Question 1 What is Streamlit and what are its main features?

Streamlit is an open source framework designed for creating and sharing data applications in Python. It is particularly popular among data scientists and analysts because it allows for the rapid development of web applications for data visualization, machine learning, and data processing without needing extensive web development knowledge.

Main Features of Streamlit

1. Ease of Use

- Pythoncentric You can build interactive applications using just Python scripts. No need for HTML, CSS, or JavaScript.
- Declarative Syntax Write straight forward Python code to describe the UI elements and their behavior. Streamlit takes care of rendering them in the browser.

2. Interactive Widgets

- Streamlit provides a variety of widgets like sliders, buttons, text inputs, and file uploaders to create interactive applications.
- Users can interact with these widgets, and the app can dynamically update based on user inputs.

3. Data Display

- Supports various data display formats, including tables, charts, and metrics.
- Integrates seamlessly with popular data visualization libraries like Matplotlib, Plotly, and Altair.

4. Live Updates

- Streamlit apps can automatically update in realtime as users interact with them or as data changes.
- Use caching to optimize performance and avoid redundant computations.

5. Layout Customization

- Offers simple methods to organize the layout of your app, including columns, expandable sections, and tabs.

6. Media Support

- Easily embed images, videos, and audio files in your application.
- Supports rendering of Markdown and LaTeX for documentation and mathematical expressions.

7. Deployment and Sharing

- Streamlit Cloud provides a platform for deploying and sharing Streamlit apps with a wider audience.
- Deployment to other platforms like Heroku, AWS, and Google Cloud is also possible.

8. Integration with Machine Learning

- Streamlit is wellsuited for building and sharing machine learning models. You can visualize model outputs, display feature importances, and create interactive model demos.

9. Community and Ecosystem

- A growing community of users and contributors.
- Extensive documentation and a variety of community contributed components and templates to help get started quickly.

Streamlit is particularly valuable for prototyping and sharing data-driven applications quickly and effectively, making it a favorite tool among data professionals for creating interactive dashboards and applications.

Question 2 How Does Streamlit Differ from Other Web Application Frameworks Like Flask or Django?

- Development Approach:
    - Streamlit: Focuses on rapid prototyping with a declarative, Python-only syntax.
    - Flask/Django: General-purpose frameworks requiring HTML, CSS, and JavaScript for front-end development.
- Use Case:
    - Streamlit: Tailored for data science applications, interactive visualizations, and machine learning demos.
    - Flask/Django: Suitable for full-scale web applications with complex back-end requirements.
- Complexity:
    - Streamlit: Minimal setup and easy to use; ideal for quick, iterative development.
    - Flask/Django: More complex with a steeper learning curve; suitable for production-ready applications.
- Architecture:
    - Streamlit: Single-script applications with automatic re-execution on interaction.
    - Flask/Django: MVC architecture with separate files for routing, views, and templates.
- Deployment and Hosting:
    - Streamlit: Simplified deployment on Streamlit Cloud and other platforms.
    - Flask/Django: Requires more configuration for deployment and hosting.

- Front-End Development:
    - Streamlit: No need for front-end code; everything is handled in Python.
    - Flask/Django: Requires knowledge of front-end technologies for designing the user interface.

In summary, Streamlit is designed for simplicity and speed in building data-centric applications, while Flask and Django are more suited for developing complex, production-grade web applications.

Question 3. What are some typical use cases for Streamlit?

Typical Use Cases for Streamlit:

1. Data Visualization Dashboards:
    - Creating interactive dashboards for visualizing data using libraries like Matplotlib, Plotly, and Altair.
2. Machine Learning Model Demos:
    - Building interactive interfaces to showcase machine learning models, including input data, predictions, and performance metrics.
3. Exploratory Data Analysis (EDA):
    - Developing tools for EDA that allow users to interactively explore datasets.
4. Real-Time Data Monitoring:
    - Setting up dashboards for monitoring real-time data streams, such as financial markets or IoT sensor data.
5. Data Entry and Collection Tools:
    - Creating forms and interfaces for data entry, data collection, and survey applications.
6. Prototyping and Rapid Development:
    - Quickly prototyping ideas and applications, especially for data-driven projects.
7. Reporting and Presentations:

- Designing interactive reports and presentations that can be shared with stakeholders.

8. Geospatial Applications:
    - Building apps for visualizing and interacting with geospatial data using libraries like Folium or PyDeck.

Streamlit is particularly well-suited for applications that require quick iteration and interactive data exploration.

Question 4. How do you create a simple Streamlit app?

Answer: Follow these 3 steps

1. Install Streamlit:

pip install streamlit

2. Create a Python Script (app.py):

```
import streamlit as st

st.title("Hello, Streamlit!")

st.write("This is a simple Streamlit app.")
```

3. Run the App:

streamlit run app.py

This script creates a basic app with a title and a text message.

Question 5. Can you explain the basic structure of a Streamlit script?

Answer: Basic Structure of a Streamlit Script:

1. Import Streamlit: import streamlit as st

2. Title and Headers:

st.title("App Title")

st.header("Section Header")

3. Widgets for User Input: user_input = st.text_input("Enter text:")

4. Display Data/Visualizations:

st.write("You entered:", user_input)

st.line_chart(data)

5. Run the App: streamlit run app.py

Question 6. How do you add widgets like sliders, buttons, and text inputs to a Streamlit app?

1 Slider:

slider_value = st.slider("Select a range:", 0, 100)

st.write("Slider value:", slider_value)

2. Button

if st.button("Click me"):

st.write("Button clicked!")

3. Text Input:

text = st.text_input("Enter text:")

```
st.write("You entered:", text)
```

7.How does Streamlit handle user interaction and state management?

Answer: Not Attempted

8.What are some best practices for organizing and structuring a Streamlit project?

Answer: Not Attempted

9.How would you deploy a Streamlit app locally?

1. Create the Streamlit Script (e.g., app.py):

```
import streamlit as st

st.title("My Local Streamlit App")

st.write("This app is running locally.")
```

2. Open a Terminal and Navigate to the Script Directory: cd path/to/your/script

3. Run the Streamlit App: streamlit run app.py

4. Access the App in Your Browser: Open a web browser and go to http://localhost:8501 to view your running Streamlit app.

10.Can you describe the steps to deploy a Streamlit app?

Answer: Not Attempted

11.What is the purpose of the requirements.txt file in the context of Streamlit deployment?

Answer: Not Attempted