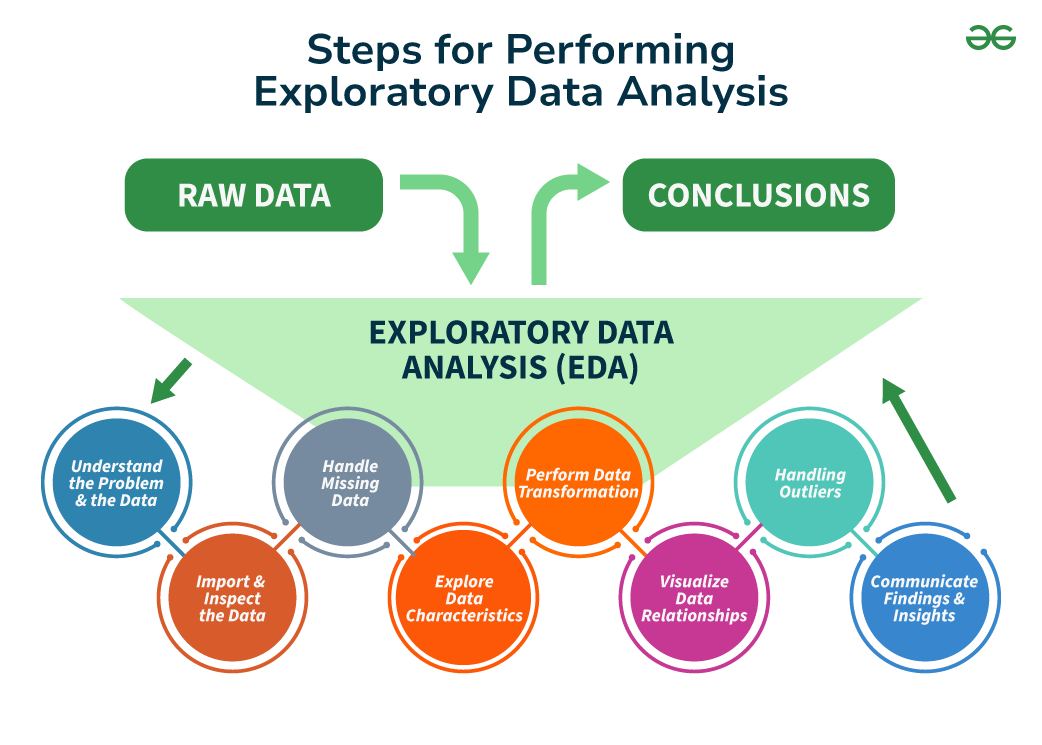
What is Exploratory Data Analysis?

**Exploratory Data Analysis (EDA)** is a important step in data science as it visualizing data to understand its main features, find patterns and discover how different parts of the data are connected. In this article, we will see more about **Exploratory Data Analysis (EDA)**.



2 / 3

**Why Exploratory Data Analysis is Important?**

Exploratory Data Analysis (EDA) is important for several reasons in the context of data science and statistical modeling. Here are some of the key reasons:

1. It helps to understand the dataset by showing how many features it has, what type of data each feature contains and how the data is distributed.
2. It helps to identify hidden patterns and relationships between different data points which help us in and model building.
3. Allows to identify errors or unusual data points (outliers) that could affect our results.
4. The insights gained from EDA help us to identify most important features for building models and guide us on how to prepare them for better performance.
5. By understanding the data it helps us in choosing best modeling techniques and adjusting them for better results.

**Types of Exploratory Data Analysis**

There are various types of EDA based on nature of records. Depending on the number of columns we are analyzing we can divide EDA into three types:

**1. Univariate Analysis**

[Univariate analysis](https://www.geeksforgeeks.org/what-is-univariate-bivariate-multivariate-analysis-in-data-visualisation/)focuses on studying one variable to understand its characteristics. It helps to describe data and find patterns within a single feature. Various common methods like histograms are used to show data distribution, box plots to detect outliers and understand data spread and bar charts for categorical data. Summary statistics like[**mean**,](https://www.geeksforgeeks.org/mean/) [**median**](https://www.geeksforgeeks.org/median/), [**mode**](https://www.geeksforgeeks.org/what-is-mode/),[**variance**](https://www.geeksforgeeks.org/variance/) and [**standard deviation**](https://www.geeksforgeeks.org/standard-deviation-formula/) helps in describing the central tendency and spread of the data

**2. Bivariate Analysis**

[Bivariate Analysis](https://www.geeksforgeeks.org/bivariate-analysis/) focuses on identifying relationship between two variables to find connections, correlations and dependencies. It helps to understand how two variables interact with each other. Some key techniques include:

* Scatter plots which visualize the relationship between two continuous variables.
* **C**orrelation coefficient measures how strongly two variables are related which commonly use [**Pearson's correlation**](https://www.geeksforgeeks.org/pearson-correlation-coefficient/) for linear relationships.
* Cross-tabulation or contingency tables shows the frequency distribution of two categorical variables and help to understand their relationship.
* **Line graphs** are useful for comparing two variables over time in time series data to identify trends or patterns.
* [**Covariance**](https://www.geeksforgeeks.org/mathematics-covariance-and-correlation/) measures how two variables change together but it is paired with the correlation coefficient for a clearer and more standardized understanding of the relationship.

**3. Multivariate Analysis**

[Multivariate Analysis](https://www.geeksforgeeks.org/multivariate-analysis-in-r/) identify relationships between two or more variables in the dataset and aims to understand how variables interact with one another which is important for statistical modeling techniques. It include techniques like:

* [**Pair plots**](https://www.geeksforgeeks.org/python-seaborn-pairplot-method/)which shows the relationships between multiple variables at once and helps in understanding how they interact.
* Another technique is [**Principal Component Analysis (PCA**](https://www.geeksforgeeks.org/principal-component-analysis-pca/)**)** which reduces the complexity of large datasets by simplifying them while keeping the most important information.
* [**Spatial Analysis**](https://www.geeksforgeeks.org/what-is-spatial-analysis/) is used for geographical data by using maps and spatial plotting to understand the geographical distribution of variables.
* [**Time Series Analysis**](https://www.geeksforgeeks.org/time-series-data-visualization-in-python/) is used for datasets that involve time-based data and it involves understanding and modeling patterns and trends over time. Common techniques include line plots, autocorrelation analysis, moving averages and [ARIMA](https://www.geeksforgeeks.org/python-arima-model-for-time-series-forecasting/) models.

**Steps for Performing Exploratory Data Analysis**

It involves a series of steps to help us understand the data, uncover patterns, identify anomalies, test hypotheses and ensure the data is clean and ready for further analysis. It can be done using different tools like:

* In Python, Pandas is used to clean, filter and manipulate data. [Matplotlib](https://www.geeksforgeeks.org/data-visualization-using-matplotlib/)helps to create basic visualizations while [Seaborn](https://www.geeksforgeeks.org/introduction-to-seaborn-python/)makes more attractive plots. For interactive visualizations Plotly is a good choice.
* In R, **ggplot2**is used for creating complex plots, **dplyr**helps with data manipulation and **tidyr**makes sure our data is organized and easy to work with.

Its step includes:

**Step 1: Understanding the Problem and the Data**

The first step in any data analysis project is to fully understand the problem we're solving and the data we have. This includes asking key questions like:

1. What is the business goal or research question?
2. What are the variables in the data and what do they represent?
3. What types of data (numerical, categorical, text, etc.) do you have?
4. Are there any known data quality issues or limitations?
5. Are there any domain-specific concerns or restrictions?

By understanding the problem and the data, we can plan our analysis more effectively, avoid incorrect assumptions and ensure accurate conclusions.

**Step 2: Importing and Inspecting the Data**

After understanding the problem and the data, next step is to import the data into our analysis environment such as Python, R or a spreadsheet tool. It’s important to find data to gain an basic understanding of its structure, variable types and any potential issues. Here’s what we can do:

1. Load the data into our environment carefully to avoid errors or truncations.
2. Check the size of the data like number of rows and columns to understand its complexity.
3. Check for missing values and see how they are distributed across variables since missing data can impact the quality of your analysis.
4. Identify data types for each variable like numerical, categorical, etc which will help in the next steps of data manipulation and analysis.
5. Look for errors or inconsistencies such as invalid values, mismatched units or outliers which could show major issues with the data.

By completing these tasks we'll be prepared to clean and analyze the data more effectively.

**Step 3: Handling Missing Data**

[Missing data](https://www.geeksforgeeks.org/ml-handling-missing-values/)is common in many datasets and can affect the quality of our analysis. During EDA it's important to identify and handle missing data properly to avoid biased or misleading results. Here’s how to handle it:

1. Understand the patterns and possible causes of missing data. Is it missing completely at random (MCAR), missing at random (MAR) or missing not at random (MNAR). Identifying this helps us to find best way to handle the missing data.
2. Decide whether to remove missing data or impute (fill in) the missing values. Removing data can lead to biased outcomes if the missing data isn’t MCAR. Filling values helps to preserve data but should be done carefully.
3. Use appropriate imputation methods like mean or median imputation, [regression](https://www.geeksforgeeks.org/regression-in-machine-learning/)imputation or machine learning techniques like[KNN](https://www.geeksforgeeks.org/k-nearest-neighbours/)or [decision trees](https://www.geeksforgeeks.org/decision-tree/)based on the data’s characteristics.
4. Consider the impact of missing data. Even after imputing, missing data can cause uncertainty and bias so understands the result with caution.

Properly handling of missing data improves the accuracy of our analysis and prevents misleading conclusions.

**Step 4: Exploring Data Characteristics**

After addressing missing data we find the characteristics of our data by checking the distribution, central tendency and variability of our variables and identifying outliers or anomalies. This helps in selecting appropriate analysis methods and finding major data issues. We should calculate summary statistics like mean, median, mode, standard deviation, skewness and kurtosis for numerical variables. These provide an overview of the data’s distribution and helps us to identify any irregular patterns or issues.

**Step 5: Performing Data Transformation**

Data transformation is an important step in EDA as it prepares our data for accurate analysis and modeling. Depending on our data's characteristics and analysis needs, we may need to transform it to ensure it's in the right format. Common transformation techniques include:

1. Scaling or normalizing numerical variables like [min-max scaling](https://www.geeksforgeeks.org/standardscaler-minmaxscaler-and-robustscaler-techniques-ml/) or [standardization](https://www.geeksforgeeks.org/what-is-standardization-in-machine-learning/).
2. Encoding categorical variables for machine learning like [one-hot encoding](https://www.geeksforgeeks.org/ml-one-hot-encoding/) or [label encoding.](https://www.geeksforgeeks.org/ml-label-encoding-of-datasets-in-python/)
3. Applying mathematical transformations like [logarithmic square root](https://www.geeksforgeeks.org/square-root-number-using-log/)to correct skewness or non-linearity.
4. Creating new variables from existing ones like calculating ratios or combining variables.
5. Aggregating or grouping data based on specific variables or conditions.

**Step 6: Visualizing Relationship of Data**

Visualization helps to find relationships between variables and identify patterns or trends that may not be seen from summary statistics alone.

1. For categorical variables, create frequency tables, bar plots and pie charts to understand the distribution of categories and identify imbalances or unusual patterns.
2. For numerical variables generate histograms, box plots, violin plots and density plots to visualize distribution, shape, spread and potential outliers.
3. To find relationships between variables use scatter plots, correlation matrices or statistical tests like Pearson’s correlation coefficient or Spearman’s rank correlation.

**Step 7: Handling Outliers**

Outliers are data points that differs from the rest of the data may caused by errors in measurement or data entry. Detecting and handling outliers is important because they can skew our analysis and affect model performance. We can identify outliers using methods like[**interquartile range (IQR)**](https://www.geeksforgeeks.org/interquartile-range-iqr/), [**Z-scores**](https://www.geeksforgeeks.org/z-score-in-statistics/) or domain-specific rules. Once identified it can be removed or adjusted depending on the context. Properly managing outliers shows our analysis is accurate and reliable.

**Step 8: Communicate Findings and Insights**

The final step in EDA is to communicate our findings clearly. This involves summarizing the analysis, pointing out key discoveries and presenting our results in a clear way.

1. Clearly state the goals and scope of your analysis.
2. Provide context and background to help others understand your approach.
3. Use visualizations to support our findings and make them easier to understand.
4. Highlight key insights, patterns or anomalies discovered.
5. Mention any limitations or challenges faced during the analysis.
6. Suggest next steps or areas that need further investigation.

Effective communication is important to ensure that our EDA efforts make an impact and that stakeholders understand and act on our insights. By following these steps and using the right tools, EDA helps in increasing the quality of our data, leading to more informed decisions and successful outcomes in any data-driven project.

[notes](https://www.geeksforgeeks.org/data-analysis/what-is-exploratory-data-analysis/)

**✅ Goal:**

If two or more rows have the **same make and model**, and some are missing price, fill the missing price with the **average or median** price of that make-model combination.

**💡 Step-by-Step Strategy**

# Step 1: Group by make and model and compute the median price

model\_price\_map = df.groupby(['make', 'model'])['price'].median()

# Step 2: Define a function to fill missing price based on group

def fill\_price(row):

if pd.isnull(row['price']):

return model\_price\_map.get((row['make'], row['model']), np.nan)

else:

return row['price']

# Step 3: Apply the function row-wise

df['price'] = df.apply(fill\_price, axis=1)

**✅ Explanation:**

* groupby(['make', 'model'])['price'].median() computes the median price per make-model group.
* .get((row['make'], row['model']), np.nan) fetches the median for that group, or leaves it as NaN if no match is found.
* apply() is used to go row-by-row and impute missing values.

**Optional: Still Missing?**

After this, if you want to **drop the remaining** rows that couldn’t be filled:

df = df.dropna(subset=['price'])

#### Optional: Drop partial duplicates (if needed):

If you think vehicles with same name, year, and mileage are essentially the same listing, try:

python

CopyEdit

df = df.drop\_duplicates(subset=['name', 'year', 'mileage'])

This helps clean up cases where the same car may be listed twice but has different colors or descriptions.

**✅ 🔧 Encoding Code (Ready for Jupyter Notebook)**

import pandas as pd

from sklearn.preprocessing import LabelEncoder

# Assuming your cleaned DataFrame is named df

# 1️⃣ Label Encoding for low-cardinality categorical features

label\_cols = ['make', 'trim', 'interior\_color', 'exterior\_color']

le = LabelEncoder()

for col in label\_cols:

df[col] = le.fit\_transform(df[col])

# 2️⃣ Frequency Encoding for high-cardinality column like 'model'

model\_freq = df['model'].value\_counts()

df['model\_encoded'] = df['model'].map(model\_freq)

# Optional: drop the original 'model' column if no longer needed

df.drop(columns=['model'], inplace=True)

# 3️⃣ One-Hot Encoding for nominal categorical features

one\_hot\_cols = ['fuel', 'transmission', 'body', 'drivetrain']

df = pd.get\_dummies(df, columns=one\_hot\_cols, drop\_first=True)

# ✅ Done: Encoded DataFrame is ready

df.head()

**🧠 Notes:**

* drop\_first=True in pd.get\_dummies() avoids the dummy variable trap (good for linear models).
* You can choose to keep or drop name, description, and engine depending on whether you’ll process them further (like TF-IDF or dropping).
* You can rename the encoded DataFrame or export it if needed.

Let me know if you want the code to:

* Export this as a new CSV
* Visualize encoding effects
* Start model training afterward

Now that your data is encoded and clean, let's move to **Feature Engineering** — a crucial step to boost your model's predictive power.

### ✅ Key Feature Engineering Ideas for Vehicle Price Prediction

Here are meaningful, domain-specific features you can engineer from your current columns:

### 1️⃣ ****Age of Vehicle****

Older vehicles generally have lower prices.

from datetime import datetime

current\_year = datetime.now().year

df['vehicle\_age'] = current\_year - df['year']

### 2️⃣ ****Mileage per Year****

Gives insight into how heavily a car has been used.

df['mileage\_per\_year'] = df['mileage'] / df['vehicle\_age'].replace(0, 1) # avoid division by zero

### 3️⃣ ****Is New****

If mileage is near 0 and age is 0, it’s likely brand new.

df['is\_new'] = ((df['vehicle\_age'] == 0) & (df['mileage'] < 100)).astype(int)

### 4️⃣ ****Cylinders Category****

Binning cylinders into categories (e.g., 4, 6, 8+).

df['cyl\_category'] = pd.cut(df['cylinders'], bins=[0, 4, 6, 12], labels=['Low', 'Medium', 'High'])

### 5️⃣ ****Luxury Brands****

Add a binary flag for premium makes.

luxury\_brands = ['BMW', 'Mercedes-Benz', 'Audi', 'Lexus', 'Tesla']

df['is\_luxury'] = df['make'].apply(lambda x: 1 if x in luxury\_brands else 0)

⚠️ If make is already label encoded, you'll need a reverse mapping to get brand names.

### 6️⃣ ****Trim Simplification****

If trim has lots of unique values, simplify to common keywords.

df['trim\_level'] = df['trim'].astype(str).str.extract(r'(\bLX|EX|Sport|Limited|Laramie|Denali|Platinum|XLT\b)', expand=False).fillna('Other')

### ✅ Optional: Drop Less Useful Columns

These are mostly text-based or IDs:

df.drop(columns=['name', 'description', 'engine'], inplace=True)

### ✅ Final Touch: Convert New Categorical Columns

df = pd.get\_dummies(df, columns=['cyl\_category', 'trim\_level'], drop\_first=True)

**✅ Why Scaling?**

Scaling is crucial for models that are sensitive to feature magnitudes, like:

* **Linear Regression**
* **KNN**
* **SVM**
* **Gradient Descent–based models**

Not strictly necessary for:

* **Tree-based models** (Random Forest, XGBoost, LightGBM)

**🎯 Common Scalers**

| **Scaler** | **Use Case** | **Description** |
| --- | --- | --- |
| StandardScaler | Most common | Mean = 0, Std = 1 |
| MinMaxScaler | Range-based models (e.g., Neural Nets) | Scales between 0 and 1 |
| RobustScaler | Outlier-resistant | Uses median and IQR |

**✅ Let’s Scale Numeric Features**

Here’s code for **Standard Scaling** (the most commonly used):

from sklearn.preprocessing import StandardScaler

# Select only numeric features for scaling

numeric\_cols = df.select\_dtypes(include=['int64', 'float64']).columns.tolist()

# Optionally exclude the target column (e.g., 'price') from scaling

numeric\_cols.remove('price')

# Initialize and fit the scaler

scaler = StandardScaler()

df[numeric\_cols] = scaler.fit\_transform(df[numeric\_cols])

# ✅ Done: Scaled features are now normalized

df.head()

**🔄 If You Prefer MinMaxScaler:**

from sklearn.preprocessing import MinMaxScaler

scaler = MinMaxScaler()

df[numeric\_cols] = scaler.fit\_transform(df[numeric\_cols])

Here's a full set of EDA visualization code blocks for your **Jupyter Notebook**, covering:

* ✅ Univariate Analysis
* ✅ Bivariate Analysis
* ✅ Multivariate Analysis

We'll use **Matplotlib**, **Seaborn**, and **Pandas**.

## 📦 Step 1: Import Libraries

import matplotlib.pyplot as plt

import seaborn as sns

import pandas as pd

# Optional: for cleaner plots

sns.set(style="whitegrid")

%matplotlib inline

## 📊 1️⃣ Univariate Analysis

### 🔸 Numeric Columns

# Histogram of price

plt.figure(figsize=(8, 5))

sns.histplot(df['price'], bins=30, kde=True)

plt.title('Distribution of Vehicle Prices')

plt.xlabel('Price')

plt.ylabel('Count')

plt.show()

### 🔸 Categorical Columns

# Count plot of body type (if not one-hot encoded already)

plt.figure(figsize=(10, 5))

sns.countplot(x='body', data=df\_original) # use original if you dropped this in encoding

plt.title('Count of Vehicles by Body Type')

plt.xticks(rotation=45)

plt.show()

📝 If you have one-hot encoded body types, you can do .sum() to visualize count:

df.filter(like='body\_').sum().sort\_values().plot(kind='barh', figsize=(8,5), title="Count of Body Types (One-hot)")

## 📊 2️⃣ Bivariate Analysis

### 🔹 Price vs Categorical (Boxplot)

plt.figure(figsize=(10, 5))

sns.boxplot(x='fuel', y='price', data=df\_original) # if fuel wasn't one-hot encoded

plt.title('Price by Fuel Type')

plt.xticks(rotation=45)

plt.show()

### 🔹 Price vs Numeric (Scatterplot)

plt.figure(figsize=(8, 5))

sns.scatterplot(x='mileage', y='price', data=df)

plt.title('Price vs Mileage')

plt.xlabel('Mileage')

plt.ylabel('Price')

plt.show()

## 📊 3️⃣ Multivariate Analysis

### 🔸 Correlation Heatmap

plt.figure(figsize=(14, 10))

corr = df.corr()

sns.heatmap(corr, annot=False, cmap='coolwarm', linewidths=0.5)

plt.title('Correlation Heatmap')

plt.show()

### 🔸 Pairplot (subset of features)

sns.pairplot(df[['price', 'mileage', 'vehicle\_age', 'cylinders']])

plt.suptitle('Pairplot of Selected Features', y=1.02)

plt.show()

### ✅ Optional: Grouped Barplot

plt.figure(figsize=(12, 5))

grouped = df\_original.groupby('make')['price'].mean().sort\_values(ascending=False).head(10)

sns.barplot(x=grouped.index, y=grouped.values)

plt.xticks(rotation=45)

plt.title('Top 10 Car Makes by Average Price')

plt.ylabel('Average Price')

plt.show()

Let’s now perform **outlier detection** and **skewness analysis** — these steps help you improve data quality and model performance, especially for regression tasks like vehicle price prediction.

## 🔍 Step 1: Outlier Detection (Multiple Methods)

We'll apply **4 methods**:

### 1️⃣ Boxplot (Visual method)

### 2️⃣ IQR (Interquartile Range Rule)

### 3️⃣ Z-Score

### 4️⃣ Isolation Forest (advanced ML-based outlier detection)

## 📦 Prerequisite: Identify numeric columns

# Select only numeric columns

numeric\_cols = df.select\_dtypes(include=['int64', 'float64']).columns.tolist()

## 🔸 1. Boxplot (Visual)

import matplotlib.pyplot as plt

import seaborn as sns

for col in numeric\_cols:

plt.figure(figsize=(6, 2))

sns.boxplot(x=df[col])

plt.title(f'Boxplot for {col}')

plt.show()

## 🔸 2. IQR Method (Programmatic)

# Detect outliers based on IQR for each column

outlier\_iqr = {}

for col in numeric\_cols:

Q1 = df[col].quantile(0.25)

Q3 = df[col].quantile(0.75)

IQR = Q3 - Q1

lower\_bound = Q1 - 1.5 \* IQR

upper\_bound = Q3 + 1.5 \* IQR

outliers = df[(df[col] < lower\_bound) | (df[col] > upper\_bound)]

outlier\_iqr[col] = len(outliers)

# Show number of outliers detected by IQR

pd.Series(outlier\_iqr).sort\_values(ascending=False)

## 🔸 3. Z-Score Method

from scipy.stats import zscore

import numpy as np

z\_outlier\_count = {}

# Z-score threshold

threshold = 3

for col in numeric\_cols:

z\_scores = zscore(df[col])

outliers = df[np.abs(z\_scores) > threshold]

z\_outlier\_count[col] = outliers.shape[0]

# Show Z-score outliers

pd.Series(z\_outlier\_count).sort\_values(ascending=False)

## 🔸 4. Isolation Forest (ML-Based)

from sklearn.ensemble import IsolationForest

iso = IsolationForest(contamination=0.05, random\_state=42)

outliers = iso.fit\_predict(df[numeric\_cols])

# Add a column showing outliers (-1 = outlier, 1 = normal)

df['outlier\_iso'] = outliers

# Count how many outliers were found

df['outlier\_iso'].value\_counts()

## 📈 Step 2: Skewness Check

# Check skewness of each numeric column

skewness = df[numeric\_cols].skew().sort\_values(ascending=False)

print("Highly skewed features (>|1|):\n")

print(skewness[abs(skewness) > 1])

### Optional: Visualize Skewed Distributions

skewed\_cols = skewness[abs(skewness) > 1].index.tolist()

for col in skewed\_cols:

plt.figure(figsize=(6, 3))

sns.histplot(df[col], kde=True)

plt.title(f'Distribution of {col} (Skewness = {skewness[col]:.2f})')

plt.show()

## 💡 Want to Fix Skewed Data?

Use **log or square root transformation**:

df['price\_log'] = np.log1p(df['price']) # log(1 + x) to avoid log(0)

Let me know if you’d like to:

* Remove outliers based on any of the above methods
* Fix highly skewed features
* Move on to model training!

## 🎯 What is Kurtosis?

* **Kurtosis** measures the **tailedness** of a distribution (how extreme the outliers are).
* It complements **skewness** which measures asymmetry.

### 📊 Types of Kurtosis:

| **Type** | **Value Range** | **Description** |
| --- | --- | --- |
| **Mesokurtic** | ≈ 3 | Normal distribution |
| **Leptokurtic** | > 3 | Heavy tails (more extreme outliers) |
| **Platykurtic** | < 3 | Light tails (fewer outliers) |

## ✅ How to Calculate Kurtosis in Your DataFrame

# Import from scipy

from scipy.stats import kurtosis

# Calculate kurtosis for numeric columns

kurt = df[numeric\_cols].kurtosis().sort\_values(ascending=False)

print("Kurtosis of numeric features:\n")

print(kurt)

## ✅ Optional: Highlight Extreme Kurtosis

print("Highly peaked distributions (kurtosis > 3):\n")

print(kurt[kurt > 3])

## 📈 Optional: Visualize High Kurtosis

high\_kurt\_cols = kurt[kurt > 3].index.tolist()

for col in high\_kurt\_cols:

plt.figure(figsize=(6, 3))

sns.histplot(df[col], kde=True)

plt.title(f'{col} (Kurtosis = {kurt[col]:.2f})')

plt.show()

# This is How I pushed my code to already existing git hub :

### ✅ Steps to Add and Push Your New Folder via CMD

Assuming you're already inside your project folder:

cd path\to\Machine-Learning

### 🔹 1. **Check Git status**

git status

This shows you the new untracked EDA/ folder and its files.

### 🔹 2. **Stage the new folder**

git add notebooks/EDA

Or to stage **everything**, just:

git add .

### 🔹 3. **Commit the changes**

git commit -m "Added EDA folder with analysis notebooks"

### 🔹 4. **Push to GitHub**

If you're already connected to a remote GitHub repo:

git push origin main

(replace *main* with *master* or your branch name if different)

### 💡 Optional: If You Haven’t Set the Remote Yet

git remote add origin https://github.com/your-username/Machine-Learning.git

git push -u origin main

### 🧠 Recap of All Commands:

cd path\to\Machine-Learning

git status

git add .

git commit -m "Added EDA folder with analysis notebooks"

git push origin main

Let me know if you want a shortcut script or you're getting any errors during push!

Great — that error is clear and easy to fix!

### ✅ Solution: Initialize Git First

In the folder C:\Users\USER\Documents\Brototype\Brocamp, run:

git init

This command **initializes an empty Git repository** in your local folder by creating a hidden .git directory.

### 🔁 Then proceed with the steps:

#### 1. **Initialize Git**

git init

#### 2. **Add remote origin**

git remote add origin https://github.com/exassaro/Datascience\_Workshop.git

#### 3. **Add your files**

git add .

#### 4. **Commit your changes**

git commit -m "Initial commit for datascience workshop"

#### 5. **Push to GitHub**

git branch -M main

git push -u origin main

### ✅ Result:

Your local folder Brocamp will now be connected to the GitHub repo Datascience\_Workshop, and your files will be uploaded without renaming the local folder.