1.**Explain the key features of Streamlit that make it suitable for data science and machine learning applications?**

Ans) Streamlit is a powerful Python library designed for building interactive data applications with minimal code.Its key features that make it suitable for data science and machine learning applications include:

1**)Ease of Use**- Streamlit allows quick development with simple Python scripts,eliminating the need for front-end knowledge.

2)**Interactive Widgets**-Provides built-in widgets like sliders,buttons and checkboxes to enable interactive data exploration and model tuning.

3)**Real-time Updates**-Automatically updates outputs when the script is rerun,making it ideal for live data visualization and model experimentation.

4)**Seamless Integration**-Supports popular ML and data science libraries like Pandas,Matplotlib,Seaborn,TensorFlow,and Scikit-learn.

—----------------------------------------------------------------------------------------------------------------------------

**2. How does Streamlit handle state management, and what are some ways to persist data across interactions?**

Ans) Streamlit is designed to run scripts from top to bottom on each interaction,which means it does not maintain state by default.However,it provides several ways to handle state management and persist data across interactions:

1. **st.session-state**

* **Purpose:**Stores variables that persist across user interactions
* **Usage:**Useful for caching user inputs,model parameters,or UI states
* **Example:**

import streamlit as st

if “counter” not in st.session\_state:

st.session\_state.counter=0

if st.button(“Increment”):

st.session\_state.counter +=1

st.write(“Counter:”,st.session\_state.counter)

1. **@st.cache**

* **Purpose:**Caches expensive computations or data to avoid recomputation.
* **Usage:**Ideal for loading datasets,running ML models,or fetching API data.
* **Example**

def load\_model():

return pd.read\_csv(“data.csv”)

data=load\_data()

st.write(data.head())

1. **@st.cache\_resource**

* **Purpose:** Caches resources like ML models or database connections.
* **Usage:** Ensures these resources are not reloaded on every interaction**.**
* **Example**

@st.cache\_resource

def load\_model():

return some\_ml\_model

model = load\_model()

—---------------------------------------------------------------------------------------------------------

**3.Compare Streamlit with Flask and Django. In what scenarios would you prefer Streamlit over these traditional web frameworks?**

**Ans)** **Scenarios Where Streamlit is Preferred**

1. **Quick Prototyping** – Best for rapidly developing interactive data apps without backend complexity.
2. **Data Science & Machine Learning Apps** – Ideal for visualizing datasets, running models, and displaying insights interactively.
3. **Minimal Web Development Knowledge** – If you want to build a UI without dealing with HTML, CSS, or JavaScript.
4. **Dashboard and Visualization** – Great for real-time data monitoring and analytics dashboards.

Table between Streamlit ,Flask and Django

| **Feature** | **Streamlit** | **Flask** | **Django** |
| --- | --- | --- | --- |
| **Primary Use** | Data science & ML apps | Lightweight web apps &APIs | Full-stack web apps |
| **Ease of Use** | Very easy | Moderate | Complex |
| **Interactivity** | Built in UI elements | Needs custom JS/HTML for interactivity | Uses templates & forms for UI |
| **State Management** | st.session\_state, caching | Needs Flask sessions/cookies | Uses Django session framework |
| **Deployment** | Rapid prototyping, ML dashboards | APIs, small web services | Large-scale, database-driven applications |

**—----------------------------------------------------------------------------------------------------------------------------**

**4. Describe the role of caching(@st.cache\_data and @st.cache\_resource)in Streamlit.How does it improve performance?**

**Ans)**Caching in Streamlit helps improve performance by reducing redundant computations and speeding up data processing. It stores the results of expensive operations so that they don’t need to be recalculated on every user interaction. Streamlit provides two main caching decorators:

### **1. @st.cache\_data (For Data & Computation Caching)**

* **Purpose**: Caches function outputs, primarily used for loading datasets or performing expensive computations.
* **Use Case**: Reading CSV files, API calls, or data preprocessing.
* **How It Works**: Stores the result of the function in memory, and if the function input remains the same, it fetches the cached result instead of re-running the function.
* **Example**

import streamlit as st

import pandas as pd

@st.cache\_data

def load\_data():

return pd.read\_csv("large\_dataset.csv")

df = load\_data()

st.write(df.head())

### **2. @st.cache\_resource (For Objects like Models & Connections)**

* **Purpose**: Caches resources like **ML models, database connections, and external services**.
* **Use Case**: Loading an ML model, establishing a database connection, or initializing a TensorFlow session.
* **Example**:

from transformers import pipeline

@st.cache\_resource

def load\_sentiment\_model():

return pipeline("sentiment-analysis")

model = load\_sentiment\_model()

st.write(model("Streamlit caching is amazing!"))

**5.How can you integrate a database with a Streamlit app? Provide an example using SQLite or PostgreSQL.**

**Ans)**  Streamlit can connect to databases like SQLite or PostgreSQL using Python libraries such as sqlite3 (for SQLite) or psycopg2 (for PostgreSQL).

### **1. Using SQLite in Streamlit**

SQLite is a lightweight database that requires no server setup.

#### **Steps to Integrate SQLite:**

1. Connect to the database using sqlite3
2. Create a table (if not exists)
3. Insert, retrieve, and display data

**Example**

import streamlit as st

import sqlite3

conn = sqlite3.connect("users.db")

cursor = conn.cursor()

cursor.execute("""

CREATE TABLE IF NOT EXISTS users (

id INTEGER PRIMARY KEY AUTOINCREMENT,

name TEXT,

age INTEGER

)

""")

conn.commit()

st.title("User Database")

name = st.text\_input("Enter Name")

age = st.number\_input("Enter Age", min\_value=1, step=1)

if st.button("Add User"):

cursor.execute("INSERT INTO users (name, age) VALUES (?, ?)", (name, age))

conn.commit()

st.success("User added successfully!")

cursor.execute("SELECT \* FROM users")

data = cursor.fetchall()

st.write("### User List:")

st.table(data)

conn.close()

### **2. Using PostgreSQL in Streamlit**

For PostgreSQL, use the psycopg2 or asyncpg library.

#### Steps to Integrate PostgreSQL:

1. Install psycopg2:
2. Connect to the PostgreSQL database
3. Perform CRUD operations

**Example:**

import streamlit as st

import psycopg2

DB\_CONFIG = {

"dbname": "your\_db",

"user": "your\_user",

"password": "your\_password",

"host": "localhost",

"port": "5432"

}

conn = psycopg2.connect(\*\*DB\_CONFIG)

cursor = conn.cursor()

cursor.execute("""

CREATE TABLE IF NOT EXISTS users (

id SERIAL PRIMARY KEY,

name VARCHAR(100),

age INTEGER

)

""")

conn.commit()

st.title("PostgreSQL Integration")

name = st.text\_input("Enter Name")

age = st.number\_input("Enter Age", min\_value=1, step=1)

if st.button("Add User"):

cursor.execute("INSERT INTO users (name, age) VALUES (%s, %s)", (name, age))

conn.commit()

st.success("User added successfully!")

cursor.execute("SELECT \* FROM users")

data = cursor.fetchall()

st.write("### User List:")

st.table(data)

cursor.close()

conn.close()

—------------------------------------------------------------------------------------------------------------------

**6. Discuss how you can deploy a Streamlit application. Mention at least two deployment platforms.?**

**Ans)** Streamlit apps can be deployed on various platforms, allowing users to access them online. Below are two common deployment methods:

1. **Deploying on Streamlit Community Cloud(Free & Easy)**

**Steps:**

1. **Push Your Code to GitHub**

* Ensure your app file(e.g.,app.py) and requirements.txt(list of dependencies)are in a GitHub repository

1. **Go to Streamlit Cloud**

\* Visit Streamlit Cloud.

\* Sign in with your GitHub account.

1. **Deploy the App**

* Click "New App", select your repository, and enter the app file path.
* Click "Deploy", and your app will be live.

**2. Deploying on AWS (Flexible & Scalable)**

#### **Steps:**

1. **Set Up an EC2 Instance**
   * Launch an Ubuntu EC2 instance on AWS.
   * Configure security groups to allow HTTP and SSH access.

2. **Install Dependencies**

sudo apt update && sudo apt install python3-pip

pip install streamlit

**3. Upload Your App Files**

* Use scp or Git to transfer your files to the server.

**4. Run the App**

streamlit run app.py --server.port 8501 --server.enableCORS false --server.enableXsrfProtection false

**5. Configure Reverse Proxy**

* Use NGINX to map your app to a domain.

—------------------------------------------------------------------------------------------------------------------------

**7. What are some limitations of Streamlit, and how can you overcome them when building production-grade applications?**

**Ans)** While Streamlit is great for rapid prototyping, it has some limitations when used in production environments. Here are key challenges and ways to overcome them:

### **1. Limited Multi-User Handling & Scalability**

**❌ Issue:** Streamlit apps are single-threaded and may not handle multiple concurrent users efficiently.  
 **✅ Solution:**

* Use **Session State (st.session\_state)** to manage user-specific data.
* Deploy behind a **load balancer** (e.g., using AWS, GCP, or Azure).
* Use **Gunicorn** with Streamlit to manage multiple processes.

### **2. No Built-in Authentication & Security**

**❌ Issue:** Streamlit lacks built-in authentication, making it unsuitable for secure applications.  
 **✅ Solution:**

* Use **OAuth, Firebase Authentication, or Auth0** for user authentication.
* Implement authentication using Flask or FastAPI as a backend.
* Restrict access using **streamlit-authenticator** (third-party package).

### **3. Performance Issues with Large Datasets**

**❌ Issue:** Handling large datasets can slow down the UI due to re-execution of scripts.  
 **✅ Solution:**

* Use **@st.cache\_data** for caching data operations.
* Process data with **Pandas, NumPy, or Dask** instead of iterating manually.
* Use **database queries** instead of loading large files in memory.

—----------------------------------------------------------------------------------------------------------------------------

**8. Explain the process of creating an interactive dashboard in Streamlit. What components would you use?**

Ans) **Creating an Interactive Dashboard in Streamlit**

**Import Required Libraries**  
  
import streamlit as st

import pandas as pd

import plotly.express as px

1. **Load & Cache Data**  
   Copy code  
   @st.cache\_data

def load\_data():

return pd.read\_csv("data.csv")

df = load\_data()

1. **Build UI Components**
   * **Sidebar Filters:** st.sidebar.selectbox(), st.sidebar.slider().
   * **Data Display:** st.dataframe(), st.table().
   * **Charts:** st.line\_chart(), st.bar\_chart(), plotly.express for interactive visuals.

**3. Create Layout**  
  
st.title("Interactive Dashboard")

category = st.sidebar.selectbox("Select Category", df["Category"].unique())

filtered\_df = df[df["Category"] == category]

st.write("### Data Overview")

st.dataframe(filtered\_df)

fig = px.bar(filtered\_df, x="Subcategory", y="Sales", title="Sales by Subcategory")

st.plotly\_chart(fig)

**4. Run the App**  
streamlit run app.py

**9. How would you implement user authentication in a Streamlit app? Provide possible solutions. Short?**

**Ans)Implementing User Authentication in Streamlit**

#### **1. Using streamlit-authenticator (Easy & Secure)**

import streamlit as st

import streamlit\_authenticator as stauth

credentials = {

"usernames": {

"admin": {"password": "hashed\_password"},

"user1": {"password": "hashed\_password"}

}

}

authenticator = stauth.Authenticate(credentials, "auth\_cookie", "abcdef", cookie\_expiry\_days=1)

name, authentication\_status, username = authenticator.login("Login", "main")

if authentication\_status:

st.success(f"Welcome {name}!")

authenticator.logout("Logout", "sidebar")

elif authentication\_status == False:

st.error("Invalid username or password")

#### **2. Using Firebase Authentication**

* Connect Streamlit to Firebase for user management.
* Authenticate via Google, email, or phone login.

—--------------------------------------------------------------------------------------------------------------

**10. Describe a real-world use case where you have implemented or would implement a Streamlit application.**

**Ans)** i have created a streamlit app where it contains button slider title