Exploratory Data Analysis Overview

Type	@datasciencebrain

1. Introduction to EDA

What is Exploratory Data Analysis?

Exploratory Data Analysis (EDA) is a crucial step in the data science pipeline that involves summarizing the main characteristics of a dataset, often using visual methods. It helps in understanding the structure of data, identifying patterns, spotting anomalies, and selecting relevant features for modeling.

Objectives of EDA

- Understand the dataset's structure and quality
- Detect missing values, outliers, and inconsistencies
- Identify important features and relationships
- Guide feature selection and engineering
- Assess data distribution and statistical properties
- Validate assumptions for model building

Types of EDA

EDA can be broadly categorized into:

1. **Univariate Analysis** – Examining each variable independently
2. **Bivariate Analysis** – Exploring relationships between two variables
3. **Multivariate Analysis** – Studying interactions among multiple variables

2. Understanding the Dataset

Before conducting EDA, it is essential to load and inspect the dataset.

Loading Data

```
import pandas as pd
import numpy as np

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

Inspecting Data Structure

```
# Display the first few rows
data.head()

# Check the shape (rows, columns)
data.shape

# Get column names
data.columns

# Data types of each column
data.dtypes
```

3. Handling Missing Values

Missing values can impact analysis and model performance. Identifying and handling them is critical.

Identifying Missing Values

```
# Count missing values per column
data.isnull().sum()
```

```
# Percentage of missing values
data.isnull().mean() * 100
```

Handling Missing Values

- **Drop missing values**: If the missing data is minimal.

  ```
  data.dropna(inplace=True)
  ```

- **Impute missing values**:

  ```
  # Fill with mean (for numerical data)
  data.fillna(data.mean(), inplace=True)

  # Fill with mode (for categorical data)
  data.fillna(data.mode().iloc[0], inplace=True)
  ```

4. Descriptive Statistics

Descriptive statistics help in summarizing numerical features.

Statistical Summary

```
# Summary statistics for numerical columns
data.describe()

# Summary for categorical columns
data.describe(include=["object"])
```

5. Univariate Analysis

Univariate analysis examines the distribution of a single variable.

Visualizing Numerical Data

```
import matplotlib.pyplot as plt
import seaborn as sns

# Histogram
sns.histplot(data['column_name'], bins=30, kde=True)
plt.show()

# Box plot
sns.boxplot(x=data['column_name'])
plt.show()
```

Visualizing Categorical Data

```
# Bar plot
sns.countplot(x=data['categorical_column'])
plt.show()
```

6. Bivariate Analysis

Bivariate analysis helps in understanding relationships between two variables.

Numerical vs Numerical

```
# Scatter plot
sns.scatterplot(x=data['feature1'], y=data['feature2'])
plt.show()

# Correlation matrix
corr_matrix = data.corr()
sns.heatmap(corr_matrix, annot=True, cmap="coolwarm")
plt.show()
```

Categorical vs Numerical

```
# Boxplot
sns.boxplot(x=data['categorical_column'], y=data['numerical_column'])
plt.show()
```

Categorical vs Categorical

```
# Crosstab analysis
pd.crosstab(data['cat1'], data['cat2'])
```

7. Outlier Detection and Treatment

Outliers can distort analysis and model performance.

Identifying Outliers

```
# Box plot method
sns.boxplot(x=data['numerical_column'])
plt.show()
```

Treating Outliers

- **Capping and Flooring**: Setting upper and lower limits

  ```
  upper_limit = data['numerical_column'].quantile(0.99)
  lower_limit = data['numerical_column'].quantile(0.01)
  data['numerical_column'] = np.clip(data['numerical_column'], lower_limit, upper_limit)
  ```

- **Removing Outliers**

  ```
  data = data[(data['numerical_column'] > lower_limit) & (data['numerical_column'] < upper_limit)]
  ```

8. Feature Engineering

Feature engineering involves creating new features from existing data to enhance model performance.

Common Feature Engineering Techniques

- **Encoding categorical variables**

```
# One-hot encoding
data = pd.get_dummies(data, columns=['categorical_column'], drop_first=
True)
```

- **Transformations** (Log, Square Root, etc.)

```
data['transformed_column'] = np.log1p(data['numerical_column'])
```

9. Data Normalization and Standardization

Scaling numerical features ensures that models do not assign disproportionate importance to certain features.

```
from sklearn.preprocessing import StandardScaler, MinMaxScaler

# Standardization (mean=0, variance=1)
scaler = StandardScaler()
data['scaled_column'] = scaler.fit_transform(data[['numerical_column']])

# Normalization (scaling between 0 and 1)
scaler = MinMaxScaler()
data['normalized_column'] = scaler.fit_transform(data[['numerical_column']])
```

10. Conclusion

EDA is a fundamental step in the data science workflow that provides insights into the dataset before modeling. It helps in making informed decisions regarding

feature selection, data preprocessing, and model optimization. By applying statistical summaries, visualizations, and transformations, one can enhance the quality of data and improve the performance of predictive models.