**Streamlit test- 21/03/2024**

**Questions:**

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

Streamlit is a powerful open-source Python library designed for quickly building interactive web applications for data science and machine learning.

The key features of streamlit are:

1. Simplicity & Ease of Use:Streamlit allows developers to create web applications with minimal code. A simple script can generate a fully functional interactive UI without requiring HTML, CSS, or JavaScript.
2. Seamless Integration with Data Science Libraries: Supports popular libraries like **Pandas, NumPy, Matplotlib, Seaborn, Plotly, Altair, and Bokeh** for easy data visualization. Works well with **TensorFlow, PyTorch, Scikit-learn, and Hugging Face Transformers** for deploying machine learning models
3. Live Model Deployment & Real-time Updates:Enables real-time updates to machine learning models and data visualizations without restarting the app.

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

Streamlit handles state management using the st.session\_state API, allowing data persistence across user interactions. By default, Streamlit reruns the entire script when a widget value changes, so st.session\_state ensures that values remain available without being reset.

1. **Session State (st.session\_state)** – A dictionary-like object to store variables across interactions. Useful for tracking user input, toggling UI elements, or caching data

2. **Caching (st.cache\_data & st.cache\_resource)** – Stores expensive computations (like loading a dataset or ML model) to avoid recalculating them on every rerun.

3. **URL Parameters (st.query\_params)** – Allows persisting data by encoding it in the URL, making it shareable.

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

| **Features** | **Streamlit** | **Flask** | **Django** |
| --- | --- | --- | --- |
| Purpose | Data apps & ML  Dashboard | Backend API  development | Full-fledged web apps |
| Ease of Use | Very Easy | Moderate | Complex |
| UI | Built-in widgets | Requires templates | Requires templates |
| State Management | st.session\_state | Cookies | ORM-based |
| Scalability | Limited | Highly scalable | Highly scalable |
| Use Case | ML Models, quick prototyping | REST API | Large-scale  applications |

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

Streamlit reruns the entire script every time a user interacts with the UI. This can slow down the app, especially if you are loading large datasets or performing expensive computations repeatedly. Caching helps optimize performance by storing and reusing results instead of recalculating them.

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

**Purpose:** Used to cache **data-related functions** (e.g., loading datasets, querying APIs).  
**Effect:** **Reuses previously computed results**, avoiding redundant computations.  
**Example Use Cases:**

1. Reading large CSV files.
2. Fetching API responses.
3. Running slow computations.

## **2. @st.cache\_resource (Resource Caching)**

**Purpose:** Used to cache **heavy resources** (e.g., database connections, ML models).  
**Effect:** Ensures **persistent objects** remain in memory **across reruns**.  
**Example Use Cases:**

1. Keeping a database connection open.
2. Loading machine learning models.
3. Caching TensorFlow/PyTorch objects.

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

Streamlit does not have built-in database support, but you can easily connect it to databases like SQLite, PostgreSQL, or MySQL using Python libraries such as sqlite3, SQLAlchemy, or psycopg2.

## **Example 1: Using SQLite with Streamlit**

SQLite is a lightweight, file-based database that is great for small-scale applications.

### **Steps to Integrate SQLite with Streamlit**

1. Connect to the database.
2. Create a table if it doesn't exist.
3. Insert and retrieve data dynamically.
4. Display the data in Streamlit

import streamlit as st

import sqlite3

# Function to connect to database

@st.cache\_resource

def get\_db\_connection():

conn = sqlite3.connect("users.db") # Creates a file-based database

return conn

# Function to create a table

def create\_table():

conn = get\_db\_connection()

cursor = conn.cursor()

cursor.execute(

"""CREATE TABLE IF NOT EXISTS users (

id INTEGER PRIMARY KEY AUTOINCREMENT,

name TEXT,

age INTEGER

)"""

)

conn.commit()

conn.close()

# Function to insert a user

def add\_user(name, age):

conn = get\_db\_connection()

cursor = conn.cursor()

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

conn.commit()

conn.close()

# Function to retrieve users

def get\_users():

conn = get\_db\_connection()

cursor = conn.cursor()

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

users = cursor.fetchall()

conn.close()

return users

# Create table if not exists

create\_table()

# Streamlit UI

st.title("SQLite Database with Streamlit")

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

age = st.number\_input("Enter Age", min\_value=0, max\_value=100, step=1)

if st.button("Add User"):

add\_user(name, age)

st.success(f"User {name} added successfully!")

# Display data

st.write("### Stored Users")

users = get\_users()

for user in users:

st.write(f" {user[0]} | \*\*{user[1]}\*\* - {user[2]} years old")

**Example 2: Using PostgreSQL with Streamlit**

PostgreSQL is a **powerful, production-ready** relational database. You can connect to it using **psycopg2**.

import streamlit as st

import psycopg2

import os

# Function to connect to PostgreSQL

@st.cache\_resource

def get\_db\_connection():

return psycopg2.connect(

dbname="your\_db\_name",

user="your\_db\_user",

password="your\_db\_password",

host="your\_db\_host",

port="your\_db\_port"

)

# Function to create a table

def create\_table():

conn = get\_db\_connection()

cursor = conn.cursor()

cursor.execute(

"""CREATE TABLE IF NOT EXISTS users (

id SERIAL PRIMARY KEY,

name TEXT,

age INTEGER

)"""

)

conn.commit()

conn.close()

# Function to insert a user

def add\_user(name, age):

conn = get\_db\_connection()

cursor = conn.cursor()

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

conn.commit()

conn.close()

# Function to retrieve users

def get\_users():

conn = get\_db\_connection()

cursor = conn.cursor()

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

users = cursor.fetchall()

conn.close()

return users

# Create table

create\_table()

# Streamlit UI

st.title("PostgreSQL Database with Streamlit")

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

age = st.number\_input("Enter Age", min\_value=0, max\_value=100, step=1)

if st.button("Add User"):

add\_user(name, age)

st.success(f"User {name} added successfully!")

# Display stored data

st.write("### Stored Users")

users = get\_users()

for user in users:

st.write(f" {user[0]} | \*\*{user[1]}\*\* - {user[2]} years old")

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

**1. Deploy on Streamlit Community Cloud (Easiest & Free)**

**Best For:** Small projects, demos, personal apps.

Steps to Deploy on Streamlit Community Cloud

1. **Push your code to GitHub** (Make sure your repo has app.py and requirements.txt).  
2. **Go to Streamlit Community Cloud** and log in.  
3. **Click "New App" → Select your GitHub repo**.  
4. **Enter branch name and app file (e.g., app.py)**.  
5. **Click "Deploy"** – Your app will be live in a few minutes

**2. Deploy on Heroku (Flexible & Free Tier Available)**

**Best For:** Medium-scale apps, more control over deployment.

Steps to Deploy on Heroku

**Prerequisites:** Install Heroku CLI, create an account.

1. **Push your code to GitHub or keep it locally.**2. **Create a Procfile (to specify how to run the app)**:

web: streamlit run app.py --server.port=$PORT

3. **Create a requirements.txt** with dependencies:

nginx

streamlit

pandas

numpy

psycopg2 # If using PostgreSQL

4. **Login to Heroku from the terminal:**

heroku login

5. **Create a new Heroku app:**

heroku create your-app-name

6. **Deploy the app to Heroku:**

git add .

git commit -m "Deploy Streamlit app"

git push heroku main

7. **Open your app in the browser:**

heroku open

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

While Streamlit is great for rapid prototyping and interactive data apps, it has some limitations when used for production-grade applications.

1. Limited State Management: Streamlit reruns the entire script on every interaction, making it hard to maintain complex user states.No built-in session handling (like Flask/Django).

Solution: Use **st.session\_state** to persist values across interactions.

2. Performance Issues for Large Apps: Slow performance when handling large datasets or ML models.Full rerun on interaction slows down response time.

solution:Use caching (@st.cache\_data, @st.cache\_resource) to avoid recomputation,Optimize large computations outside the main script using background jobs.

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

Streamlit makes it easy to create interactive dashboards for data visualization and analysis. Here’s a step-by-step guide on how to build one.

1.Install Streamlit & Required Libraries:

Command: pip install streamlit pandas matplotlib seaborn plotly

2. Load & Preprocess Data:

import streamlit as st

import pandas as pd

@st.cache\_data # Cache the data to avoid reloading

def load\_data():

df = pd.read\_csv("sales\_data.csv") # Replace with your file

df["Date"] = pd.to\_datetime(df["Date"]) # Ensure correct date format

return df

df = load\_data()

st.write(df.head()) # Display the first few rows

3. Add Sidebar Filters:

st.sidebar.header("Filter Options")

# Date Range Filter

start\_date = st.sidebar.date\_input("Start Date", df["Date"].min())

end\_date = st.sidebar.date\_input("End Date", df["Date"].max())

# Product Filter

product\_filter = st.sidebar.multiselect("Select Products", df["Product"].unique(), default=df["Product"].unique())

# Apply Filters

filtered\_data = df[(df["Date"] >= pd.to\_datetime(start\_date)) &

(df["Date"] <= pd.to\_datetime(end\_date)) &

(df["Product"].isin(product\_filter))]

4. Visualize Data Using Charts

import plotly.express as px

st.subheader("📊 Sales Trend Over Time")

fig = px.line(filtered\_data, x="Date", y="Sales", color="Product", title="Sales Trend")

st.plotly\_chart(fig)

5. Add Interactive Widgets:

if st.checkbox("Show Raw Data"):

st.write(filtered\_data)

if st.button("Show Summary Statistics"):

st.write(filtered\_data.describe())

6. Deploy the Dashboard: command:streamlit run app.py

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

**1**. Using st.session\_state for Basic Authentication (Simple & Quick)

import streamlit as st

# Fake user database

USER\_CREDENTIALS = {"admin": "password123", "user": "userpass"}

def login():

st.title(" User Login")

username = st.text\_input("Username")

password = st.text\_input("Password", type="password")

if st.button("Login"):

if username in USER\_CREDENTIALS and USER\_CREDENTIALS[username] == password:

st.session\_state["authenticated"] = True

st.session\_state["user"] = username

st.success(f" Welcome, {username}!")

else:

st.error("Invalid username or password")

def main\_app():

st.title(" Welcome to the Main App")

st.write(f"Hello, {st.session\_state['user']}!")

if "authenticated" not in st.session\_state:

login()

elif st.session\_state["authenticated"]:

main\_app()

**2.** Using streamlit-authenticator for Secure Login (Recommended)

pip install streamlit-authenticator

import streamlit as st

import streamlit\_authenticator as stauth

# User credentials (hashed passwords generated using stauth.Hasher)

credentials = {

"usernames": {

"admin": {"name": "Admin", "password": "$2b$12$8WZ..."}, # Hashed password

"user": {"name": "User", "password": "$2b$12$QmA..."} # Hashed password

}

}

# Initialize authenticator

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

# Login widget

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

if authentication\_status:

st.write(f"Welcome, \*\*{name}\*\*")

st.button("Logout", on\_click=authenticator.logout)

elif authentication\_status is False:

st.error(" Incorrect username or password")

elif authentication\_status is None:

st.warning("Please log in")

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

Use Case: Crop Yield Prediction using Streamlit

Objective: Develop a Streamlit application for predicting crop yield using machine learning models.

Components:

· Upload Dataset using st.file\_uploader() for CSV files.

· Data Preprocessing with pandas and st.dataframe().

· Model Integration – Load trained RandomForestRegressor from .pkl.

· User Inputs – Select crop and region using st.selectbox().

· Visualization – Display results using st.pyplot() or st.plotly\_chart().

Why Streamlit:

Easy deployment of ML models.

Interactive UI for user input and predictions.

No need for complex web development.