Register Number														

Set -

SRM Institute of Science and Technology
College of Engineering and Technology
School of Computing
SRM Nagar, Kattankulathur – 603203, Chengalpattu District, Tamil Nadu
Academic Year: 2024-25 (EVEN)

Test: FT4 Date: 29-04-2025
Course Code & Title: 21CSS303T-Data Science Duration: Two periods
Year& Sem: III Year /VI Sem Max.Marks:50

Course Articulation Matrix:

Course Outcome	PO1	PO2	PO3	PO4	PO5	PO6	PO7	PO8	PO9	PO10	PO11	PO12
CO3	-	-	-	-	1	-	-	-	-	-	-	-
CO4	-	-	-	-	1	-	-	-	-	-	-	-
CO5	-	-	-	-	1	-	-	-	-	-	-	-

Note: CO3 – To identify data manipulation and cleaning techniques using pandas
CO4 – To constructs the Graphs and plots to represent the data using python packages
CO5 – To apply the principles of the data science techniques to predict and forecast the outcome of real-world problem

Part – A (10 x 1 = 10 Marks)

Instructions:
1) Answer **ALL** questions.
2) The duration for answering Part A is **15 minutes** (this sheet will be collected after 15 minutes).
3) **Encircle the correct answer**.

S.No	Question	Marks	BL	CO	PO	PI Code
1	**In data wrangling, what does the term "imputation" refer to?** A. Dropping columns B. Filling in missing values C. Renaming variables D. Removing duplicates	1	1	3	5	
2	**What does `df1.join(df2, how='outer')` do?** A. Performs an outer join on columns B. Merges df2 into df1 on index, including all entries from both C. Merges by common column D. Appends rows	1	1	3	5	

3	**What is the output of the code?**
	`s = "abcdefghijk"`
	`result = s[8:2:-2]`
	`print(result)`
	A. "igec"
	B. "igda"
	C. "igca"
	D. "hfdb"
		1	1	3	5	

4	**In which scenario would the following code fail to detect outliers?**
	`z_scores = stats.zscore(data)`
	`outliers = np.where(np.abs(z_scores) > 3)`
	A. If data is normally distributed
	B. If outliers are beyond ±3 standard deviations
	C. If outliers are within ±3 standard deviations
	D. If data has no variation
		1	2	3	5	

5	**What is the output of the code?**
	`s = "one,two,three,four"`
	`result = "-".join([word.upper() for word in s.split(",")])`
	`print(result)`
	A. "ONE-TWO-THREE-FOUR"
	B. "one-two-three-four"
	C. "ONE,TWO,THREE,FOUR"
	D. An error occurs
		1	2	3	5	

6	**What does this annotation code do?** `plt.annotate('Peak', xy=(5, 10), xytext=(6, 12), arrowprops=dict(facecolor='black', shrink=0.05))` A. Adds a legend with an arrow B. Labels a point and draws an arrow C. Adds a title to the figure D. Plots an arrow without annotation	1	1	4	5	
7	**Consider the code below that creates a scatter plot with Seaborn:** `sns.relplot(x="sepal_length", y="sepal_width", data=iris, hue="species", kind="scatter", alpha=0.7)` Which of the following statements best explains the use of alpha=0.7? A. It reduces the marker size. B. It adjusts the transparency to help visualize overlapping points. C. It changes the color palette. D. It increases the line width for plot boundaries.	1	1	4	5	
8	**What does the following Matplotlib code snippet do?** `plt.text(0.5, 0.5, 'Hello, World!', fontsize=14, rotation=45,ha='center', va='center', color='red')` A. Places the text at the center of the figure with a 45° clockwise rotation B. Centers the text at (0.5, 0.5) of the axes coordinate system with 45° rotation and red color C. Rotates the text by 45° around the origin and aligns left D. Places the text at data coordinates (0.5, 0.5) with no rotation	1	1	4	5	
9	**In the following code snippet, what is the role of the `rstride` and `cstride` parameters?** `surf = ax.plot_surface(X, Y, Z, cmap='viridis', rstride=1, cstride=1)` A. They define the number of rows and columns in the data grid. B. They control the sampling (row and column stride) of the input data for rendering the surface. C. They set the resolution of the color mapping. D. They adjust the transparency of the surface.	1	2	5	5	

10	**Consider the following code snippet. What does it accomplish?**
	`fig = plt.figure()`
	`ax = fig.add_subplot(111, projection='3d')`
	`X, Y = np.meshgrid(np.linspace(-5, 5, 50), np.linspace(-5, 5, 50))`
	`Z = np.sin(np.sqrt(X**2 + Y**2))`
	`surf = ax.plot_surface(X, Y, Z, cmap='plasma', edgecolor='none')`
	A. It creates a wireframe 3D surface plot of a sine function.
	B. It generates a smooth 3D surface plot using a sine function with the 'plasma' colormap and no edge lines.
	C. It plots a scatter plot of sine values in 3D space.
	D. It creates a contour plot on a 3D axis.
		1	2	5	5	

Register Number

Set -

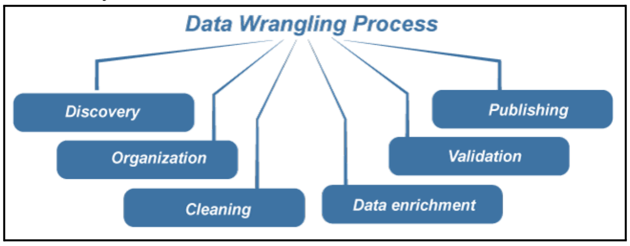
SRM Institute of Science and Technology
College of Engineering and Technology
School of Computing
SRM Nagar, Kattankulathur – 603203, Chengalpattu District, Tamil Nadu
Academic Year: 2024-25 (EVEN SEM)

Test: FT4 | Date:29-04-2025
Course Code & Title: 21CSS303T-Data Science | Duration: Two periods
Year& Sem: III Year /VI Sem | Max.Marks:50

Part – B (4 x 5 = 20 Marks)
Instructions: Answer **ANY FOUR** Questions

Q. No	Question	Marks	BL	CO	PO	PI Code
11	**Explain the process of data wrangling. Describe at least three key steps involved, discuss why data wrangling is important in data analysis, and provide a brief example to illustrate your answer.**	5	2	3	5	

- Data Wrangling is one of those technical terms that are more or less self-descriptive.
- The term "wrangling" refers to rounding up information in a certain way.



- Discovery: Before starting the wrangling process, it is critical to think about what may lie beneath your data.
- Organization: After you've gathered your raw data within a particular dataset, you must structure your data.
- Cleaning: When your data is organized, you can begin cleaning your data. Data cleaning involves removing outliers, formatting nulls, and eliminating duplicate data.
- Data enrichment: This step requires that you take a step back from your data to determine if you have enough data to proceed.
- Validation: After determining you gathered enough data, you will need to apply validation rules to your data. Validation rules, performed in repetitive sequences, confirm that data is consistent throughout your dataset.
- Publishing: The final step of the data munging process is data publishing. Data providing notes and documentation of your wrangling process and creating access for other users and applications.

Example:
Suppose you have a dataset on customer purchases with the following columns: customer_id, purchase_date, amount_spent, and coupon_used. The data may have issues like missing values in amount_spent, duplicates in customer_id, and inconsistent date formats.
Steps involved in data wrangling for this example:

1. Remove Duplicates:

	`data.drop_duplicates(subset='customer_id', inplace=True)`
	2. Handle Missing Values:
	`data['amount_spent'].fillna(data['amount_spent'].mean(), inplace=True)`
	3. Convert Date Format:
	`data['purchase_date'] = pd.to_datetime(data['purchase_date'], format='%Y-%m-%d')`					

12	**Explain how merging using indices differs from merging on columns in pandas. In your answer, describe the key steps and benefits of merging on an index and provide a brief Python code example to illustrate this method.**	5	3	3	5	

In pandas, merging can be done on column values or on index labels, depending on how your data is structured.

Merging on Columns

This is the default behavior of `pd.merge()`, where you specify one or more columns from both DataFrames to match rows.

Example:

```
import pandas as pd

df1 = pd.DataFrame({'id': [1, 2, 3], 'name': ['Alice', 'Bob', 'Charlie']})
df2 = pd.DataFrame({'id': [1, 2], 'score': [85, 90]})

merged = pd.merge(df1, df2, on='id')
print(merged)
```

```
output:
  id   name  score
0   1  Alice     85
1   2    Bob     90
```

Rows are matched where values in the id column are equal.

Merging Using Indices

When merging on indices, pandas uses the row labels (index values) to align and join rows instead of specific columns. This is done with:

- `df1.join(df2)` – by default joins on index
- `pd.merge(df1, df2, left_index=True, right_index=True)`

Benefits of Merging on Index:

1. Simplifies merging when the index holds meaningful identifiers (like time series data or grouped keys).
2. Avoids resetting indexes or adding redundant ID columns.
3. Supports hierarchical (multi-level) indices in complex datasets.

Example: Merging on Index

```
import pandas as pd

# Create two DataFrames with custom indices
df1 = pd.DataFrame({'name': ['Alice', 'Bob', 'Charlie']}, index=[101, 102, 103])
df2 = pd.DataFrame({'score': [88, 92]}, index=[101, 102])

# Merge using index
merged = df1.join(df2)  # same as df1.join(df2, how='left')
```

	print(merged) Output: name score 101 Alice 88.0 102 Bob 92.0 103 Charlie NaN The join is done based on the index values, not a column. Index 103 has no match, so NaN is inserted.					

13	Give a credit risk model for a fintech startup. The dataset includes columns: credit_score, income, loan_amount, defaulted (Yes/No), and age. Perform the following task to prepare the data for modeling.
	a. Group credit_score into risk categories: 'Poor', 'Fair', 'Good', 'Excellent'.
	b. Standardize income and loan_amount.
	c. Summarize the average loan amount and default rate for each credit risk category.
	d. Explain why binning and standardization are important in this context.
	Step-by-Step Data Preparation
	a. Group credit_score into risk categories
	categorize credit scores into bins:
	`import pandas as pd`
	`import numpy as np`
	`# Example DataFrame`
	`df = pd.DataFrame({`
	`'credit_score': [580, 660, 710, 780, 620],`
	`'income': [30000, 45000, 60000, 80000, 35000],`
	`'loan_amount': [5000, 7000, 10000, 12000, 6000],`
	`'defaulted': ['Yes', 'No', 'No', 'No', 'Yes'],`
	`'age': [25, 35, 45, 50, 30]`
	`})`
	
	`# Define credit score bins`
	`bins = [0, 599, 659, 719, 850]`
	`labels = ['Poor', 'Fair', 'Good', 'Excellent']`
	
	`# Create risk category`
	`df['risk_category'] = pd.cut(df['credit_score'], bins=bins, labels=labels)`
	`print(df)`
	
	Output:
	credit_score income loan_amount defaulted age risk_category
	0 580 30000 5000 Yes 25 Poor
	1 660 45000 7000 No 35 Good
	2 710 60000 10000 No 45 Good
	3 780 80000 12000 No 50 Excellent
	4 620 35000 6000 Yes 30 Fair
	b. Standardize income and loan_amount
	Standardization centers values to a mean of 0 and a standard deviation of 1:
	`from sklearn.preprocessing import StandardScaler`
		5	2	3	5	

```
scaler = StandardScaler()
df[['income_scaled',      'loan_amount_scaled']]      =
scaler.fit_transform(df[['income', 'loan_amount']])
print(df)
```

Output:

credit_score	income	loan_amount	defaulted	age	risk_category \
0 580	30000	5000	Yes	25	Poor
1 660	45000	7000	No	35	Good
2 710	60000	10000	No	45	Good
3 780	80000	12000	No	50	Excellent
4 620	35000	6000	Yes	30	Fair

	income_scaled	loan_amount_scaled
0	-1.100964	-1.150447
1	-0.275241	-0.383482
2	0.550482	0.766965
3	1.651446	1.533930
4	-0.825723	-0.766965

c. Summarize average loan and default rate per risk category

```
# Convert 'defaulted' to binary
df['defaulted_binary']  =  df['defaulted'].map({'Yes':
1, 'No': 0})

# Group by credit risk
summary = df.groupby('risk_category').agg({
    'loan_amount': 'mean',
    'defaulted_binary': 'mean'
}).rename(columns={
    'loan_amount': 'avg_loan_amount',
    'defaulted_binary': 'default_rate'
})

print(summary)
```

output:

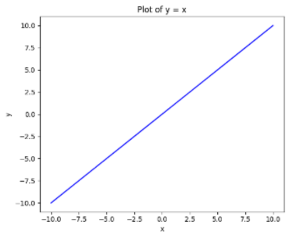
	avg_loan_amount	default_rate
risk_category		
Poor	5000.0	1.0
Fair	6000.0	1.0
Good	8500.0	0.0
Excellent	12000.0	0.0

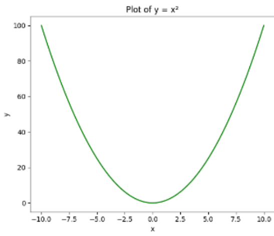
d. Why are binning and standardization important?

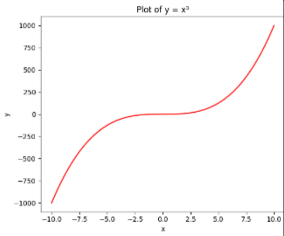
- **Binning (Grouping Credit Scores):**
  - **Simplifies modeling** by converting continuous scores into understandable categories.
  - Enables models and stakeholders to easily interpret risk levels ("Fair", "Good", etc.).
  - Helps capture **non-linear relationships** between credit score and default probability.
- **Standardization:**

	• Ensures **numerical features** like income and loan amount are on the **same scale**. • Crucial for algorithms sensitive to scale • Prevents high-magnitude variables from **dominating model weights**.					
14	**Write a Python program using Matplotlib to create a single figure with three subplots arranged in 1 row and 3 columns. Plot the following functions in each subplot:** 1. **First subplot: plot** $y=x$ 2. **Second subplot: plot** $y=x^2$ 3. **Third subplot: plot** $y=x^3$ **Use the range x=-10 to x=10 for all plots. Add titles to each subplot and label the x and y axes appropriately.**	5	3	4	5	

```
import matplotlib.pyplot as plt
import numpy as np

# Define the range of x values
x = np.linspace(-10, 10, 400)

# Define y values for each function
y1 = x
y2 = x**2
y3 = x**3

# Create a figure and subplots
fig, axes = plt.subplots(1, 3, figsize=(18, 5))  # 1
row, 3 columns

# First subplot: y = x
axes[0].plot(x, y1, color='blue')
axes[0].set_title('Plot of y = x')
axes[0].set_xlabel('x')
axes[0].set_ylabel('y')

# Second subplot: y = x^2
axes[1].plot(x, y2, color='green')
axes[1].set_title('Plot of y = x²')
axes[1].set_xlabel('x')
axes[1].set_ylabel('y')

# Third subplot: y = x^3
axes[2].plot(x, y3, color='red')
axes[2].set_title('Plot of y = x³')
axes[2].set_xlabel('x')
axes[2].set_ylabel('y')

# Adjust layout to prevent overlapping
plt.tight_layout()

# Display the plots
plt.show()

output:
```

<table>
<tr><td></td><td>



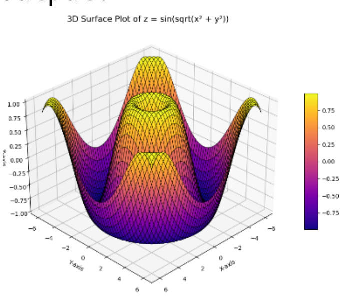

</td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td>15</td><td>

Write a Python program that demonstrates the use of 3D plotting by doing the following:

- Create a 3D plot using any mathematical function or parametric equations of your choice.
- Plot the data using a 3D axis (ax = fig.add_subplot(..., projection='3d')).
- Customize the plot using color maps, line styles, or markers for better visualization.

```
import numpy as np
import matplotlib.pyplot as plt

# Create the figure and 3D axes
fig = plt.figure(figsize=(10, 7))
ax = fig.add_subplot(111, projection='3d')

# Generate data for x, y
x = np.linspace(-6, 6, 100)
y = np.linspace(-6, 6, 100)
X, Y = np.meshgrid(x, y)
Z = np.sin(np.sqrt(X**2 + Y**2))

# Plot the surface with a color map
surf = ax.plot_surface(X, Y, Z, cmap='plasma',
edgecolor='k', linewidth=0.5, antialiased=True)

# Add a color bar for reference
fig.colorbar(surf, ax=ax, shrink=0.5, aspect=10)

# Customize labels
ax.set_title('3D Surface Plot of z = sin(sqrt(x² +
y²))', fontsize=14)
ax.set_xlabel('X-axis')
ax.set_ylabel('Y-axis')
ax.set_zlabel('Z-axis')

# Adjust view angle
ax.view_init(elev=30, azim=45)

# Show the plot
plt.tight_layout()
plt.show()
```

Output:



</td><td>5</td><td>3</td><td>5</td><td>5</td><td></td></tr>
</table>

Part – C (2 x 10 = 20 Marks)

Instructions: Answer ALL questions.

Q. No	Question	Marks	BL	CO	PO	PI Code
16 a	Consider the basic dataset that contains student details collected during admissions. The dataset contains errors and inconsistencies that need to be addressed before it can be used for reporting and visualization.	10	2	3	5	

student_id	**Name**	**Age**	**Email**	**grade**
1	John Smith	20	john.smith@email.com	A
2	SARA	-1	sara123@email.com	B+
3	Riya Kapoor	NaN	riya_kapoor@gmail	A
4	Tom Brown	19	tom.brown@email.com	None
5		22		B
6	alex johnson	0	alex.j@email.com	A+

Write Python code to perform the following data cleaning operations:

a. Identify and remove rows where the name or email is missing or blank.
b. Replace invalid age values (e.g., 0, -1, or NaN) with the **mean age** of valid entries.
c. Strip extra spaces in the name column and convert all names to proper title case.
d. Standardize grade values by replacing None with "Incomplete".
e. Remove rows with **invalid email addresses** (those without "@" or a "." after the "@").
f. Display a **summary** of the cleaned dataset using df.describe() or df.info().
g. Explain **two potential risks** if this dataset is used in its raw form for decision-making.

Python code:

```
import pandas as pd

import numpy as np

data = {

    'student_id': [1, 2, 3, 4, 5, 6],

    'Name': ['John Smith', 'SARA', 'Riya
Kapoor', 'Tom Brown', '', 'alex johnson'],

    'Age': [20, -1, np.nan, 19, 22, 0],

    'Email': ['john.smith@email.com',
'sara123@email.com', 'riya_kapoor@gmail',
'tom.brown@email.com', '',
'alex.j@email.com'],

    'grade': ['A', 'B+', 'A', None, 'B', 'A+']

}
```

```
df = pd.DataFrame(data)

print(df)
```

Identify and remove rows where the name or email is missing or blank.

```
df = df[(df['Name'].notna()) &
(df['Name'].str.strip() != '') &

        (df['Email'].notna()) &
(df['Email'].str.strip() != '')]

print(df)
```

output:

student_id Name Age Email grade

0 1 John Smith 20.0 john.smith@email.com A

1 2 SARA -1.0 sara123@email.com B+

2 3 Riya Kapoor NaN riya_kapoor@gmail A

3 4 Tom Brown 19.0 tom.brown@email.com None

5 6 alex johnson 0.0 alex.j@email.com A+

Replace invalid age values (e.g., 0, -1, or NaN) with the mean age of valid entries.

```
valid_ages = df['Age'][df['Age'] > 0]

mean_age = valid_ages.mean()

df['Age'] = df['Age'].apply(lambda x: mean_age
if pd.isna(x) or x <= 0 else x)

print(df)
```

output:

student_id Name Age Email grade

0 1 John Smith 20.0 john.smith@email.com A

1 2 SARA 19.5 sara123@email.com B+

2 3 Riya Kapoor 19.5 riya_kapoor@gmail A

3 4 Tom Brown 19.0 tom.brown@email.com None

5 6 alex johnson 19.5 alex.j@email.com A+

Strip extra spaces in the name column and convert all names to proper title case.

```
df['Name'] =
df['Name'].str.strip().str.title()

print(df)
```

Output:

student_id Name Age Email grade

0 1 John Smith 20.0 john.smith@email.com A

1 2 Sara 19.5 sara123@email.com B+

	2 3 Riya Kapoor 19.5 riya_kapoor@gmail A 3 4 Tom Brown 19.0 tom.brown@email.com None 5 6 Alex Johnson 19.5 alex.j@email.com A+				

Standardize grade values by replacing None with "Incomplete".

```
df['grade'] = df['grade'].fillna('Incomplete')

print(df)
```

Output:

student_id Name Age Email grade

0 1 John Smith 20.0 john.smith@email.com A

1 2 Sara 19.5 sara123@email.com B+

2 3 Riya Kapoor 19.5 riya_kapoor@gmail A

3 4 Tom Brown 19.0 tom.brown@email.com Incomplete

5 6 Alex Johnson 19.5 alex.j@email.com A+

Remove rows with invalid email addresses (those without "@" or a "." after the "@").

```
def is_valid_email(email):

    if "@" in email:

        local, _, domain =
email.partition("@")

        return "." in domain

    return False

df = df[df['Email'].apply(is_valid_email)]

print(df)
```

Output:

student_id Name Age Email grade

0 1 John Smith 20.0 john.smith@email.com A

1 2 Sara 19.5 sara123@email.com B+

3 4 Tom Brown 19.0 tom.brown@email.com Incomplete

5 6 Alex Johnson 19.5 alex.j@email.com A+

Explain two potential risks if this dataset is used in its raw form for decision-making.

- Misleading Insights Due to Invalid or Missing Data

If such data is used to analyze age distributions, assign age-based benefits, or segment students demographically, it could lead to biased or incorrect conclusions. For example, a scholarship program for students over 18 might be inaccurately designed based on the skewed average age.

- Communication Failures and Operational Errors

Using this data for sending admission decisions or updates could lead to failed communications or privacy issues (e.g., emails sent to the wrong recipients). This undermines trust in institutional processes and may result in lost opportunities or legal liability.

(OR)

16 b	Given two datasets:	10	3	3	5	

customers.csv

Customer_ID	Name	Age	City
C001	Alice	30	New York
C002	Bob	45	Chicago
C003	Charlie	35	San Diego

transactions.csv

Customer_ID	Date	Purchase_Amount
C001	2024-10-01	250
C002	2024-10-02	100
C004	2024-10-02	300

a. Write the code to merge customers.csv with transactions.csv using Customer_ID.
b. Explain the difference between inner, left, and outer joins in this context.
c. Use pd.concat() to vertically combine the customers and a new small DataFrame with more customer entries.
d. Explain how .combine_first() works and when it is useful.
e. Briefly explain the use of .stack() and .unstack() in reshaping hierarchical indexes

a. Code to merge customers.csv with transactions.csv using Customer_ID:

```
import pandas as pd

# Simulating the datasets
customers = pd.DataFrame({
    'Customer_ID': ['C001', 'C002', 'C003'],
    'Name': ['Alice', 'Bob', 'Charlie'],
    'Age': [30, 45, 35],
    'City': ['New York', 'Chicago', 'San Diego']
})

transactions = pd.DataFrame({
    'Customer_ID': ['C001', 'C002', 'C004'],
          'Date':  ['2024-10-01',  '2024-10-02',
'2024-10-02'],
    'Purchase_Amount': [250, 100, 300]
})

# Merging on Customer_ID
merged_df  =  pd.merge(customers,  transactions,
on='Customer_ID')
print(merged_df)
```

b. Difference between inner, left, and outer joins in this context:

Join Type	**Description**	**Result**
Inner Join (how='inner')	Only includes rows with matching Customer_ID in **both** DataFrames.	Drops C003 (no transaction) and C004 (not in customers).

Left Join (how='left')	Keeps all rows from customers, adds matching transactions if available.	Keeps C001, C002, C003; C003 will have NaNs for transaction columns.
Outer Join (how='outer')	Includes **all** rows from both DataFrames, matches where possible.	Keeps all customer and transaction entries (C001, C002, C003, C004). Unmatched parts get NaNs.

c. Combine customers with new customers using pd.concat():

```
new_customers = pd.DataFrame({
    'Customer_ID': ['C005', 'C006'],
    'Name': ['David', 'Eva'],
    'Age': [29, 41],
    'City': ['Houston', 'Seattle']
})
all_customers = pd.concat([customers, new_customers], ignore_index=True)
print(all_customers)
```

d. Explanation of .combine_first():

.combine_first() is used to fill missing values in a DataFrame with values from another DataFrame **with the same index and columns**.

If df1 has missing values and df2 has some overlapping rows/columns with non-null values, you can write:

```
df_combined = df1.combine_first(df2)
```

It fills in missing values in df1 with corresponding values from df2.

Useful for: filling gaps in incomplete data from a backup or fallback dataset.

e. Brief explanation of .stack() and .unstack() for reshaping:

- **.stack()**: Converts columns into rows; it moves the **inner level of columns to rows**, producing a Series with a MultiIndex.
    - Useful to **long-form** reshape a DataFrame.
- **.unstack()**: Does the reverse—it pivots the **inner row index level to columns**.
    - Converts a hierarchical index DataFrame into a **wide format**.

Example:

```
df = pd.DataFrame({
    'Category': ['A', 'A', 'B', 'B'],
    'Type': ['X', 'Y', 'X', 'Y'],
    'Value': [10, 20, 30, 40]
}).set_index(['Category', 'Type'])

# Stack moves 'Value' to inner row index
stacked = df.stack()
# Unstack moves 'Type' to column level
unstacked = df.unstack()
```

17 a	**Explain the functionalities and plotting techniques provided by the Seaborn library in Python. Discuss its advantages over Matplotlib and describe in detail at least three major types of plots with appropriate code examples and use cases. Also, explain how Seaborn handles datasets using built-in functions and how it integrates with Pandas for effective data visualization.**	10	2	4	5	

Seaborn is a high-level Python data visualization library built on top of **Matplotlib** and tightly integrated with **Pandas**. It provides an interface for drawing attractive and informative statistical graphics with just a few lines of code.

Key Functionalities of Seaborn

1. **Statistical Plotting:** Supports regression, distribution, categorical, and matrix plots.
2. **Automatic Aesthetics:** Uses beautiful default themes and color palettes.
3. **Pandas Integration:** Accepts DataFrames directly and uses column names for axes, hue, style, etc.
4. **Built-in Datasets:** Offers sample datasets for practice (e.g., tips, iris, penguins).
5. **Faceting:** Easily creates subplots by category (with FacetGrid, catplot, etc.).
6. **Aggregation:** Aggregates data behind the scenes for meaningful summaries (e.g., barplot shows mean by default).

Advantages Over Matplotlib

Feature	**Seaborn**	**Matplotlib**
Ease of Use	High-level API, less code	Low-level, more manual configuration
Built-in Aggregation	Yes (e.g., mean, CI)	No
Aesthetics	Better default styling and themes	Requires manual customization
Pandas Integration	Seamless (df, col names)	Requires conversion or manual mapping
Statistical Tools	Built-in regression, KDE, violin plots	Needs manual setup or SciPy

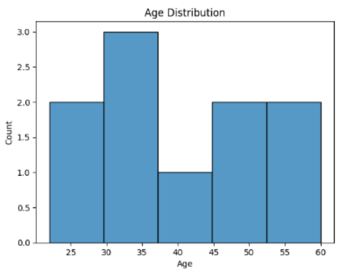
---

Three Major Plot Types with Code and Use Cases

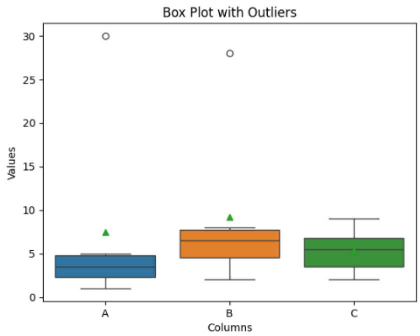
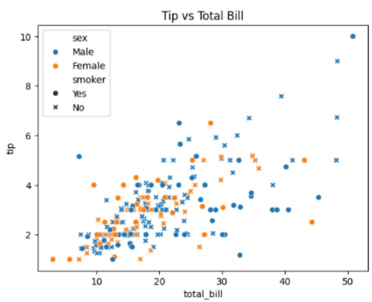
1. Distribution Plot (sns.histplot, sns.kdeplot)

Used for analyzing the distribution of a numeric variable.

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

# Sample data
df = pd.DataFrame({'Age': [22, 25, 30, 30, 35, 40, 45, 50, 55, 60]})

# Histogram with KDE
sns.histplot(df['Age'], kde=True, bins=5)
plt.title("Age Distribution with KDE")
plt.show()
```



2. Categorical Plot (sns.boxplot, sns.violinplot, sns.barplot)

	Used for comparing distributions or aggregated values across categories.
	```import seaborn as sns```
	```import pandas as pd```
	```import matplotlib.pyplot as plt```
	
	```# Sample data with outliers```
	```data = {```
	``` "A": [1, 2, 3, 4, 5, 30], # 30 is an outlier```
	``` "B": [2, 4, 6, 8, 7, 28], # 28 is an outlier```
	``` "C": [3, 6, 9, 5, 2, 7]```
	```}```
	```# Convert data to DataFrame for better visualization```
	```df = pd.DataFrame(data)```
	
	```# Create a box plot with outliers explicitly shown```
	```sns.boxplot(data=df, showmeans=True, whis=1.5)```
	
	```# Add a title and labels```
	```plt.title("Box Plot with Outliers")```
	```plt.xlabel("Columns")```
	```plt.ylabel("Values")```
	
	```# Show the plot```
	```plt.show()```
	
	
	3. Relational Plot (sns.scatterplot, sns.lineplot)
	Visualizes relationships between two numeric variables.
	# Scatterplot
	```sns.scatterplot(data=tips, x='total_bill', y='tip', hue='sex', style='smoker')```
	```plt.title("Tip vs Total Bill")```
	```plt.show()```
	
	
	● hue adds color for a third variable.					

- style changes markers for different categories.

Built-in Dataset Handling
Seaborn provides a variety of built-in datasets for practice, accessible

```
sns.get_dataset_names()            # List available datasets
df = sns.load_dataset('iris')    # Load a dataset as a DataFrame
```

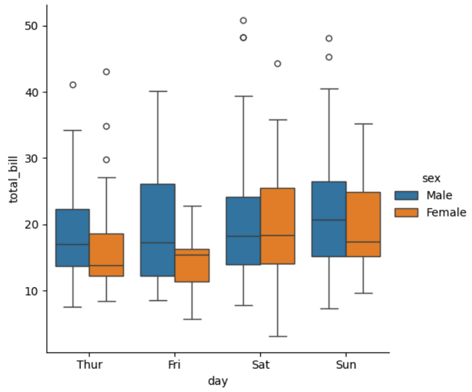
These datasets are automatically returned as **Pandas DataFrames**, making them easy to explore and plot without extra loading steps.

Integration with Pandas
Seaborn is **pandas-aware**, meaning:

- You can pass entire DataFrames to functions.
- Specify variables with column names (x='col1', y='col2').
- Use groupby-like semantics via hue, col, row for easy faceting.
- Automatically handles missing values and categorical data.

Example: Multiple plots with Pandas-style semantics

```
sns.catplot(data=tips,          x='day', y='total_bill', hue='sex', kind='box')
plt.show()
```



(OR)

Q. No.	Question	Marks				
17 b	**Describe annotation techniques used in data visualization using Python. Explain the importance of annotations in plots and demonstrate how annotations can be added using Matplotlib and Seaborn with appropriate code examples. Include different types of annotations such as text, arrows, and labels on bar charts, line plots, and scatter plots.**	10	3	5	5	

Annotations are crucial in data visualization as they help highlight important information, clarify data points, and guide interpretation. In Python, both Matplotlib and Seaborn support annotation techniques—since Seaborn builds on Matplotlib, annotations typically use Matplotlib's functions under the hood.

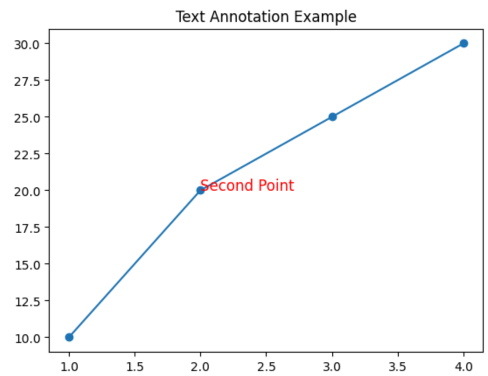
Importance of Annotations in Plots

- Emphasize key data points (e.g., max/min values, outliers).
- Explain trends in time series or correlations.
- Label elements in bar or scatter plots.
- Make plots more informative and presentation-ready.

Annotation Techniques in Matplotlib
1. Adding Text with plt.text()

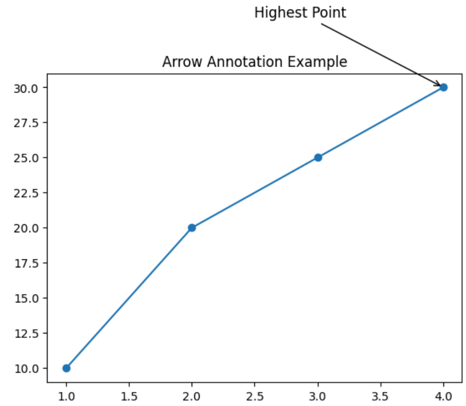
```
import matplotlib.pyplot as plt

x = [1, 2, 3, 4]
```

```
y = [10, 20, 25, 30]

plt.plot(x, y, marker='o')
plt.text(2, 20, 'Second Point', fontsize=12,
color='red')
plt.title("Text Annotation Example")
plt.show()
```



2. Using plt.annotate() with Arrows

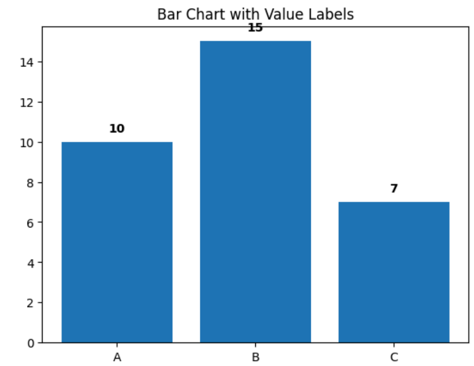
```
plt.plot(x, y, marker='o')
plt.annotate(
    'Highest Point',
     xy=(4, 30),                    # Point to
annotate
    xytext=(2.5, 35),        # Text location
             arrowprops=dict(facecolor='black',
arrowstyle='->'),
    fontsize=12
)
plt.title("Arrow Annotation Example")
plt.show()
```



3.Annotations in Bar Charts

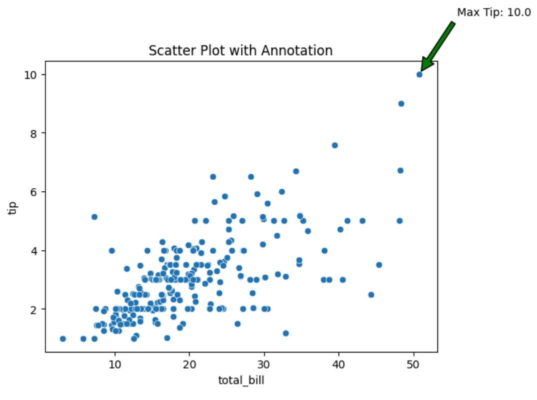
Bar Chart with Text Labels

```
categories = ['A', 'B', 'C']
values = [10, 15, 7]

plt.bar(categories, values)
for i, v in enumerate(values):
     plt.text(i, v + 0.5, str(v), ha='center',
fontweight='bold')
plt.title("Bar Chart with Value Labels")
plt.show()
```



Annotations in Scatter Plots using Seaborn

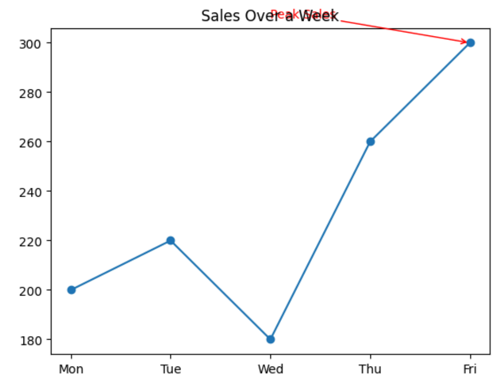
```
import seaborn as sns
import pandas as pd

# Sample data
df = sns.load_dataset('tips')
sns.scatterplot(data=df,       x='total_bill',
y='tip')

# Annotate a specific point
max_tip = df.loc[df['tip'].idxmax()]
plt.annotate(
    f"Max Tip: {max_tip['tip']}",
                     xy=(max_tip['total_bill'],
max_tip['tip']),
          xytext=(max_tip['total_bill']  +  5,
max_tip['tip'] + 2),
             arrowprops=dict(facecolor='green',
shrink=0.05)
)
plt.title("Scatter Plot with Annotation")
plt.show()
```



Annotations in Line Plots

```
days = ['Mon', 'Tue', 'Wed', 'Thu', 'Fri']
sales = [200, 220, 180, 260, 300]

plt.plot(days, sales, marker='o')
plt.title("Sales Over a Week")

# Annotate peak
plt.annotate(
    'Peak Sales',
    xy=('Fri', 300),
    xytext=('Wed', 310),
```

```
                    arrowprops=dict(arrowstyle='->',
color='red'),
    color='red'
)
plt.show()
```



Annotation Techniques

Technique	Function	Use Case
plt.text()	Add static text	Labeling bars or points
plt.annotate()	Text + arrows	Highlighting specific features
ax.bar_label()	Bar label shortcut	Labeling each bar
Seaborn + annotate	Highlights in plots	Same as Matplotlib, post-plot

Course Outcome (CO) and Bloom's level (BL) Coverage in Questions