



Introduction to Streamlit



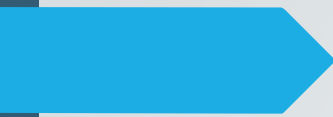
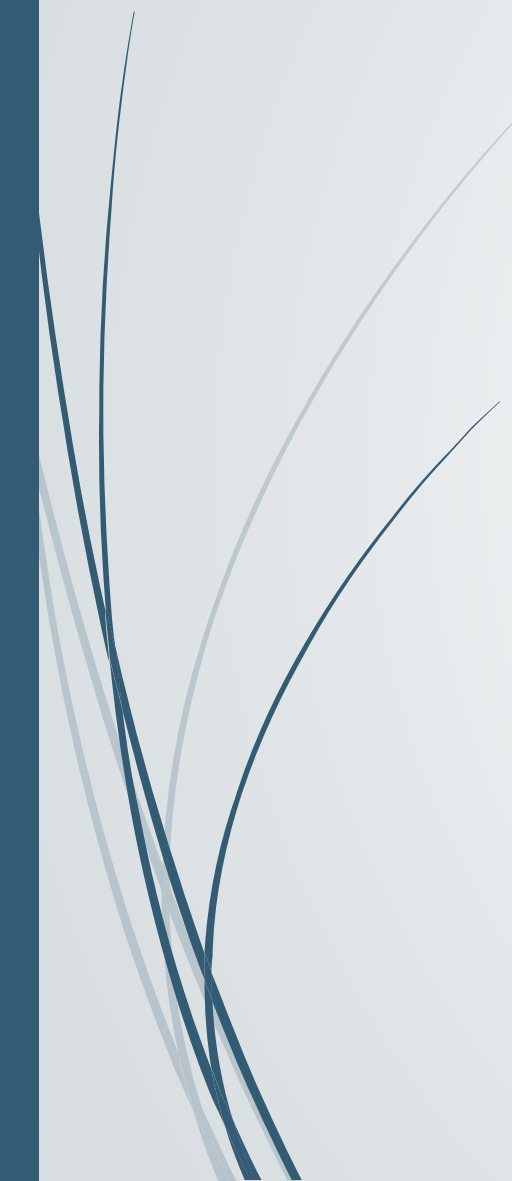
Streamlit

- Streamlit is an open-source Python library used for creating custom web applications for data science and machine learning projects.
- It allows data scientists and developers to convert data scripts into shareable web apps quickly and easily.
- Streamlit provides a simple and intuitive interface, enabling users to focus on data visualization and interactivity without extensive web development knowledge.
- Streamlit is designed for rapid prototyping and deployment of data-driven web applications.



Key features

- **Streamlit Magic:** Automatic data rendering and widget generation from Python code.
- **Interactive Widgets:** Enables user interaction with the app using widgets like sliders, buttons, and text inputs.
- **Data Visualization:** Intuitive integration with popular data visualization libraries like Matplotlib and Plotly.

- 
- 
- **Layout Customization:** Flexibility in organizing app components and customizing the app layout.
 - **Real-Time Updates:** Ability to update visualizations and app content in real-time based on user input.



Advantages of Using Streamlit for Web App Development

- **Easy and Fast:** Rapidly convert data scripts into web apps without extensive coding.
- **Data-Driven:** Perfect for showcasing data visualizations and interactive components.
- **Pythonic:** Full-stack development using Python only, no need to learn additional languages.

- 
- 
- **Open-Source and Active Community:** Regular updates, bug fixes, and community contributions.
 - **Seamless Deployment:** Streamlit apps can be easily deployed on various platforms, including Heroku, AWS, and Streamlit Sharing.



Setting Up a Streamlit Development Environment

- Install Streamlit using pip or conda: **pip install streamlit** or **conda install streamlit**.
- Create a new Python script (e.g., **app.py**) for your Streamlit app.
- Use the Streamlit command **streamlit run app.py** to run the app locally.



Streamlit Layout System



Layout of a Streamlit App

- Streamlit apps are organized as a sequence of Python statements.
- Each statement generates a corresponding component or widget on the app's web page.
- The order of the statements defines the layout and appearance of the app.



Creating a Simple Streamlit App with Basic Components

- Start with a minimal Streamlit app containing a title and text.
- Add additional components like data visualizations and widgets step-by-step.
- Observe how each new statement updates the app layout



Using Markdown and HTML in Streamlit Apps

- **Markdown:** Use `st.write()` or `st.markdown()` to display formatted text, headings, and lists.
- **HTML:** Utilize `st.write()` with raw HTML tags for more advanced text formatting or styling.
- Explain how to include images, links, and other HTML elements in the app.



Example 1: Creating a Basic Streamlit App

Import the Streamlit library

```
import streamlit as st
```

Add a title to the app

```
st.title("My First Streamlit App")
```

Add text using Markdown

```
st.markdown("Welcome to this introductory Streamlit app!")
```



Organizing App Components with Columns and Containers

- **Columns:** Arrange components in multiple columns using `st.columns()`.
- **Containers:** Group components into sections with `st.container()` to improve organization.
- Demonstrate how to create complex layouts with multiple columns and containers.

Example-2





```
import streamlit as st
```

```
# Create two columns  
col1, col2 = st.columns(2)
```

```
# Add content to the first column  
with col1:
```

```
    st.title("This is column 1")  
    # Add your content for the first column here  
    st.write("This is the left column.")  
    st.image("https://upload.wikimedia.org/wikipedia/commons/thumb/c/c3/Python-logo-notext.svg/1200px-Python-logo-notext.svg.png")
```

```
# Add content to the second column  
with col2:
```

```
    st.title("This is column 2")  
    st.button("Click Me")  
    # Add your content for the second column here
```



Example-3

```
import streamlit as st
```

```
# Create two rows and three columns
```

```
row1_col1, row1_col2, row1_col3 = st.columns(3)
```

```
row2_col1, row2_col2, row2_col3 = st.columns(3)
```

```
# Add content to the cells
```

```
with row1_col1:
```

```
    st.write("Row 1, Column 1")
```

```
    # Add your content for this cell
```

```
with row1_col2:
```

```
    st.write("Row 1, Column 2")
```

```
    # Add your content for this cell
```




with row1_col3:

```
st.write("Row 1, Column 3")
```

```
# Add your content for this cell
```

with row2_col1:

```
st.write("Row 2, Column 1")
```

```
# Add your content for this cell
```

with row2_col2:

```
st.write("Row 2, Column 2")
```

```
# Add your content for this cell
```

with row2_col3:

```
st.write("Row 2, Column 3")
```

```
# Add your content for this cell
```

Example-4

```
import streamlit as st
```

```
# Define the column widths as a list of percentages
```

```
column_widths = [30, 40, 20, 10]
```

```
# Create the table header
```

```
table_header = """
```

```
<table>
```

```
<tr>
```

```
<th style="width: {}%;">Column 1</th>
```

```
<th style="width: {}%;">Column 2</th>
```

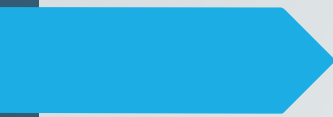
```
<th style="width: {}%;">Column 3</th>
```

```
<th style="width: {}%;">Column 4</th>
```

```
</tr>
```

```
</table>
```

```
""".format(*column_widths)
```



```
# Display the table header
st.markdown(table_header, unsafe_allow_html=True)
```

```
# Sample data for the table
table_data = [
    ["Row 1, Cell 1", "Row 1, Cell 2", "Row 1, Cell 3", "Row 1, Cell 4"],
    ["Row 2, Cell 1", "Row 2, Cell 2", "Row 2, Cell 3", "Row 2, Cell 4"],
]
```

```
# Create the table body
table_body = "<table>"
for row in table_data:
    table_row = "<tr>"
    for cell in row:
        table_row += "<td>{}</td>".format(cell)
    table_row += "</tr>"
    table_body += table_row
table_body += "</table>"
```

```
# Display the table body
st.markdown(table_body, unsafe_allow_html=True)
```



Interactive Widgets



Introduction to Widgets

- Widgets are interactive elements that allow user input and interaction with the app.
- Streamlit provides a variety of built-in widgets like sliders, buttons, text inputs, and select boxes.
- Widgets are essential for creating dynamic and user-friendly apps.



Creating Interactive Elements such as Sliders, Buttons, and Text Inputs

- Use `st.slider()` to create a numeric slider widget.
- Demonstrate `st.button()` and `st.text_input()` for interactive buttons and text input.

Example 5: Adding Interactive Widgets

```
import streamlit as st
```

```
# Add a title to the app
```

```
st.title("Interactive Widget Example")
```

```
# Add a slider widget
```

```
selected_value = st.slider("Select a value", 1, 10)
```

```
# Display a message based on the selected value
```

```
st.write(f"You selected: {selected_value}")
```



Using Widgets to Control Data Visualization and App Behavior

- Link widgets to data visualizations to enable user-driven updates.
- Show examples of dynamically changing plots based on slider values.
- Implement widgets to control data filtering and manipulation



Handling User Input and Reacting to Widget Events

- Streamlit apps respond to widget events like button clicks or slider changes.
- Use Python functions as event handlers to update app content based on widget input.
- Explain the importance of event-driven programming for interactive apps.



Dynamic Visualization



Utilizing Matplotlib and Plotly Libraries for Data Visualization

- Streamlit seamlessly integrates with popular visualization libraries.
- Show examples of creating basic plots using Matplotlib and Plotly.



Generating Dynamic Plots Based on User Input and Widget Interactions

- Use widgets to control plot parameters like data range, type, and style.
- Show how dynamic plots are updated in real-time as users interact with the app.
- Demonstrate reactive programming and automatic re-rendering of the app.



Updating Plots and Visualizations in Real-Time

- Show how to use Streamlit's caching mechanisms to optimize app performance.
- Explain the benefits of caching for efficiently rendering dynamic visualizations.



Enhancing Data Visualizations with Custom Styling and Annotations

- Customize plot aesthetics, such as colors, labels, and titles.
- Annotate plots with text, lines, and shapes to provide additional insights.
- Show how to create a visually appealing dashboard with interactive plots.

Example 6: Dynamic Plotting

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

# Add a title to the app
st.title("Dynamic Plot Example")

# Create data
x = np.linspace(0, 10, 100)
y = x ** 2

# Create a figure
fig, ax = plt.subplots()
```

Example 6: Dynamic Plotting...

```
# Plot initial data
```

```
line, = ax.plot(x, y)
```

```
ax.set_xlabel("X")
```

```
ax.set_ylabel("Y")
```

```
# Add a slider widget for controlling the exponent
```

```
exponent = st.slider("Select an exponent", 1, 10)
```

```
# Update plot based on user input
```

```
line.set_ydata(x ** exponent)
```

```
# Display the plot using Streamlit
```

```
st.pyplot(fig)
```




Working with Data in Streamlit



Loading Data into the Streamlit App

- Import data from various sources, such as CSV or Excel files.
- Use Pandas to read and preprocess the data.



Displaying Data Tables and Interactive Data Exploration

- Showcase data tables using **st.dataframe()** or **st.table()**.
- Enable interactive exploration of data with widgets and plots.



Creating Data-Driven Visualizations with User-Selected Data

- Use widgets to allow users to select data subsets or columns.
- Generate visualizations based on user-selected data.

Example 7: Displaying Data Tables

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

```
# Load data from a CSV file  
data = pd.read_csv("data.csv")
```

```
# Add a title to the app  
st.title("Data Table Example")
```

```
# Display the data as a table  
st.dataframe(data)
```



Dashboard Creation with Streamlit





What a dashboard is in the context of data visualization

A dashboard is like the control center of your data. It's a user interface that displays key information, metrics, and insights from various data sources in one place. Think of it as a real-time, interactive snapshot of your data's most important aspects.



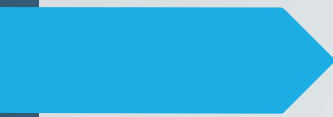
Why dashboards are essential in data visualization

- ▶ "Dashboards enhance data communication by presenting information in a visually appealing and accessible format."
- ▶ "They empower users to make data-driven decisions by providing real-time insights."
- ▶ "Dashboards facilitate data exploration, allowing users to interact with data, apply filters, and drill down for details"

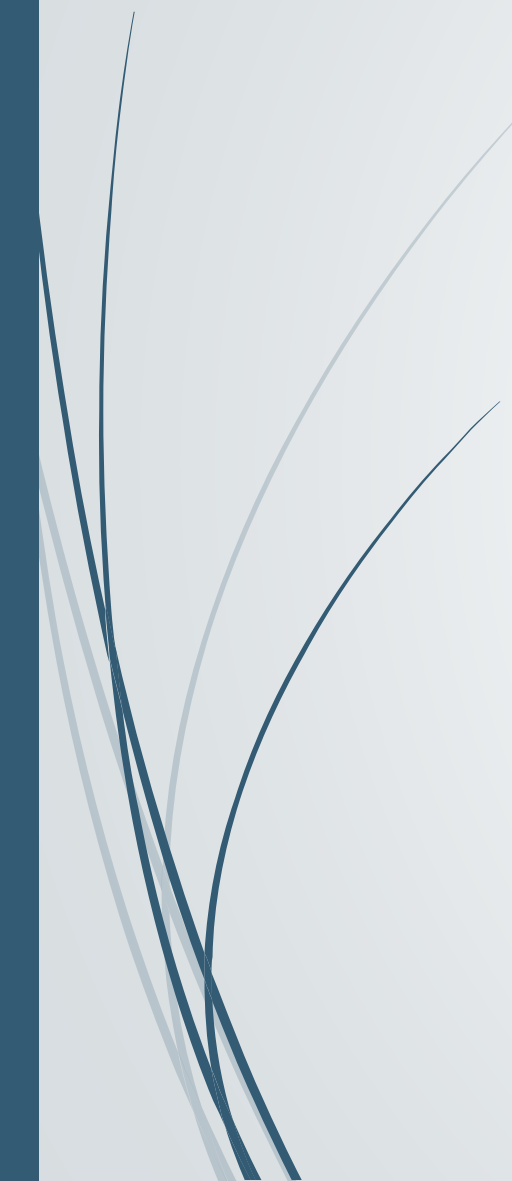


Key Components

- Visualization Widgets
 - Charts and Graphs
 - Tables
 - Maps.
 - Indicators
- Data Source
- Filters and Controls
 - Dropdowns:
 - Sliders
 - Buttons
 - Search Bars
- Dashboard Title and Header
- Navigation and Menu
- Data Insights and Annotations
- Performance Metrics
- Alerts and Notifications
- Time Series and Trends
- User Interactivity
- Security and Permissions
- Export and Sharing
- Help and Documentation
- Responsive Design
- Feedback Mechanism
- Refresh and Update
- Customization Options



importance of user-friendly design in dashboards

- ▶ An effective dashboard is not just about data; it's about the user experience. A well-designed dashboard is intuitive, visually appealing, and easy to navigate. It empowers users to interact with data effortlessly and gain insights quickly.
- 



Key purposes of dashboards in presenting data insights

- ▶ **Data Visualization:** Dashboards use charts, graphs, tables, and other visual elements to represent complex data. Visualizations make it easier for users to grasp information quickly and identify trends, outliers, and patterns.
- ▶ **Real-Time Monitoring:** Dashboards can display data in real-time, allowing users to monitor important metrics as they change. This is especially valuable in scenarios where up-to-the-minute data is crucial, such as financial markets or website analytics.
- ▶ **Performance Tracking:** Dashboards provide a centralized location to track and measure the performance of an organization, project, or process. Users can see whether they are meeting their goals and objectives.
- ▶ **Data Interactivity:** Many dashboards are interactive, allowing users to filter, drill down, or explore data further. This interactivity empowers users to tailor the information to their specific needs and answer ad-hoc questions.
- ▶ **Decision Support:** Dashboards help users make data-driven decisions. When all relevant information is presented clearly, decision-makers can assess situations, identify issues, and choose appropriate actions more effectively.
- ▶ **Efficiency:** Dashboards save time by eliminating the need to sift through large datasets or run multiple reports. Users can access the most critical information with a glance, freeing up their time for more strategic tasks.

Key purposes of dashboards in presenting data insights..

- Communication: Dashboards facilitate communication within an organization. They provide a common platform where teams can align their understanding of data and collaborate based on shared insights.
- Transparency: Dashboards promote transparency by making data accessible to a broader audience. When data is easily visible to stakeholders, it enhances trust and accountability.
- Goal Alignment: Dashboards align individuals and teams with organizational goals. By visualizing progress toward objectives, they motivate employees and foster a sense of purpose.
- Risk Identification: Dashboards help identify potential risks or issues early. By monitoring trends and outliers, users can detect anomalies and take proactive measures to mitigate risks.
- Customization: Dashboards are often customizable, allowing users to tailor them to their specific roles or preferences. This ensures that individuals see the information most relevant to them.
- Accessibility: Dashboards can be accessed from various devices, making it easy for users to stay informed even when they are not in the office.



Designing and Organizing Dashboard Components

- Plan the layout and components of the dashboard.
- Use columns and containers to organize the app's structure.



Integrating Various Visualizations and Widgets into a Cohesive Dashboard

- Combine multiple plots and widgets into the dashboard layout.
- Ensure a smooth and interactive user experience.

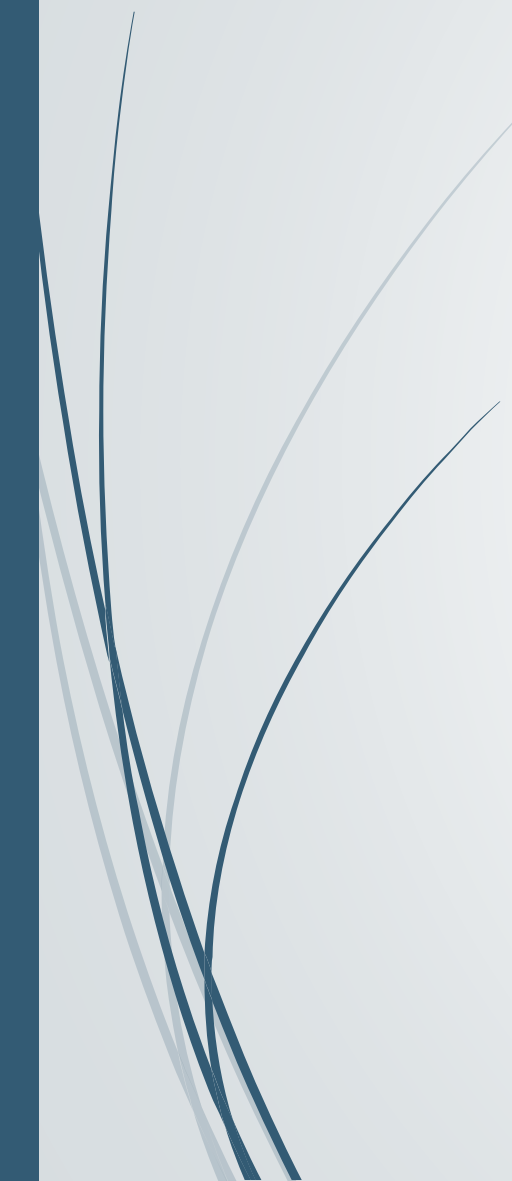
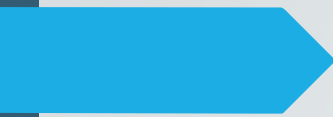
Combining multiple plots and widgets into a Streamlit dashboard for an interactive user experience

```
import streamlit as st
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

# Create a sample DataFrame with consistent data length
data = pd.DataFrame({
    'Date': pd.date_range(start='2022-01-01', periods=366),
    'Value': np.random.randn(366)
})

# Set the title of the web app
st.title('Interactive Financial Dashboard')

# Create a sidebar for user input
st.sidebar.header('Dashboard Settings')
start_date = pd.Timestamp(st.sidebar.date_input('Start Date', pd.to_datetime('2022-01-01')))
end_date = pd.Timestamp(st.sidebar.date_input('End Date', pd.to_datetime('2022-12-31')))
```



```
# Filter the data based on user input
filtered_data = data[(data['Date'] >= start_date) & (data['Date'] <= end_date)]

# Display a line chart of the filtered data
st.header('Line Chart of Financial Data')
st.line_chart(filtered_data.set_index('Date')['Value'])

# Create widgets for user interaction
st.header('Interactive Widgets')
selected_dates = st.multiselect('Select Dates', filtered_data['Date'].unique())
if selected_dates:
    st.write('Selected Dates:', selected_dates)

# Display descriptive statistics
st.header('Descriptive Statistics')
st.write('Mean:', filtered_data['Value'].mean())
st.write('Median:', filtered_data['Value'].median())
st.write('Standard Deviation:', filtered_data['Value'].std())
```




Create a histogram of the data

st.header('Histogram')

plt.hist(filtered_data['Value'], bins=20, alpha=0.5, color='b')

plt.xlabel('Value')

plt.ylabel('Frequency')

st.pyplot(plt)

Create a box plot of the data

st.header('Box Plot')

plt.boxplot(filtered_data['Value'], vert=False)

plt.xlabel('Value')

st.pyplot(plt)



Deploying the Streamlit Dashboard for Sharing and Collaboration

- Explore deployment options for sharing the app with others.
- Discuss best practices for deploying Streamlit apps.