STREAMLIT TEST (21/03/2025)

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

Streamlit is a powerful tool for data science and machine learning because it's simple, interactive, and requires only Python—no need for frontend or backend development. It provides built-in widgets like sliders, buttons, and file uploaders, making it easy to create interactive applications. It also supports popular data visualization libraries like Matplotlib, Seaborn, and Plotly, helping you explore and present data effectively.

For machine learning, you can load models, take user inputs, and display predictions in real time. It auto- refreshes when code changes, has caching to improve performance, and makes deployment easy with options like Streamlit Cloud. Overall, it’s great for quickly prototyping and sharing ML projects without worrying about complex web development.

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

Streamlit is **stateless** by default, meaning it reruns the script from top to bottom on every interaction. This can be an issue when you need to persist data across interactions, like keeping user selections or ML model outputs.

To handle state management, Streamlit provides:

* 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.

1. **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 |

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

* **Caching** helps improve performance by storing results of expensive computations, reducing redundant processing.
* **@st.cache\_data**:
  + Caches function outputs, useful for data processing and API calls.
  + Re-executes only when input parameters change.
* **@st.cache\_resource**:
  + Caches objects like ML models, database connections, or large datasets.
  + Useful when you want to reuse expensive resources across multiple interactions.
* **Performance Benefits**:
  + Reduces execution time for repetitive tasks.
  + Optimizes resource usage, making apps run smoother.

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

Steps to integrate a database in Streamlit:

* 1. Connect to the database.
  2. Create a table if it doesn’t exist.
  3. Insert and retrieve data using Streamlit widgets.

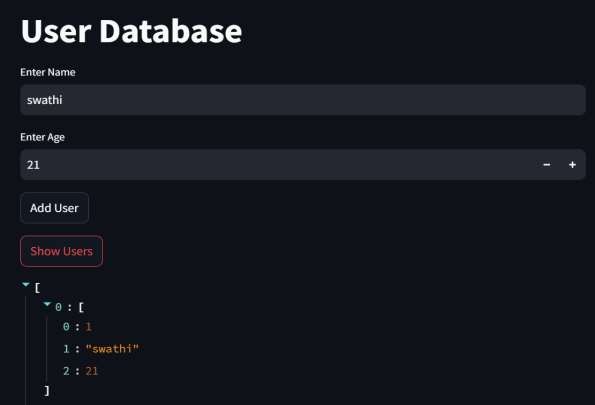
1. import sqlite3
2. import streamlit as st 6.
3. # Connect to SQLite database
4. conn = sqlite3.connect("my\_database.db")
5. cursor = conn.cursor() 10.
6. # Create table if not exists
7. cursor.execute("CREATE TABLE IF NOT EXISTS users (id INTEGER PRIMARY KEY,

name TEXT, age INTEGER)")

1. conn.commit()

14.

1. # Streamlit UI
2. st.title("User Database") 17.
3. # Insert data
4. name = st.text\_input("Enter Name")
5. age = st.number\_input("Enter Age", min\_value=1, step=1) 21.
6. if st.button("Add User"):
7. cursor.execute("INSERT INTO users (name, age) VALUES (?, ?)", (name, age))
8. conn.commit()
9. st.success("User added!")
10. # Fetch and display data
11. if st.button("Show Users"):
12. cursor.execute("SELECT \* FROM users")
13. users = cursor.fetchall()
14. st.write(users)
15. # Close connection
16. conn.close() 35.



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

# Deploying a Streamlit App

🔹 **Streamlit Community Cloud** (Free, simple)

* Push code to **GitHub** and deploy via Streamlit Cloud.

🔹 **Heroku**

* Add requirements.txt and Procfile.
* Run:

bash CopyEdit git init

heroku create my-app git push heroku main

* Open app with heroku open.

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

**Limitations of Streamlit & Solutions**

🔹 **Limited Customization → Use HTML/CSS & JavaScript for advanced UI.**

🔹 **Single-threaded (Not Ideal for Heavy Loads) → Deploy with FastAPI/Flask as backend.**

🔹 **Limited Authentication → Use OAuth, Firebase, or third-party authentication.**

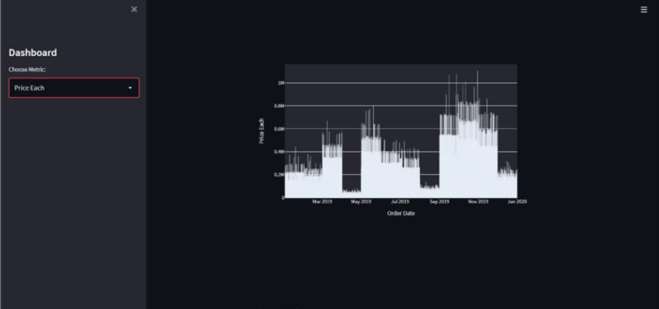
🔹 **No Built-in Database Support → Connect to PostgreSQL, Firebase, or MongoDB.**

🔹 **Deployment Constraints → Use Docker, AWS, or Heroku for scalable deployment.**

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

**Components to Create an Interactive Dashboard in Streamlit**

st.sidebar() – Sidebar navigation.

st.dataframe() – Display tabular data. st.plotly\_chart() – Data visualizations. st.selectbox(), st.slider() – User inputs. st.expander() – Collapsible sections.

import streamlit as st import pandas as pd

import plotly.express as px

st.sidebar.title("Dashboard")

option = st.sidebar.selectbox("Choose Metric:", ["Category", "Price Each"])

df = pd.read\_csv("AllYearsales.csv")

1. **How would you implement user authentication in a Streamlit app? Provide possible solutions.**
2.  **st.text\_input() + Password Validation** – Basic authentication.
3.  **Streamlit-Authenticator** – Secure login with hashed passwords.
4.  **OAuth (Google, GitHub, etc.)** – Use third-party authentication.
5.  **JWT (JSON Web Tokens)** – Secure token-based authentication.
6.  **Session State** – Maintain login sessions.
7. import streamlit as st
8. import streamlit\_authenticator as stauth

11.

1. # Define credentials
2. credentials = {"usernames": {"admin": {"name": "Admin", "password": "hashed\_password"}}}
3. # Initialize authenticator
4. authenticator = stauth.Authenticate(credentials, "app\_cookie", "auth\_key", cookie\_expiry\_days=1)
5. name, authentication\_status, username = authenticator.login("Login", "main")
6. if authentication\_status:
7. st.write(f"Welcome {name}!")
8. authenticator.logout("Logout", "sidebar")
9. elif authentication\_status == False:
10. st.error("Username or password is incorrect")
11. elif authentication\_status == None:
12. st.warning("Please enter your credentials") 27.
13. **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**.