Streamlit→21/03/2025 Test

Q1) Explain the key features of Streamlit that make it suitable for data science and machine learning applications.

- It is very easy to use with python syntax one can easily achieve UI creation
- Automatically renders the UI such as widgets, buttons, checkboxes, etc.
- Changes in the script reflect instantly without restarting the server.
- Supports libraries like Matplotlib, Seaborn.
- It has a built in cashe that Improves performance by memoizing expensive computations.
- Easy integration with TensorFlow, Scikit-learn, and PyTorch.
- Can be deployed on Streamlit Community Cloud, AWS, GCP, or Heroku.

Q2) How does Streamlit handle state management, and what are some ways to persist data across interactions?

Streamlit runs scripts from top to bottom on each user interaction, making state management non-persistent by default. However, persistence can be achieved using:

1. **Session State (** `st.session_state` **)** – Maintains variables across reruns.
2. **Caching (** `@st.cache_data` **&** `@st.cache_resource` **)** – Stores expensive computations and resources.

3. **Database Storage** – Saves and retrieves data from databases like SQLite, PostgreSQL.
4. **External Storage (e.g., Redis, Firebase)** – Stores data across multiple user sessions.

Q3) Compare Streamlit with Flask and Django. In what scenarios would you prefer Streamlit over these traditional web frameworks?

Feature	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 application

Q4) Describe the role of caching `(@st.cache_data and @st.cache_resource)` in Streamlit. How does it improve performance?

- Caching stores the results of slow function calls, so they only need to run once. This makes your app much faster and helps with persisting objects across reruns.

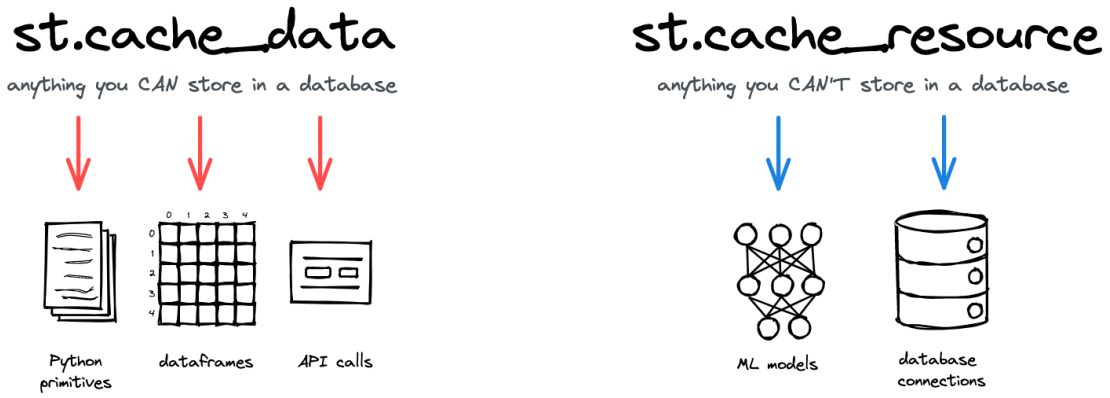
▼ `st.cache_data`

- It is the recommended way to cache computations that return data such as loading a DataFrame from CSV or transforming a NumPy array, etc.
- Its creates a new copy of the data at each function call, making it safe against mutations and race conditions.

▼ `st.cache_resource`

- It is the recommended way to cache global resources like ML models or database connections – unserializable objects that you don't want to load multiple times.

- Using it, you can share these resources across all reruns and sessions of an app without copying or duplication.



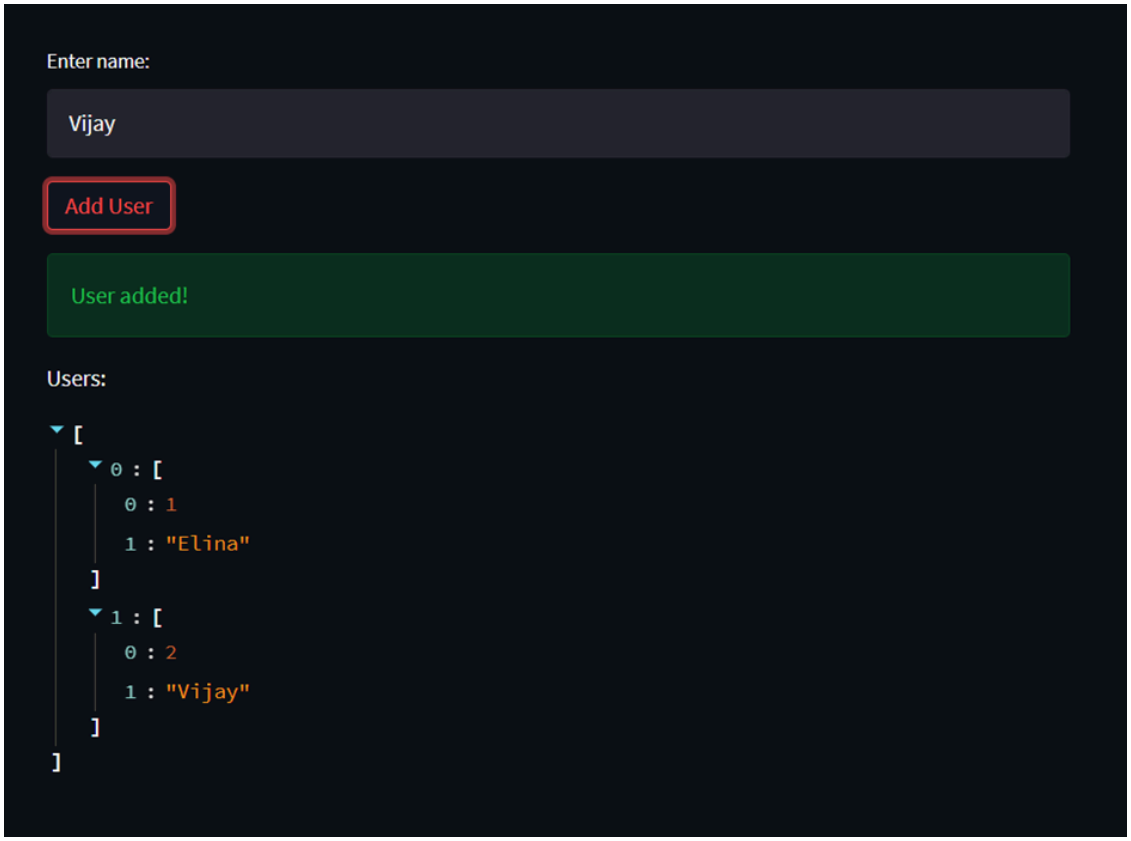
Q5) How can you integrate a database with a Streamlit app? Provide an example using SQLite.

```
import streamlit as st
import sqlite3

# Database connection
conn = sqlite3.connect("users.db")
cursor = conn.cursor()
cursor.execute("CREATE TABLE IF NOT EXISTS users (id INTEGER PRIMARY K
conn.commit()

# UI
name = st.text_input("Enter name:")
if st.button("Add User"):
    cursor.execute("INSERT INTO users (name) VALUES (?)", (name,))
    conn.commit()
    st.success("User added!")

# Display Data
users = cursor.execute("SELECT * FROM users").fetchall()
st.write("Users:", users)
```



Q6) Discuss how you can deploy a Streamlit application. Mention at least two deployment platforms.

- **Streamlit Community Cloud** (Free, simple)
  - Push code to GitHub and deploy via Streamlit Cloud
- **Heroku**
  - Requires `requirements.txt` and `Procfile`
  - Deployment steps:

```
git init
heroku create my-app
git push heroku main
```

Q7) What are some limitations of Streamlit, and how can you overcome them when building production-grade applications?

Limitation	Workaround
No multi-page support	Use `st.experimental_rerun()` or `st.switch_page()`
Not suitable for large-scale apps	Combine with FastAPI for backend APIs
Limited session handling	Use `st.session_state` or store sessions in Redis
Minimal authentication	Use Firebase Auth or OAuth integrations

Q8) Explain the process of creating an interactive dashboard in Streamlit. What components would you use?

Components:

1. `st.sidebar()` – Sidebar navigation.
2. `st.dataframe()` – Display tabular data.
3. `st.plotly_chart()` – Data visualizations.
4. `st.selectbox()`, `st.slider()` – User inputs.
5. `st.expander()` – Collapsible sections.

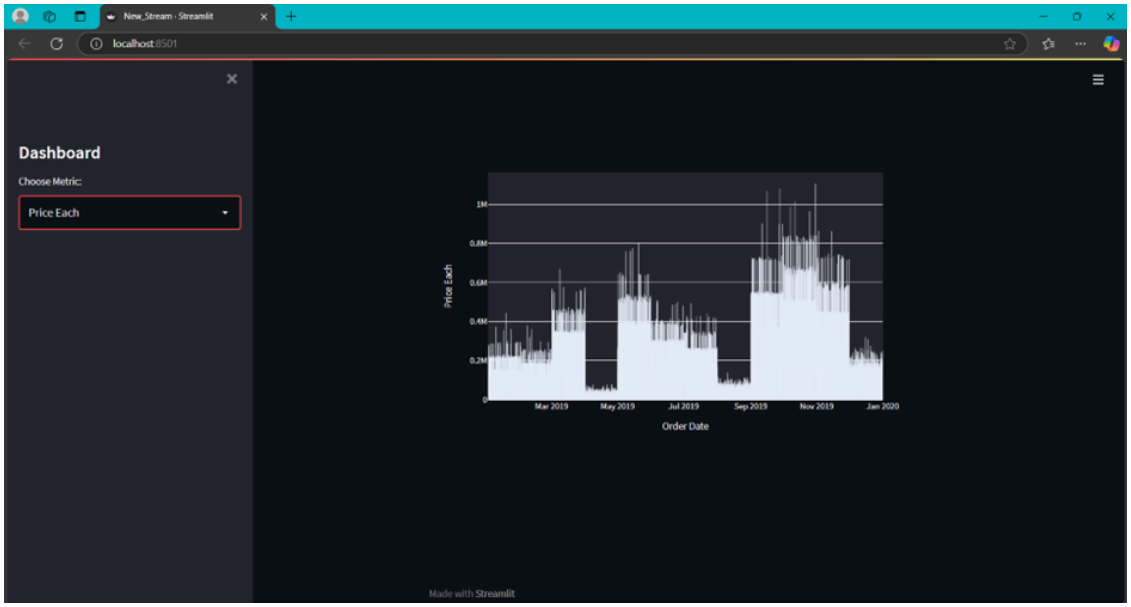
```
import streamlit as st
import pandas as pd
import plotly.express as px

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

df = pd.read_csv("AllYearsales.csv")

if option == "Category":
    fig = px.line(df, x="Order Date", y="Category")
else:
    fig = px.bar(df, x="Order Date", y="Price Each")

st.plotly_chart(fig)
```



Q9) How would you implement user authentication in a Streamlit app? Provide possible solutions.

- **Using `st.text_input()` & Password Hashing**

```
python
CopyEdit
import streamlit as st

users = {"admin": "password123"}

username = st.text_input("Username")
password = st.text_input("Password", type="password")

if st.button("Login"):
    if username in users and users[username] == password:
        st.success("Login Successful")
    else:
        st.error("Invalid Credentials")
```

- **OAuth Authentication (e.g., Firebase, Auth0)**
  - Use Streamlit OAuth library (`streamlit-authenticator`).
  - Authenticate via Google, GitHub, etc.

Q10) Describe a real-world use case where you have implemented or would implement a Streamlit application.

Use Case: Machine Learning Model Deployment for Sign Language Recognition

- **Objective:** Deploy a trained Mediapipe model for real-time sign language recognition.
- **Components:**
  - Upload user hand gesture images.
  - Use OpenCV for real-time webcam input.
  - Process frames with Mediapipe.
  - Display recognized gestures dynamically.

Why Streamlit?

- Quick deployment of ML models.
- Interactive UI for real-time gesture detection.
- Eliminates complex web development.