edunet foundation

HANDBOOK

Development and Data Visualization using Streamlit

Table of Contents

Table of Contents

Course Objectives

After completing this handbook, learner will be able to

- Apply modern web development skills using AI-assisted tools like GitHub Copilot and Cursor AI.
- Enable rapid prototyping and data visualization through Streamlit for building interactive web applications.
- Introduce AI integration techniques using tools like Gemini API for smart feature development (e.g., chatbots, content generation).
- Develop proficiency in version control and cloud deployment using Git, GitHub, and Streamlit Community Cloud.
- Empower faculty to transfer industry-relevant skills to students, enhancing employability and innovation in academic projects.

Chapter 2: Development and Data Visualization using Streamlit

Learning Outcomes:

- Understand the structure and workflow of a multi-page Streamlit application.
- Design and build intuitive application pages.
- Use forms and input widgets to capture and process user input within an interactive app.
- Display dynamic data using basic charts and understand where and when to use each chart type effectively.

Introduction to Streamlit

About Streamlit

Streamlit is an open-source Python library that allows users to create interactive web applications for data and machine learning projects. It has been designed to make it easy for people who work with data, such as analysts, researchers, and educators, to present their work as a web application without learning complex web development technologies. In traditional web application development, knowledge of multiple languages such as HTML, CSS, and JavaScript is required. Streamlit simplifies this by allowing users to build an application entirely using Python.

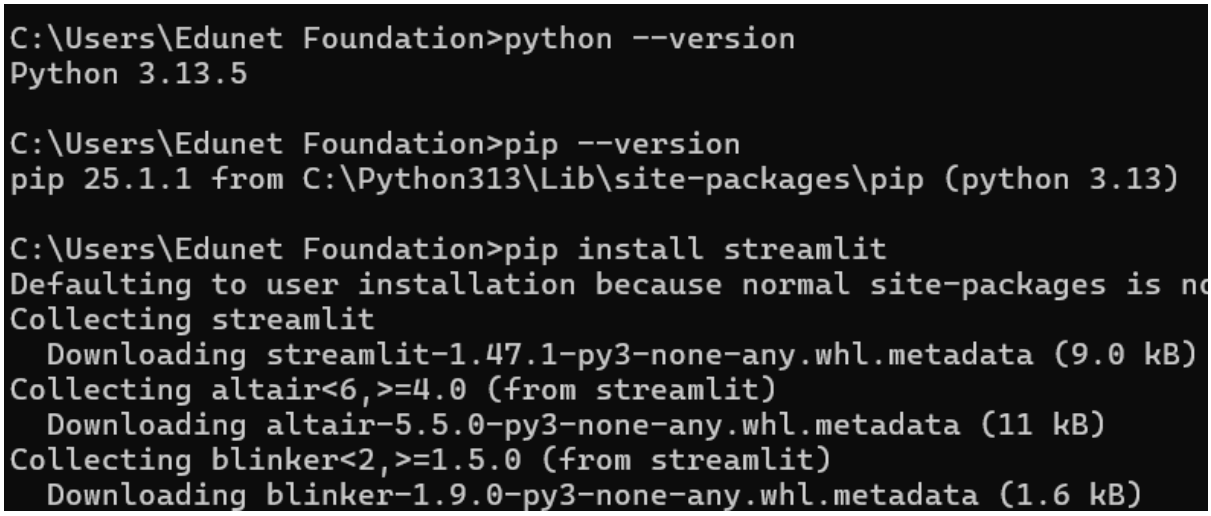
Streamlit is particularly useful for people who already know Python but do not have experience in web development. For example, a teacher who analyses student performance data using Python can use Streamlit to quickly build a simple dashboard that shows results through charts and interactive inputs. This can be done without needing to hire a web developer or learn front-end technologies. The main idea behind Streamlit is that writing Python scripts and creating an interactive web interface should be as simple as writing the code for data analysis itself. Streamlit works in a way where each script is run from top to bottom every time a user interacts with the application, so that the output on the screen is always updated. This makes the process straightforward and reduces complexity. Another important feature of Streamlit is that it supports a wide variety of visualization libraries like Matplotlib, Plotly, Altair, and others. This enables users to present their data in different forms such as line charts, bar charts, maps, and interactive plots. All these outputs can be displayed on a single page or across multiple pages of a Streamlit application.

A real-world example that explains Streamlit can be understood by thinking of how a faculty member may want to share a data-driven project with students. Without Streamlit, the faculty might prepare the analysis in Python, generate some static graphs, take screenshots, paste them into slides, and then present them during class. With Streamlit, the same faculty can prepare a small application where students can open a browser, interact with the data by applying filters, and see updated results instantly. This improves engagement and helps in understanding the data better.

Installing and Launching First Application

Once the purpose of Streamlit is clear, the next step is to install it and create a first basic application. Streamlit is a Python package, so it can be installed using the command line. Before starting the installation, Python and pip (Python package manager) need to be available in the system. The installation steps are as follows:

1. Open the command prompt on your computer.
2. Verify that Python is installed by typing ***python --version*** and pressing Enter. This will show the version of Python that is currently installed.
3. Verify that ***pip*** is installed by typing ***pip --version*** and pressing Enter. Pip is required to install Python libraries.
4. Install Streamlit by typing ***pip install streamlit*** and pressing Enter. This command will download and install Streamlit and its required components.



Source: Screenshot

After the installation is complete, Streamlit can be tested by creating a small application. The process of creating and launching a first application involves the following steps:

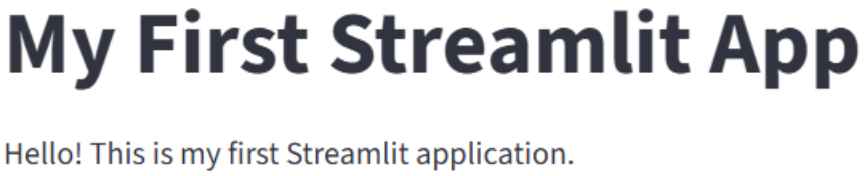
1. Create a new Python file with a simple name. For example, the file can be called ***app.py***.
2. Open this file in any code editor and write the following code:

```
import streamlit as st

st.title("My First Streamlit App")

st.write("Hello! This is my first Streamlit application.")
```

3. Save the file.
4. Open the command prompt, move to the directory where this file is saved using the cd command, and then type the following command:

   python -m streamlit run app.py

5. Press Enter. This will start a local server and open the default web browser with a new tab that displays the application.



Source: Screenshot

When the application is launched, a simple page will appear in the browser with the title "***My First Streamlit App***" and the text "***Hello! This is my first Streamlit application.***" This confirms that Streamlit has been installed and is running successfully.

This first application is very simple but demonstrates the workflow of Streamlit:

1. Writing Python code in a file
2. Running it using the Streamlit command
3. Seeing the output as an interactive web page in the browser.

This process is repeated whenever an application is created. More elements, such as charts, text boxes, buttons, and layouts, can be added to make the application richer and more interactive. An important point to remember is that each time changes are made in the code file, the browser page can be refreshed to see the changes. Streamlit automatically tracks changes in the file and updates the application, which makes the development cycle very quick. This feature is similar to how, in a classroom, any updates made to a board are immediately visible to students without any extra effort.

Streamlit vs. Traditional Web Frameworks

Streamlit and traditional web frameworks are two different approaches to creating web applications. While both can be used to build applications, they are designed for different purposes and audiences. The key points of difference are shown in the table below:

Aspect	Traditional Web Frameworks	Streamlit
Technologies Required	Requires knowledge of multiple technologies such as HTML for structure, CSS for styling, JavaScript for interactivity, and a server-side language like Python.	Requires only Python. Streamlit takes care of creating the interface automatically.
Learning Curve	Steeper learning curve as multiple tools and frameworks must be learned and integrated.	Easier for Python users because no additional web development skills are needed.
Development Time	Development can be slow due to the need to handle design, layout, and front-end logic separately.	Fast development because developers focus on Python code, and the application updates automatically.
Flexibility and Features	Offers complete control over all aspects of the application, making it suitable for very complex and feature-rich applications.	More focused on creating data-driven applications, dashboards, and machine learning interfaces. Limited flexibility for very complex features.
Deployment	Requires careful configuration and deployment steps. Often involves integrating a web server and managing multiple files.	Streamlined deployment process. Applications can be deployed quickly with simpler commands or cloud services.
Best Use Case	Building large-scale, multi-functional applications such as portals, e-commerce sites, or social networks.	Building analytical dashboards, quick prototypes, and interactive machine learning tools.

Application Page

The application page is the screen that opens in a web browser when a Streamlit app is launched. It acts as the front face of the application where users interact with the content. In simple terms, it is similar to the first slide of a presentation or the first page of a report that is shown to an audience. Everything that is displayed or interacted with in a Streamlit app appears on this page.

When a Streamlit application is run using the ***python – m streamlit run filename.py*** command, Streamlit starts a local server. After a few seconds, it automatically opens the browser window with the application page. This page shows the elements that are defined in the Python file, such as titles, text, charts, tables, and widgets. The layout of the page follows the order of commands written in the script.

The page is dynamic, which means that whenever a change is made in the script and saved, the application page can be refreshed to see the updated version. This is very useful because the developer does not need to restart the entire application every time a small change is made. The changes appear in the browser almost instantly.

The application page in Streamlit is structured from top to bottom. The commands written in the Python file are executed in sequence, and their outputs are placed on the page in that same order. This process is similar to how notes written line by line on a board in a classroom appear in sequence for students to follow.

Steps for Creating a Simple Application Page

The steps below show how a simple application page can be created:

1. Open a code editor and create a new Python file, for example ***dashboard.py***.
2. Import the Streamlit library at the top of the file:

   import streamlit as st

3. Use Streamlit functions to add elements to the page:

   st.title("Student Performance Dashboard")

   st.header("Overview")

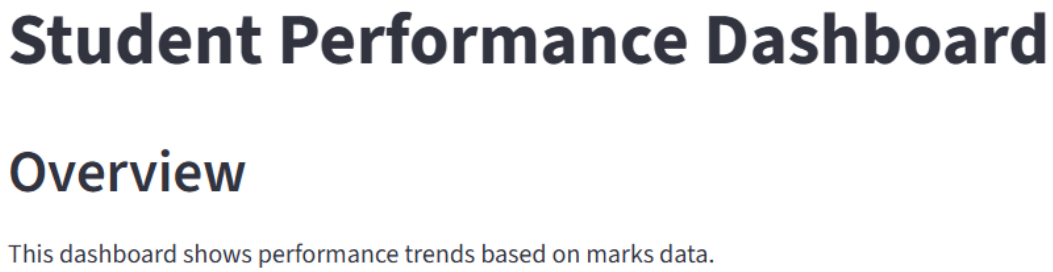
   st.write("This dashboard shows performance trends based on marks data.")

4. Save the file.
5. Open the command prompt and move to the folder where the file is saved.
6. Run the application using the command:

   python -m streamlit run app.py

7. The browser will open and display a page with the title, header, and text that were written in the code.

Output



Source: Screenshot

Remember:

- The application page opens automatically in the browser when a Streamlit script is run.
- The contents of the page follow the sequence of commands written in the script.
- Any update to the script file can be reflected immediately by refreshing the browser.
- Multiple elements such as text, images, tables, charts, and widgets can be added to the application page.

Page Configuration

By default, when a Streamlit application runs, the page in the browser uses a standard layout with the Streamlit title and icon. For many applications, it is useful to customize the appearance of this page. Streamlit provides a function called ***st.set_page_config()*** that is used at the beginning of the script to control how the page looks and behaves.

The page configuration allows developers to change the page title, the icon displayed on the browser tab, the layout of the page, and whether the sidebar is open or closed by default. These settings make the application look more organized and aligned with the purpose of the project.

The ***st.set_page_config()*** function should be placed at the very top of the Streamlit script. It can take parameters such as:

- **page_title**: Sets the title of the browser tab.
- **page_icon**: Sets the small icon that appears in the browser tab. This can be an emoji or an image. ***Make sure you have the icon in the same folder with the python file.***

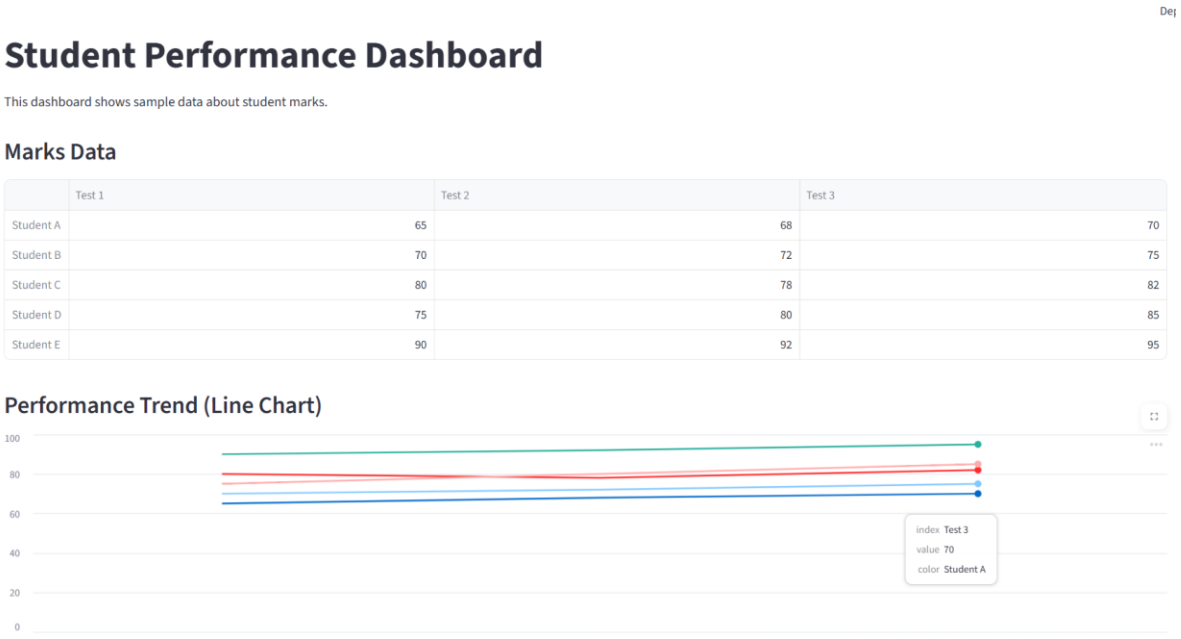
- **layout**: Can be set to "centered" or "wide". A wide layout allows more content to be shown horizontally, which is useful for dashboards with charts.
- **initial_sidebar_state**: Controls whether the sidebar is "expanded" or "collapsed" when the page first loads.

Example of Page Configuration

The following code demonstrates how to configure the page:

```
import streamlit as st
import pandas as pd
import numpy as np

# Page configuration
st.set_page_config(
    page_title="Student Dashboard",
    page_icon="icon.png",
    layout="wide",
    initial_sidebar_state="expanded"
)

# Page content
st.title("Student Performance Dashboard")
st.write("This dashboard shows sample data about student marks.")

# Create sample data
# Suppose there are marks of 5 students over 3 tests
data = {
    "Test 1": [65, 70, 80, 75, 90],
    "Test 2": [68, 72, 78, 80, 92],
    "Test 3": [70, 75, 82, 85, 95]
}

# Convert data into a DataFrame
df = pd.DataFrame(data, index=["Student A", "Student B", "Student C", "Student
D", "Student E"])

# Display the DataFrame
st.subheader("Marks Data")
st.dataframe(df)

# Display a line chart
st.subheader("Performance Trend (Line Chart)")
st.line_chart(df.T)
```

Output



Source: Screenshot

When this script is run, the browser tab will display “Student Dashboard” as the title and show an icon of a chart. The page will open in wide layout, which is more suitable for showing tables and charts side by side, and the sidebar will be expanded.

Why Page Configuration is Important

- It gives the application a more professional appearance.
- It ensures that the layout is suited to the type of content being displayed.
- It helps the users of the application to easily identify the purpose of the page by looking at the title and icon.

Project Directory Structure

As Streamlit applications grow beyond a single file, it becomes important to organize the project files in a clear and logical way. A well-planned directory structure makes it easier to manage code, reuse components, and share the application with others. For a small one-page Streamlit application, all the code can be placed in a single file. However, when an application includes multiple pages, images, datasets, and custom modules, keeping everything in one file can make it confusing and difficult to maintain.

Basic Structure for a Simple Project

A simple Streamlit project might look like this:

project_folder/

app.py

Here, app.py contains all the Streamlit code. This structure is fine when the application is very small.

Structure for a Multi-File Project

For larger applications, a more organized structure is recommended. For example:

```
student_dashboard/
app.py
pages/
   page_1.py
   page_2.py
data/
   marks.csv
images/
   logo.png
modules/
   helper_functions.py
```

- **app.py**: The main entry point of the application. It contains the home page or the main dashboard.
- **pages/**: A folder containing additional Python files for multi-page applications. Streamlit automatically detects these files and shows them as separate pages in the application.
- **data/**: A folder that stores datasets, for example CSV files that contain data to be displayed or analyzed.
- **images/**: A folder for image files that may be displayed on the dashboard or used as icons.
- **modules/**: A folder for Python files that contain helper functions or reusable components, keeping the main file simpler.

How Streamlit Uses the Pages Folder

When there is a folder named pages in the project directory, Streamlit automatically creates a navigation menu in the application. Each Python file in the pages folder is treated as a separate page. This feature is useful when creating different sections of an application, such as one page for uploading data and another page for displaying charts.

Best Practices for Directory Structure

1. Keep the main script (for example, app.py) simple and focused on the main functionality.
2. Use a pages folder when the application has multiple sections.
3. Keep datasets and image files in separate folders so that they can be easily located and managed.
4. If there are reusable functions or custom logic, put them in a separate module file and import them into the main script.

A clean structure improves readability and helps others who might want to collaborate on the same application.

Core UI Component

Display Elements

Display Elements – Text and Formatting

Text is one of the most basic and important components of any Streamlit application. It helps to communicate information, explain data, and provide instructions to users. Streamlit provides several functions to display text in different styles and sizes, allowing the application page to be structured clearly. The functions commonly used for text and formatting are:

1. **st.title()**
   This function is used to display the main title of the application page. The text is shown in a large bold font at the top of the page.

   st.title("Student Performance Dashboard")

2. **st.header()**
   This function displays a header, which is slightly smaller than the title. It is useful for dividing content into sections.

   st.header("Marks Analysis")

3. **st.subheader()**
   This function displays a subheading that is smaller than a header. It helps in further breaking down the sections into smaller parts.

   st.subheader("Subject-wise Performance")

4. **st.write()**
   This function displays plain text, numbers, data frames, and even charts, depending on the type of data passed to it. It is the most flexible text function.

   st.write("This dashboard provides insights into marks scored by students.")

5. **st.text()**
   This function displays text exactly as it is, without additional formatting. It is often used when showing fixed information.

   st.text("Data source: Marks recorded for the semester")

6. **st.markdown()**
   Markdown is a simple way to style text with formatting options such as bold, italics, and bullet points. Streamlit allows you to use markdown syntax for better presentation.

   *st.markdown("**Key Insights:** Overall performance has improved.")*

   st.markdown(" Use the filters in the sidebar to select a section.")*

Formatting with Markdown

Markdown supports:

- **Bold text**: **text**
- *Italic text*: *text*
- Bullet lists:

  - Point 1

  - Point 2
- Numbered lists:

  1. Step one

  2. Step two

This makes it easier to present points clearly.

Remember

- Use st.title() only once at the top of the page.
- Use headers and subheaders to organize sections clearly.
- Use st.write() for general-purpose text and st.markdown() when formatting is needed.
- Use consistent formatting throughout the application for a professional and clear layout.

Display Elements – Data Display Element

A key purpose of Streamlit applications is to present data clearly so that users can understand and analyze it. Streamlit provides built-in functions for displaying structured data such as tables and data frames directly on the application page. These functions make it easy to present datasets without any manual formatting.

Common Functions for Displaying Data

1. **st.dataframe()**
    - Displays data in an interactive table format.
    - Users can scroll through rows and columns, and even sort the data by clicking on column headers.
    - Best suited for large datasets where interaction is useful.

```
import pandas as pd
import streamlit as st

data = {
    "Student": ["A", "B", "C"],
    "Marks": [85, 90, 78]
}
df = pd.DataFrame(data)
st.dataframe(df)
```

2. **st.table()**
    - Displays data as a static table without interaction features.
    - Suitable for small tables where data is fixed and does not need sorting or scrolling.

```
st.table(df)
```

3. **st.metric()**
    - Displays a single key value, often with a label and an optional delta (change).
    - Useful for dashboards where highlighting key numbers (like average marks or growth) is important.

```
st.metric(label="Average Marks", value="84%", delta="3%")
```

4. **st.json()**
    - Displays structured JSON data in a formatted way.
    - Helpful when working with data that comes in JSON format.

```
data_json = {"name": "Student A", "marks": {"Math": 85, "Science": 90}}
st.json(data_json)
```

When to Use These Functions

- Use st.dataframe() when users need to explore or scroll through a dataset.
- Use st.table() for smaller datasets or summary results that should be shown as they are.
- Use st.metric() to show key highlights at the top of a dashboard.
- Use st.json() to display structured JSON data for debugging or explanation.

Remember

- Choose the data display function based on the size of the dataset and the need for interactivity.
- Keep summary information (metrics) separate from detailed tables so that important information is not missed.
- Use meaningful labels and ensure that the data is clean and easy to read.

Display Elements – Text and Input Widget

In addition to displaying static content, Streamlit allows users to interact with an application by entering text or values. These input widgets make the application dynamic because users can provide inputs that change the content shown on the page. Streamlit provides a variety of widgets for entering text, numbers, and other types of information.

Common Input Widgets

1. **st.text_input()**
    - Allows users to type a single line of text.
    - Useful for capturing simple inputs such as a name or a small search term.

```
name = st.text_input("Enter student name")
st.write("You entered:", name)
```

2. **st.text_area()**
    - Allows users to enter multiple lines of text.
    - Suitable for capturing longer inputs like comments or feedback.

```
feedback = st.text_area("Enter feedback")
st.write("Feedback:", feedback)
```

3. **st.number_input()**
    - Allows users to enter a number by typing or using small arrows to increase or decrease the value.
    - Can also set minimum and maximum values.

```
marks = st.number_input("Enter marks", min_value=0, max_value=100, step=1)
st.write("Marks entered:", marks)
```

4. **st.date_input()**
    - Allows users to select a date from a calendar.

```
exam_date = st.date_input("Select exam date")
st.write("Exam date selected:", exam_date)
```

5. **st.time_input()**
    - Allows users to select a time.

```
exam_time = st.time_input("Select exam time")
st.write("Exam time selected:", exam_time)
```

How These Widgets Work

When a user enters text or a value into an input widget, Streamlit automatically runs the script from top to bottom. This means that the page is updated immediately, and the new value is reflected wherever it is used in the code. This interaction makes the application feel responsive without requiring additional code for event handling.

Remember

- Input widgets make applications interactive.
- Use meaningful labels so that users know what to enter.
- The application updates immediately when a user provides input, so you can use the input values in calculations, filtering data, or generating charts.

Button and Action Widgets in Streamlit

In many applications, we want to allow the user to trigger a specific task or action, such as submitting a form, running a calculation, or refreshing a result. Streamlit provides interactive widgets like buttons and checkboxes that help in triggering actions. These widgets do not accept input data like text fields but allow control over when something should happen. These are particularly useful when the app should wait for user confirmation before executing a piece of logic.

1. st.button()

- A simple clickable button.
- Often used to trigger a specific block of code only when the button is pressed.

```
if st.button("Calculate Total"):
    total = 85 + 90 + 78
    st.write("Total Marks:", total)
```

In this example, the total is only calculated and shown **after** the button is clicked.

2. st.checkbox()

- A toggle switch (True/False).
- Useful to show or hide specific content based on user choice.

```
show_result = st.checkbox("Show Result")
if show_result:
    st.write("Congratulations! You have passed.")
```

This is like an optional reveal; the user controls whether a section of the app is visible.

3. st.radio()

- A selection widget for choosing one option among many.

```
gender = st.radio("Select Gender", ["Male", "Female", "Other"])
st.write("You selected:", gender)
```

4. st.selectbox()

- Similar to a dropdown menu with one selectable option.

```
subject = st.selectbox("Choose subject", ["Math", "Science", "English"])
st.write("Selected subject:", subject)
```

5. st.multiselect()

- Allows selection of multiple options from a list.

```
hobbies = st.multiselect("Select your hobbies", ["Reading", "Music", "Sports",
"Art"])
st.write("You selected:", hobbies)
```

6. st.slider() and st.select_slider()

- For selecting a numeric or categorical value using a slider.

```
marks = st.slider("Select marks", 0, 100, 50)
st.write("You selected:", marks)
```

How Action Widgets Work

Each of these widgets returns a value (like True/False for a checkbox, or a selected option for a dropdown), which can be used in if statements to control the flow of the application. This gives the user control over what is displayed and when certain calculations should be triggered. Unlike text inputs that run the script on every change, buttons only perform their action once per click. This makes them ideal for controlled actions.

Remember

- Action widgets are used to control the execution of specific parts of the code.
- Use buttons when you want to delay an action until user confirmation.
- Use checkboxes and radio buttons for conditional display and logical branching.
- Combine these widgets to build rich, interactive user interfaces in Streamlit.

Display Elements – Selection Widgets

Selection widgets in Streamlit allow users to choose one or more options from a predefined list. These widgets make the application interactive by enabling conditional logic, custom displays, and user-driven navigation. They are most useful when the choices are limited and predefined, and when the selection influences the flow or content of the app. In a typical classroom scenario, a teacher might ask students to choose an elective subject from a list. Similarly, in a data application, selection widgets guide users to filter results, navigate sections, or tailor views based on preferences.

1. st.selectbox()

This widget displays a dropdown list and allows the user to **pick one option**.

```
subject = st.selectbox("Choose your subject", ["Mathematics", "Science",
"English", "History"])
st.write("You selected:", subject)
```

- Best used when the list of options is long.
- The dropdown ensures a compact layout.
- Returns the selected option as a string.

Example Use Case: Selecting a report type in a school dashboard.

2. st.multiselect()

This allows users to select **multiple options** from a list.

```
skills = st.multiselect("Select your skills", ["Python", "Excel",
"Communication", "Public Speaking"])
st.write("Selected skills:", skills)
```

- Returns a list of selected items.
- Best used when more than one selection is allowed or required.
- Can be useful in surveys, resume data collection, or configuration panels.

Example Use Case: A teacher selecting multiple responsibilities for the upcoming academic year.

3. st.radio()

Presents a list of options vertically with radio buttons, allowing selection of only **one** option at a time.

```
gender = st.radio("Select your gender", ["Male", "Female", "Other"])
st.write("You selected:", gender)
```

- Suitable when you want all options visible on screen at once.
- Good for short lists with fewer than five items.
- More visible than selectbox, hence more suited for quick and intuitive selection.

Example Use Case: Choosing the grading system for student evaluation (e.g., Marks or Grades).

4. st.slider()

Allows selection of a **single numeric value** (or a range) using a slider interface.

```
marks = st.slider("Select marks", 0, 100, 50)
st.write("You selected:", marks)
```

- Accepts numeric ranges.
- Supports integer and float values.
- Can also be used with dates or custom step sizes.

Example Use Case: Filtering students by minimum required marks in a subject.

5. st.select_slider()

Similar to slider, but instead of numbers, it can work with **categorical data** or predefined labels.

```
grade = st.select_slider("Choose your grade", options=["A", "B", "C", "D",
"E", "F"])
st.write("You selected:", grade)
```

- Used when data is not numeric but still needs ordered navigation.
- Ideal for performance levels, grade categories, or phases.

Example Use Case: Selecting the level of training from Beginner to Expert.

Introduction to Navigation and Multi-Page Layouts

As Streamlit apps grow more complex, organizing the content into multiple pages becomes essential. A single-page layout is fine for small tools, but larger dashboards, reports, or apps benefit greatly from a multi-page structure. Just like a school website has sections like Home, About Us, Admissions, and Contact, a Streamlit app can also be structured into multiple pages for better readability, maintainability, and user experience. Streamlit provides a simple and flexible way to create and manage multiple pages using a folder-based structure and the Streamlit sidebar for navigation.

Key Concepts of Multi-Page Applications

1. **Each Page is a Python File**
    - Streamlit treats each .py file inside a folder named pages/ as a separate page.
    - These are automatically listed in the sidebar with their filenames (titles can be customized).
    - Each file behaves like an independent app section, but shares the same overall configuration.
2. **Default Page is the Main File (app.py or main.py)**
    - This is the page shown first when the app is launched.
    - It typically contains an overview, dashboard, or landing content.
3. **Sidebar Used for Navigation**
    - Streamlit automatically adds a navigation bar in the sidebar.
    - The names of the Python files become page labels (can include emojis or spaces).
4. **Pages Are Executed Separately**
    - Each page runs in isolation when selected.
    - You can share data between pages using session state if needed.

Steps to Create a Multi-Page Streamlit App

1. **Create a Main File** (e.g., app.py):

```
import streamlit as st
st.set_page_config(page_title="School Dashboard", layout="wide")
st.title("Welcome to the School Dashboard")
st.write("Use the sidebar to navigate to different sections of the app.")
```

2. **Create a pages/ Folder in the Same Directory**
    - Add new Python files like Student_Info.py, Exam_Results.py, etc.
3. **Write Code in Each Page File**

For example, inside **Student_Info.py:**

```
import streamlit as st
st.title("Student Information")
st.write("Here you can view and manage student profiles.")
```

Subject_Enrollment.py

```
import streamlit as st
st.title("Subject Enrollment")
subjects = st.multiselect("Select your subjects", ["Math", "Physics",
"Economics", "History"])
st.write("You have enrolled in:", subjects)
```

Note:

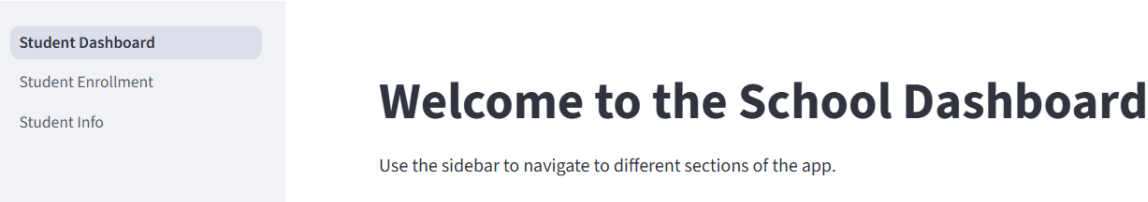
- The python file '***Student_Dasboard.py***' has to be present in a new folder.
- Within that folder, ***create another folder 'pages'*** and make sure the other ***two python files are present inside this folder.***

4. **Run the App**
   Use the command: ***python -m streamlit run app.py***
5. **Use the Sidebar Navigation**
   When the app loads, Streamlit will automatically display a sidebar list with all pages in the pages/ folder.

When the app runs, the user sees a sidebar:

- Student Dashboard
- Subject Enrollment
- Student Info

Output



Source: Screenshot

Introduction to Custom Layout and Columns

Streamlit apps are designed to be simple and responsive by default. However, for more advanced applications - such as dashboards, data input forms, or custom workflows - you often need more control over how content is arranged on the screen. This is where layout management becomes important. Streamlit allows you to arrange components side by side or in custom blocks using layout features like st.columns() and st.container().

Why Layout Management is Useful

Using columns, containers, and expanders, you can create a more professional layout.

1. st.columns() – Side-by-Side Layout

The st.columns() function allows you to create multiple columns side by side.

Syntax:

```
col1, col2 = st.columns(2)
```

Each column acts like its own mini-section where you can add Streamlit widgets or content.

Example:

```
col1, col2 = st.columns(2)
with col1:
    st.subheader("Upload CSV")
    st.file_uploader("Choose a file", type="csv")
with col2:
    st.subheader("App Info")
    st.write("This app helps visualize student data.")
```

This will display two sections side by side - one for file upload and the other for information.

2. st.container() – Group Elements Together

A container is like a visual section or block that can be reused or grouped for logic and clarity.

Syntax:

```
with st.container():
    st.write("Grouped content")
    st.button("Click me")
```

This is useful for adding multiple components together as a logical unit.

3. st.expander() – Collapsible Sections

If you want to save screen space or keep less important info hidden by default, use st.expander().

Example:

```
with st.expander("See student tips"):
    st.write("""
        1. Keep your ID card ready
        2. Submit fees before deadline
        3. Contact class teacher for updates
    """)
```

This creates a clickable section that expands when the user wants to read more.

Combining Columns with Containers

You can nest these components for advanced layouts.

Example:

```
col1, col2 = st.columns(2)
with col1:
    with st.container():
        st.subheader("Basic Details")
        st.text_input("Name")
        st.number_input("Age")
with col2:
    with st.container():
        st.subheader("Academic Info")
        st.selectbox("Class", ["9th", "10th", "11th", "12th"])
        st.slider("Marks (%)", 0, 100)
```

This example creates two side-by-side input forms: one for personal info and another for academic info.

Column Widths and Ratios

By default, all columns are equal in width. But you can customize the width using ratios:

```
col1, col2, col3 = st.columns([2, 1, 3])
```

Here:

- col1 is twice as wide as col2
- col3 is three times as wide as col2

This helps when one part (like a graph) needs more space than another (like a label).

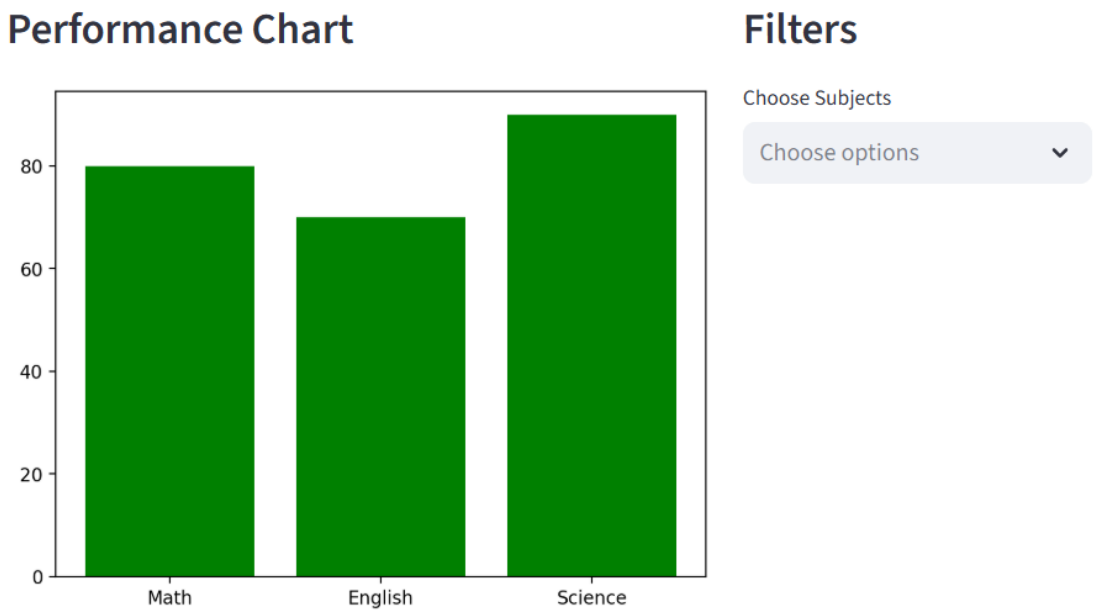
Use Case Example: Class Dashboard

A school wants a dashboard with: A ***chart on the left*** and ***Filters on the right***

```
import streamlit as st
import pandas as pd
import matplotlib.pyplot as plt
# Sample data
data = {'Subject': ['Math', 'English', 'Science'], 'Marks': [80, 70, 90]}
df = pd.DataFrame(data)

col1, col2 = st.columns([2, 1])

with col1:
    st.subheader("Performance Chart")
    fig, ax = plt.subplots()
    ax.bar(df['Subject'], df['Marks'], color='green')
    st.pyplot(fig)

with col2:
    st.subheader("Filters")
    st.multiselect("Choose Subjects", options=df['Subject'])
```

This layout creates a clean and functional interface with a wider area for the chart and a narrower filter panel.

Output



Source: Screenshot

Visualization

Line Chart

What is a Line Chart?

A line chart is a basic type of chart used to display information that changes continuously over time. It represents data points connected by straight lines. The x-axis usually shows time (like days, months, or years), and the y-axis shows the values (like marks, sales, or temperature). This chart helps identify trends, patterns, and fluctuations in data.

Where are Line Charts Used?

Line charts are commonly used in education, business, and research to track progress over a period. For example:

- A teacher tracking students' performance in weekly tests.
- A school administrator monitoring monthly attendance percentages.
- An engineering department analyzing sensor data collected over time.
- A faculty member displaying the usage of the lab facilities across semesters.

Why Use Line Charts in a Dashboard?

Line charts allow decision-makers to:

- Compare performance over multiple periods.
- Detect patterns like growth, decline, or cycles.

- Quickly interpret long-term trends using visual cues.

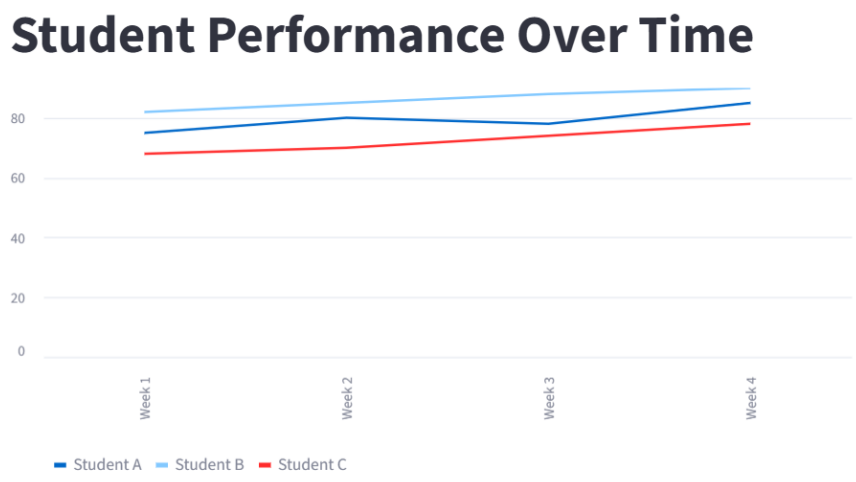
Creating a Line Chart in Streamlit

Streamlit provides a simple way to generate a line chart using built-in methods like st.line_chart(). It works best when using Pandas DataFrame or structured data such as lists or dictionaries.

Example: Displaying Students' Weekly Marks

```
import streamlit as st
import pandas as pd
import numpy as np
st.title("Student Performance Over Time")
# Sample data: Weekly test scores for 3 students
data = {
    "Week 1": [75, 82, 68],
    "Week 2": [80, 85, 70],
    "Week 3": [78, 88, 74],
    "Week 4": [85, 90, 78]
}

# Create DataFrame with student names as rows
df = pd.DataFrame(data, index=["Student A", "Student B", "Student C"])

# Transpose to make weeks on x-axis
df = df.T

st.line_chart(df)
```

Output



Source: Screenshot

Explanation:

- The data dictionary stores weekly scores.
- The DataFrame uses student names as row labels.
- Transposing the DataFrame (df.T) allows weeks to appear on the x-axis and student scores as lines.
- st.line_chart(df) draws the chart directly.

Bar Chart

What is a Bar Chart?

A bar chart is a type of data visualization that uses rectangular bars to represent and compare values across different categories. Each bar's height (in vertical bar charts) or length (in horizontal bar charts) shows the magnitude of the data. Unlike line charts, bar charts are better for comparing individual values, rather than showing trends over time.

Where are Bar Charts Used?

Bar charts are useful when you want to compare:

- The number of students in different classes.
- The pass percentages of various subjects.
- The count of students choosing different streams (Science, Commerce, Humanities).
- Survey responses or voting results.

Why Use Bar Charts in a Dashboard?

Bar charts allow for:

- Easy comparison between different categories.
- Quick understanding of which group is larger or smaller.
- Visual display of rankings or distributions.

Creating a Bar Chart in Streamlit

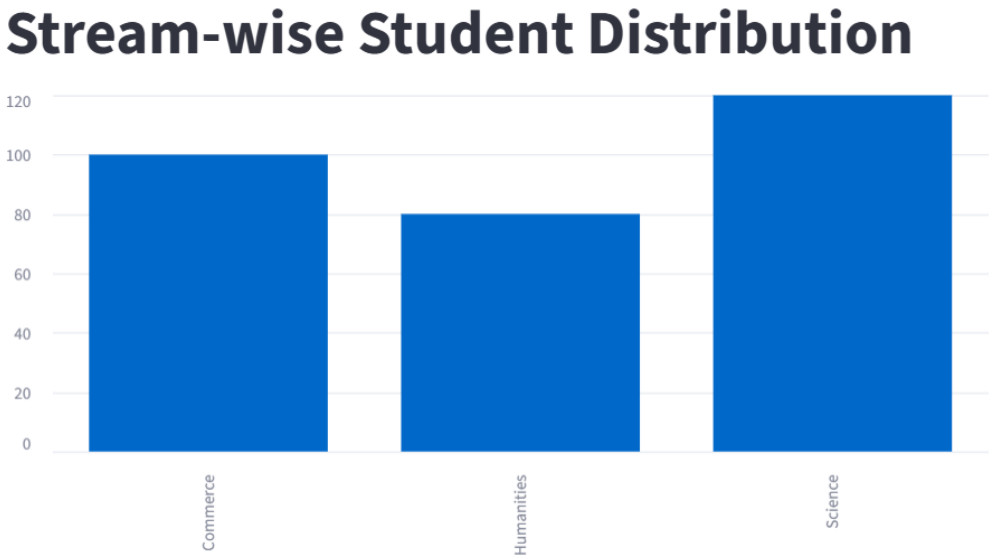
Streamlit makes it simple to display a bar chart using the st.bar_chart() function. It accepts structured data like Pandas DataFrame or a list of values.

Example: Displaying Number of Students in Different Streams

```
import streamlit as st
import pandas as pd

st.title("Stream-wise Student Distribution")
```

```
# Sample data
data = {
    "Streams": ["Science", "Commerce", "Humanities"],
    "Students": [120, 100, 80]
}

# Create DataFrame
df = pd.DataFrame(data)

# Set 'Streams' as index for correct labeling
df.set_index("Streams", inplace=True)

# Plot bar chart
st.bar_chart(df)
```

Output



Source: Screenshot

Explanation:

- The chart compares the number of students in three streams.
- The Streams column becomes the x-axis labels.
- The height of each bar shows how many students are in each stream.

This visualization helps make quick decisions based on data.

Scatterplot Chart

What is a Scatterplot Chart?

A scatterplot chart is a type of data visualization that displays individual data points on a two-dimensional graph using dots. Each dot represents one observation and is plotted based on two variables: one on the x-axis and one on the y-axis. Scatterplots are helpful in understanding relationships or patterns between variables.

Where are Scatterplot Charts Used?

Scatterplots are used when you want to:

- Check correlation between two factors (e.g., marks vs. attendance).
- Spot patterns or clusters in data.
- Detect outliers in student performance or responses.
- Understand performance distribution across two metrics (e.g., test score vs. hours studied).

Why Use Scatterplots in a Dashboard?

Scatterplots help users:

- Observe how one metric may affect another.
- Identify linear or non-linear relationships.
- See how spread out the data is (variance).
- Detect unusual values or behaviours.

Creating a Scatterplot in Streamlit

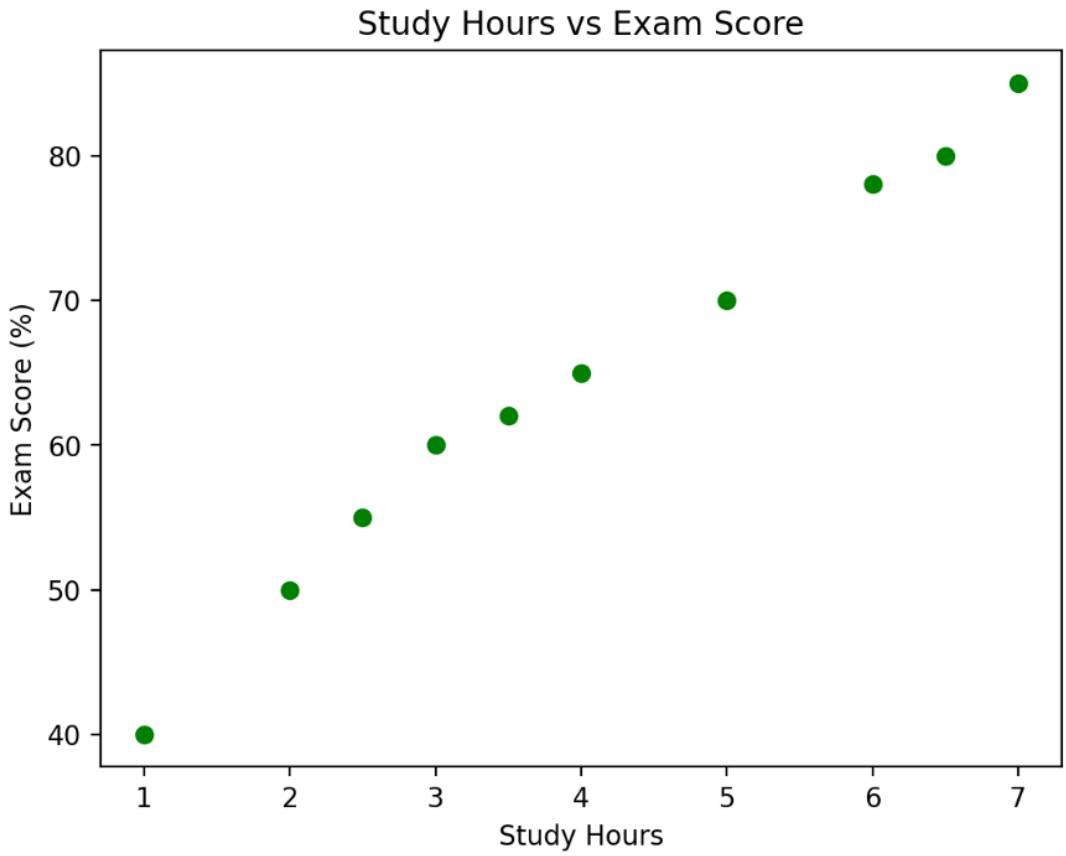
Streamlit itself does not have a built-in function like st.scatter_chart(). However, you can easily create scatterplots using external libraries such as Matplotlib or Plotly, and then render them with st.pyplot() or st.plotly_chart().

Example: Student Performance vs. Study Hours

```
import streamlit as st
import matplotlib.pyplot as plt

# Sample data
study_hours = [1, 2, 2.5, 3, 3.5, 4, 5, 6, 6.5, 7]
exam_scores = [40, 50, 55, 60, 62, 65, 70, 78, 80, 85]

# Create the plot
fig, ax = plt.subplots()
ax.scatter(study_hours, exam_scores, color='green')
ax.set_title("Study Hours vs Exam Score")
ax.set_xlabel("Study Hours")
ax.set_ylabel("Exam Score (%)")
```

```
# Display in Streamlit
st.pyplot(fig)
```

Output



Source: Screenshot

Explanation:

- Each dot shows one student's data (how many hours they studied and what score they received).
- The pattern indicates that as study hours increase, exam scores also tend to improve.
- This kind of insight can help faculty suggest better study strategies.

st.pyplot

What is st.pyplot?

Streamlit provides the st.pyplot() function to embed Matplotlib visualizations directly into a Streamlit web app. While Streamlit has some built-in charting capabilities like st.line_chart() or st.bar_chart(), they offer limited customization. If you need fine-

grained control over your charts - like custom axis labels, titles, annotations, colors, or multiple plots - then Matplotlib + st.pyplot() is the best choice.

When to Use st.pyplot()?

Use st.pyplot() when:

- You want to create custom charts beyond the basic chart types.
- You already use Matplotlib in your data analysis workflow.
- You need to add extra details like legends, grid lines, multiple data series, or annotations.

Basic Steps to Use st.pyplot() in a Streamlit App

1. Import necessary libraries:

   import matplotlib.pyplot as plt

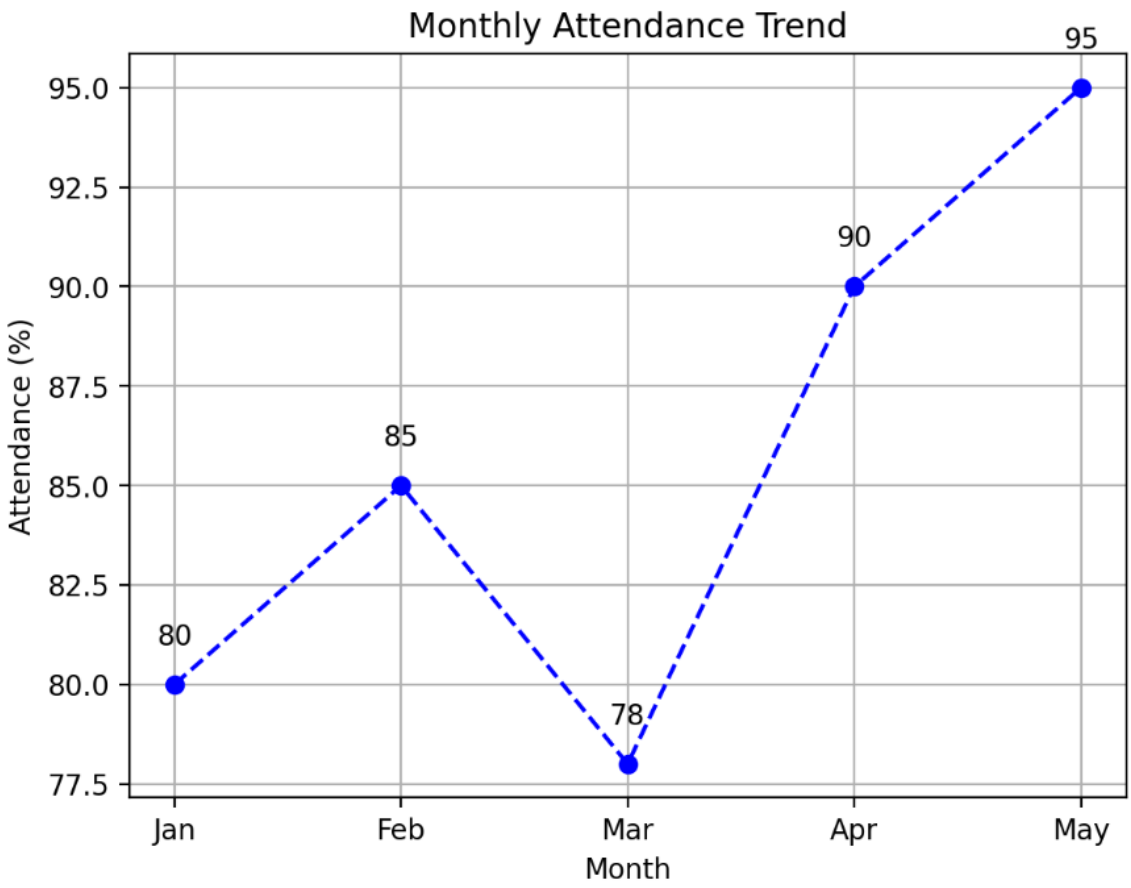
   import streamlit as st

2. Create your Matplotlib plot using the usual steps.
3. Use st.pyplot(fig) to display it in the Streamlit app.

Custom Line Plot with Annotations

```
import streamlit as st
import matplotlib.pyplot as plt

# Sample data
study_hours = [1, 2, 2.5, 3, 3.5, 4, 5, 6, 6.5, 7]
exam_scores = [40, 50, 55, 60, 62, 65, 70, 78, 80, 85]

# Create the plot
fig, ax = plt.subplots()
ax.scatter(study_hours, exam_scores, color='green')
ax.set_title("Study Hours vs Exam Score")
ax.set_xlabel("Study Hours")
ax.set_ylabel("Exam Score (%)")

# Display in Streamlit
st.pyplot(fig)
```

Output



Source: Screenshot

Output Explanation

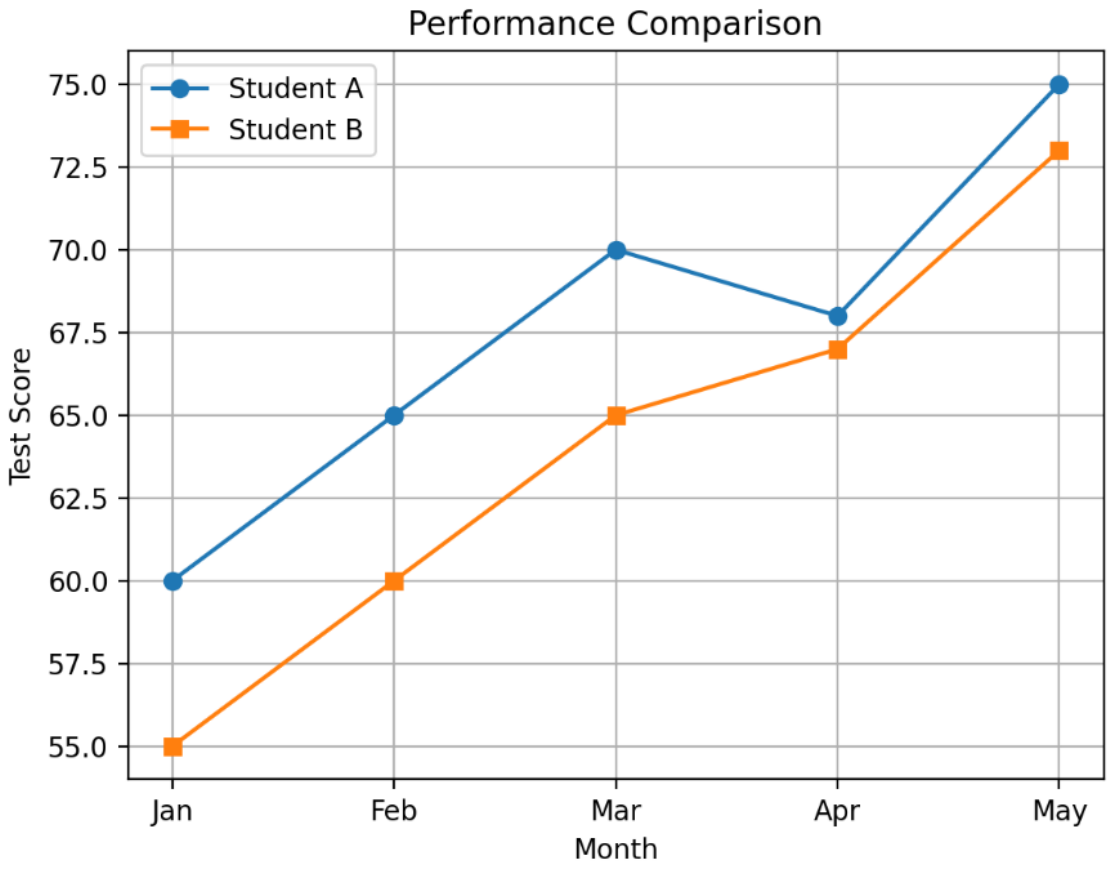
- This chart shows monthly attendance with connected dots.
- Markers and annotation make each point stand out.
- Helpful for academic dashboards to monitor classroom discipline or regularity.

Example 2: Multiple Lines in a Single Plot

```
import matplotlib.pyplot as plt
import streamlit as st

# Sample data for two students
months = ['Jan', 'Feb', 'Mar', 'Apr', 'May']
student_A = [60, 65, 70, 68, 75]
student_B = [55, 60, 65, 67, 73]

# Create the plot
fig, ax = plt.subplots()
ax.plot(months, student_A, label="Student A", marker='o')
ax.plot(months, student_B, label="Student B", marker='s')
```

```
ax.set_title("Performance Comparison")
ax.set_xlabel("Month")
ax.set_ylabel("Test Score")
ax.legend()
ax.grid(True)

# Show chart
st.pyplot(fig)
```

Output



Source: Screenshot

Output Explanation

- This chart shows monthly test scores for two students, using connected dots (line plot) for each student's trend over five months.
- Markers (circles for Student A, squares for Student B) make each score stand out clearly at each month, helping viewers distinguish performance changes.
- A legend labels the two lines, while grid lines make it easier to read exact values.
- The line plot highlights patterns - for example, steady improvement, drop in scores, or consistent performance - across time.

References

- Chart elements - Streamlit Docs. (n.d.). https://docs.streamlit.io/develop/api-reference/charts
- Create a multipage app - Streamlit Docs. (n.d.). https://docs.streamlit.io/get-started/tutorials/create-a-multipage-app
- Examples — Matplotlib 3.10.5 documentation. (n.d.). https://matplotlib.org/stable/gallery/index.html
- GeeksforGeeks. (2025, July 16). A Beginners guide to Streamlit. GeeksforGeeks. https://www.geeksforgeeks.org/python/a-beginners-guide-to-streamlit/
- st.bar_chart - Streamlit Docs. (n.d.). https://docs.streamlit.io/develop/api-reference/charts/st.bar_chart
- St.Page - Streamlit Docs. (n.d.). https://docs.streamlit.io/develop/api-reference/navigation/st.page
- St.pyplot - Streamlit docs. (n.d.). https://docs.streamlit.io/develop/api-reference/charts/st.pyplot
- streamlit. (2025, July 25). PyPI. https://pypi.org/project/streamlit/
- st.scatter_chart - Streamlit Docs. (n.d.). https://docs.streamlit.io/develop/api-reference/charts/st.scatter_chart
- st.set_page_config - Streamlit Docs. (n.d.). https://docs.streamlit.io/develop/api-reference/configuration/st.set_page_config
- Text elements - Streamlit Docs. (n.d.). https://docs.streamlit.io/develop/api-reference/text