EDA
(EXPLORATORY DATA ANALYSIS)

OUTLINES

Introduction to Exploratory Data Analysis (EDA)

Steps in EDA

Data Types

Data Transformation

Introduction to Missing data, handling missing data

Data Visualization using Matplotlib, Seaborn

INTRODUCTION

- **EDA** is a crucial step in the data analysis process.
- It involves investigating datasets to summarize their main characteristics, often using visual methods. It uses graphs and stats to find patterns and weird things in the data without guessing or proving anything.
- EDA helps in understanding the data structure, detecting anomalies, discovering patterns, and checking assumptions, all of which inform further analysis.
- By checking each piece of data and how they relate, EDA helps us understand what's in the data. As well as decide what to do next.
- It's like a guide that helps us make sense of all the information we have, so we can make smart decisions.

STEPS IN EDA

1- Understand the Data Structure
- Data Collection
-Data Description

2-Data Cleaning
-Handling Missing Values
-Removing Duplicates
-Data Correction
-Outlier Detection

3-Data Transformation
-Feature Engineering
-Normalization/Standardization
-Encoding Categorical Variables

4-Data Visualization

-Univariate Analysis-histograms, box plots, and density plots

-Bivariate Analysis-scatter plots, correlation matrices, and line charts

-Multivariate -Analysis-pair plots and heat maps

5-Data Summarization-

Descriptive Statistics- mean, mode, median

Correlation Analysis- correlation, covariance

Distribution Analysis- normal distribution

6-Data Interpretation

Pattern Recognition

Hypothesis Testing

7-Report Findings-Summarize insights and findings from the EDA process.

DATA TYPES

1.Numerical Data-

- Continuous
- Discrete

2.Categorical Data

- Nominal:
- ordinal

3.Binary Data

4.Text Data

5.Time Series Data

Date/Time Data

DATA TRANSFORMATION

DATA TRANSFORMATION

- Data transformation is a crucial step in the Exploratory Data Analysis (EDA) process.
- It involves converting data into a suitable format or structure to improve its quality and usability.
- This can help to uncover hidden patterns, identify anomalies, and ensure that the data is ready for modeling.

BENEFITS OF DATA TRANSFORMATION

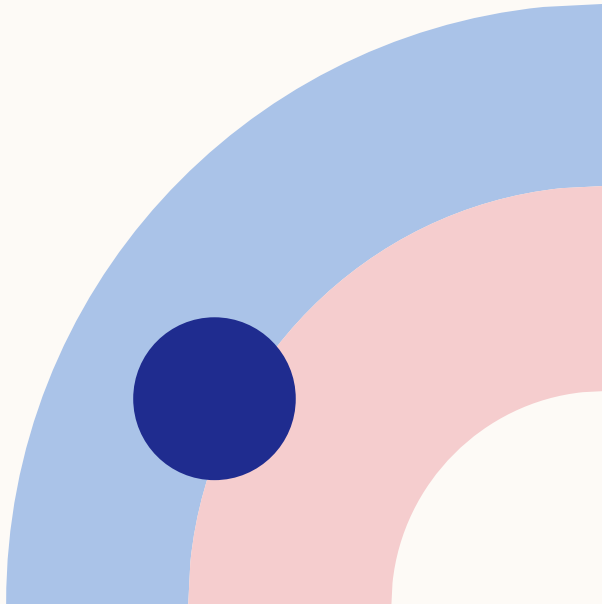
- **Improved Model Performance**
- **Enhanced Data Visualizations**
- **Handling Outliers**
- **Better Convergence-**Faster Training, Stability, better Performance
- **Dimensionality Reduction**
- **Better Insights from Feature Engineering**

DISADVANTAGES

- Information Loss
- Risk of Overfitting
- Data Leakage
- Increased Complexity
- Assumption Violation

TRANSFORMATION TECHNIQUES

1. **Normalization and Standardization**
2. **Log Transformation**
3. 3. **Box-Cox Transformation**
4. **Square Root and Cube Root Transformations**
5. **Categorical Encoding**
6. **Binning or Discretization**
7. **Handling Missing Values**
8. **Scaling**
9. **Feature Engineering**
10. **Dimensionality Reduction**



NORMALIZATION AND STANDARDIZATION

- Are two common techniques in machine learning for scaling or transforming features, especially when working with data that vary significantly in magnitude or units.
- Proper scaling often improves model performance, particularly in algorithms that are sensitive to feature magnitudes, such as gradient-based models and distance-based models.

NORMALIZATION

- Normalization typically refers to scaling the features to a specific range, usually [0, 1] or [-1, 1].
- This ensures that the values of all features lie within a uniform range, preserving the relationships among data points while changing their scale.
- **Formula:**
- $$X_{norm} = X - X_{min} / X_{max} - X_{min}$$
- Where:
- X is the original feature value,
- Xmin and Xmax are the minimum and maximum values of the feature.

STANDARDIZATION

- Standardization involves transforming the features to have a **mean of 0 and a standard deviation of 1**. This process shifts the distribution of the data so that it is centered around 0, with unit variance. It is particularly useful when features have different units or are distributed on vastly different scales.
- **Formula:**
- $$Xstd = X - \mu / \sigma$$
- Where:
- μ is the mean of the feature,
- σ is the standard deviation of the feature.

DISCRETIZATION AND BINNING

- **Binning** (or discretization) is the process of converting continuous data into discrete bins or categories.
- It is often used in data analysis and preprocessing to simplify the data, reduce the effects of small observation errors, and make patterns more visible.
- Binning can also help in handling skewed data and outliers, improving the performance of some machine learning models.

TYPES OF BINNING

1. Fixed-width Binning (Equal-width Binning)-

Divides the range of data into intervals (bins) of equal size.

Example: If you have a continuous range of data from 0 to 100 and you decide to use 5 bins, each bin would have a width of 20 (0-20, 20-40, 40-60, 60-80, 80-100).

2. Quantile Binning (Equal-frequency Binning)-

Divides the data into bins so that each bin contains an equal number of data points.

Example: For a dataset of 100 values, if you create 4 bins using quantile binning, each bin will have 25 values

3.Custom Binning-

The user defines the bin edges based on domain knowledge or specific requirements.

Example: Age categories like 0-18 (child), 19-35 (young adult), 36-60 (middle-aged), 61+ (senior).

4. K-Means Binning-

Uses clustering techniques, like K-Means, to create bins where data points in each bin are similar.

This method is less common but can be useful for creating bins with similar properties.

Advantages of Binning

- Simplification
- Handling Outliers
- Improved Model Interpretability
- Data Normalization

Disadvantages of Binning

- Loss of Information
- Choice of Bins
- Introduces Bias

```python
import pandas as pd

# Sample data
data = {'age': [12, 25, 47, 51, 62, 78, 15, 36, 29, 52, 85]}
df = pd.DataFrame(data)

# Fixed-width binning
df['age_bin_fixed'] = pd.cut(df['age'], bins=3)  # 3 equal-width bins

# Quantile binning
df['age_bin_quantile'] = pd.qcut(df['age'], q=3)  # 3 equal-frequency bins

# Custom binning
bins = [0, 18, 35, 60, 100]
labels = ['Child', 'Young Adult', 'Middle-aged', 'Senior']
df['age_bin_custom'] = pd.cut(df['age'], bins=bins, labels=labels)

print(df)
```

OUTPUT

	age	age_bin_fixed	age_bin_quantile	age_bin_custom
0	12	(11.927, 36.333]	(11.999, 31.333]	Child
1	25	(11.927, 36.333]	(11.999, 31.333]	Young Adult
2	47	(36.333, 60.667]	(31.333, 51.667]	Middle-aged
3	51	(36.333, 60.667]	(31.333, 51.667]	Middle-aged
4	62	(60.667, 85.0]	(51.667, 85.0]	Senior
5	78	(60.667, 85.0]	(51.667, 85.0]	Senior
6	15	(11.927, 36.333]	(11.999, 31.333]	Child
7	36	(11.927, 36.333]	(31.333, 51.667]	Middle-aged
8	29	(11.927, 36.333]	(11.999, 31.333]	Young Adult
9	52	(36.333, 60.667]	(51.667, 85.0]	Middle-aged
10	85	(60.667, 85.0]	(51.667, 85.0]	Senior

INTRODUCTION TO MISSING DATA, HANDLING MISSING DATA

INTRODUCTION TO MISSING DATA

- **Missing data** is a common issue in data analysis, where certain values or observations are absent from a dataset.
- This can happen for various reasons, such as human error, data entry problems, or sensor malfunctions.
- Missing data can significantly affect the quality of the analysis and the performance of machine learning models.

IDENTIFYING MISSING DATA:

- **Visualization:** Use heatmaps, bar plots, or matrix plots to visualize the distribution and pattern of missing data.

- **Summary Statistics:** Use functions like .isnull().sum() in pandas to count missing values per column.

HANDLING MISSING DATA:

- **Removing Missing Data:**

- **Dropping Rows:** Remove rows with missing data if the proportion of missing data is small.

- **Dropping Columns:** Remove columns with a high proportion of missing values, especially if they are not critical to the analysis.
- Example- df.dropna()

```
import pandas as pd
import numpy as np
from sklearn.impute import SimpleImputer

# Create a sample DataFrame with missing values
data = {'Name': ['Alice', 'Bob', 'Charlie', 'David', 'Eve'],
        'Age': [24, np.nan, 22, 30, np.nan],
        'Salary': [50000, 54000, np.nan, 60000, 59000]}

df = pd.DataFrame(data)

# Show the data with missing values
print("Original Data:")
print(df)
```

OUTPUT

```
Original Data:
      Name   Age   Salary
0    Alice  24.0  50000.0
1      Bob   NaN  54000.0
2  Charlie  22.0      NaN
3    David  30.0  60000.0
4      Eve   NaN  59000.0
```

```
# Option 1: Drop rows with missing values

df_dropped = df.dropna()

print("\nAfter Dropping Rows with Missing Values:")

print(df_dropped)
```

OUTPUT

```
After Dropping Rows with Missing Values:
    Name   Age   Salary
0  Alice  24.0  50000.0
3  David  30.0  60000.0
```

DROP COLUMN

```
# Drop the 'Salary' column
df_dropped = df.drop('Salary', axis=1)
print(df_dropped)
```

	Name	Age
0	Alice	24.0
1	Bob	NaN
2	Charlie	22.0
3	David	30.0
4	Eve	NaN

```
# Option 2: Fill missing values with a specific value (e.g., mean, median)
# Filling missing 'Age' with the mean value

df['Age_filled'] = df['Age'].fillna(df['Age'].mean())

# Filling missing 'Salary' with the median value

df['Salary_filled'] = df['Salary'].fillna(df['Salary'].median())

print("\nAfter Filling Missing Values:")
print(df)
```

```
After Filling Missing Values:
      Name   Age   Salary  Age_filled  Salary_filled
0    Alice  24.0  50000.0   24.000000        50000.0
1      Bob   NaN  54000.0   25.333333        54000.0
2  Charlie  22.0      NaN   22.000000        56500.0
3    David  30.0  60000.0   30.000000        60000.0
```

IMPUTATION:

- **Mean/Median/Mode Imputation**: Replace missing values with the mean, median, or mode of the column. This is simple but can introduce bias.

- **Forward/Backward Fill**: Fill missing values using the previous (forward fill) or next (backward fill) values. This is useful for time-series data.

- **Interpolation**: Estimate missing values by interpolating betwe isting values.
- Example: df.fillna(df.mean())

in pandas to fill missing values with the mean.

```
# Option 3: Imputation using SimpleImputer from sklearn
# Filling missing values using the mean for 'Age' and median for 'Salary'
imputer = SimpleImputer(strategy='mean')
df['Age_imputed'] = imputer.fit_transform(df[['Age']])

imputer = SimpleImputer(strategy='median')
df['Salary_imputed'] = imputer.fit_transform(df[['Salary']])

print("\nAfter Imputation Using SimpleImputer:")
print(df)
```

```
After Imputation Using SimpleImputer:
      Name   Age   Salary  Age_filled  Salary_filled  Age_imputed  \
0    Alice  24.0  50000.0   24.000000        50000.0    24.000000
1      Bob   NaN  54000.0   25.333333        54000.0    25.333333
2  Charlie  22.0      NaN   22.000000        56500.0    22.000000
3    David  30.0  60000.0   30.000000        60000.0    30.000000
4      Eve   NaN  59000.0   25.333333        59000.0    25.333333

   Salary_imputed
0         50000.0
1         54000.0
2         56500.0
3         60000.0
4         59000.0
```

DATA VISUALIZATION USING MATPLOTLIB, SEABORN

- Line plot
- Scatter Plot
- Heat Map
- Box plot

NORMALIZATION AND STANDARDIZATION

```
import pandas as pd
import numpy as np
from sklearn.preprocessing import MinMaxScaler, StandardScaler

# Sample dataset
data = {'Age': [25, 30, 22, 35, 28],
        'Salary': [50000, 60000, 52000, 80000, 55000]}

df = pd.DataFrame(data)

# Display original data
print("Original Data:")
print(df)
```

OUTPUT

```
Age  Salary
0   25   50000
1   30   60000
2   22   52000
3   35   80000
4   28   55000
```

NORMALIZATION


```
# === Normalization using MinMaxScaler === #
# Create a MinMaxScaler object
min_max_scaler = MinMaxScaler()

# Apply normalization to the dataset
normalized_data = min_max_scaler.fit_transform(df)

# Convert the result back to a DataFrame
df_normalized = pd.DataFrame(normalized_data, columns=df.columns)

print("\nNormalized Data (MinMaxScaler):")
print(df_normalized)
```

OUTPUT

	Age	Salary
0	0.230769	0.000000
1	0.615385	0.500000
2	0.000000	0.100000
3	1.000000	1.000000
4	0.461538	0.166667

STANDARDIZATION

```
# === Standardization using StandardScaler === #
# Create a StandardScaler object
standard_scaler = StandardScaler()

# Apply standardization to the dataset
standardized_data = standard_scaler.fit_transform(df)

# Convert the result back to a DataFrame
df_standardized = pd.DataFrame(standardized_data, columns=df.columns)

print("\nStandardized Data (StandardScaler):")
print(df_standardized)
```

OUTPUT

	Age	Salary
0	-0.707107	-1.149082
1	0.707107	-0.229816
2	-1.414214	-1.034466
3	1.414214	1.609073
4	0.000000	-0.804291

FORWARD FILL


```
import pandas as pd
import numpy as np

# Sample DataFrame with missing values
data = {'Date': ['2023-10-01', '2023-10-02', '2023-10-03', '2023-10-04', '2023-10-05'],
        'Value': [10, np.nan, 20, np.nan, 30]}

df = pd.DataFrame(data)

# Display original DataFrame
print("Original DataFrame:")
print(df)

# Apply forward fill to fill missing values
df_ffill = df.fillna(method='ffill')

# Display DataFrame after forward fill
print("\nDataFrame After Forward Fill:")
print(df_ffill)
```

Original DataFrame:

	Date	Value
0	2023-10-01	10.0
1	2023-10-02	NaN
2	2023-10-03	20.0
3	2023-10-04	NaN
4	2023-10-05	30.0

DataFrame After Forward Fill:

	Date	Value
0	2023-10-01	10.0
1	2023-10-02	10.0
2	2023-10-03	20.0
3	2023-10-04	20.0
4	2023-10-05	30.0

BACKWARD FILL


```
import pandas as pd
import numpy as np

# Sample DataFrame with missing values
data = {'Date': ['2023-10-01', '2023-10-02', '2023-10-03', '2023-10-04', '2023-10-05'],
        'Value': [10, np.nan, 20, np.nan, 30]}

df = pd.DataFrame(data)

# Display original DataFrame
print("Original DataFrame:")
print(df)

# Apply backward fill to fill missing values
df_bfill = df.fillna(method='bfill')

# Display DataFrame after backward fill
print("\nDataFrame After Backward Fill:")
print(df_bfill)
```

OUTPUT

Original DataFrame:

	Date	Value
0	2023-10-01	10.0
1	2023-10-02	NaN
2	2023-10-03	20.0
3	2023-10-04	NaN
4	2023-10-05	30.0

DataFrame After Backward Fill:

	Date	Value
0	2023-10-01	10.0
1	2023-10-02	20.0
2	2023-10-03	20.0
3	2023-10-04	30.0
4	2023-10-05	30.0

INTERPOLATE MISSING VALUES

In machine learning, **interpolating missing values** involves filling in the missing data points by estimating their values based on the surrounding data. This can be especially useful when you have time-series or continuous data, where missing values can be inferred from existing data points.

INTERPOLATION TECHNIQUES

There are several interpolation techniques, including:

1. **Linear Interpolation**: Assumes a straight line between two known points.
2. **Polynomial Interpolation**: Fits a polynomial function between the points.
3. **Spline Interpolation**: Fits a spline (piecewise polynomial) between points.
4. **Time-based Interpolation**: If the index of your data is a time series, this method takes the time spacing into account.

```
import pandas as pd
import numpy as np

# Create a sample dataframe with missing values
data = {'A': [1, 2, np.nan, 4, 5],
        'B': [5, np.nan, np.nan, 8, 10]}

df = pd.DataFrame(data)

print("Original Data:")
print(df)

# Perform linear interpolation
df_interpolated = df.interpolate(method='linear’)


print("\nInterpolated Data (Linear):")
print(df_interpolated)
```

DATA DEDUPLICATION

In **Machine Learning (ML)**, data deduplication refers to the process of identifying and removing duplicate data entries from the dataset to ensure that the training data is unique, clean, and efficient for model training.

Redundant or duplicate data can lead to biased models, inefficient training processes, and increased computational overhead.

By eliminating duplicates, deduplication enhances the quality of the dataset, reduces storage requirements, and improves the model's generalization ability.

COMMON TECHNIQUES USED FOR DATA DEDUPLICATION

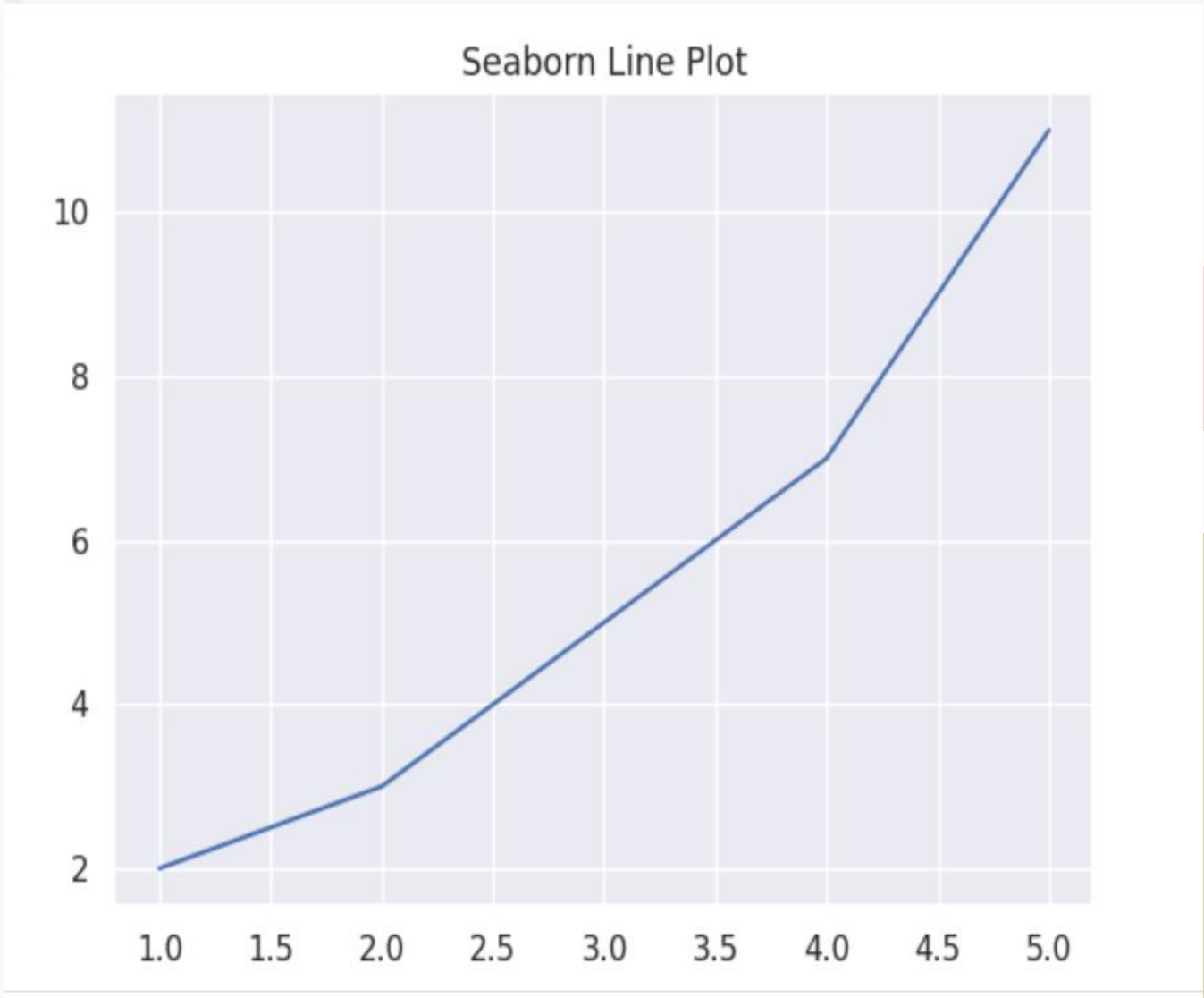
1. Exact Match Deduplication
2. Near-Duplicate Matching (Fuzzy Matching)
3. Hash-Based Deduplication
4. Feature-Based Deduplication
5. Clustering-Based Deduplication
6. Cross-Dataset Deduplication
7. Image-Based Deduplication (for Computer Vision)

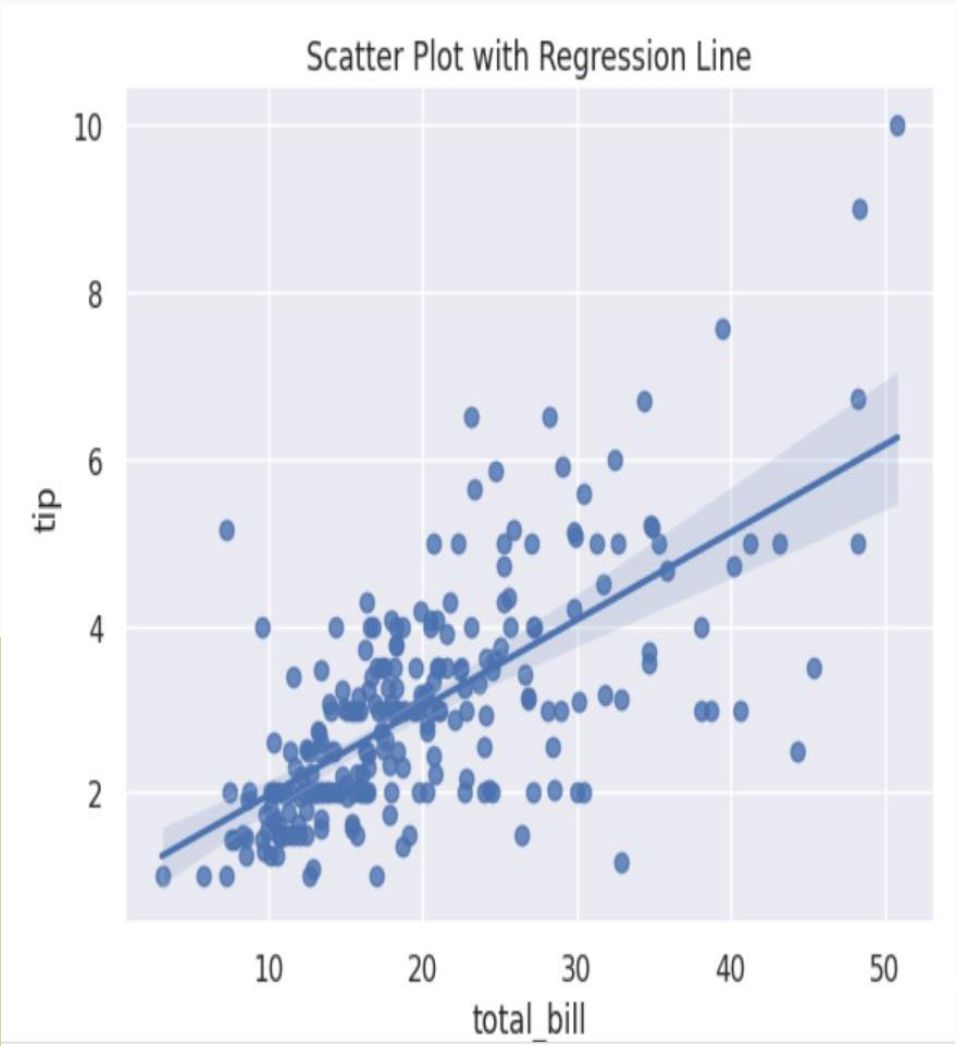
```
# Example in pandas
df.drop_duplicates(inplace=True)
```

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

# Sample data
sns.set(style="darkgrid")
sns.lineplot(x=[1, 2, 3, 4, 5], y=[2, 3, 5, 7, 11])

plt.title('Seaborn Line Plot')
plt.show()
```






```
# Scatter plot with regression line
tips = sns.load_dataset("tips")
sns.regplot(x='total_bill', y='tip', data=tips)

plt.title('Scatter Plot with Regression Line')
plt.show()
```