1) Define data wrangling and explain its importance in data science.

➢ Data wrangling (or data munging) is the process of **cleaning, transforming, and preparing raw data** into a usable format for analysis or modeling.
➢ It involves handling missing values, removing duplicates, converting data types, and normalizing features.
➢ **Importance:**
  ➢ Ensures **accuracy and consistency** of data.
  ➢ Makes data **suitable for model training** and analysis.
  ➢ Saves time and improves **model performance**.
  ➢ Helps uncover patterns and insights that might be hidden in raw data.

2) What is the role of Scikit-learn in data preprocessing?

➢ Scikit-learn provides a wide range of **preprocessing tools** to prepare data before training models.
➢ It helps with **scaling, encoding, normalizing, and splitting datasets** efficiently.
➢ Key functions include:
  a) `StandardScaler`, `MinMaxScaler` - for feature scaling.
  b) `LabelEncoder`, `OneHotEncoder` - for categorical encoding.
  c) `train_test_split` - for dividing data into training/testing sets.
  These built-in tools simplify and automate preprocessing tasks, ensuring consistency and reproducibility.

3) Name three common data wrangling tasks handled using Scikit-learn utilities.

➢ Common data wrangling tasks handled using Scikit-learn include:
  1. **Data Cleaning** – handling missing or duplicate values.
  2. **Feature Scaling** – normalizing or standardizing numerical data.
  3. **Encoding Categorical Data** – converting text labels into numeric form.
  4. **Feature Extraction & Selection** – transforming raw features into useful ones.
  5. **Data Splitting** – separating datasets into training and testing sets.
  These utilities automate and streamline data preparation before modeling

4) Write a short note on datasets available in Scikit-learn.

➢ Scikit-learn provides a variety of **built-in datasets** for learning, testing, and research.
➢ Types include:
  ➢ **Toy Datasets:** e.g., `load_iris()`, `load_boston()`, `load_digits()`.
  ➢ **Real Datasets:** e.g., `fetch_20newsgroups()`, `fetch_olivetti_faces()`.

➢ **Dataset Generators:** e.g., `make_classification()`, `make_blobs()`. These datasets help users quickly experiment, test algorithms, and understand model behavior without needing external data.

5) What are estimators, transformers, and pipelines in Scikit-learn?

➢ In Scikit-learn:

- **Estimators** are objects that learn from data using the `.fit()` method — for example, `LinearRegression()` or `KMeans()`.
- **Transformers** are used to modify or preprocess data using `.fit_transform()` — for example, `StandardScaler()` or `PCA()`.
- **Pipelines** combine multiple steps (transformers and estimators) into a single workflow, ensuring that preprocessing and modeling occur sequentially and consistently

6) Explain the role of the .fit() and .transform() methods in Scikit-learn.

➢ `.fit()` → **Learns patterns or parameters** from the input data.
Example: `scaler.fit(X)` learns the mean and standard deviation.

➢ `.transform()` → **Applies those learned parameters** to modify or standardize data.
Example: `scaler.transform(X)` scales the dataset using learned values.

➢ Together, these methods separate the **learning phase** from the **application phase**, ensuring cleaner and reproducible workflows.

7) Differentiate between class and object in Scikit-learn.

Concept	Class	Object
Definition	Blueprint or template defining structure and behavior	Actual instance of a class
Example	`StandardScaler`	`scaler = StandardScaler()`
Purpose	Describes what operations are possible	Performs real data processing
Relation	Used to create objects	Created from classes

In Scikit-learn, models like `LogisticRegression` are classes, and when we instantiate them, we get objects that can `.fit()` or `.predict()`.

8) What is the purpose of using a pipeline in Scikit-learn?

➢ A **pipeline** automates the machine learning workflow by chaining preprocessing and modeling steps together.
Purpose:
- Ensures that **data transformation and model training** happen sequentially.
- Simplifies **code management** and **reproducibility**.
- Prevents **data leakage** between training and testing phases.
Example: A pipeline that scales input features, applies PCA, and then fits a classifier in one step.

9) List three practical use cases of Scikit-learn in data wrangling.

➢ **Feature Scaling:** Using `StandardScaler` or `MinMaxScaler` to normalize numeric data.
➢ **Categorical Encoding:** Using `LabelEncoder` or `OneHotEncoder` to handle text data.
➢ **Missing Value Imputation:** Using `SimpleImputer` to replace missing entries.
➢ **Data Splitting:** Using `train_test_split()` to divide datasets for training/testing.
These are essential preprocessing steps in any data science pipeline

10) What is feature extraction? Give one example.

➢ Feature extraction is the process of **converting raw data into numerical features** that capture important information.
➢ It reduces dimensionality and improves model efficiency.
➢ **Example:**
- Extracting **text features** using `CountVectorizer` or `TfidfVectorizer`.
- Extracting **image features** using edge detection or pixel intensity histograms.
This helps models learn patterns more effectively from raw data.

11) Define preprocessing and transformation with respect to data science.

➢ **Preprocessing** refers to all steps applied to **clean, standardize, and prepare** data before model training (e.g., removing nulls, scaling, encoding).
➢ **Transformation** is a specific step that **modifies the representation** of data, often through mathematical or statistical operations (e.g., log-transforming skewed data, standardization).
➢ Both are crucial to improve data quality and model accuracy.

12) Mention one real-world application where Scikit-learn is used for data cleaning.

➢ Scikit-learn is widely used in **financial fraud detection**, where large transaction datasets contain missing or inconsistent values.
➢ Example: Using `SimpleImputer` to fill missing entries, `StandardScaler` to normalize transaction amounts, and `OneHotEncoder` to process categorical fields before training a fraud detection model.
➢ Such preprocessing ensures clean, uniform data for reliable predictions

13) What is hashing? Why is it useful in machine learning?

➢ **Hashing** is a process of converting input data (like words or numbers) into a fixed-size numerical value using a **hash function**.
➢ In machine learning, it is useful for efficiently representing large or variable-length data (like text) into fixed-size numerical features, making computation and storage faster.
➢ It helps in text vectorization, memory efficiency, and scalability for large datasets.

14) Write advantages of the hashing trick.

➢ The **hashing trick** offers several advantages in machine learning:
  - **Memory Efficiency:** It avoids storing large vocabularies by mapping features directly to fixed-size indices.
  - **Speed:** It is faster than traditional vectorization since it doesn't require building a dictionary or lookup table.
  - **Scalability:** Works efficiently with large-scale or streaming data.
  - **Simplicity:** Easy to implement using tools like `HashingVectorizer` in Scikit-learn.

15) Explain the term "feature hashing" in simple words.

➢ Feature hashing, also called the **hashing trick**, is a method that converts features (like words or categories) into a fixed-length numerical vector using hash functions - useful for handling large or unknown feature spaces efficiently.

16) What are the limitations of the hashing trick?

➢ Collision Problem: Different features may map to the same index, causing information loss.
➢ No Inverse Mapping: You cannot recover the original feature names from the hashed output.

17) State properties of a good hash function.

➢ A good hash function should have the following properties:
- **Determinism:** The same input must always produce the same hash value.
- **Uniformity:** Hash values should be evenly distributed to minimize collisions.
- **Speed:** It should compute hash values quickly, even for large data.
- **Low Collision Probability:** Different inputs should rarely produce the same hash value

18) Give examples of hash functions available in Python.

➢ Common hash functions in Python include:
- `hash()` (built-in hash function)
- From the `hashlib` module:
  - `hashlib.md5()`
  - `hashlib.sha1()`
  - `hashlib.sha256()`

19) Why should a hash function be deterministic?

A hash function must be deterministic so that the **same input feature always maps to the same hash value**, ensuring consistent preprocessing for both training and testing data

20) Define uniformity in the context of hashing.

Uniformity means that a hash function should **distribute data evenly across all hash buckets**, preventing clustering or bias and reducing the chance of collisions.

21) What is HashingVectorizer in Scikit-learn?

`HashingVectorizer` is a Scikit-learn class that implements the hashing trick to convert text into numerical feature vectors using a hash function, without storing feature names — making it efficient for large-scale text data

22) Differentiate between CountVectorizer and HashingVectorizer.

Aspect	CountVectorizer	HashingVectorizer
Stores Vocabulary	Yes	No
Memory Usage	Higher	Lower
Inverse Mapping	Possible	Not possible
Collisions	None	Possibl

23) What is the main benefit of using hashing over TF-IDF vectorization?

➢ Hashing does not require building or storing a vocabulary, making it **faster, more memory-efficient**, and **suitable for streaming or very large datasets** compared to TF-IDF.

24) What does deterministic selection mean?

➢ **Deterministic selection** means that a process or algorithm always produces the **same result given the same input**.
➢ In data processing or hashing, it ensures that the same data point (e.g., a word or sample) always maps to the same feature index or output.
➢ This consistency is crucial when training and testing models, as it prevents mismatches in feature representation.
➢ Hence, determinism ensures stability, reliability, and reproducibility of results

25) Explain the use of random_state in Scikit-learn.

➢ The `random_state` parameter in Scikit-learn is used to **control randomness** in algorithms that involve random sampling, splitting, or initialization (like train-test splits, KMeans, or RandomForest).
➢ Setting a fixed `random_state` value ensures that the results remain **reproducible** across multiple runs.

26) Why is determinism important for reproducibility?

➢ Determinism ensures that every time you run an experiment with the same input and parameters, you get the **same results**.
➢ This is vital for scientific research and machine learning experiments to **verify and compare outcomes**.
➢ Without determinism, small changes in random initialization could lead to different model results.
➢ Hence, reproducibility increases **trust and reliability** in data science workflows

27) What happens if the hashing process is not deterministic?

➢ If hashing is not deterministic, the same input feature (e.g., “apple”) may map to **different indices** each time the program runs.
➢ This inconsistency causes mismatched features between training and testing data, leading to **model errors or poor predictions**.
➢ Non-deterministic hashing can also make results **irreproducible**, which is undesirable in machine learning.

28) What is the purpose of using timeit in Python?

➢ The `timeit` module measures how long it takes to execute a small piece of code.
➢ It helps in **benchmarking and performance analysis** by comparing different code implementations.

29) List any two ways to improve code performance.

➢ The following methods can help improve code performance in data wrangling and machine learning tasks:
  - **Use Vectorization:** Replace loops with NumPy or Pandas operations for faster execution.
  - **Optimize Data Structures:** Use efficient types (like arrays, dictionaries) instead of lists where possible.
  - **Use Parallel Processing:** Utilize multiprocessing or joblib to run tasks on multiple CPU cores.
  - **Profile and Optimize Hotspots:** Use tools like `timeit` or `memory_profiler` to find and improve slow sections of code

30) What are the main factors affecting code performance in data wrangling?

➢ **Data Size:** Larger datasets require more time and memory to process.
➢ **Algorithm Efficiency:** Inefficient algorithms slow down computation.
➢ **Memory Usage:** Poor memory management leads to bottlenecks.
➢ **I/O Operations:** Reading and writing large files can affect speed.

Additional factors include use of loops instead of vectorization and lack of parallel processing

31) Explain Scikit-learn's API design principles. Illustrate with examples of .fit(), .transform(), and .predict() methods.

➢ Scikit-learn follows a **consistent and simple API design** so all models and preprocessing tools work in a unified way.
➢ Key principles include:
  - **Consistency:** All estimators share common methods like `.fit()`, `.transform()`, and `.predict()`.
  - **Simplicity:** The same syntax applies across models and transformers.
  - **Composition:** Tools can be combined into pipelines.
  - **Inspection:** Model parameters and results are easily accessible through attributes like `.coef_` or `.predict()`

☐ from sklearn.preprocessing import StandardScaler
☐ from sklearn.linear_model import LinearRegression

```
from sklearn.pipeline import Pipeline

pipe = Pipeline([
    ('scaler', StandardScaler()),
    ('model', LinearRegression())
])

pipe.fit(X_train, y_train)          # Fits scaler + model
pred = pipe.predict(X_test)         # Predicts using fitted model
```

➢ Here, `.fit()` trains the model, `.transform()` scales the data, and `.predict()` generates predictions

32) Discuss the main applications of Scikit-learn in data wrangling and feature extraction with examples.

➢ Scikit-learn is widely used for **data wrangling** and **feature extraction** as part of data preprocessing.

➢ **Applications include:**

- **Handling Missing Values:** Using `SimpleImputer()` to replace NaN values.
- **Scaling and Normalization:** Standardizing data using `StandardScaler()` or `MinMaxScaler()`.
- **Encoding Categorical Data:** Applying `OneHotEncoder()` for converting text labels to numeric form.
- **Feature Extraction:** Extracting features from text using `CountVectorizer()` or images using feature extraction modules.

```
from sklearn.preprocessing import OneHotEncoder
enc = OneHotEncoder()
encoded = enc.fit_transform([['red'], ['blue'], ['red']])
```

➢ Thus, Scikit-learn simplifies data cleaning and representation before modeling.

33) What is the Hashing Trick? Explain how hashing helps in feature representation.

➢ The **Hashing Trick**, or **Feature Hashing**, is a method to convert input features (like text tokens) into a fixed-length numerical vector using a **hash function**.

➢ Each feature (e.g., word) is hashed to an index in a fixed-size array.

➢ **Advantages:**

- Avoids storing feature names or vocabularies.
- Scales efficiently with large datasets.
- Reduces memory usage.

➢ **Example:**

- If “apple” and “orange” are hashed to indices 2 and 5 respectively, they become part of a fixed-size feature vector.
- Hashing ensures constant vector size regardless of the number of unique words, making it ideal for large-scale text data

34) What is the purpose of database connectivity in Python?

➢ The purpose of database connectivity in Python is to enable programs to:

- Interact with databases (like MySQL) directly through Python code.
- Perform operations such as **creating tables**, **inserting**, **updating**, **deleting**, and **retrieving** data.
- Develop **data-driven applications** where data is stored and managed efficiently using a database.

35) List the steps required to connect Python with a MySQL database.

➢ The typical steps are:

1. Install MySQL Connector:
   - ☐ pip install mysql-connector-python
2. Import the module:
   - ☐ import mysql.connector
3. Establish connection:
   - ☐ conn = mysql.connector.connect(host="localhost", user="root", password="1234", database="college_db")
4. Create a cursor object:
   - ☐ cursor = conn.cursor()
5. Execute SQL queries using cursor.
6. Commit changes (if any updates).
7. Close the connection

36) What is the role of cursor() in MySQL-Python connectivity?

➢ The `cursor()` method creates a **cursor object**, which acts as a control structure to execute SQL commands and retrieve results.
➢ It is used to:
  a) Execute SQL statements (`cursor.execute()`).
  b) Fetch data from query results (`cursor.fetchone()`, `cursor.fetchall()`).
  c) Manage database transactions.
➢ Example:
  - ☐ cursor = conn.cursor()
  - ☐ cursor.execute("SELECT * FROM students")

37) How can you verify if the MySQL connector module is properly installed in Python?

➢ You can verify it using a simple import test in Python
  - ☐ import mysql.connector
  - ☐ print("MySQL connector installed successfully!")
➢ If there's no error message, the module is installed.
➢ Alternatively, check via command line:
  - ☐ pip show mysql-connector-python
➢ This displays version and installation path details

38) Write the command to connect to a MySQL database named college_db using mysql.connector.

```
import mysql.connector

conn = mysql.connector.connect(
    host="localhost",
    user="root",
    password="1234",
    database="college_db"
)
```

39) What is the difference between `execute()` and `executemany()` methods?

Method	Description	Example
`execute()`	Executes a single SQL query or command.	`cursor.execute("INSERT INTO student VALUES (1, 'Raj')")`
`executemany()`	Executes the same SQL command multiple times with different data values.	`cursor.executemany("INSERT INTO student VALUES (%s, %s)", [(1,'Raj'),(2,'Riya')])`

➢ **Use:**

- `execute()` → for one statement.
- `executemany()` → for bulk insert/update operations.

40) Explain in detail how to create a database table using Python and MySQL connector. Include an example.

1. **Install connector** (once): `pip install mysql-connector-python`.
2. **Import module**: `import mysql.connector`.
3. **Establish connection** using credentials (host, user, password, database).
4. **Create a cursor object** from the connection — the cursor executes SQL statements.
5. **Write a CREATE TABLE statement** (use `IF NOT EXISTS` to avoid errors if the table already exists).

6. **Execute the statement** with `cursor.execute(...)`.
7. **Commit** if needed (DDL operations are typically auto-committed by some servers, but calling `conn.commit()` is safe).
8. **Close cursor and connection** to release resources.

Important points:

- ➢ Use `VARCHAR`, `INT`, `DATE`, etc. as appropriate for columns.
- ➢ Define `PRIMARY KEY` and optionally `UNIQUE`, `NOT NULL`, `FOREIGN KEY`.
- ➢ Use parameterized queries for DML (INSERT/UPDATE/DELETE). For DDL like CREATE TABLE you usually build the SQL string directly (no user input).
- ➢ Wrap connection/operations in `try/except/finally` to handle errors and always close resources.

```
import mysql.connector
from mysql.connector import Error

try:
    # 1. Connect to MySQL
    conn = mysql.connector.connect(
        host="localhost",
        user="root",
        password="1234",
        database="college_db"   # database must exist; else create it first
    )

    if conn.is_connected():
        cursor = conn.cursor()

        # 2. Create table SQL (if not exists)
        create_table_sql = """
        CREATE TABLE IF NOT EXISTS employees (
            emp_id INT AUTO_INCREMENT PRIMARY KEY,
            first_name VARCHAR(50) NOT NULL,
            last_name VARCHAR(50),
            email VARCHAR(100) UNIQUE,
            hire_date DATE,
            salary DECIMAL(10,2)
        )
```

```
        """

        # 3. Execute the CREATE TABLE statement
        cursor.execute(create_table_sql)

        # 4. Commit (safe practice)
        conn.commit()
        print("Table 'employees' is ready.")

except Error as e:
    print("Error while creating table:", e)

finally:
    # 5. Clean-up
    if 'cursor' in locals() and cursor:
        cursor.close()
    if 'conn' in locals() and conn.is_connected():
        conn.close()
```

41) Discuss the steps involved in MySQL database connection, query execution, and result fetching in Python.

Step-by-step workflow:

1. Import connector and errors: import mysql.connector and optionally from mysql.connector import Error.
2. Establish connection: call mysql.connector.connect(host, user, password, database). Check conn.is_connected() to ensure success.
3. Create cursor: cursor = conn.cursor() (or cursor = conn.cursor(dictionary=True) if you want results as dicts).
4. Prepare SQL statement: write SELECT / INSERT / UPDATE / DELETE / DDL SQL. For statements using user data, use parameterized placeholders (%s) rather than string formatting.
5. Execute SQL:
    - Single: cursor.execute(sql, params)
    - Multiple (bulk): cursor.executemany(sql, seq_of_params)
6. Fetch results (for SELECT):
    - cursor.fetchone() - fetch a single row.
    - cursor.fetchmany(size=n) - fetch next n rows.
    - cursor.fetchall() - fetch all remaining rows.
7. Process result set: iterate rows or use them to build application objects.

8. Commit or rollback:
    - For DML (INSERT / UPDATE / DELETE) call conn.commit() to persist changes.
    - On error, call conn.rollback() to revert.
9. Close resources: cursor.close() and conn.close() (or use context managers / finally block).
10. Error handling: wrap operations in try/except to catch mysql.connector.Error and handle it (log, rollback, clean-up).

Transaction note: By default, many MySQL connectors use autocommit = False so you control transaction boundaries with commit() and rollback().

42) Compare Python's MySQL connectivity with SQLite and PostgreSQL connectivity. Discuss the advantages and limitations of each

Feature	**MySQL**	**SQLite**	**PostgreSQL**
Type	Client–server RDBMS	File-based DB	Enterprise-level RDBMS
Connector	mysql-connector-python, pymysql	Built-in sqlite3 module	psycopg2, asyncpg
Setup	Requires MySQL server installation	No server required	Requires PostgreSQL server
Speed	Fast for read/write in web apps	Extremely fast local database	Efficient with large datasets
Concurrency	Supports multiple users	Limited concurrency	Excellent concurrency handling
Transactions	Good	Good	Very strong transactional integrity
Use Cases	Websites, CMS, ERP systems	Mobile apps, small apps, prototyping	Banking, analytics, big enterprise

43) Write a Python program to perform complete CRUD operations (Create, Read, Update, Delete) on a students table in MySQL.

```
import mysql.connector
from mysql.connector import Error

def connect():
    return mysql.connector.connect(
        host="localhost",
        user="root",
        password="1234",
        database="college_db"
    )

# --------------------------
# CREATE TABLE
# --------------------------
def create_table():
    try:
        conn = connect()
        cursor = conn.cursor()
        sql = """
        CREATE TABLE IF NOT EXISTS students(
            id INT PRIMARY KEY,
            name VARCHAR(50),
            age INT,
            city VARCHAR(50)
        )
        """
        cursor.execute(sql)
        conn.commit()
        print("Table created successfully.")
    except Error as e:
        print("Error:", e)
```

```
    finally:
        cursor.close()
        conn.close()

# --------------------------
# INSERT OPERATION
# --------------------------
def insert_student(id, name, age, city):
    try:
        conn = connect()
        cursor = conn.cursor()
        sql = "INSERT INTO students VALUES(%s, %s, %s, %s)"
        cursor.execute(sql, (id, name, age, city))
        conn.commit()
        print("Student inserted.")
    except Error as e:
        print("Insert Error:", e)
    finally:
        cursor.close()
        conn.close()

# --------------------------
# READ OPERATION
# --------------------------
def read_students():
    try:
        conn = connect()
        cursor = conn.cursor()
        cursor.execute("SELECT * FROM students")
        rows = cursor.fetchall()
        for r in rows:
            print(r)
    except Error as e:
```

```
        print("Error:", e)
    finally:
        cursor.close()
        conn.close()

# --------------------------
# UPDATE OPERATION
# --------------------------
def update_student(city, id):
    try:
        conn = connect()
        cursor = conn.cursor()
        sql = "UPDATE students SET city=%s WHERE id=%s"
        cursor.execute(sql, (city, id))
        conn.commit()
        print("Record updated.")
    except Error as e:
        print("Update Error:", e)
    finally:
        cursor.close()
        conn.close()

# --------------------------
# DELETE OPERATION
# --------------------------
def delete_student(id):
    try:
        conn = connect()
        cursor = conn.cursor()
        sql = "DELETE FROM students WHERE id=%s"
        cursor.execute(sql, (id,))
        conn.commit()
        print("Record deleted.")
```

```
    except Error as e:
        print("Delete Error:", e)
    finally:
        cursor.close()
        conn.close()

# MAIN EXECUTION
create_table()
insert_student(1, "Riya", 20, "Ahmedabad")
insert_student(2, "Karan", 21, "Surat")
read_students()
update_student("Baroda", 1)
delete_student(2)
read_students()
```