Name :- Vijay Gurung

ID Number : MST03 – 0070

Date : 27-June-24

Topic : * Streamlit - Deployment Module *

Q.1 What is Streamlit and what are its main features?

Answer :-

Streamlit is a free and open-source Python library that allows data scientists and AI & Machine learning engineers to quickly build and deploy interactive web applications without requiring extensive web development knowledge. Some of Streamlit's main features include:

1.Streamlit enables you to create stunning-looking applications with just a few lines of Python code, without needing to know HTML, CSS, or JavaScript.

2. Streamlit provides a wide range of interactive widgets such as buttons, sliders, text inputs, and dropdowns that allow users to interact with our application.

3. Streamlit integrates seamlessly with popular data visualization libraries like Matplotlib, Plotly, and Altair, making it easy to create interactive charts and plots within your application.

Streamlit app that allows users to input their height and weight and displays their BMI:

```
+ Code  + Text

import streamlit as st

st.title("BMI Calculator")

height = st.number_input("Enter your height (in meters)", min_value=0.0, step=0.01)
weight = st.number_input("Enter your weight (in kg)", min_value=0.0, step=0.1)

bmi = weight / (height ** 2)

if bmi < 18.5:
    st.write(f"Your BMI is {bmi:.2f}, which means you are underweight.")
elif bmi < 25:
    st.write(f"Your BMI is {bmi:.2f}, which means you are at a healthy weight.")
elif bmi < 30:
    st.write(f"Your BMI is {bmi:.2f}, which means you are overweight.")
else:
    st.write(f"Your BMI is {bmi:.2f}, which means you are obese.")
```

Q.2.How does Streamlit different from other web application frameworks like Flask or Django?

Answer :-

Streamlit differs from web frameworks like Flask and Django in several key ways are as follows :-

1.Streamlit is designed to be extremely easy to use, even for those with minimal web development experience. It requires just a few lines of Python code to create interactive web apps.

2.Streamlit is optimized for building data science, AI & Machine learning applications. It integrates seamlessly with popular data visualization libraries like Matplotlib and Plotly.

3. Streamlit enables very fast development and deployment of data apps, as it automatically handles the underlying web development tasks.

4.While Streamlit is simple to use, it provides less flexibility and customization options compared to Flask and Django.

Scalability: Streamlit is best suited for small to medium-sized data apps. For large, complex web applications, Flask and Django may be more scalable options.

Below is the example of a Streamlit app that displays a scatter plot of random data:

```
import streamlit as st
import numpy as np

# Generate sample data
data = np.random.randn(1000, 2)

# Create Streamlit app
st.title("Streamlit Scatter Plot")
st.scatter_chart(data)
```

3.What are some typical use cases for Streamlit?

Answer :-

Streamlit makes it easy to create interactive data visualizations and dashboards. So, we can use popular libraries like Matplotlib, Plotly and Altair to create charts, graphs and plots that update in real-time as the user interacts with the app.

```
import streamlit as st
import pandas as pd
import altair as alt

# Load data
df = pd.read_csv('data.csv')

# Create chart
chart = alt.Chart(df).mark_bar().encode(
    x='category',
    y='value'
)

# Display chart in Streamlit app
st.title("Data Visualization Example")
st.altair_chart(chart, use_container_width=True)
```

4.How do you create a simple Streamlit app?

Asnswer :-

The create a simple Streamlit Apps below is the process we need to follow :

(A) Install Streamlit: If you haven't already, install Streamlit using pip:

bash

pip install streamlit

(B) Create a new Python file: Create a new Python file (e.g., app.py) and import the necessary libraries:

python

import streamlit as st

(C) Add content to our app: Use Streamlit's functions to add content to your app. For example:

python

st.title("Hello, Streamlit!")

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

(D) Run our app: Save the file and run the following command in your terminal:

bash

streamlit run app.py

This will launch our app in a web browser.

```python
import streamlit as st
import random

st.title("Random Number Generator")

if st.button("Generate Number"):
    number = random.randint(1, 100)
    st.write(f"Your random number is: {number}")
else:
    st.write("Click the button to generate a random number.")
```

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

Answer :-

Below is the explanation with the proper code about the basic structure of a Streamlit script.

To demonstrates the basic structure of a Streamlit script, including setting the page configuration, adding content and interactive widgets, loading and displaying data and adding conditional logic and buttons.

```
import streamlit as st
import pandas as pd

st.set_page_config(
    page_title="My Streamlit App",
    page_icon="🚀",
    layout="wide",
    initial_sidebar_state="expanded"
)

st.title("Welcome to my Streamlit App")
st.write("This is a simple example of a Streamlit app.")

name = st.text_input("What's your name?", "Type your name here...")
age = st.number_input("What's your age?", min_value=0, step=1)

data = pd.read_csv("data.csv")
st.dataframe(data)
st.line_chart(data["column_name"])

if age < 18:
    st.write(f"Hello, {name}! You are a minor.")
else:
    st.write(f"Hello, {name}! You are an adult.")

if st.button("Click me!"):
    st.write("You clicked the button!")
```

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

Answer :-

We demonstrate the use of various widgets, including sliders, buttons, text inputs, dropdowns, checkboxes and file uploads. Each widget is added using a specific Streamlit function, and the user's input is then displayed or used in some way.

The key to adding widgets is to understand the different functions provided by Streamlit, such as st.slider(), st.button(), st.text_input(), st.selectbox(), st.checkbox(), and st.file_uploader(). These functions allow you to create interactive elements in your Streamlit app.

```
[ ]  import streamlit as st

     st.title("Streamlit Widgets Example")

     # Slider
     value = st.slider("Select a value", 0, 100, 50)
     st.write(f"The selected value is: {value}")

     # Button
     if st.button("Click me!"):
         st.write("You clicked the button!")

     # Text input
     name = st.text_input("What's your name?", "Type your name here...")
     st.write(f"Hello, {name}!")

     # Dropdown
     options = ["Option 1", "Option 2", "Option 3"]
     selected_option = st.selectbox("Select an option", options)
     st.write(f"You selected: {selected_option}")

     # Checkbox
     agree = st.checkbox("I agree to the terms and conditions")
     if agree:
         st.write("You agreed to the terms and conditions.")

     # File upload
     uploaded_file = st.file_uploader("Choose a file")
     if uploaded_file is not None:
         st.write(f"You uploaded: {uploaded_file.name}")
```

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

Answer :-

Streamlit handles user interaction and state management through a combination of widgets, callbacks, and session state. Below is the short explanation with a coding demo:

```
import streamlit as st

if "count" not in st.session_state:
    st.session_state.count = 0

def increment_count():
    st.session_state.count += 1

st.title("Counter App")
st.write(f"Current count: {st.session_state.count}")

if st.button("Increment", on_click=increment_count):
    pass
```

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

Answer :-

By following these practices, We can create a well-structured, scalable and maintainable Streamlit project.

Step : 1

Modularize :-

Break your app into smaller, reusable modules or components. This makes your code more maintainable.

app.py

import components.data_loader, components.visualizations, components.widgets

Step : 2

Use a Config File :-

config.py

DATA_SOURCE = "data.csv"

Step 3 :

Implement Caching

Use Streamlit's built-in caching to improve performance for expensive operations.

@st.cache_data

def load_data():

    return pd.read_csv("data.csv")

Step 4 :

Separate Concerns

Keep your Streamlit script focused on UI and interaction. Move complex logic to separate modules.

app.py

import services.data_service

insights = services.data_service.get_insights()

Step 5 :

Use Relative Imports

Use relative imports to reference other parts of your codebase

app.py

from .components.data_loader import load_dat

Q.9 How would you deploy a Streamlit app locally?

Answer :-

To deploy a Streamlit app locally, We can follow these below steps :

(A) Install Streamlit: If you haven't already, install Streamlit using pip:

pip install streamlit

(B) Create your Streamlit app: Write your Streamlit app in a Python script, for example, app.py:

```
import streamlit as st
st.title("My Streamlit App")
st.write("This is a simple Streamlit app running locally.")
```

(C) Run the Streamlit app: In your terminal, navigate to the directory containing your app.py file and run the following command:

streamlit run app.py

Finally because of this it will launch your Streamlit app in a new web browser window.

```python
import streamlit as st
import pandas as pd

# Load some data
data = pd.DataFrame({
    'A': [1, 2, 3, 4, 5],
    'B': [10, 20, 30, 40, 50]
})

# Create the Streamlit app
st.title("Local Streamlit App Demo")
st.write("This is a simple Streamlit app running locally.")

# Display the data
st.dataframe(data)

# Add a button
if st.button("Click me!"):
    st.write("You clicked the button!")
```

When we run this script using streamlit run app.py, it will launch a local Streamlit app in your web browser. We can then interact with the app, such as clicking the button and see the changes reflected in real-time.

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

Answer :-

Below is the steps that we can follow to deploy a Streamlit app :

(A) Prepare the Streamlit app: Develop your Streamlit app locally and ensure it's working correctly.

(B) Choose a deployment platform: Popular options include Streamlit Sharing, Heroku, AWS, or your own server.

(C) Set up the deployment environment: Install Streamlit and any other required dependencies in your deployment environment.

(D) Configure the app: Depending on the platform, you may need to set environment variables, configure a web server, or create a deployment script.

(E) Deploy the app: Push your Streamlit app code to your deployment platform, either manually or through automation.

```
# app.py
import streamlit as st
import pandas as pd

# Your Streamlit app code here

if __name__ == '__main__':
    st.title("My Streamlit App")
    st.write("This is a deployed Streamlit ap

 # requirements.txt
streamlit
pandas

# Procfile
web: sh setup.sh && streamlit run app.py

# setup.sh
mkdir -p ~/.streamlit/
echo "\
[general]\n\
email = \"your-email@domain.com\"\n\
" > ~/.streamlit/credentials.toml
echo "\
[server]\n\
headless = true\n\
enableCORS=false\n\
port = $PORT\n\
" > ~/.streamlit/config.toml
```

In this example, we have done followed some steps :-

(A) Prepare the Streamlit app in app.py.
(B) Create a requirements.txt file to list the dependencies.
(C) Create a Procfile to specify the command to run the app on Heroku.
(D) Create a setup.sh script to configure the Streamlit server settings.
(E) Then, we can deploy the app to Heroku by connecting the GitHub repository or using the Heroku CLI.

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

Answer :-

The requirements.txt file is used to specify the dependencies required to run a Streamlit application in a deployment environment.

When deploying a Streamlit app, the deployment platform (e.g., Heroku, AWS, etc.) needs to know which Python packages and their versions to install in order to run your app. The requirements.txt file serves this purpose.

For Example :-

a. streamlit
b. pandas
c. numpy
d. matplotlib

This requirements.txt file specifies that the deployment environment needs to install the streamlit, pandas, numpy, and matplotlib packages in order to run the Streamlit app.

When you deploy your Streamlit app, the deployment platform will read the requirements.txt file and install the specified dependencies, ensuring that your app can run correctly in the deployment environment.

This is important because the deployment environment may not have the same Python packages installed as your local development environment. The requirements.txt file ensures that all the necessary dependencies are installed, making the deployment process smoother and more reliable.

```
[ ]  headless = true\n\
     enableCORS=false\n\
     port = $PORT\n\
     " > ~/.streamlit/config.toml
```

```
# app.py
import streamlit as st
import pandas as pd
import matplotlib.pyplot as plt

# Load some data
data = pd.DataFrame({
    'A': [1, 2, 3, 4, 5],
    'B': [10, 20, 30, 40, 50]
})

# Create a simple Streamlit app
st.title('My Streamlit App')
st.write(data)

fig, ax = plt.subplots()
ax.plot(data['A'], data['B'])
st.pyplot(fig)
```

In this example, the Streamlit app requires the streamlit, pandas and matplotlib packages to run. To deploy this app, we would create a requirements.txt file in the same directory as the app.py file, with the following contents:

a. streamlit
b. pandas
c. matplotlib

This requirements.txt file ensures that the deployment environment installs the necessary packages, allowing our Streamlit app to run correctly.