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

Streamlit is designed to make it easy for data scientists and machine learning engineers to create interactive web applications. Some key features include:

- Simple and Fast Development: Uses a Pythonic syntax, allowing developers to turn data scripts into web apps with minimal effort.
- Interactive Widgets: Provides built-in widgets (slider, select box, button) for user interaction.
- Live Code Updates: Auto-refreshing UI when code changes, making rapid prototyping easier.
- Seamless Integration: Works well with libraries like Pandas, Matplotlib, Seaborn, Plotly, and TensorFlow.
- Caching for Performance: Caches expensive computations using @st.cache_data and @st.cache_resource.
- Media Support: Can display images, audio, and video using image, audio, and video.
- Markdown and LaTeX Support: Allows rich text formatting and mathematical expressions.

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

Since Streamlit runs scripts top-to-bottom on each interaction, state management requires additional handling. Some ways to persist data across interactions:

- Session State (st.session_state): Stores variables that persist across reruns.
- Caching (@st.cache_data & @st.cache_resource): Stores computation results to avoid redundant processing.
- Database Storage: Saves persistent data externally in databases like SQLite, PostgreSQL, or Firebase.
- Local Storage (Pickle, JSON, CSV): Saves intermediate results locally

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

Comparison: Streamlit vs. Flask vs. Django

- Streamlit is best suited for quickly building data-driven applications with minimal effort. It is highly interactive, requires no frontend development, and is optimized for data visualization and machine learning model deployment. However, it is not designed for handling complex user authentication or large-scale web applications.
- Flask is a lightweight micro-framework that provides flexibility in building web applications. It is more suitable for developers who need control over routing, database management, and API development. Unlike Streamlit, Flask requires manual integration of frontend technologies (HTML, CSS, JavaScript) for interactivity.
- Django is a full-fledged web framework that follows the Model-View-Template (MVT) pattern. It is ideal for building scalable, database-driven applications with robust authentication, security, and user management. Django is better suited for production-grade applications with complex business logic.

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

Caching in Streamlit (@st.cache_data & @st.cache_resource)

Caching improves performance by storing expensive computations:

- @st.cache_data: Caches function output (data processing results).
- @st.cache_resource: Caches resources like database connections, API clients.

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

```python
import streamlit as st
import sqlite3

conn = sqlite3.connect("nothing.db")
cursor = conn.cursor()

cursor.execute("CREATE TABLE IF NOT EXISTS users (id INTEGER PRIMARY KEY, name TEXT, age
INTEGER)")
conn.commit()

st.title("nothing Database App")

name = st.text_input("Enter Name")
age = st.number_input("Enter Age", min_value=1, max_value=100, step=1)

if st.button("Add User"):
    cursor.execute("INSERT INTO users (name, age) VALUES (?, ?)", (name, age))
    conn.commit()
    st.success("User added successfully!")

st.subheader("User List")
users = cursor.execute("SELECT * FROM users").fetchall()
for user in users:
    st.write(user)

conn.close()
```

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

Deploying a Streamlit Application

1. Streamlit Community Cloud – Easiest method, free hosting for public apps.
2. Heroku – Use Procfile to run Streamlit with Gunicorn.
3. AWS (EC2, Lambda, ECS) – Requires Dockerization.
4. Google Cloud Run – Serverless deployment using Docker.

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

Limitations of Streamlit & solutions to overcome them

1. Stateless by Default

Streamlit re-runs the script on every interaction, losing previous state.

Solution: Use st.session_state to persist values across interactions or store data in a database.

2. Limited UI Customization

Lacks flexibility for complex frontend designs.

Solution: Use st.markdown() for custom HTML/CSS or embed JavaScript components via st.components.v1.html().

3. Not Ideal for Large-Scale Applications

Not designed for full-stack apps or multi-page navigation.

Solution: Integrate with Flask/FastAPI for backend logic and use Streamlit for interactive dashboards.

4. Performance Issues with Large Data

Handling large datasets can slow down execution.

Solution: Use @st.cache_data to cache computations, load data in chunks, or use optimized data formats (e.g., Parquet).

5. No Built-in Authentication

Lacks native login or user authentication support.

Solution: Implement authentication using Firebase, Auth0, or a custom session-based login system.

6. Limited Multi-User Support

Single-threaded execution can cause slow performance in multi-user environments.

Solution: Use database-backed session storage and deploy on scalable infrastructure like Kubernetes or AWS Lambda.

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

Components to Use:

- st.sidebar – For filters & navigation.
- st.selectbox, st.slider – User inputs.
- st.plotly_chart, st.dataframe – Data visualization.
- st.map – Geospatial data visualization.

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

st.title("Dashboard")

df = pd.read_csv("data.csv")
category = st.sidebar.selectbox("Select Category", df["Category"].unique())
```

```python
filtered_df = df[df["Category"] == category]
fig = px.bar(filtered_df, x="Subcategory", y="Value")

st.plotly_chart(fig)
```

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

Using st.session_state with a Simple Login

```python
import streamlit as st

users = {"admin": "password5655"}

if "authenticated" not in st.session_state:
    st.session_state.authenticated = False

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.session_state.authenticated = True
        st.success("Login Successful")
    else:
        st.error("Invalid")

if st.session_state.authenticated:
    st.write("Welcome")
```

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

A Streamlit app that analyzes network traffic logs, applies machine learning models, and visualizes anomalies for proactive threat detection.

Key Features:

Upload network log files (CSV) for analysis.
Apply machine learning models like XGBoost for anomaly detection.
Display high-risk IPs, attack patterns, and model predictions.
Interactive visualizations using Plotly for real-time monitoring.