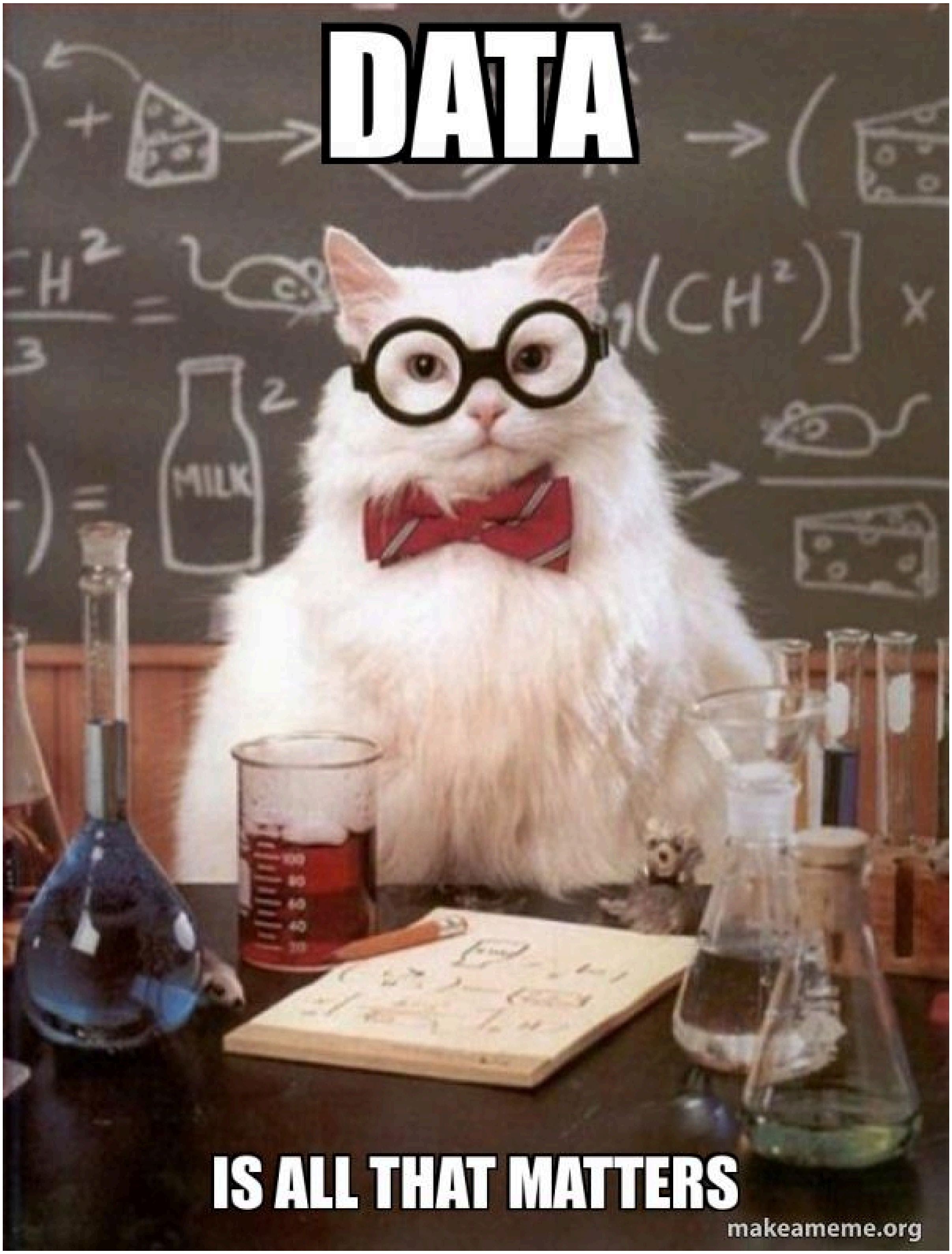
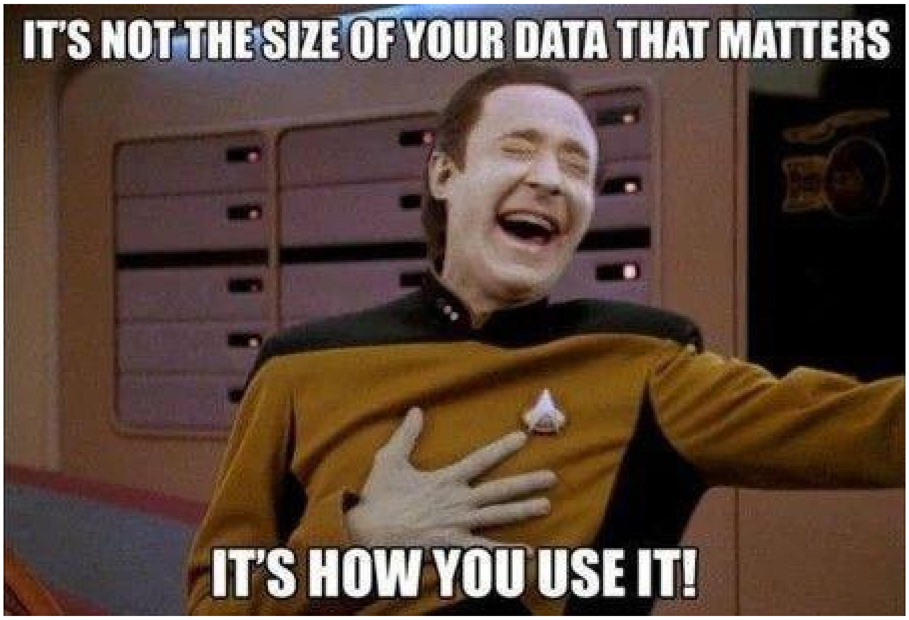

DATA
IS ALL THAT MATTERS
makeameme.org


DATA
DATA EVERYWHERE



1. Understanding the Dataset

The first step in Exploratory Data Analysis (EDA) is to understand the dataset. This helps in knowing the structure, content, and quality of the data before performing any analysis or modeling.

- 🔹 Key Aspects of Understanding the Dataset
  - 1️⃣ **Load the dataset** (`pd.read_csv()`)
  - 2️⃣ **Check the shape** (`df.shape`)
  - 3️⃣ **Check column names** (`df.columns`)
  - 4️⃣ **Identify data types** (`df.dtypes`)
  - 5️⃣ **Check for missing values** (`df.isnull().sum()`)
  - 6️⃣ **Check for duplicates** (`df.duplicated().sum()`)

7️⃣ **Get summary statistics** (`df.describe()`)

8️⃣ **Check unique values** (`df['col'].unique()`)

9️⃣ **Check data balance (for classification)**

(`df['target'].value_counts()`)

This step **lays the foundation** for deeper **EDA**, including visualization, outlier detection, correlation analysis, and feature engineering. 🚀

2. Handling Missing Values

Missing values in a dataset can cause problems in analysis and modeling. How we handle missing data depends on the type of data and the reasons why values are missing.

1️⃣ **Identify Missing Values**

- Use `.isnull().sum()` to check missing values.
- Calculate missing percentage using `(df.isnull().sum() / len(df)) * 100`.

2️⃣ **Understand the Type of Missing Data**

- **MCAR (Missing Completely at Random)** – No pattern.

- **MAR (Missing at Random)** – Missingness depends on other variables.
- **MNAR (Missing Not at Random)** – Missing for a specific reason.

3️⃣ Handle Missing Values

(A) Removing Missing Values

- **Drop Rows** → `df.dropna()` (If missing values are few).
- **Drop Columns** → `df.drop(columns=['Column_Name'])` (If >50% missing).

(B) Imputing Missing Values

1. **Numerical Data**
    - **Mean** → `df['Column'].fillna(df['Column'].mean())` (for normal distribution).
    - **Median** → `df['Column'].fillna(df['Column'].median())` (for skewed data).

2. **Categorical Data**
    - **Mode** → `df['Column'].fillna(df['Column'].mode(`

`)[0])` (for categorical values).

3. **Time-Series Data**
   - **Forward Fill** → `df.fillna(method='ffill')` (uses previous value).
   - **Backward Fill** → `df.fillna(method='bfill')` (uses next value).

4. **Machine Learning Imputation**
   - **KNN Imputation** → `KNNImputer(n_neighbors=5)` (predicts missing values).

4️⃣ Verify Handling of Missing Values

1. Run `.isnull().sum()` again to ensure no missing values remain.

3. Removing Duplicates

Duplicates in a dataset can cause bias in analysis and incorrect model predictions. Removing them ensures data consistency and accuracy

1️⃣ Identifying Duplicate Rows

- Use `df.duplicated().sum()` to count duplicate rows.

- Use `df[df.duplicated()]` to display duplicate rows.

2️⃣ Removing Duplicate Rows

- **Remove all duplicates (keep first occurrence)**
  ```
  df_cleaned = df.drop_duplicates()
  ```
- **Remove all duplicates (keep last occurrence)**
  ```
  df_cleaned = df.drop_duplicates(keep='last')
  ```
- **Remove all duplicates (keep none, strict removal)**
  ```
  df_cleaned = df.drop_duplicates(keep=False)
  ```

3️⃣ Removing Duplicates Based on Specific Columns

- Remove duplicates based on selected columns:
  ```
  df_cleaned =
  df.drop_duplicates(subset=['Column1',
  'Column2'])
  ```

4️⃣ Removing Duplicates While Ignoring NaN Values

- Ignore index while removing duplicates:
  ```
  df_cleaned =
  df.drop_duplicates(ignore_index=True)
  ```

5️⃣ Verify Duplicate Removal

- Use `df_cleaned.duplicated().sum()` to ensure duplicates are removed.

4. Generating Summary Statistics

Summary statistics help understand the distribution, central tendency, and variability of the dataset. These insights guide data preprocessing, feature engineering, and model selection.

Before interpreting summary statistics, check for missing values and data types.

1️⃣ Generating Basic Summary Statistics

The `.describe()` method provides key statistical metrics.

📌 **Output Includes:**

- **count** → Total non-null values in each column.
- **mean** → Average value.
- **std** → Standard deviation (spread of values).
- **min** → Minimum value.
- **25% (Q1)** → First quartile (25th percentile).
- **50% (Q2)** → Median (50th percentile).
- **75% (Q3)** → Third quartile (75th percentile).

- **max** → Maximum value.

2️⃣ Summary Statistics for Categorical Data

Use `.describe(include='object')` for categorical columns.

📌 **Output Includes:**

- **count** → Total non-null values.
- **unique** → Number of unique values.
- **top** → Most frequent category.
- **freq** → Frequency of the top category.

✅ **Example:** If `Gender` column has "Male" and "Female," it will show which occurs most frequently.

5. Univariate Analysis

Univariate analysis involves analyzing a **single** variable (column) at a time to understand its distribution, central tendency, spread, and presence of outliers.

It helps answer:

✅ How is the data distributed?

✅ Are there missing values or outliers?

✅ What are the key statistics (mean, median, mode)?

1️⃣ **Numerical Data Analysis**

✔ **Summary Statistics** → `df['Column'].describe()`

✔ **Histogram** → `plt.hist(df['Column'])`

✔ **Box Plot** → `sns.boxplot(x=df['Column'])`

✔ **Skewness & Kurtosis** → `df['Column'].skew()`, `df['Column'].kurt()`

✔ **Detect Outliers (IQR Method)** → `Q1 - 1.5*IQR`, `Q3 + 1.5*IQR`

2️⃣ **Categorical Data Analysis**

✔ **Frequency Count** → `df['CategoryColumn'].value_counts()`

✔ **Bar Plot** → `sns.countplot(x=df['CategoryColumn'])`

✔ **Pie Chart** → `df['CategoryColumn'].value_counts().plot.pie()`

6. Bivariate Analysis

Bivariate analysis examines the relationship between two variables to identify patterns, correlations, and dependencies.

It helps answer:
✅ **How does one variable affect another?**
✅ **Is there a correlation between variables?**
✅ **Are categories related in any way?**

Bivariate analysis helps in understanding how two variables are related, leading to better insights & feature selection in data science projects!

1. Bivariate Analysis for Numerical vs. Numerical Data

A. Correlation Analysis

Measures how two numerical variables are related.

- **Positive correlation** → Both increase together
- **Negative correlation** → One increases, the other decreases
- **No correlation** → No relationship

✔ Helps identify strongly related variables.

✔ Useful for feature selection in ML.

1️⃣ Correlation Matrix & Heatmap

✔ Helps identify strongly related variables.

✔ Useful for feature selection in ML.

2️⃣ Scatter Plot (Visualizing Relationships)

A scatter plot helps in identifying trends between two numerical variables.

✔ If points form a line → Strong correlation

✔ If points are scattered → Weak or no correlation

3️⃣ Regression Plot (Best-Fit Line)

Shows the trend between two numerical variables.

✔ Steeper line = Stronger relationship

✔ Helps in predictive modeling

2. Bivariate Analysis for Categorical vs. Numerical Data

A. Box Plot (Comparing Distributions)

Used to compare distributions of a numerical variable across different categories.

✔ Median comparison across categories

✔ Identifies outliers in each category

B. Violin Plot (Density & Distribution)

A violin plot combines a box plot & KDE (Kernel Density Estimation).

✔ Shows data distribution more clearly.

C. Bar Plot (Mean Value per Category)

Shows the mean of a numerical variable for each category.

✔ Comparing averages across groups.

3. Bivariate Analysis for Categorical vs. Categorical Data

A. Crosstab (Frequency Table)

Shows the count of categories in relation to another categorical variable

✔ Analyzing relationships between categorical variables.

B. Grouped Bar Plot

Visualizes relationships between two categorical variables.

✔ Helps detect imbalances in categorical data.

C. Chi-Square Test (Statistical Dependency)

Used to check if two categorical variables are dependent.

7. Detecting Outliers

What is an Outlier?

- An outlier is a data point that deviates significantly from the rest of the data.

- It can be due to errors, natural variability, or rare events.

Why Detect Outliers?

✔ Prevents **skewed statistical analysis**.
✔ Improves **model accuracy**.
✔ Identifies **data quality issues or anomalies**.

Methods to Detect Outliers

1️⃣**Visual Methods**

📌 **Helps in quick identification**

✅ **Box Plot (IQR Method)** → Outliers appear as points outside the whiskers.
✅ **Scatter Plot** → Helps identify extreme values.
✅ **Histogram & KDE Plot** → Shows distribution and extreme values.

2️⃣Statistical Methods

📌 Mathematically identifies outliers

✅ Interquartile Range (IQR) Method → Uses Q1 & Q3 to detect extreme values.
✅ Z-Score Method → Outliers have |Z| > 3 (for normally distributed data).

3️⃣**Machine Learning Methods**

📌 **Useful for high-dimensional data & anomalies**

✅ **Isolation Forest** → Detects outliers by isolating observations.
✅ **DBSCAN (Density-Based Clustering)** → Identifies anomalies as noise points.

Choosing the Right Method

Method	Best For
IQR Method	Skewed data
Z-Score	Normally distributed data
Modified Z-Score	Skewed data
Box Plot	Quick visualization
Scatter Plot	Spotting extreme values
Isolation Forest	High-dimensional data
DBSCAN	Clustered datasets

Next Steps After Outlier Detection

✅ **Remove** (If caused by errors).
✅ **Cap or Transform** (If needed for model accuracy).
✅ **Investigate Further** (If outliers contain useful insights).

8. Feature Engineering

Feature engineering is the process of transforming raw data into meaningful features that improve machine learning model performance. It involves **creating, modifying, or selecting** features to enhance predictive power.

Step	Techniques
Handling Missing Values	Imputation (mean, median, mode), indicator columns
Handling Outliers	IQR, Z-score, transformations (log, sqrt)
Feature Creation	Time-based, ratio, polynomial features
Encoding Categorical Data	Label Encoding, One-Hot Encoding, Target Encoding
Feature Scaling	Standardization, Min-Max Scaling, Robust Scaling
Feature Selection	Correlation analysis, Low variance filtering, Recursive Feature Elimination
Dimensionality Reduction	PCA, t-SNE, UMAP

9. Analyzing Correlation

Correlation measures the **strength and direction** of a relationship between two variables. It helps identify dependencies and eliminate redundant features.

Step	Purpose
Pearson Correlation	Measures linear relationships between continuous variables.
Spearman Correlation	Used for monotonic relationships (even if non-linear).

Kendall’s Tau	Works well for ordinal variables and small datasets.
Heatmap	Best for a quick correlation overview.
Pair Plot	Helps in visualizing relationships between features.
Feature Selection	Drop highly correlated features to avoid redundancy & multicollinearity.

10. Transforming and Scaling Data

Machine learning models perform better when features have a consistent scale and distribution. Transformation and scaling ensure:

✔ **Improved Model Performance** – Reduces bias toward large values.

✔ **Faster Convergence** – Optimizers like gradient descent work better.

✔ **Better Interpretability** – Some models assume normally distributed data.

Technique	Effect	Best Use Case
Min-Max Scaling	Scales between 0 and 1	When data has fixed bounds.
Standardization (Z-score)	Mean = 0, Std Dev = 1	Data is normally distributed.
Robust Scaling	Uses median and IQR	Data contains outliers.
Log Transformation	Reduces right skewness	Highly skewed positive data.

Power Transformation	Makes data normal-like	When data isn't normally distributed.

11. Reducing Dimensionality

High-dimensional data can lead to:
✔ **Curse of Dimensionality** – More dimensions → Sparse data → Poor model performance.
✔ **Longer Training Time** – More features increase computational cost.
✔ **Overfitting** – Too many irrelevant features can make models complex.

Reducing dimensionality **improves efficiency** while **retaining key information**.

Method	Type	Best Use Case
Feature Selection	Removes irrelevant features	When some features add no value.
PCA	Transforms data into principal components	When features are correlated.
LDA	Maximizes class separability	Classification tasks.
t-SNE	Preserves local structure in 2D/3D	Data visualization.

Autoencoders	Learns compressed representations	Deep learning & non-linear data.