**Perform Standard Data Import, Joining, and Aggregation Tasks**

Import Data from Flat Files (CSV) into Python

Description: Load data from flat files like CSV into a DataFrame using pandas. This is crucial for working with structured datasets.

Utility: Read CSV files for data analysis or preprocessing tasks.

Keywords: import CSV, read data, pandas DataFrame, flat file, read\_csv.

Example:

import pandas as pd

data = pd.read\_csv('file.csv')

Import Data from Databases into Python

Description: Fetch data directly from databases using SQLAlchemy or sqlite3 in Python and convert it to a DataFrame.

Utility: Useful for pulling data from SQL databases for analysis.

Keywords: database import, SQL, fetch data, SQLAlchemy, sqlite3, connection.

Example:

import sqlite3

import pandas as pd

conn = sqlite3.connect('database.db')

data = pd.read\_sql\_query("SELECT \* FROM table\_name", conn)

Aggregate Numeric, Categorical Variables, and Dates by Groups

Description: Use groupby in pandas to aggregate data based on numeric, categorical, or date columns.

Utility: Summarize or aggregate data, often used for statistical reports.

Keywords: groupby, aggregate, sum, mean, aggregate dates, categorical.

Example:

# Aggregating numerical columns by a categorical column

data\_grouped = data.groupby('category\_column')['numeric\_column'].sum()

Combine Multiple Tables by Rows or Columns

Description: Use concat or merge in pandas to combine data from different sources, either by rows or columns.

Utility: Combine datasets for broader analysis, handling data spread across multiple tables.

Keywords: combine tables, merge, concat, join tables, append.

Example:

# Combining by columns

combined = pd.concat([df1, df2], axis=1)

# Combining by rows

combined = pd.concat([df1, df2], axis=0)

Filter Data Based on Different Criteria

Description: Use boolean indexing in pandas to filter data based on conditions.

Utility: Select specific subsets of data for targeted analysis.

Keywords: filter, conditional, subset data, query.

Example:

# Filtering rows where the value in 'column' is greater than 10

filtered\_data = data[data['column'] > 10]

**Perform Standard Cleaning Tasks to Prepare Data for Analysis**

Match Strings in a Dataset with Specific Patterns

Description: Use the str.contains() or str.match() methods in pandas or regular expressions (re module) to identify or extract strings matching a pattern.

Utility: Useful for identifying specific patterns in text, such as finding rows with email addresses, phone numbers, etc.

Keywords: string matching, regex, pattern matching, contains, regular expression.

Example:

import pandas as pd

# Filter rows where 'column' contains the word 'apple'

filtered\_data = data[data['column'].str.contains('apple', na=False)]

Convert Values Between Data Types

Description: Use the astype() method to convert columns between different data types (e.g., string to integer or float to string).

Utility: Essential when handling inconsistent data formats, ensuring compatibility for analysis.

Keywords: convert data type, astype, type conversion, string to int, float to int.

Example:

# Convert a column to integer

data['column'] = data['column'].astype(int)

Clean Categorical and Text Data by Manipulating Strings

Description: Use string manipulation functions like str.lower(), str.strip(), str.replace() to clean categorical or textual data.

Utility: Standardize and clean text data by removing whitespace, converting cases, and replacing unwanted characters.

Keywords: clean text, string manipulation, replace, lowercase, strip whitespace, categorical cleaning.

Example:

# Remove leading/trailing spaces and convert to lowercase

data['category\_column'] = data['category\_column'].str.strip().str.lower()

Clean Date and Time Data

Description: Use the pd.to\_datetime() function to parse and convert string dates into datetime objects for analysis and manipulation.

Utility: Ensures consistent formatting and allows for operations like filtering or calculating time differences.

Keywords: date cleaning, convert to datetime, time data, pd.to\_datetime, date format.

Example:

# Convert string column to datetime format

data['date\_column'] = pd.to\_datetime(data['date\_column'], format='%Y-%m-%d')

**Assess Data Quality and Perform Validation Tasks**

Identify and Replace Missing Values

Description: Use functions like isnull() or fillna() in pandas to locate and handle missing values in datasets. You can either fill them with a default value or drop them based on the context.

Utility: Ensures data quality by handling missing data appropriately, which is essential for maintaining consistency in analysis.

Keywords: missing values, NaN, null values, fillna, dropna.

Example:

import pandas as pd

# Identify missing values

missing\_data = data.isnull().sum()

# Replace missing values with the mean of the column

data['column'] = data['column'].fillna(data['column'].mean())

Perform Different Types of Data Validation

Description: Validate data for consistency, enforce constraints, ensure values fall within an expected range, and check for uniqueness using conditional statements and built-in pandas methods.

Utility: Guarantees the reliability and validity of the dataset by ensuring that data adheres to specific rules.

Keywords: data validation, range validation, consistency check, constraints, unique values.

Example:

# Range validation: ensure values in a column are between 0 and 100

valid\_data = data[(data['column'] >= 0) & (data['column'] <= 100)]

# Check for uniqueness in a column

unique\_values = data['column'].is\_unique

Identify and Validate Data Types

Description: Use the dtypes attribute to identify the data types of columns, and astype() or infer\_objects() to validate or convert them.

Utility: Ensures that each column has the correct data type, which is essential for applying further analysis or calculations.

Keywords: data type validation, dtypes, astype, infer\_objects, validate types.

Example:

# Identify data types in the dataset

data\_types = data.dtypes

# Convert a column to float type if necessary

data['column'] = data['column'].astype(float)

Collect Data from Non-Standard Formats by Modifying Existing Code

Adapt Provided Code to Import Data from an API

Description: Use the requests library in Python to retrieve data from an API and convert it into a DataFrame using pandas.

Utility: Allows you to collect live or real-time data from web APIs, such as financial data, weather reports, or social media metrics.

Keywords: API data, requests, GET request, retrieve API, import JSON.

Example:

import requests

import pandas as pd

# Send a GET request to the API

response = requests.get('https://api.example.com/data')

# Convert the JSON response to a DataFrame

data = pd.DataFrame(response.json())

Identify the Structure of HTML and JSON Data and Parse Them

Description: Use BeautifulSoup from bs4 to parse HTML and json library to parse JSON data into usable formats for analysis in Python.

Utility: Extract relevant data from HTML pages (web scraping) or process JSON data (common in APIs) for structured data analysis.

Keywords: HTML parsing, JSON parsing, web scraping, BeautifulSoup, json, from\_dict.

Example:

# Parsing JSON data

import json

# Load JSON data

json\_data = '{"name": "John", "age": 30, "city": "New York"}'

parsed\_data = json.loads(json\_data)

# Converting JSON data to a DataFrame

data = pd.DataFrame([parsed\_data])

# Parsing HTML using BeautifulSoup

from bs4 import BeautifulSoup

html = '<html><body><h1>Hello World</h1></body></html>'

soup = BeautifulSoup(html, 'html.parser')

# Extracting data from HTML

heading = soup.find('h1').text

Parsing JSON

Description: The json.loads() method helps convert a JSON object into a Python dictionary for further manipulation.

Utility: Useful for decoding JSON responses from APIs and working with structured data in Python.

Keywords: json, loads, parse JSON, dictionary.

Example:

import json

# Parse JSON string

data\_dict = json.loads('{"key": "value"}')

**Calculate Metrics to Report Characteristics of Data and Relationships Between Features**

Calculate Measures of Center (Mean, Median, Mode)

Description: Use mean(), median(), and mode() from pandas or scipy to calculate central tendencies in datasets.

Utility: These measures summarize data by indicating the central or typical value for a dataset.

Keywords: mean, median, mode, central tendency, average.

Example:

import pandas as pd

from scipy import stats

# Calculate mean, median, and mode

mean\_value = data['column'].mean()

median\_value = data['column'].median()

mode\_value = stats.mode(data['column'])[0]

Calculate Measures of Spread (Range, Standard Deviation, Variance)

Description: Use functions like std() and var() from pandas to measure how data is dispersed. Range can be calculated by subtracting the minimum value from the maximum.

Utility: Helps in understanding the variability and distribution of data.

Keywords: range, standard deviation, variance, spread, dispersion.

Example:

# Calculate standard deviation, variance, and range

std\_dev = data['column'].std()

variance = data['column'].var()

data\_range = data['column'].max() - data['column'].min()

Calculate Skewness

Description: Use skew() from scipy.stats or pandas to determine the asymmetry of data distributions.

Utility: Skewness helps identify whether the data distribution leans more towards higher or lower values.

Keywords: skewness, distribution, asymmetry, left-skew, right-skew.

Example:

from scipy.stats import skew

# Calculate skewness

skewness\_value = skew(data['column'])

Calculate Missingness and Its Influence

Description: Use isnull() and sum() to calculate the number of missing values. Missing data can distort metrics, lead to biased estimates, or affect the reliability of data relationships.

Utility: Identifying and handling missing data is critical to maintain accurate reporting and analysis.

Keywords: missing values, missingness, NaN, influence on data.

Example:

# Calculate missing values

missing\_count = data['column'].isnull().sum()

# Calculate percentage of missing values

missing\_percentage = (missing\_count / len(data)) \* 100

Calculate Correlation Between Variables

Description: Use the corr() function in pandas to calculate Pearson correlation coefficients between two or more variables.

Utility: Correlation indicates the strength and direction of linear relationships between variables.

Keywords: correlation, correlation coefficient, relationship between variables, Pearson.

Example:

# Calculate correlation between two variables

correlation\_value = data['column1'].corr(data['column2'])

# Calculate correlation matrix for the entire dataset

correlation\_matrix = data.corr()

**Create Data Visualizations in Python to Demonstrate Data Characteristics**

Create and Customize Bar Charts

Description: Use matplotlib or seaborn to create bar charts, representing categorical data with rectangular bars.

Utility: Bar charts are great for comparing quantities across categories.

Keywords: bar chart, categorical data, bar width, bar height, matplotlib, seaborn.

Example:

import matplotlib.pyplot as plt

# Create a bar chart

plt.bar(data['category\_column'], data['value\_column'])

# Customize the bar chart

plt.title('Bar Chart Example')

plt.xlabel('Category')

plt.ylabel('Value')

plt.show()

Create and Customize Box Plots

Description: Use seaborn or matplotlib to generate box plots that visualize the distribution, including medians, quartiles, and outliers.

Utility: Box plots are ideal for displaying the spread and identifying outliers in continuous data.

Keywords: box plot, outliers, quartiles, distribution, seaborn.

Example:

import seaborn as sns

# Create a box plot

sns.boxplot(x='category\_column', y='value\_column', data=data)

# Customize the box plot

plt.title('Box Plot Example')

plt.show()

Create and Customize Line Graphs

Description: Use matplotlib or seaborn to create line graphs, ideal for showing trends over time or continuous data.

Utility: Line graphs help in analyzing trends, particularly for time-series data.

Keywords: line graph, time series, trend, continuous data, matplotlib.

Example:

# Create a line graph

plt.plot(data['date\_column'], data['value\_column'])

# Customize the line graph

plt.title('Line Graph Example')

plt.xlabel('Date')

plt.ylabel('Value')

plt.show()

Create and Customize Histograms

Description: Use matplotlib or seaborn to create histograms that represent the distribution of continuous data.

Utility: Histograms display the frequency of data within specified ranges.

Keywords: histogram, distribution, frequency, bins, seaborn.

Example:

# Create a histogram

plt.hist(data['value\_column'], bins=10)

# Customize the histogram

plt.title('Histogram Example')

plt.xlabel('Value')

plt.ylabel('Frequency')

plt.show()

Create Data Visualizations in Python to Represent Relationships Between Features

Create and Customize Scatterplots

Description: Use matplotlib or seaborn to create scatterplots that show relationships between two continuous variables.

Utility: Scatterplots reveal correlations and patterns between variables.

Keywords: scatterplot, relationship, correlation, continuous data, seaborn.

Example:

# Create a scatterplot

plt.scatter(data['column1'], data['column2'])

# Customize the scatterplot

plt.title('Scatterplot Example')

plt.xlabel('Variable 1')

plt.ylabel('Variable 2')

plt.show()

Create and Customize Heatmaps

Description: Use seaborn to create heatmaps that visualize the correlation between variables or intensity of values.

Utility: Heatmaps are useful for analyzing the correlation matrix or frequency distributions.

Keywords: heatmap, correlation, intensity, seaborn.

Example:

# Create a heatmap for a correlation matrix

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

# Customize the heatmap

plt.title('Heatmap Example')

plt.show()

Create and Customize Pivot Tables

Description: Use pandas to create pivot tables for summarizing and aggregating data. Visualize the output using seaborn or matplotlib.

Utility: Pivot tables are ideal for summarizing categorical data.

Keywords: pivot table, summarize data, categorical data, aggregation, pandas.

Example:

# Create a pivot table

pivot\_table = data.pivot\_table(index='category\_column', values='value\_column', aggfunc='sum')

# Plot the pivot table data

pivot\_table.plot(kind='bar')

plt.title('Pivot Table Example')

plt.show()

**Statistical Operations Using Python**

Sample Data from a Statistical Distribution

Description: Use numpy and scipy to generate random samples from common statistical distributions like normal, binomial, Poisson, and exponential.

Utility: Sampling allows you to simulate and analyze data that follows specific theoretical distributions.

Keywords: normal distribution, binomial distribution, Poisson distribution, exponential distribution, random samples.

Example:

import numpy as np

# Generate samples from a normal distribution

normal\_samples = np.random.normal(loc=0, scale=1, size=1000)

# Generate samples from a binomial distribution

binomial\_samples = np.random.binomial(n=10, p=0.5, size=1000)

# Generate samples from a Poisson distribution

poisson\_samples = np.random.poisson(lam=3, size=1000)

# Generate samples from an exponential distribution

exponential\_samples = np.random.exponential(scale=1, size=1000)

Calculate a Probability from a Statistical Distribution

Description: Use functions from scipy.stats to calculate probabilities for various distributions, such as cumulative probabilities or point probabilities.

Utility: Calculate the likelihood of events based on the properties of statistical distributions.

Keywords: calculate probability, PDF, CDF, normal distribution, binomial, Poisson, exponential.

Example:

from scipy.stats import norm, binom, poisson, expon

# Calculate the cumulative probability for a normal distribution

normal\_prob = norm.cdf(1, loc=0, scale=1)

# Calculate the probability of exactly 5 successes in a binomial distribution

binomial\_prob = binom.pmf(5, n=10, p=0.5)

# Calculate the probability of exactly 3 events in a Poisson distribution

poisson\_prob = poisson.pmf(3, mu=3)

# Calculate the probability of a value being less than or equal to 2 in an exponential distribution

exponential\_prob = expon.cdf(2, scale=1)

Statistical Tests Using Python

Run Statistical Tests

Description: Use scipy.stats to run common statistical tests such as t-tests, ANOVA, and chi-square tests.

Utility: Statistical tests are used to determine if there are significant differences or relationships between data groups.

Keywords: t-test, ANOVA, chi-square test, significance, statistical hypothesis, p-value.

Example:

from scipy.stats import ttest\_ind, f\_oneway, chi2\_contingency

# Run an independent t-test

t\_stat, t\_p\_value = ttest\_ind(group1, group2)

# Run a one-way ANOVA test

f\_stat, f\_p\_value = f\_oneway(group1, group2, group3)

# Run a chi-square test

chi2\_stat, chi2\_p\_value, \_, \_ = chi2\_contingency(contingency\_table)

Analyze Results of Statistical Tests

Description: Interpret the test statistic and p-value to determine if the null hypothesis can be rejected.

Utility: Allows you to conclude whether there is statistical significance between groups or variables.

Keywords: p-value, statistical significance, null hypothesis, reject null, confidence level.

Example:

# Analyze t-test results

if t\_p\_value < 0.05:

print("Reject null hypothesis, significant difference between groups")

else:

print("Fail to reject null hypothesis, no significant difference")

# Analyze ANOVA test results

if f\_p\_value < 0.05:

print("Reject null hypothesis, significant difference between groups")

else:

print("Fail to reject null hypothesis, no significant difference")

# Analyze chi-square test results

if chi2\_p\_value < 0.05:

print("Reject null hypothesis, significant association between variables")

else:

print("Fail to reject null hypothesis, no significant association")

**Joining Data with pandas in Python**

Description:

Use pandas functions like merge(), join(), and concat() to combine two or more dataframes. The method of joining depends on the structure and requirements of the data (e.g., inner, outer, left, right joins). This allows you to combine data from multiple sources or tables for analysis.

Utility:

Joining data allows you to combine datasets with common columns or indices, making it easier to work with larger and more complex datasets.

Common use cases include merging data from different sources (e.g., combining sales data with customer data).

Keywords:

merge, join, concat, inner join, outer join, left join, right join, combining dataframes.

Common Joining Methods

Inner Join:

Merges only the rows with keys present in both dataframes.

Example:

merged\_df = pd.merge(df1, df2, on='key\_column', how='inner')

Left Join:

Merges all rows from the left dataframe, and only matching rows from the right.

Example:

merged\_df = pd.merge(df1, df2, on='key\_column', how='left')

Right Join:

Merges all rows from the right dataframe, and only matching rows from the left.

Example:

merged\_df = pd.merge(df1, df2, on='key\_column', how='right')

Outer Join:

Merges all rows from both dataframes. Missing values are filled with NaN.

Example:

merged\_df = pd.merge(df1, df2, on='key\_column', how='outer')

Concatenating DataFrames:

Used to stack dataframes either vertically (row-wise) or horizontally (column-wise).

Example (row-wise):

concatenated\_df = pd.concat([df1, df2], axis=0)

Join on Index:

Merges two dataframes based on their indices.

Example:

joined\_df = df1.join(df2, how='inner')

Example of a Left Join:

import pandas as pd

# Sample data

df1 = pd.DataFrame({

'ID': [1, 2, 3, 4],

'Name': ['Alice', 'Bob', 'Charlie', 'David']

})

df2 = pd.DataFrame({

'ID': [1, 2, 5],

'Score': [85, 90, 95]

})

# Perform a left join on the 'ID' column

merged\_df = pd.merge(df1, df2, on='ID', how='left')

**Data Manipulation with pandas in Python**

Description:

pandas is a powerful Python library used for data manipulation and analysis. It provides data structures like DataFrames and Series that make it easy to work with structured data. Common tasks include filtering, sorting, grouping, transforming, and reshaping data.

Utility:

pandas helps in cleaning, modifying, and transforming datasets to make them ready for analysis.

You can handle missing data, calculate new columns, aggregate data, and reshape it to fit your requirements.

Keywords:

filter, sort, groupby, transform, reshape, manipulate data, aggregate, pandas.

Common Data Manipulation Tasks in pandas

Filtering Data

Description: Extract specific rows based on conditions.

Example:

# Filter rows where 'age' is greater than 30

filtered\_df = df[df['age'] > 30]

Sorting Data

Description: Sort the DataFrame by one or more columns.

Example:

# Sort by 'age' in descending order

sorted\_df = df.sort\_values(by='age', ascending=False)

Creating New Columns

Description: Create new columns based on existing data or calculations.

Example:

# Create a new column 'total' as the sum of 'col1' and 'col2'

df['total'] = df['col1'] + df['col2']

Handling Missing Data

Description: Fill, drop, or manipulate missing values.

Example:

# Fill missing values in 'column\_name' with the mean

df['column\_name'].fillna(df['column\_name'].mean(), inplace=True)

Grouping and Aggregating Data

Description: Group data by one or more columns and calculate aggregates like sum, mean, etc.

Example:

# Group by 'category' and calculate the mean of 'value' column

grouped\_df = df.groupby('category')['value'].mean()

Renaming Columns

Description: Rename one or more columns in the DataFrame.

Example:

# Rename a single column

df.rename(columns={'old\_name': 'new\_name'}, inplace=True)

Dropping Columns or Rows

Description: Remove unnecessary columns or rows.

Example:

# Drop a column

df.drop(columns=['column\_to\_drop'], inplace=True)

Reshaping Data

Description: Use functions like pivot() or melt() to reshape the DataFrame.

Example (Pivot):

# Pivot the DataFrame to reorganize data

pivot\_df = df.pivot(index='category', columns='type', values='value')

Merging or Joining Data

Description: Combine multiple DataFrames using merge or join operations.

Example:

# Merge two DataFrames on a common column

merged\_df = pd.merge(df1, df2, on='key\_column', how='inner')

Example of Data Manipulation:

import pandas as pd

# Sample DataFrame

data = {

'Name': ['Alice', 'Bob', 'Charlie', 'David', 'Eve'],

'Age': [25, 30, 35, 40, 45],

'Salary': [50000, 60000, 70000, 80000, 90000],

'Department': ['HR', 'Finance', 'IT', 'HR', 'Finance']

}

df = pd.DataFrame(data)

# Filtering data for employees older than 30

filtered\_df = df[df['Age'] > 30]

# Adding a new column 'Bonus' as 10% of 'Salary'

df['Bonus'] = df['Salary'] \* 0.10

# Grouping by 'Department' and calculating average 'Salary'

grouped\_df = df.groupby('Department')['Salary'].mean()

# Sorting by 'Age'

sorted\_df = df.sort\_values(by='Age')

Handling Missing Data Example:

python

import numpy as np

# Introduce missing data

df.loc[2, 'Salary'] = np.nan

# Fill missing 'Salary' with the mean of the column

df['Salary'].fillna(df['Salary'].mean(), inplace=True)

**Introduction to Statistics in Python**

Description:

Statistics is a branch of mathematics that deals with collecting, analyzing, interpreting, presenting, and organizing data. Python, with libraries like NumPy, pandas, and SciPy, provides powerful tools for performing statistical analysis.

Utility:

Understanding statistics is essential for data analysis, as it helps in making informed decisions based on data. Python's libraries make it easy to perform statistical tests, summarize data, and visualize results.

Keywords:

descriptive statistics, inferential statistics, mean, median, mode, standard deviation, probability distributions, hypothesis testing.

Basic Statistical Concepts

Descriptive Statistics:

Summary measures that describe the main features of a dataset.

Common Measures:

Mean: The average of a dataset.

Median: The middle value when data is sorted.

Mode: The most frequently occurring value.

Standard Deviation: A measure of the dispersion or spread of the data.

Example:

import pandas as pd

# Sample data

data = [1, 2, 2, 3, 4, 5, 5, 5, 6]

# Convert to DataFrame

df = pd.DataFrame(data, columns=['Values'])

# Descriptive statistics

mean = df['Values'].mean()

median = df['Values'].median()

mode = df['Values'].mode()[0] # Mode can return multiple values

std\_dev = df['Values'].std()

print(f"Mean: {mean}, Median: {median}, Mode: {mode}, Std Dev: {std\_dev}")

Inferential Statistics:

Techniques that allow us to use a sample to make inferences about a population.

Common Techniques:

Confidence Intervals: A range of values that is likely to contain the population parameter.

Hypothesis Testing: A method to determine if there is enough evidence to reject a null hypothesis.

Probability Distributions:

Functions that describe the likelihood of obtaining the possible values that a random variable can take.

Common Distributions:

Normal Distribution: Bell-shaped curve, characterized by its mean and standard deviation.

Binomial Distribution: Represents the number of successes in a fixed number of trials.

Example:

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

# Generate a normal distribution

data = np.random.normal(loc=0, scale=1, size=1000)

# Plot the distribution

sns.histplot(data, bins=30, kde=True)

plt.title('Normal Distribution')

plt.xlabel('Value')

plt.ylabel('Frequency')

plt.show()

Hypothesis Testing:

A process to test assumptions or claims about a population based on sample data.

Common Tests:

t-test: Compares the means of two groups.

Chi-square test: Tests the relationship between categorical variables.

Example:

from scipy import stats

# Sample data

group1 = [20, 21, 23, 24, 30]

group2 = [22, 25, 28, 29, 31]

# Perform a t-test

t\_stat, p\_value = stats.ttest\_ind(group1, group2)

print(f"T-statistic: {t\_stat}, P-value: {p\_value}")

if p\_value < 0.05:

print("Reject the null hypothesis")

else:

print("Fail to reject the null hypothesis")

**Data Visualization with Seaborn in Python**

Description:

Seaborn is a powerful Python visualization library based on Matplotlib that provides a high-level interface for drawing attractive and informative statistical graphics. It simplifies the creation of complex visualizations and offers built-in themes and color palettes.

Utility:

Seaborn is particularly useful for visualizing data distributions, relationships between variables, and categorical data. Its integration with pandas allows for easy plotting directly from DataFrames.

Keywords:

data visualization, Seaborn, scatter plot, bar plot, box plot, heatmap, pair plot, catplot.

Common Visualization Types in Seaborn

Scatter Plot:

Description: Displays values for typically two variables for a set of data.

Example:

import seaborn as sns

import matplotlib.pyplot as plt

# Sample dataset

tips = sns.load\_dataset('tips')

# Create a scatter plot

sns.scatterplot(data=tips, x='total\_bill', y='tip', hue='time', style='time')

plt.title('Scatter Plot of Total Bill vs Tip')

plt.show()

Bar Plot:

Description: Shows the relationship between a categorical variable and a continuous variable.

Example:

# Create a bar plot

sns.barplot(data=tips, x='day', y='total\_bill', estimator=sum)

plt.title('Total Bill by Day')

plt.show()

Box Plot:

Description: Summarizes the distribution of a dataset through its quartiles.

Example:

# Create a box plot

sns.boxplot(data=tips, x='day', y='total\_bill', palette='Set2')

plt.title('Box Plot of Total Bill by Day')

plt.show()

Histogram:

Description: Shows the distribution of a single continuous variable.

Example:

# Create a histogram

sns.histplot(data=tips, x='total\_bill', bins=20, kde=True)

plt.title('Histogram of Total Bill')

plt.show()

Heatmap:

Description: Visualizes data through variations in color, typically used for correlation matrices.

Example:

# Calculate the correlation matrix

corr = tips.corr()

# Create a heatmap

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

plt.title('Heatmap of Correlation Matrix')

plt.show()

Pair Plot:

Description: Displays pairwise relationships in a dataset.

Example:

# Create a pair plot

sns.pairplot(tips, hue='time')

plt.title('Pair Plot of Tips Dataset')

plt.show()

Facet Grid:

Description: A grid of subplots based on a categorical variable, useful for visualizing the distribution of a variable across different categories.

Example:

# Create a facet grid

g = sns.FacetGrid(tips, col='time')

g.map(sns.histplot, 'total\_bill')

plt.subplots\_adjust(top=0.8)

g.fig.suptitle('Total Bill Distribution by Time')

plt.show()

Customization Options

Color Palettes: Seaborn provides various color palettes that can be easily applied to your plots.

sns.set\_palette('husl') # Set a color palette

Themes: You can change the overall style of your plots using:

sns.set\_style('whitegrid') # Options: darkgrid, whitegrid, dark, white, ticks