Exploratory Data Analysis (EDA)

🔍 1. What is Exploratory Data Analysis (EDA)?

Definition:
EDA is the process of analyzing and visualizing datasets to summarize their main characteristics, often before applying any machine learning models.

Purpose:

- Understand data structure, distributions, and relationships.
- Identify trends, outliers, and anomalies.
- Guide feature engineering and data preprocessing.
- Generate hypotheses for modeling.

Core Tools:

- `Pandas` for data manipulation.
- `Matplotlib` & `Seaborn` for visual analysis.

🧰 2. Libraries for EDA

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

- **Pandas**: Fast, powerful, and flexible data analysis library.
- **NumPy**: Used for numeric operations.
- **Matplotlib**: Low-level visualization; highly customizable.
- **Seaborn**: Built on top of Matplotlib for high-level statistical graphics.

📥 3. Data Loading and Initial Exploration

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

Key Functions

Function	Description
`df.head()`	View first 5 rows
`df.tail()`	View last 5 rows
`df.shape`	Tuple of (rows, columns)
`df.columns`	List column names
`df.info()`	Summary with dtypes and nulls
`df.describe()`	Statistical summary of numeric fields

Initial data exploration provides a bird's-eye view of the dataset, allowing us to plan cleaning and transformations.

---

4. Data Cleaning

What is Data Cleaning?

Data Cleaning is the process of fixing or removing incorrect, corrupted, incorrectly formatted, duplicate, or incomplete data within a dataset.

4.1 Handling Missing Values

```
df.isnull().sum()             # Count missing values
df.dropna()                   # Drop rows with missing values
df.fillna(0)                  # Replace missing with 0
df.fillna(method='ffill')     # Forward fill
```

- Missing values can distort data insights.
- Filling them depends on the type (mean, median, mode for numeric; 'Unknown' for category).

4.2 Handling Duplicates

```
df.duplicated().sum()
df.drop_duplicates(inplace=True)
```

Duplicates can cause skew in statistical analysis. Removing them improves reliability.

4.3 Data Type Conversion

```
df['date'] = pd.to_datetime(df['date'])
df['cat'] = df['cat'].astype('category')
```

Correct data types save memory, improve performance, and unlock specific operations (e.g., datetime indexing).

---

5. Univariate Analysis (One Variable)

Goal: Understand the distribution, range, frequency, and nature of individual features.

5.1 Numerical Variables

Histogram

```
sns.histplot(df['age'], kde=True)
```

- Shows frequency distribution.
- Use KDE to show density curve.

Boxplot

```
sns.boxplot(x=df['salary'])
```

- Shows median, IQR, and outliers.

KDE Plot

```
sns.kdeplot(df['score'])
```

- Smooth curve representing data density.

5.2 Categorical Variables

Countplot

```
sns.countplot(x='gender', data=df)
```

Pie Chart

```
df['gender'].value_counts().plot.pie(autopct='%1.1f%%')
```

Univariate plots help you detect skewed distributions, imbalances in categories, and plan binning strategies.

---

6. Bivariate Analysis (Two Variables)

Goal: Study relationships between pairs of variables.

6.1 Numerical vs Numerical

Scatter Plot

```
sns.scatterplot(x='age', y='salary', data=df)
```

Correlation Heatmap

```
sns.heatmap(df.corr(), annot=True, cmap='coolwarm')
```

Regression Plot

```
sns.regplot(x='study_hours', y='score', data=df)
```

- Identify linear/non-linear relationships.
- High correlation could indicate redundancy.

6.2 Categorical vs Numerical

Boxplot

```
sns.boxplot(x='department', y='salary', data=df)
```

Bar Plot

```
sns.barplot(x='gender', y='score', data=df)
```

Violin Plot

```
sns.violinplot(x='team', y='experience', data=df)
```

- Helpful for comparing distributions across groups.
- Barplot shows aggregated means by default.

6.3 Categorical vs Categorical

Countplot with Hue

```
sns.countplot(x='department', hue='gender', data=df)
```

Crosstab

```
pd.crosstab(df['role'], df['education'])
```

---

7. Multivariate Analysis

Goal: Explore interactions among three or more variables.

Pairplot

```
sns.pairplot(df, hue='result')
```

LM Plot (with Hue)

```
sns.lmplot(x='age', y='score', hue='gender', data=df)
```

Heatmap of Features

```
sns.heatmap(df.corr(), annot=True)
```

---

8. GroupBy and Aggregation

What is `groupby()` in Pandas?

The `groupby()` function in pandas is used to **split** the data into groups based on some criteria, **apply** a function to each group, and then **combine** the results.

This is often referred to as the **Split-Apply-Combine** strategy:

1. **Split** the data into groups based on one or more keys.
2. **Apply** a function (aggregation, transformation, or filtration).
3. **Combine** the results into a data structure.

```
df.groupby(by=None, axis=0, level=None, as_index=True, sort=True,
group_keys=True, dropna=True)
```

Parameters:

- **by**: column name(s) or array-like – the key(s) to group by.
- **axis**: 0 or 1 – default is 0 (rows).
- **as_index**: if True, the group labels become the index (default=True).
- **sort**: whether to sort the group keys (default=True).
- **dropna**: if True, NaN values in group keys are excluded (Pandas 1.1.0+).

Example DataFrame:

```
import pandas as pd  data = {     'Department': ['HR', 'IT', 'HR', 'Finance',
'IT', 'Finance'],     'Employee': ['Alice', 'Bob', 'Charlie', 'David', 'Eve',
'Frank'],     'Salary': [50000, 60000, 52000, 58000, 62000, 61000] }

df = pd.DataFrame(data)
```

Group by Department and get mean salary:

```
python
grouped = df.groupby('Department')['Salary'].mean() print(grouped)`
```

Output:

Multiple Aggregations

You can apply multiple aggregation functions to each group:

```
df.groupby('Department')['Salary'].agg(['mean', 'max', 'min'])`
```

Group by Multiple Columns

```
df.groupby(['Department', 'Employee'])['Salary'].sum()`
```

Common Aggregation Functions:

Function	Description
`sum()`	Sum of values
`mean()`	Mean of values
`size()`	Size of each group
`count()`	Non-null value count
`std()`	Standard deviation

Function	Description
`min()`	Minimum value
`max()`	Maximum value
`agg()`	Apply multiple aggregations
`transform()`	Transform each group
`filter()`	Filter groups by condition

Multiple Aggregations

```
df.groupby('department')['salary'].agg(['mean', 'median', 'count'])
```

Pivot Table

```
pd.pivot_table(df, index='region', columns='year', values='sales',
aggfunc='sum')
```

- `groupby()` is essential for summarizing subsets of data.
- Helps identify patterns across groups (e.g., sales per region).

---

9. Feature Engineering for EDA

- **Binning**: Convert continuous to categorical (e.g., age groups)
- **Datetime**: Extract month, year, weekday
- **Length-based features**: Text, reviews
- **Ratios**: Price per unit, Engagement per post
- **Interaction features**: Multiplication/division of numeric features

---

10. EDA Plot Summary Table

Plot	Best For	Lib
Histogram	Distribution (Univariate)	Seaborn
Boxplot	Outliers, Spread	Seaborn

Plot	Best For	Lib
Countplot	Frequency of categories	Seaborn
Scatterplot	Relation between numerics	Seaborn
Heatmap	Correlation matrix	Seaborn
Violin Plot	Distribution + IQR + KDE	Seaborn
Pairplot	Multivariate analysis	Seaborn
Lineplot	Time series	Seaborn
Crosstab	Category vs category	Pandas