

Streamlit Essentials

From basics to advanced data app development



Surabhi Pandey

bpb

Streamlit

Essentials

From basics to advanced data app development



Surabhi Pandey

bpb

Streamlit Essentials

*From basics to advanced data
app development*

Surabhi Pandey



www.bpbonline.com

First Edition 2025

Copyright © BPB Publications, India

ISBN: 978-93-65890-822

All Rights Reserved. No part of this publication may be reproduced, distributed or transmitted in any form or by any means or stored in a database or retrieval system, without the prior written permission of the publisher with the exception to the program listings which may be entered, stored and executed in a computer system, but they can not be reproduced by the means of publication, photocopy, recording, or by any electronic and mechanical means.

LIMITS OF LIABILITY AND DISCLAIMER OF WARRANTY

The information contained in this book is true to correct and the best of author's and publisher's knowledge. The author has made every effort to ensure the accuracy of these publications, but publisher cannot be held responsible for any loss or damage arising from any information in this book.

All trademarks referred to in the book are acknowledged as properties of their respective owners but BPB Publications cannot guarantee the accuracy of this information.

To View Complete
BPB Publications Catalogue
Scan the QR Code:



www.bpbonline.com

Dedicated to

*My partner, the fundamental constant in my
equation.*

About the Author

Surabhi is a data enthusiast with over a decade of experience across various data-focused roles. She holds a Bachelor's Degree in Computer Science from Amrita Vishwavidyapeetham. She enjoys the challenges and opportunities that come with turning raw data into meaningful insights and creating data-driven solutions that empower business decision-making.

Her background includes a mix of programming languages and tools; her current favorite toolkit includes DBT, Airflow, and Great Expectations, and her go-to language is Python. She has a keen interest in building frameworks to improve and streamline data quality and governance. Currently based in Kuala Lumpur, Surabhi enjoys a good cup of coffee and is an avid reader of epic fantasy novels.

About the Reviewers

❖ **Afaque** is an enthusiastic machine learning engineer who has gained experience in data science, AI, and the advancing fields of MLOps and GenAI. He approaches his work with curiosity and a willingness to learn, utilizing various tools and technologies.

His current focus is on GenAI projects, emphasizing deployment, scaling, latency optimization, and monitoring. Outside of his professional role, Afaque shares insights and experiences through his Substack and Medium platforms, exploring various aspects of the data science and AI realm.

❖ **Vishu Aasliya** is an emerging tech enthusiast with a strong passion for machine learning and artificial intelligence. She has extensive experience in Docker and DevOps, focusing on developing and automating solutions that enhance efficiency and reduce manual intervention. Alongside her academic pursuits, Vishu is also a freelance developer, where she customizes web platforms and implements solutions to optimize user experience and drive engagement. She has worked on various projects, gaining hands-on experience in programming, containerization technologies, and process automation. Currently pursuing a degree in computer science with a specialization in artificial intelligence, Vishu is always eager to explore new technologies and broaden her skill set. In her free time, she enjoys exploring new tools, contributing to open-source projects,

and staying updated with the latest advancements in tech.

- ❖ **Smaranjit Ghose** is a freelance machine learning engineer and lifelong learner specializing in data and algorithms to build fantastic tools and scalable applications. He is currently focused on developing distributed systems for applications serving millions of users, with particular emphasis on computer vision and **natural language processing (NLP)** implementations. Smaranjit is passionate about creating efficient, automated solutions and avoiding manual work wherever possible.

As a technical reviewer, he has contributed to various IT books covering machine learning, React, TensorFlow, **large language models (LLMs)**, and Python and JavaScript ecosystems. Beyond his professional endeavors, Smaranjit is an avid history enthusiast and UFC fan.

Smaranjit is also dedicated to mentoring, having guided over 10,000 students across India, the US, UAE, Ireland, and Mexico in building and contributing to open-source organizations and research labs. His commitment to education and open-source development reflects his belief in collaborative innovation and knowledge sharing in the tech community.

Acknowledgement

I would like to express my sincere gratitude to everyone who contributed to completing this book.

First and foremost, I extend my heartfelt appreciation to my family for their unwavering support and encouragement; their love and encouragement have been a constant source of motivation.

I would also like to thank the BPB Publications team for their excellent work. I would also like to acknowledge the reviewers and technical experts who provided valuable feedback and contributed to the refinement of this manuscript. Their insights and suggestions have significantly enhanced the quality of the book.

Some people leave a lasting impression when they cross your path. I had the pleasure of working with Dennis Lau and Raymond Au Yong a few years back. Dennis introduced me to Streamlit (and many other tools), reigniting my spark for experimentation. Raymond was always encouraging and supportive of the ideas I brought to the table, offering valuable feedback that refined my ideas and improved my design thinking. Their belief and their expertise helped me build confidence and trust in my abilities. Their influence still resonates in my tech journey today, and I am grateful to have crossed paths with them.

Last but not least, we want to express our gratitude to the readers who have shown interest in our book. Your support and encouragement have been deeply appreciated.

Thank you to everyone who has played a part in making this book a reality.

Preface

“Streamlit Essentials: From basics to advanced data app development” provides a comprehensive introduction to Streamlit, a powerful framework that simplifies the process of building and deploying data apps.

The chapters in this book take readers from the fundamentals of Streamlit to advanced techniques. It begins with the basics, covering **Introduction to Streamlit and Getting Started with Streamlit**, ensuring that the readers understand the platform’s needs and core concepts.

As you progress, you will explore the intricacies of **Layouts and Widgets** and how to integrate data to create visually appealing **Data Visualizations**. With a focus on real-world applications, you will also learn about **Machine Learning Model Deployments**.

The later chapters focus on more advanced topics, such as **Advanced Streamlit Concepts** and **Deploying Streamlit Apps**, helping you leverage Streamlit’s features like secret management, callbacks, Database connectivity, App testing and streamlining your workflow to make your application production-ready. In addition, hands-on projects in the book will enable you to showcase your skills by building a professional portfolio.

Building and Enhancing your Portfolio will help to use the power of networking and community building to get the most out of Streamlit and create a portfolio that resonates with potential employers and collaborators. **Enhancing Streamlit development with AI Tools** introduces AI-

assisted development tools that can help speed up the process of application building from ideation to deployment.

This book is designed for data professionals, from beginners looking to explore Streamlit for the first time to advanced users seeking to optimize their development process. With practical examples and hands-on projects, “**Streamlit Mastery**” is a valuable resource for anyone aiming to include Streamlit in their data tooling arsenal and build interactive, data-driven applications.

Chapter 1: Introduction to Streamlit: This chapter discusses the challenges of data presentation and how this gap can be bridged. The most commonly used tools and also the challenges faced by the data developers while communicating and understanding the non-technical stakeholders. We discuss various challenges and explore why Streamlit fits in as an ideal solution for faster prototyping and custom data application development.

Chapter 2: Getting Started with Streamlit: This chapter covers the installation, basic setup, and core concepts that make Streamlit so accessible. Here, readers will gain strong foundation of Streamlit’s working and understand how to get Streamlit up and running quickly. We will also dive into building the first interactive Data application, which we will enhance throughout future chapters.

Chapter 3: Exploring Streamlit Widgets: This chapter covers Streamlit-provided widgets, which help in faster development of the applications. Knowing what Streamlit supports natively, can help readers visualize their ideas faster. From creating form-based applications to embedding media, Streamlit widgets help create a fully functional data app in no time.

Chapter 4: Styling and Layouts in Streamlit: This chapter covers layout options provided by Streamlit.

Although not a full-fledged web application development framework, Streamlit provides great tools to enhance the design of your data application and apply design principles to build an aesthetic data application. We will discuss Streamlit state management, custom components library, and what it means to design an engaging and well-structured application.

Chapter 5: Data Visualization with Streamlit: This chapter focuses on integrating and creating compelling visualizations with Streamlit. Whether it's simple charts or complex real-time data-driven insights, transforming raw data into visually engaging content can be achieved using Streamlit. We will also explore Streamlit chat widgets and third-party visualization libraries and compare their functionalities and use cases.

Chapter 6: Streamlit and Machine Learning: This chapter dives into integrating machine learning models with Streamlit, showcasing how to build and deploy interactive apps powered by machine learning. Readers will learn how to load pre-trained models, visualize predictions, and make the models accessible to a wider audience.

Chapter 7: Advanced Streamlit Concepts: In this chapter, we explore more advanced features, including caching, secret management, app testing, and optimizing performance. We will also discuss the recommended design patterns and pitfalls, ensuring that the data application.

Chapter 8: Deployment of Streamlit Apps: In this chapter, we will explore the steps required to deploy Streamlit applications using various options like Streamlit Sharing or cloud providers. We will deploy our Product Recommender application using Streamlit Sharing and Google Cloud Platform and compare various considerations to note when deciding on the deployment platform.

Chapter 9-11: Hands-On Projects: Easy-Advanced: In these chapters, we dive into real-world projects from easy to advanced. These projects can help you streamline your learning and explore tooling options available with Streamlit.

Chapter 12: Build and Enhance Your Portfolio: This chapter gives a walkthrough on Streamlit's active and vibrant global community boards and how to leverage them to enhance your portfolio. From contributing to open-source projects to marketing your apps, you will learn practical tips for getting your work noticed in the broader data and tech community.

Chapter 13: Enhancing Streamlit Development with AI Tools: The final chapter introduces how AI tools can complement Streamlit development. Readers will gain insights into how AI can speed up coding, assist in generating ideas, and even optimize the development process. We will also touch upon the ethical considerations when using AI tools in your projects.

Code Bundle and Coloured Images

Please follow the link to download the **Code Bundle** and the **Coloured Images** of the book:

<https://rebrand.ly/cfyi5zj>

The code bundle for the book is also hosted on GitHub at **<https://github.com/bpbpublications/Streamlit-Essentials>**. In case there's an update to the code, it will be updated on the existing GitHub repository.

We have code bundles from our rich catalogue of books and videos available at **<https://github.com/bpbpublications>**. Check them out!

Errata

We take immense pride in our work at BPB Publications and follow best practices to ensure the accuracy of our content to provide with an indulging reading experience to our subscribers. Our readers are our mirrors, and we use their inputs to reflect and improve upon human errors, if any, that may have occurred during the publishing processes involved. To let us maintain the quality and help us reach out to any readers who might be having difficulties due to any unforeseen errors, please write to us at :

errata@bpbonline.com

Your support, suggestions and feedbacks are highly appreciated by the BPB Publications' Family.

Did you know that BPB offers eBook versions of every book published, with PDF and ePub files available? You can upgrade to the eBook version at www.bpbonline.com and as a print book customer, you are entitled to a discount on the eBook copy. Get in touch with us at :

business@bpbonline.com for more details.

At www.bpbonline.com, you can also read a collection of free technical articles, sign up for a range of free newsletters, and receive exclusive discounts and offers on BPB books and eBooks.

Piracy

If you come across any illegal copies of our works in any form on the internet, we would be grateful if you would provide us with the location address or website name. Please contact us at **business@bpbonline.com** with a link to the material.

If you are interested in becoming an author

If there is a topic that you have expertise in, and you are interested in either writing or contributing to a book, please visit www.bpbonline.com. We have worked with thousands of developers and tech professionals, just like you, to help them share their insights with the global tech community. You can make a general application, apply for a specific hot topic that we are recruiting an author for, or submit your own idea.

Reviews

Please leave a review. Once you have read and used this book, why not leave a review on the site that you purchased it from? Potential readers can then see and use your unbiased opinion to make purchase decisions. We at BPB can understand what you think about our products, and our authors can see your feedback on their book. Thank you!

For more information about BPB, please visit www.bpbonline.com.

Join our book's Discord space

Join the book's Discord Workspace for Latest updates, Offers, Tech happenings around the world, New Release and Sessions with the Authors:

<https://discord.bpbonline.com>



Table of Contents

1. Introduction to Streamlit

- Introduction
- Structure
- Objectives
- Challenges of data presentation
- Need for Streamlit
- Conclusion
- Exercises
- Answers*

2. Getting Started with Streamlit

- Introduction
- Structure
- Objectives
- Setting up Streamlit
 - Comparing installation options*
 - Running Streamlit locally using virtual environment*
 - Running Streamlit using Docker*
 - Running Streamlit using Streamlit community cloud*
- Streamlit core concepts
 - Streamlit app model*
 - Exploring the Streamlit hello app*
- Hands-on: Product recommender app

Project structure

Running your first Streamlit app

Conclusion

Exercises

Answers

3. Exploring Streamlit Widgets

Introduction

Structure

Objectives

Widgets overview

Streamlit core widgets

Display widgets

Text input widgets

Numerical input widgets

Date and time widgets

Other input widgets

Single choice widgets

Multiple choice widgets

Button widgets

Data widgets

Streamlit magic

Hands-on: Adding widgets to enhance the app

Choosing the right widgets

Conclusion

Exercises

Answers

4. Styling and Layouts in Streamlit

Introduction

Structure
Objectives
Designing engaging and well-structured application
User interface design with Streamlit
Streamlit layouts and containers
Streamlit utility widgets
Streamlit control flow widgets
Streamlit theming
Rendering HTML and CSS
State management
Streamlit custom components
Hands-on enhancing product recommender application
Conclusion
Exercises
Answers

5. Data Visualization with Streamlit

Introduction
Structure
Objectives
Importance of data visualization in data analysis
Streamlit visualization options
Streamlit core chart elements
Summarizing the chart types and comparing use cases
Streamlit supported charting libraries
Streamlit display and aesthetics widgets
Display progress and status widgets
Display message widgets

Media widgets
Metric widget
Streamlit chat widgets
Hands-on: Integrating visual elements in Product Recommender
Conclusion
Exercises
Answers

6. Streamlit and Machine Learning

Introduction
Structure
Objectives
Introduction to machine learning in Streamlit
Setting up your machine learning environment
Essential Python libraries
Preparing your machine learning models
Starting with a simple machine learning model
Integrating machine learning trained model with Streamlit
Dynamically adjusting model hyperparameters
Understanding model hyperparameters
Implementing interactive controls for hyperparameters
Tips to improve the performance
Hands-on: Integrating ML in Product Recommender
Conclusion
Exercises
Answers

7. Advanced Streamlit Concepts

Introduction

Structure

Objectives

Streamlit advanced topics

Callbacks

Secret management

Best practices for Secret Management using Streamlit

Working with timezones

Best practices when using different timezones in application

Building mutating charts with Streamlit

Best practices when using mutating charts

Connecting to databases with Streamlit

Best practices for database connections

App testing with Streamlit

Best practices for app testing with Streamlit

Design patterns pitfalls to avoid for Streamlit

Strategies for defining custom classes

Conclusion

Exercises

Answers

8. Deployment of Streamlit Apps

Introduction

Structure

Objectives

Hosting considerations for Streamlit applications

Hosting Streamlit applications

Hands-on hosting application with Streamlit Share
Debugging tips

Hands-on hosting application using GCP cloud run
Google Cloud Platform
Google Cloud Run
Deploying the application
Setting up continuous integration and deployment for the data application
Getting the metrics and alert information from Cloudrun
Other Streamlit deployment options

Conclusion

Exercises

Answers

9. Hands-On Projects: Easy

Introduction

Structure

Objectives

Streamlit projects abstract: Easy

Project 1: Weather dashboard

Implementation guide

Project 2: Basic stock price viewer

Implementation guide

Project 3: To-do list

Implementation guide

Project 4: Image gallery

Implementation guide

Project 5: Movie recommendation system

Implementation guide

Project 6: BMI calculator
Implementation guide

Project 7: Simple expense tracker
Implementation guide

Project 8: Currency converter
Implementation guide

Project 9: Interactive quiz
Implementation guide

Project 10: Random quote generator
Implementation guide

Hands-on: Implementing sample project on the Weather dashboard
Project structure for Weather data app
Project structure for currency converter data app

Conclusion

10. Hands-On Projects: Intermediate

Introduction

Structure

Objectives

Streamlit project abstracts: Intermediate

Project 1: Multi-page data analysis dashboard
Implementation guide

Project 2: Job application tracker
Implementation guide

Project 3: Budget planner with visualizations
Implementation guide

Project 4: Personal fitness tracker
Implementation guide

Project 5: Multi-page portfolio website

Implementation guide

Project 6: Task management application

Implementation guide

Project 7: Multi-page product catalog

Implementation guide

Project 8: Sports team management application

Implementation guide

Project 9: Multi-page event planner

Implementation guide

Project 10: Multi-page recipe organizer

Implementation guide

Hands-on: Implementing sample project

Conclusion

11. Hands-On Projects: Advanced

Introduction

Structure

Objectives

Streamlit projects abstract: Advanced

Project 1: Real-time data analytics dashboard

Implementation guide

Project 2: Secure financial portfolio manager

Implementation guide

Project 3: Machine learning model deployment

Implementation guide

Project 4: Customizable news aggregator

Implementation guide

Project 5: Advanced time series forecasting application

Implementation guide

Project 6: Interactive Geographic Information System Implementation guide

Hands-on: Implementing sample project on news aggregator

Conclusion

12. Build and Enhance Your Portfolio

Introduction

Structure

Objectives

Streamlit journey and next steps

Importance of community engagement

Key resources and how to engage

Official forums and discussion boards

Streamlit community forum

Streamlit Official App Gallery

Streamlit Component Gallery

Crafting your professional portfolio

The idea behind a comprehensive portfolio

Leveraging tools for effective documentation

Marketing your skills through your portfolio

Conclusion

13. Enhancing Streamlit Development with AI Tools

Introduction

Structure

Objectives

Introduction to AI-assisted development tools

Shift from manual coding to AI-assisted development

Advantages and guardrails of using AI-assisted tools

Exploring categories of AI tools

- Code completion tools*
- AI-powered coding assistants*
- Prompt engineering platforms*
- Bug detection and code review tools*
- Developer bots*

Comparison of tool category and Streamlit usage

Hands-on: Using ChatGPT in Streamlit development

- ChatGPT overview*
- ChatGPT data privacy, intellectual property, ethical usage recommendations*
- Data privacy with ChatGPT*
- Intellectual property guardrails with ChatGPT*
- Other important terms and conditions with ChatGPT*

Conclusion

Appendix A: Streamlit Cheat Sheet

- Objectives
- Streamlit commands

Appendix B: Additional Resources and References

- Introduction
- Streamlit projects
- User Experience design
- Future of AI and ethical usage

Appendix C: Docker 101: Beginner's Guide to Containers

- Objectives
- Introduction to containers

Understanding containers with traveling stage show analogy

Docker: The stage crew for your traveling show

Docker basics

Run docker image

Getting started

Cleaning up

Building custom images

Index

CHAPTER 1

Introduction to Streamlit

Introduction

In the modern digital realm, where data has earned its reputation as the new oil, our interaction with this invaluable resource has been rapidly transforming. The journey from traditional spreadsheets to intricate BI tools and now to intuitive data apps showcases the ever-evolving canvas of data interactivity. This is where Streamlit shines brightly. It is an open-source Python library that is reshaping how we build and visualize data applications.

Streamlit empowers data professionals to transform their intricate analyses and models into dynamic web apps by serving as a bridge between data science and app development. Designed with Python enthusiasts in mind, it eliminates the need for exhaustive knowledge of front-end development, making data app creation accessible and straightforward.

Structure

The chapter discusses the following topics:

- Challenges of data presentation

- Need for Streamlit

Objectives

In this chapter, we will understand the role of Streamlit in a data project, highlighting how it empowers data professionals to create impactful, accessible, and interactive data applications that resonate with a broad audience, including those with minimal technical expertise.

Challenges of data presentation

As businesses and organizations become increasingly data-driven, the gap between complex data analysis and actionable insights for decision-makers and stakeholders becomes more evident.

Data scientists often find themselves at a crossroads when it is time to present their findings, discoveries, or models. It is prudent that the stakeholders are kept in the loop at every project stage to avoid any miscommunication and differences at later stages. Given ahead is an overview of the high-level stages of data projects.

Let us walk through each stage of the project to understand the challenges better:

- **Problem definition, data collection, and preparation:** The project begins with understanding the business problem, defining the scope, and setting clear objectives. This is an important stage of the project; defining correct boundaries and the definition of done is of utmost importance for a successful project execution. This also acts as a guardrail in the later stages of a project when new requirements are brought in. Challenges include dealing with large volumes of data, ensuring data quality, and addressing missing or inconsistent data. Challenges at this stage

include aligning the project with business goals and ensuring stakeholder agreement on the project's direction. Data collection involves gathering relevant data from various sources and preprocessing it for analysis. It is important to communicate the findings and data quality issues to the stakeholders, too. It might not seem important to convey at the start, but it helps to set the right expectations from the data.

- **Exploratory Data Analysis (EDA):** Here, data scientists explore the data to understand patterns, anomalies, and relationships. Challenges include making sense of complex data patterns and summarizing key findings in a way that is understandable to non-technical stakeholders.
- **Model building and insight generation:** This stage involves selecting, building, and validating machine learning models, as well as model results, to generate actionable insights. Challenges include translating technical findings into business insights and validating these insights with stakeholders.
- **Deployment and maintenance:** The final model is deployed into a production environment. Challenges include integrating the model with existing systems and ensuring its performance over time.

Each of the stages has its varied technical hurdles. However, one common thing across all the steps is the challenge to constantly communicate and keep the stakeholders in the loop. When communicating with stakeholders who might come from different areas of expertise like sales, advertising, and marketing, it is crucial to ensure that the presentation tools are selected carefully to communicate the key points effectively and precisely.

Visualization serves as a powerful tool in bridging the gap between technical and non-technical stakeholders in such projects. It translates complex data sets into intuitive, easy-to-understand graphical formats, enabling stakeholders who may not have the technical expertise to grasp intricate patterns and insights at a glance. This visual approach not only facilitates clearer communication but also aids in collaborative decision-making. By presenting data in a visually engaging and accessible manner, stakeholders can more effectively comprehend the implications of the data, leading to informed discussions and more efficient decision-making processes. Visualization is a universal language, democratizing access to insights and fostering a shared understanding among diverse teams. Also, it is important to remember that the end goal is always to assist in providing insights to the business.

There are many dashboarding options available in the market today. Most tools cater to specific use cases; some are good at churning out heavy-weight BI insights or building a fancy dashboard, but the fanciest tool might not be the answer here. A simple tool is required that can build faster without over-engineering.

There are some traditional methods like using Jupyter Notebook for building faster, but these might present challenges such as:

- **The complexity of sharing:** Raw data, detailed algorithms, or even Jupyter notebooks with non-technical stakeholders can lead to confusion. It demands an intricate dance of explanations, which might fail to drive the point home.
- **Static visuals:** While visuals like charts and graphs are a step forward, they are static. They showcase a snapshot in time rather than a dynamic flow of data,

which might not encapsulate the entire story or the potential variations.

- **Lack of interactivity:** Presenting a model without interactivity means stakeholders cannot experiment with inputs to understand potential outcomes. It becomes a one-way street of information flow, lacking the essential feedback loop.

Need for Streamlit

Turning a data analysis or a machine learning model into an interactive web application required a blend of front-end development skills and a stack of web frameworks. This often meant that data scientists needed to collaborate with web developers or invest time learning these frameworks, which is time-consuming and sometimes cumbersome.

Streamlit fills this void by offering a platform where the usual Python code can magically turn into an interactive web application. Think about when you wanted to share your analysis or a prototype model with a non-technical stakeholder. Instead of navigating through Jupyter Notebooks or sharing static visuals, Streamlit provides a medium to make these insights dynamic, interactive, and web-ready.

The current industry is full of tools for data visualization and app creation. From Dash by Plotly to Shiny for R, the options are varied. However, Streamlit has carved its niche, especially when fast prototyping is the goal, for the following reasons:

- **Ease of use:** Unlike some platforms that require a deep dive into the nuances of web frameworks, Streamlit offers a simpler Python-centric approach. If you are comfortable with Python, you are already halfway there.

Streamlit's promise is to deliver a functional web application with minimal Python code. Gone are the days when JavaScript, HTML, or CSS were barriers.

- **Interactivity at its core:** While tools like Dash offer interactivity, Streamlit's widget ecosystem is designed to be integrated seamlessly, making the building process intuitive and less cumbersome.
- **Widgets** like sliders, buttons, and text inputs are not just add-ons; they are core to Streamlit. They transform static data insights into dynamic narratives.
- **Speed of delivery:** Rapid prototyping is one of Streamlit's forte. The immediate feedback and live-coding feature ensure a quick transition from code to a visual prototype, a feature not all tools can boast of.

The essence of *what you see is what you get* is embodied in Streamlit's live-coding capability, making prototyping swift and iterative.

When comparing Streamlit with other dashboarding tools in the market, an interesting key feature is the ability to save visualizations as code, which brings several significant advantages over traditional dashboarding tools. When you implement your visualizations as code, there are a few advantages that technical engineers might appreciate:

- **Version control and change management:** By saving visualizations as code, Streamlit allows integration with version control systems like Git. This means any changes made to the dashboard are trackable and reversible, similar to how software development teams manage code changes. This capability is a game-changer for data teams, as it ensures that updates or modifications to the dashboard are documented, can be reviewed, and rolled back if necessary.

- **One point of change:** Streamlit's code-centric approach allows changes to a visualization or dashboard that do not require navigating through complex GUI settings. Instead, updates are made directly in the code. This ensures that there is a single point of change, which is particularly beneficial in maintaining consistency and reducing errors or conflicts that might arise from multiple adjustment points.
- **Collaborative development:** Streamlit's compatibility with standard development tools and workflows makes it an ideal platform for collaborative projects. Multiple team members can work on different aspects of the dashboard simultaneously, with changes being merged and reviewed through standard code collaboration practices. This collaborative approach is less cumbersome than traditional dashboard tools, where collaboration often means working on the same GUI interface or passing files back and forth.
- **Scalability and flexibility:** As projects grow in complexity, Streamlit's code-based dashboards can be scaled more effectively. Since the underlying logic and visualizations are code-driven, they can be modularized, reused, and integrated with other systems and data sources with ease. This flexibility is often limited in traditional dashboarding tools, which might not scale well with complex or evolving data projects.

As we learn more concepts about Streamlit, we will be incorporating the learning by iteratively building the data application and integrating features as concepts are introduced. This will help to build a more in-depth understanding of the concepts and their application.

Embark on this journey with an open heart and mind. Streamlit is not just another tool in your arsenal; it is a canvas waiting for your masterstroke. Dive in, explore, and let your data narratives captivate and inform.

Conclusion

In this chapter, we have discussed the innate challenges data scientists face in sharing their invaluable insights and how Streamlit steps in, making the transition from code to interactive web applications feel almost effortless. Our discussion on the need for Streamlit sheds light on its edge in the saturated market of data visualization tools, emphasizing its simplicity, interactivity, and rapid prototyping capabilities.

In the next chapter, our hands-on project, the E-commerce Product Recommender, will represent the confluence of coding, machine learning, and deployment, illustrating how a well-thought-out idea can metamorphose into a tool of real-world significance, revolutionizing the e-commerce landscape.

Exercises

- 1. Which of the following best describes the primary purpose of Streamlit?**
 - a. It is a tool for static data visualizations like charts and graphs.
 - b. It serves as a bridge between data science and app development, allowing data professionals to create dynamic web apps without exhaustive knowledge of front-end development.
 - c. It is a replacement for web development frameworks like Django.

- d. It is a tool primarily for sharing Jupyter notebooks with non-technical stakeholders.
- 2. What challenges do traditional methods of data presentation, such as sharing raw data or static visuals, often face?**
- a. They are too interactive and can confuse stakeholders.
 - b. They cannot capture dynamic flows of data and might not present the full story or potential variations.
 - c. They are usually developed using Django.
 - d. There are no challenges.
- 3. What makes Streamlit stand out, especially for fast prototyping, compared to other data visualization tools?**
- a. Its dependency on JavaScript, HTML, and CSS.
 - b. It requires deep dives into web frameworks' nuances.
 - c. It is the same as others.
 - d. Its ease of use, with a focus on a Python-centric approach, interactivity through widgets, and rapid prototyping capabilities.

Answers

- 1. b**
- 2. b**
- 3. d**

Join our book's Discord space

Join the book's Discord Workspace for Latest updates, Offers, Tech happenings around the world, New Release and Sessions with the Authors:

[https://discord\(bpbonline\).com](https://discord(bpbonline).com)



CHAPTER 2

Getting Started with Streamlit

Introduction

In the second chapter, we will explore different options for setting up Streamlit. These options will help us pick up the most familiar toolset and begin the journey to explore Streamlit. We will also introduce and understand Streamlit's core concepts and building blocks. Equipped with this knowledge, we can start with our first iteration of the Product recommender application.

Structure

The chapter discusses the following topics:

- Setting up Streamlit
- Streamlit core concepts
- Hands-on: Product recommender app

Objectives

In this chapter, we will walk through the simple steps to set up Streamlit. Whether you are an expert or a beginner, this chapter will be useful. We will also discuss different installation options and why they matter. We will also look at the benefits of virtual environments and the magic of Docker. The chapter will also

examine how Streamlit compiles the code at the back end and its core concept. By the end of this chapter, you will have your very first Streamlit app up and running.

Setting up Streamlit

For data enthusiasts and professionals diving into new tools, a complex setup can be a significant deterrent. By recognizing this, the creators of Streamlit prioritized ease of use, starting from the installation process. In fact, one of the best things about Streamlit is its installation process, which is as easy as it comes with respect to current tools.

It is intuitive and user-friendly. With just a single command in your terminal or command prompt, you can have Streamlit up and running, ready to turn your data scripts into interactive web applications. By making the installation process straightforward, Streamlit has lowered the entry barrier for many. Whether you are a seasoned data scientist or someone just starting on their data journey, Streamlit ensures that everyone has a fair shot at creating and sharing their data stories.

Comparing installation options

Before we begin, let us explore the options for setting up the development environment. There is no right or wrong; it is dependent on the ease of development and flexibility in the deployment of the code.

Running Streamlit locally using virtual environment

It is always a best practice to create a virtual environment when using a local development environment. Each project comes with its unique set of requirements, dependencies, and configurations. Managing these for multiple projects on a single machine can quickly become overwhelming. It is a commitment to cleaner, more organized, and more reproducible coding practices.

Running Streamlit using Docker

Using Docker does not only guarantee efficiency but also ensures a consistent and reliable setup across various platforms. An

additional advantage is that it is easily deployable, whether to a cloud provider or any server.

At its essence, Docker offers a way to package applications into containers; think of these as lightweight, self-sufficient boxes that contain everything an application needs to run. This includes the code, runtime, system tools, libraries, and settings. Leveraging Docker for Streamlit is akin to discovering a secret ingredient in a recipe. It enhances the process, ensures consistency, and elevates the result. We will use this approach once we start with our hands-on project.

Running Streamlit using Streamlit community cloud

This is a good way to get a feel of a shareable Streamlit dashboard. It has its limitations, but the free tier provides enough to get a basic app up and running. Its free tier is great for hobbyists, students, or anyone looking to test and showcase their apps without incurring costs. If your app is not performance-critical and you are not expecting a huge load, then this is the way to go. Even if you are not a seasoned developer, you can get your Streamlit app up and running on the web with just a few simple commands. The apps deployed are private by default, ensuring that only those with the link can access them. This provides a layer of security for initial deployments.

Overall, the Streamlit Community Cloud provides a fantastic platform if you are looking to quickly deploy and share your Streamlit apps without the overhead of managing infrastructure. However, for more complex projects or apps with specific infrastructure needs, one might need to weigh the pros and cons and consider other deployment options.

The prerequisites are as follows:

- Python 3.8 - 3.11
- pip
- IDE/ TextEditor (I use Pycharm)

The installation process for Windows is as follows:

1. Install the `virtualenv` package using pip as follows:

```
pip install virtualenv
```

2. Create a virtual environment. The current directory will be where your virtual environment will be placed. You can choose the name for your environment as follows:

```
Python -m venv mystreamlit_venv
```

3. Activate the virtual environment by using the following code:

```
myenv\Scripts\activate
```

4. Install Streamlit using the following code:

```
pip install streamlit
```

5. Test your installation using the following code:

```
streamlit hello
```

Streamlit will be loaded in the localhost, as shown in the following figure:

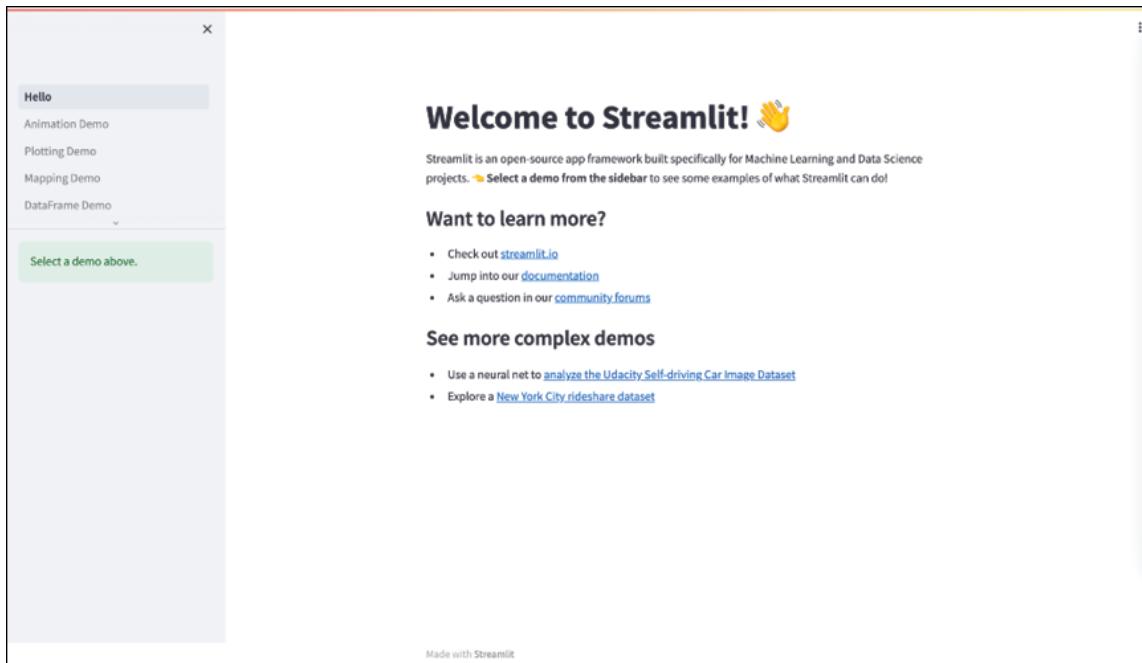


Figure 2.1: Streamlit page hello application

6. For every subsequent run, you can run your Streamlit app from the same virtual environment as follows:

```
streamlit run myfile.py
```

Note: You can use other virtual environment creation tools like Anaconda, pipenv also.

The steps for installation for MacOS/Linux are as follows:

1. Install the pipenv package as follows:

```
brew install pipenv # if you have brew installed  
pip3 install pipenv # if you don't have brew  
installed
```

2. Install Streamlit using pipenv as follows:

```
pipenv install streamlit
```

3. Start the pipenv shell, as shown in the following code:

```
pipenv shell
```

4. Test your Streamlit installation using the following code:

```
streamlit hello
```

Streamlit will be loaded in the localhost as shown in the following figure:

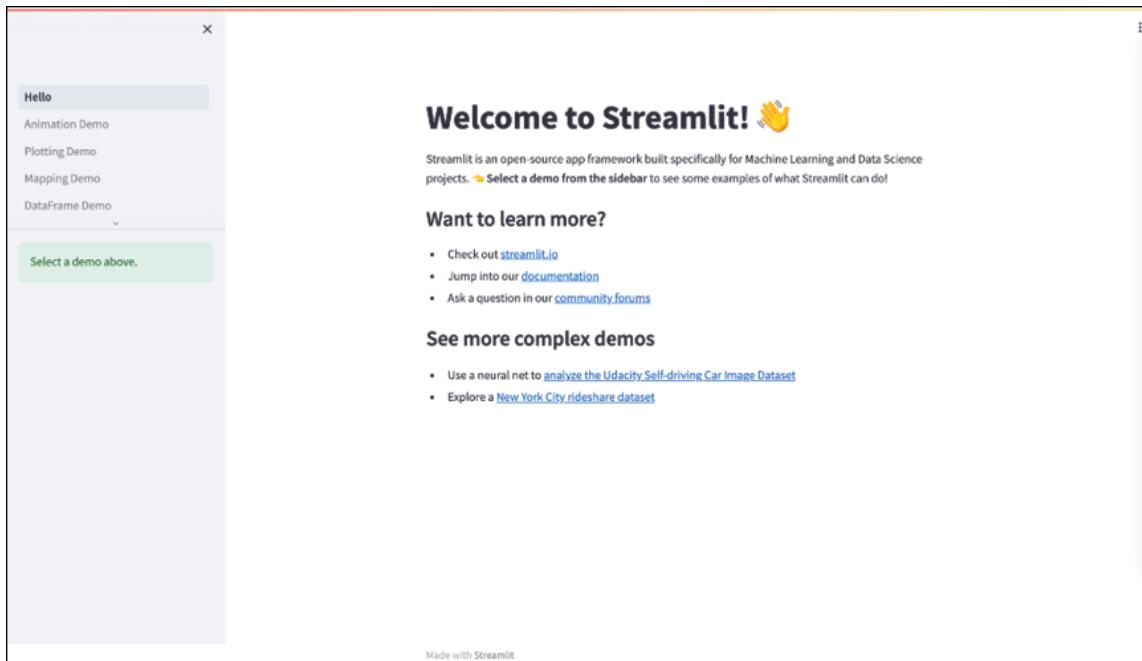


Figure 2.2: Streamlit Hello application home page

5. For every subsequent run, you can run your Streamlit app from the same virtual environment:

```
streamlit run myfile.py
```

Streamlit core concepts

When looking at the world of Streamlit, it is easy to get captivated by its ease of use and rapid prototyping capabilities. However, as with any tool or technology, it is crucial to look beneath the surface and understand the mechanics that power it.

Streamlit, in its essence, excels at what it is designed for, turning data scripts into shareable web apps without the usual web development overhead. If you need to show off a new data visualization, a machine learning model, or a quick dashboard, you can use Streamlit. Its design allows for quick iterations, making it useful for prototyping. However, when it comes to building a full-fledged web application with user authentication, database integrations, complex back-end logic, and other advanced features, that is when we need to turn to more robust frameworks like Django/flask.

So, before we start building applications, let us understand how Streamlit works.

Streamlit app model

Let us look at the basics of how Streamlit runs:

- Streamlit is a Python framework that runs Python scripts from top to bottom.
- As the script is run, Streamlit renders its output live on the browser.
- Every time a webpage is opened, or any interaction on the webpage triggers the Streamlit page to be rerun. An interaction could be a click or widget state change. This is important to note when developing apps that have to maintain the state. (More on this in later chapters.)

- Streamlit caches the result to avoid reloading and recomputing expensive operations.

Exploring the Streamlit hello app

Once you run the Streamlit hello command, you can see the first Streamlit app loaded in your localhost. This app is a comprehensive guide, demonstrating the Streamlit capabilities and links to documentation, tutorials, and demos. This is a multipage app; you can browse and load the various pages based on the sidebar selection. The screen, as shown in *Figure 2.3*, comes up.

You can check the entire source code and documentation of this application in the Streamlit GitHub repository:

<https://github.com/streamlit/hello>.

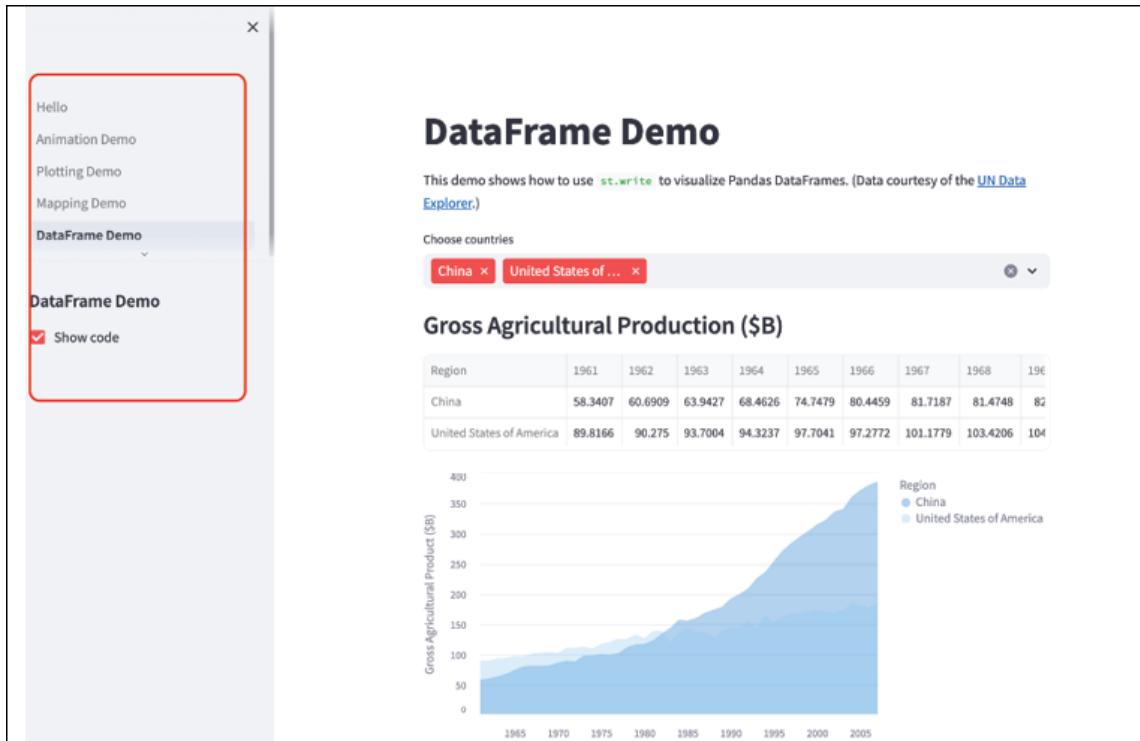
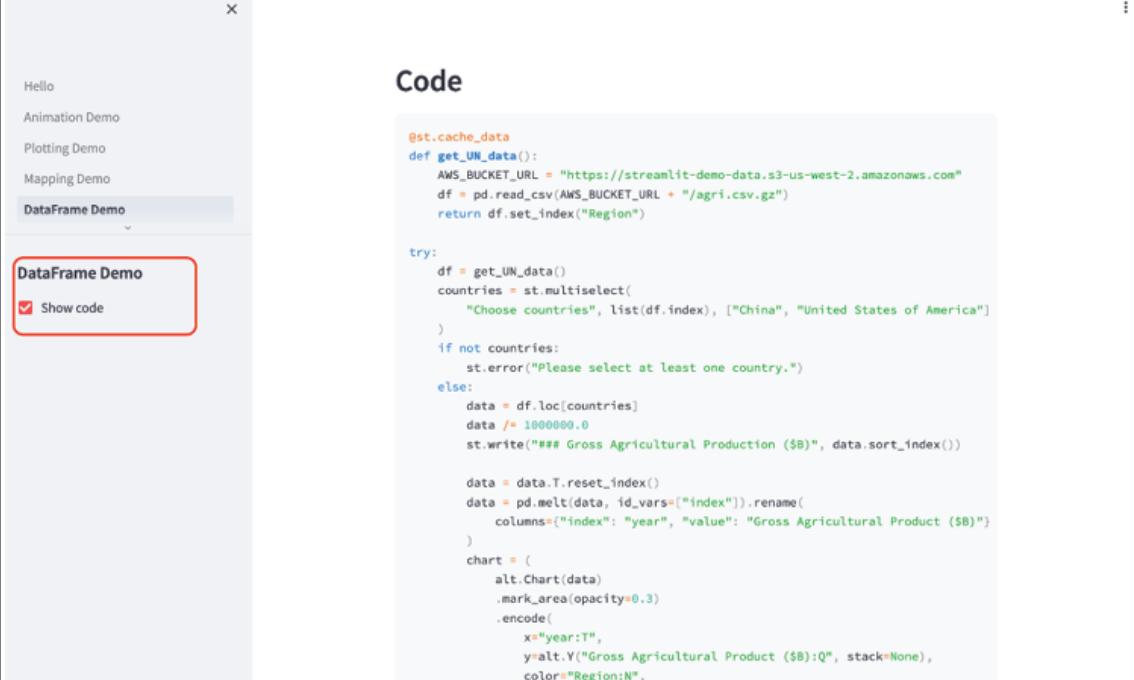


Figure 2.3: Streamlit Page for Dataframe demo.

You can also check the code for the respective page by clicking the show code toggle button. The screen is as follows:



The screenshot shows a Streamlit application window. On the left, a sidebar lists several demo options: Hello, Animation Demo, Plotting Demo, Mapping Demo, and DataFrame Demo. The DataFrame Demo option is selected and expanded, revealing a sub-menu with a checkbox labeled "Show code". A red box highlights this checkbox. The main content area is titled "Code" and contains the Python code for the DataFrame Demo. The code uses Streamlit's caching mechanism (@st.cache_data) to fetch data from an AWS bucket and then processes it to create a stacked area chart showing Gross Agricultural Product (\$B) over time for specific countries.

```

@st.cache_data
def get_UN_data():
    AWS_BUCKET_URL = "https://streamlit-demo-data.s3-us-west-2.amazonaws.com"
    df = pd.read_csv(AWS_BUCKET_URL + "/agri.csv.gz")
    return df.set_index("Region")

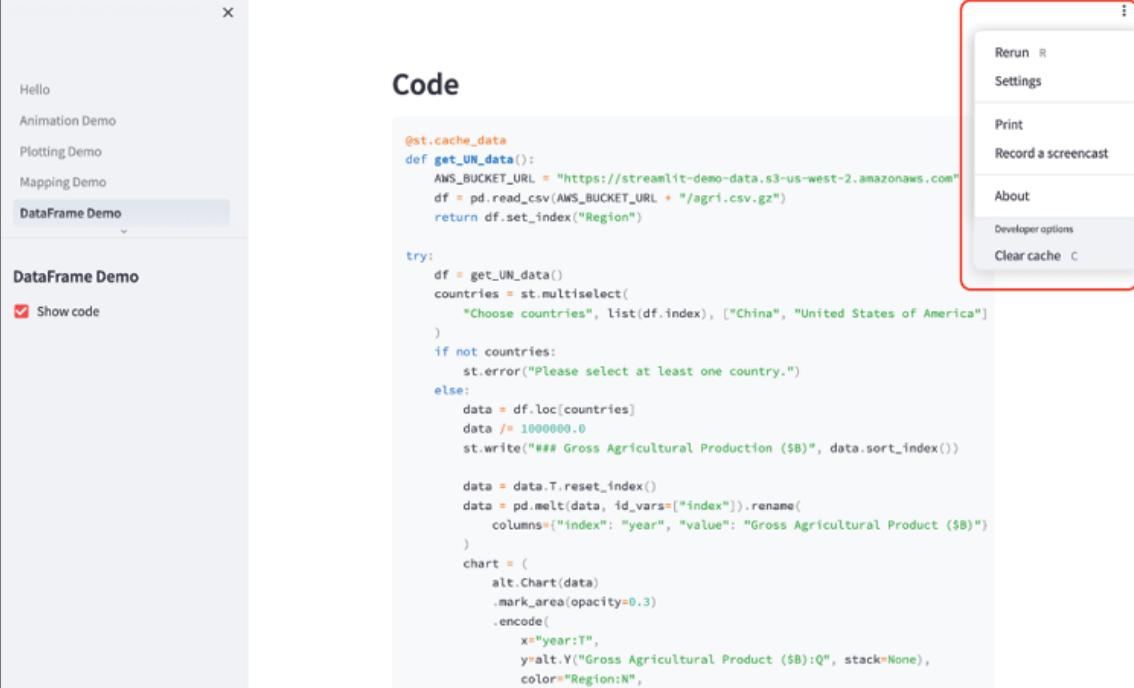
try:
    df = get_UN_data()
    countries = st.multiselect(
        "Choose countries", list(df.index), ["China", "United States of America"]
    )
    if not countries:
        st.error("Please select at least one country.")
    else:
        data = df.loc[countries]
        data /= 1000000.0
        st.write("### Gross Agricultural Production ($B)", data.sort_index())

        data = data.T.reset_index()
        data = pd.melt(data, id_vars=[ "index"]).rename(
            columns={"index": "year", "value": "Gross Agricultural Product ($B)"}
        )
        chart = (
            alt.Chart(data)
            .mark_area(opacity=0.3)
            .encode(
                x="year:T",
                y=alt.Y("Gross Agricultural Product ($B):Q", stack=None),
                color="Region:N",
            )
        )

```

Figure 2.4: Checkbox to explore the code behind the dataframe demo page

Let us understand each option provided in the submenu in the top-right corner, as shown in the following figure:



The screenshot shows the same Streamlit application as Figure 2.4, but with a red box highlighting the top-right corner of the main content area. This area contains a vertical submenu with several options: Rerun (with a 'R' icon), Settings, Print, Record a screencast, About, Developer options, and Clear cache (with a 'C' icon). The "Developer options" option is currently selected.

Figure 2.5: The submenu has additional options to configure applications

- **Rerun:** It reruns the code for the current page. This is useful when you have made changes in the code in your file and you want to rerun the Streamlit file to reflect the changes.
- **Settings:** You can set the appearance, theme, and other options like re-running the code every time to save the code base file.

Additional settings options for the application are as follows:

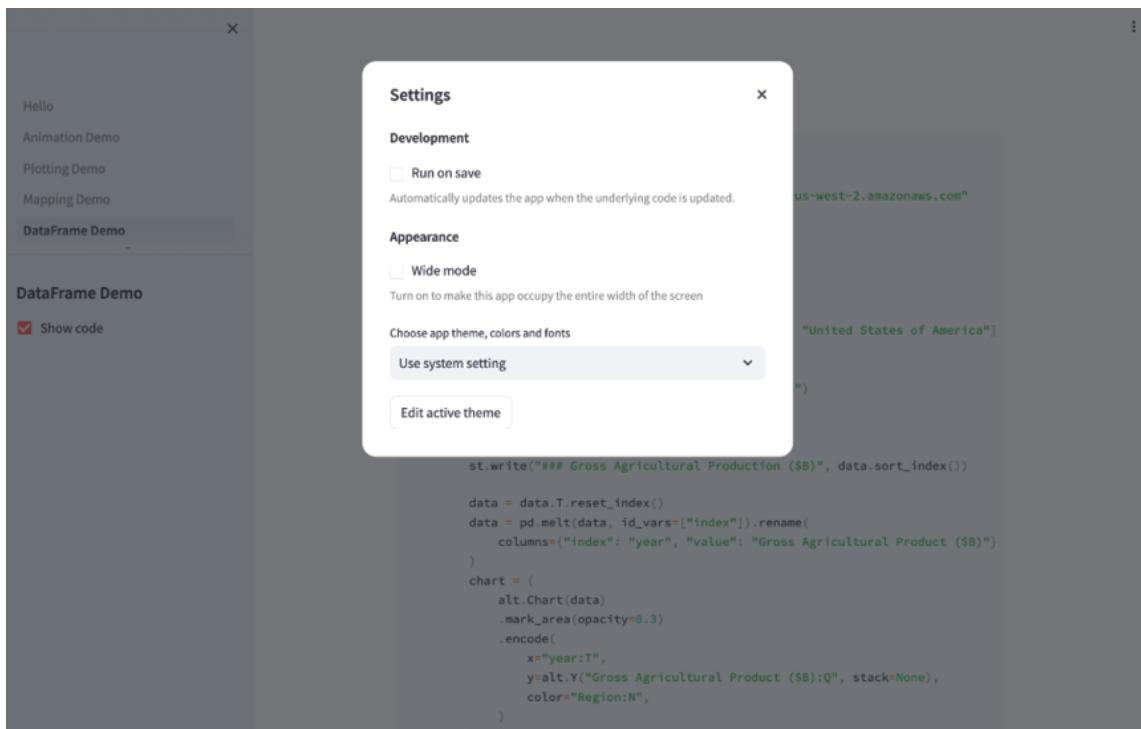


Figure 2.6: The setting options for the Streamlit application

- **Print:** You can print/save the entire webpage
- **Record a screencast:** It lets you record your screen. This is extremely useful when you want to share your Streamlit app demo offline. This also lets you record the audio, so you can also guide the user on how to use the application.
- **About:** It provides you with the Streamlit version of the app.

Clicking on About provides information about the Streamlit version being used, as shown in the following figure:

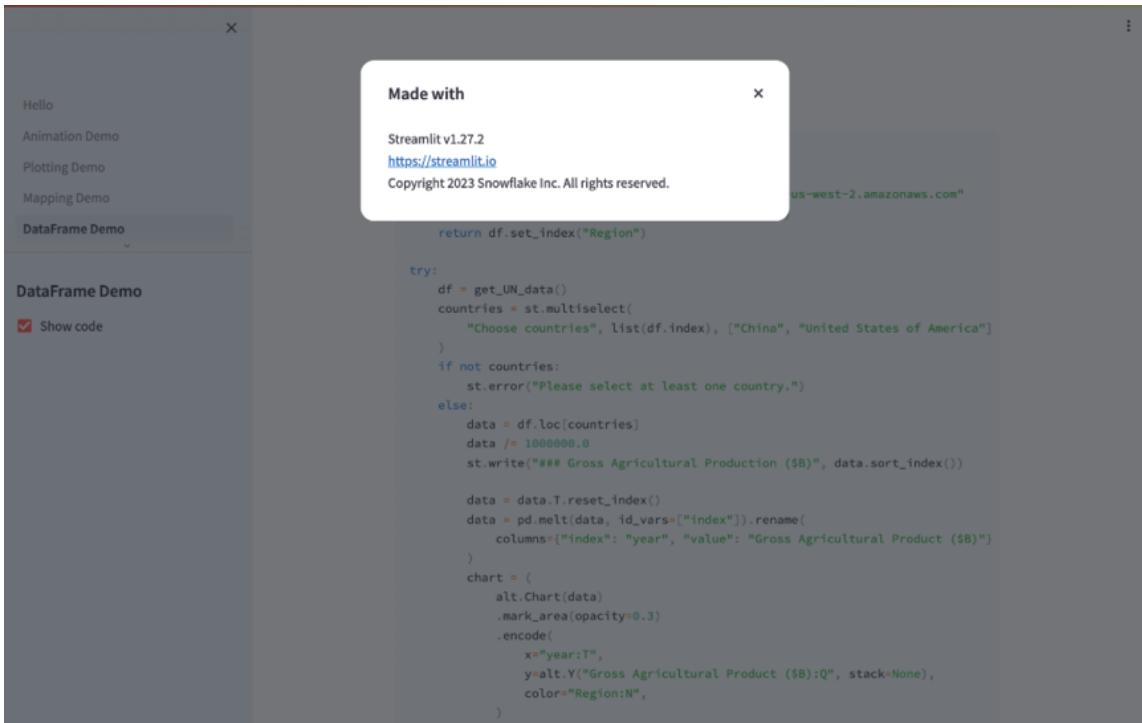


Figure 2.7: Streamlit version information

- **Clear cache:** Streamlit provides a helpful feature. By default, Streamlit caches the results fetched from the database or the results of expensive computations that have been performed. This helps the application perform better and saves costs.

You can build custom logic in your application to flush the cache and retain the results for a certain time. Streamlit provides multiple functions to achieve this. We will cover this under advanced concepts, but you can achieve this from the front end using this option.

Clicking on the **Clear cache** options provides the dialogue box to clear the cache for the application, as shown in the following figure:

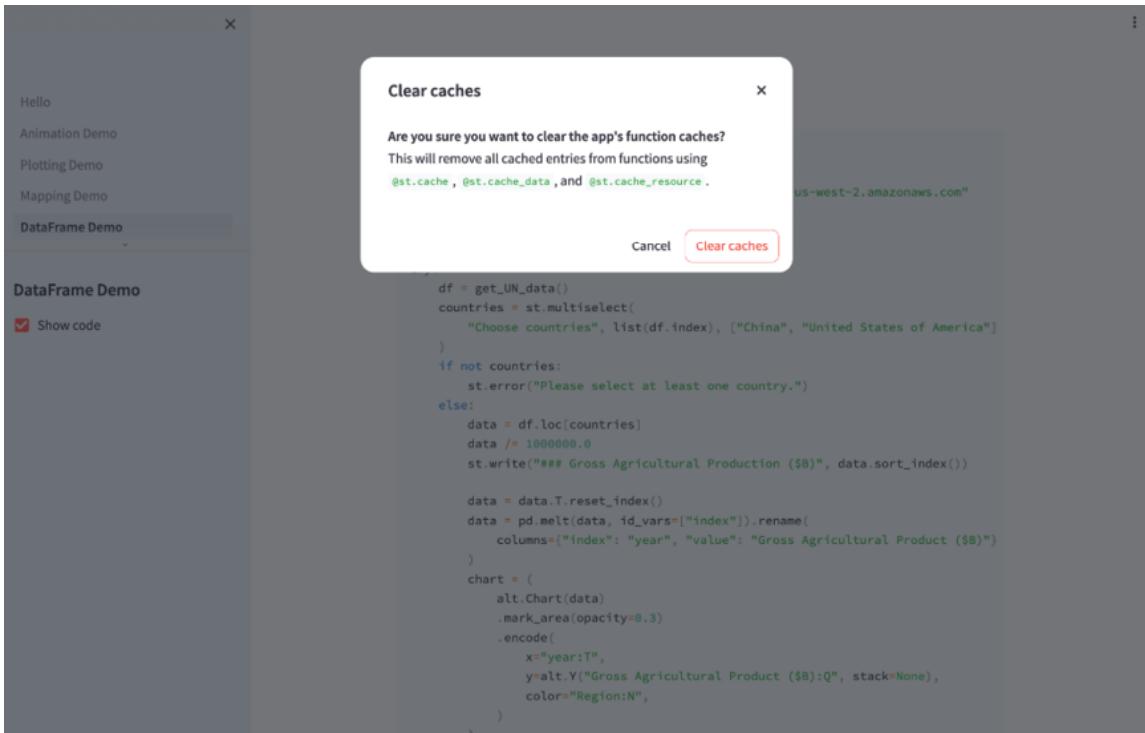


Figure 2.8: Clicking on the clear cache button clears the cache for the application

Hands-on: Product recommender app

In this section, we start by building our base project structure and running it using Docker. The prerequisites are as follows:

- Docker
- Favorite IDE

Project structure

Figure 2.9 represents our base project structure. We will build on this as we add more functionality to our application.



Figure 2.9: Project Structure

The functions of each of these are as follows:

- **Folder name: Product_Recommender:** This is the base folder for your application. When running the Streamlit code locally on a virtual environment for ease of use, you can place this folder inside the folder where you created the virtual environment.
- **data_app.py: Streamlit entry script:** This is the script that will be our main Streamlit entry point. It is important to note, as discussed in the above section, that when executing this script, Streamlit will run it from top to bottom. Every interaction on the webpage will trigger this execution. When designing or adding the components for Streamlit, we will constantly revise this principle to ensure our app is responsive and user-friendly.

Our current code for data application is very straightforward. The code is as follows:

```
import streamlit as st #Import the streamlit library
```

```
st.title('Product Recommender') #Set title of the webpage
```

- **requirements.txt:** In this file, we will track our dependencies. It is a good practice to maintain requirements files separately for every project, especially if you are building a containerized application. This helps to ensure your dependencies are tracked correctly and consistently.

The following is a sample **requirements.txt file**; since we just want Streamlit installed in our container, the requirements file contains only Streamlit:

```
streamlit==1.37.0
```

To get the latest version of the Streamlit, check the official Streamlit page.

<https://docs.streamlit.io/develop/quick-reference/release-notes>

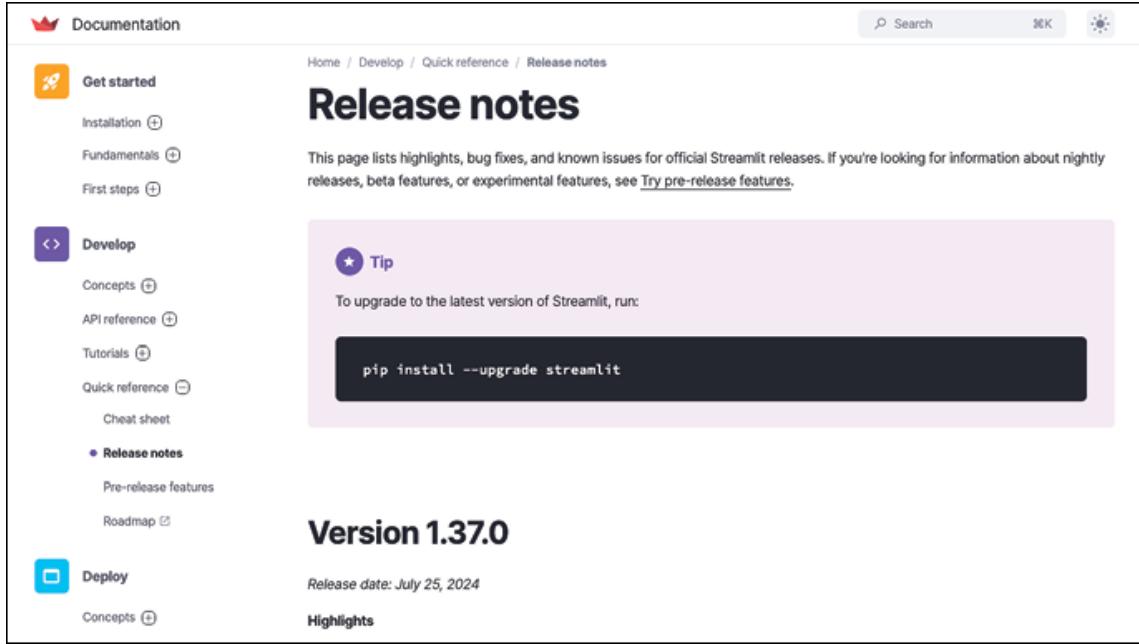


Figure 2.10: Streamlit official release page

- **Dockerfile:** This is our Docker file, which will help us build the container. It consists of instructions for the libraries and files to be installed and copied in your container. If you are not familiar with using containers for application development, you can refer to Docker 101 (Appendix C).

Sample Docker file first iteration of Producer recommender application. The code is as follows:

```
FROM Python:3.9-slim
```

```
EXPOSE 8501
```

```
RUN pip install --upgrade pip
```

```
WORKDIR /app
```

```
COPY requirements.txt .
```

```
RUN pip install --no-cache-dir -r requirements.txt
```

```
COPY data_app.py .
```

```
CMD streamlit run data_app.py --server.port=8501 --  
browser.serverAddress="0.0.0.0"
```

Running your first Streamlit app

Since we are using the Docker container to run the Streamlit app, we will be building our container first and then running the container as follows:

1. Make sure you are in the Base project directory (the directory where all your project files are, including Dockerfile and Streamlit Python script)
2. Execute the docker build command

```
cd Product_Recommender
```

```
docker build -t product_recommender:v1 .
```

Note: We use the -t flag to tag our image. It is a good practice to tag each revision of the image.

For version tracking and faster recovery. You can read about more available options in the docker official documentation.

<https://docs.docker.com/engine/reference/commandline/build/>

3. The successful output of the build should look like this. This implies your docker image is built and ready. The output screen is as shown in the following figure:

```

StreamlitDemo >> cd Product_Recommender
StreamlitDemo >> docker build -t product_recommender:v1 .
[+] Building 0.3s (11/11) FINISHED
=> [internal] load build definition from Dockerfile
=> => transferring dockerfile: 303B
=> [internal] load .dockerrcignore
=> => transferring context: 2B
=> [internal] load metadata for docker.io/library/python:3.9-slim
=> [1/6] FROM docker.io/library/python:3.9-slim
=> [internal] load build context
=> => transferring context: 147B
=> CACHED [2/6] RUN pip install --upgrade pip
=> CACHED [3/6] WORKDIR /app
=> CACHED [4/6] COPY requirements.txt .
=> CACHED [5/6] RUN pip install --no-cache-dir -r requirements.txt
=> [6/6] COPY data_app.py .
=> exporting to image
=> => exporting layers
=> writing image sha256:11c07a4b299034880947a386aafe13481083ee897fafadaf09cc399463e1c0d9
=> => naming to docker.io/library/product_recommender:v1

```

Figure 2.11: Sample docker build logs

- Run the image. Running the image means that you are running your Streamlit code inside the container.

```
docker run -p 8501:8501 product_recommender:v1
```

Note: With the `-p` flag, we are mapping port **8501** of the container (which we exposed in our dockerfile) to **8501** of our physical machine.

For a more elaborate guide and more options, you can check the official docker documentation.

<https://docs.docker.com/engine/reference/commandline/run/>

- The successful output of the execution should give you a running docker container with a link to your local host to access the Streamlit page, as shown in the following figure:

```

StreamlitDemo >> docker run -p 8501:8501 product_recommender:v1

Collecting usage statistics. To deactivate, set browser.gatherUsageStats to False.

You can now view your Streamlit app in your browser.

URL: http://0.0.0.0:8501

```

Figure 2.12: Docker Run for Streamlit application with hosted URL

- Once you access the webpage URL, you can see the Streamlit page:



Figure 2.13: Home page for Product Recommender application

This is the first Streamlit page. Although this does not seem too fancy, we are just getting started. We have added the title and deployed the Streamlit using the docker container.

Conclusion

We explored how easy it is to set up Streamlit and turn our data scripts into web apps. We looked at various installation options and learned about virtual environments and Docker. Get more details on docker from Appendix C. You should now have a solid foundation and your very first Streamlit app ready. The best learning comes from practicing; hence, we should keep playing around with what we have covered.

In the next chapter, we will learn about the Streamlit core widgets and how to use them to enhance our product recommender application. Streamlit core widgets are the building blocks of Streamlit's functionality.

Exercises

- 1. Why is Streamlit considered user-friendly for beginners?**
 - a. It requires multiple complex installation steps.
 - b. It is only suitable for seasoned data scientists.

- c. It turns data scripts into complicated web apps.
- d. With a single command, you can have Streamlit up and running.

2. Why is it a good practice to use a virtual environment for local development?

- a. It allows for multiple projects to share the same dependencies.
- b. It prevents you from installing new packages.
- c. It ensures a unique set of requirements and configurations for each project.
- d. It slows down the coding process.

3. What is Docker's primary benefit for Streamlit?

- a. It makes the setup less reliable.
- b. It packages applications into containers with all necessary components.
- c. It is only suitable for experts in coding.
- d. It makes the application harder to deploy on cloud platforms.

4. What is the purpose of the Streamlit hello command?

- a. To say “hello” to the Streamlit community.
- b. To check the version of Streamlit installed.
- c. To validate the correct installation of Streamlit, load the first Streamlit app with guides and demos.
- d. To terminate a running Streamlit app.

5. For what reason might someone use the Streamlit Community Cloud?

- a. To make their app more complex to run.

- b. To deploy and share Streamlit apps without managing infrastructure.
 - c. To store personal files and images.
 - d. To incur additional costs for basic app deployment.
6. **What does Streamlit excel at in its fundamental design?**
- a. Making web apps more complicated.
 - b. Requiring user authentication for every feature.
 - c. Turning data scripts into shareable web apps without typical web development overhead.
 - d. Serving as a database for large scale storage.
7. **How does Streamlit behave when a user interacts with a Streamlit-powered webpage, like clicking a widget?**
- a. The Streamlit page never reruns, making interactions static.
 - b. Only the widget changes, without any effect on the Streamlit script.
 - c. The Streamlit page reruns from top to bottom, reflecting the interaction effects.
 - d. It freezes the page to prevent further interactions.
8. **Why does Streamlit cache the results by default?**
- a. To make the application run slower.
 - b. To require manual updates for every change.
 - c. To avoid reloading and recomputing expensive operations every time the user interacts with the webpage.
 - d. To limit the amount of data that can be shown.

Answers

1. d

2. c

3. b

4. c

5. b

6. c

7. c

8. c

Join our book's Discord space

Join the book's Discord Workspace for Latest updates, Offers, Tech happenings around the world, New Release and Sessions with the Authors:

[https://discord\(bpbonline\).com](https://discord(bpbonline).com)



CHAPTER 3

Exploring Streamlit Widgets

Introduction

In the third chapter, we will learn and get comfortable with Streamlit widgets. Widgets are the building blocks of any application. Streamlit provides different types of widgets for the users to get started with the application development faster. These widgets are straightforward to use, allowing you to bring your ideas to life more rapidly. This chapter will guide you through the different types of widgets available and show you how to use them effectively to create interactive and user-friendly applications.

Structure

The chapter discusses the following topics:

- Widgets overview
- Streamlit core widgets
- Streamlit magic
- Hands-on: Adding widgets to enhance the app

Objectives

Widgets are tools that let your app's users talk back to it. They are the levers, buttons, and dials that make your app respond and

interact with those who use it. From simple buttons to sliders to text inputs, widgets are the building blocks of a two-way conversation between the app and its audience. By the end of this chapter, you will get a hands-on understanding of the most versatile and user-friendly widgets that Streamlit offers.

Widgets overview

Widgets are important because, without them, an app is just a static display, a one-way street of information. With widgets, however, your app becomes a two-way road full of interaction and engagement. Imagine a cooking app where you cannot adjust the serving size, or a music player where you cannot control the volume. That is what an app without widgets would look like.

Streamlit core widgets

Streamlit offers a variety of elements to enhance your data app. We have broken down the widgets into different sections based on their functions and usage for better segregation of the options available.

Display widgets

These widgets are the foundational blocks for any streamlit app and form the backbone of user interaction. Streamlit enables easy display of text, markdown, code, captions, or mathematical equations, allowing for rich content formatting within your application. Display widgets include the following:

- **st.title**: Adds title to your page
- **st.header**: Adds header in your page
- **st.subheader**: Adds **subheader** in the page
- **st.write, st.text**: Adds text in your page
- **st.markdown**: Adds markdown in the page
- **st.caption**: Puts the text in a caption
- **st.code**: Adds code block in the page

- `st.latex`: Adds complex mathematical equations
- `st.divider`: Adds a horizontal divider
- `st.json`: Displays a JSON in the page

The following code is an example of using the display widgets.

```
import streamlit as st

#title and header
st.title('This is a title')
st.header('This is a header')
st.subheader('This is a subheader')

# text and markdown
st.write('Hello, World!')
st.text('This is some text.')
st.markdown('**Bold** and _Italic_ Markdown Text')
st.caption('This is a caption that explains something above.')

# displays code
code = '''def hello_streamlit():
    print("This is code!")'''
st.code(code, language='python')
# displays mathematical functions
st.latex(r'''
    a + ar + a r^2 + a r^3 + \cdots + a r^{n-1} =
    \sum_{k=0}^{n-1} a r^k =
    ''')
```

```
a \left(\frac{1-r^n}{1-r}\right)
```

```
'''
```

```
# draws horizontal ruler
```

```
st.divider()
```

```
st.json({
```

```
    'foo': 'bar',
```

```
    'baz': 'boz',
```

```
    'stuff': [
```

```
        'stuff 1',
```

```
        'stuff 2',
```

```
        'stuff 3',
```

```
        'stuff 5',
```

```
    ],
```

```
})
```

The compiled Streamlit page for the aforementioned code snippet looks as follows:

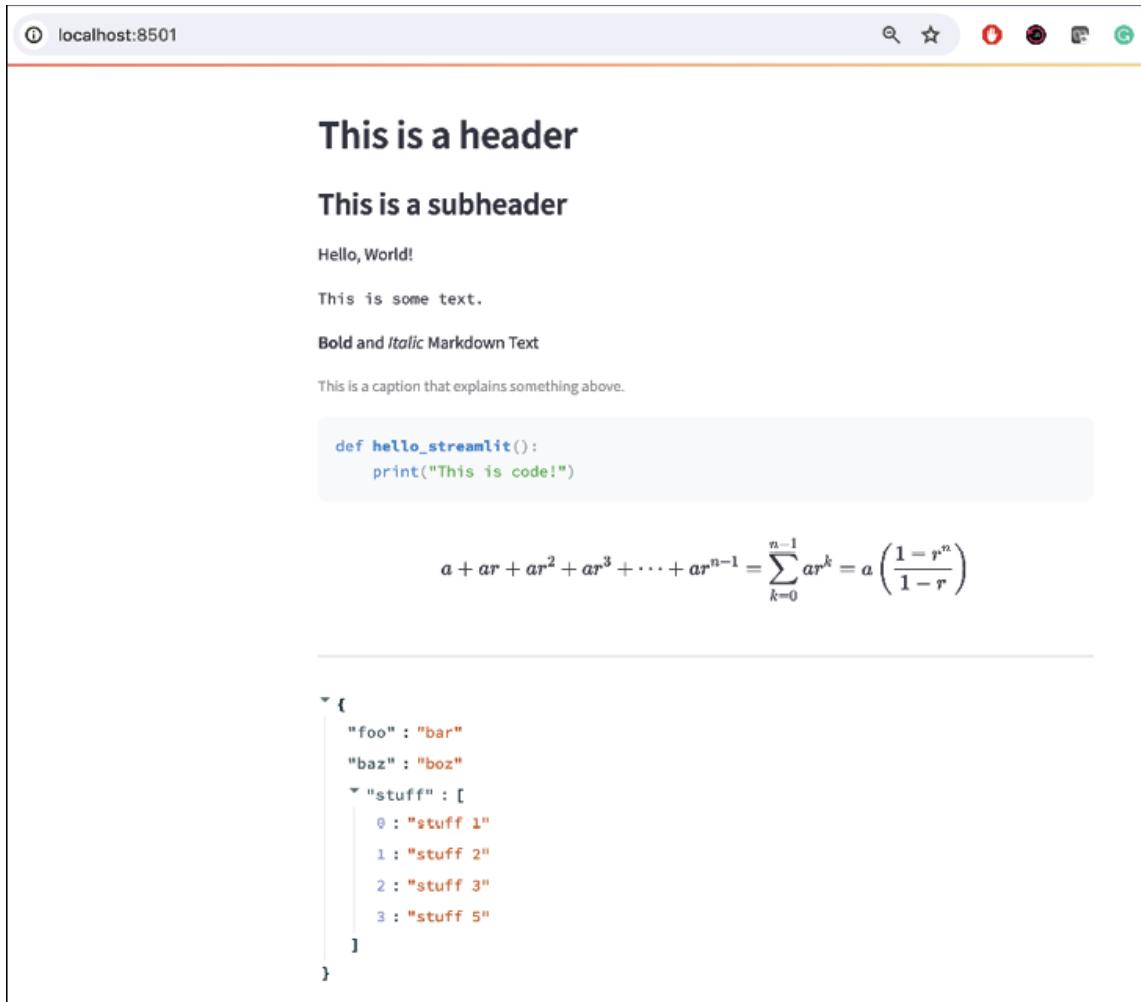


Figure 3.1: Streamlit application with display widgets

Text input widgets

These widgets allow users to input text through simple input fields or larger text areas for more extensive. Text input widgets include:

- `st.text_input`: Accept text input in the page.
- `st.text_area`: Accept large text input in the page

The following code is to build a page to use the text input widgets:

```
import streamlit as st
```

```
st.text_input('Enter name:', 'Type here...')  
st.text_area('Enter your message:', 'Type here...')
```

The compiled Streamlit page for the code looks as follows:

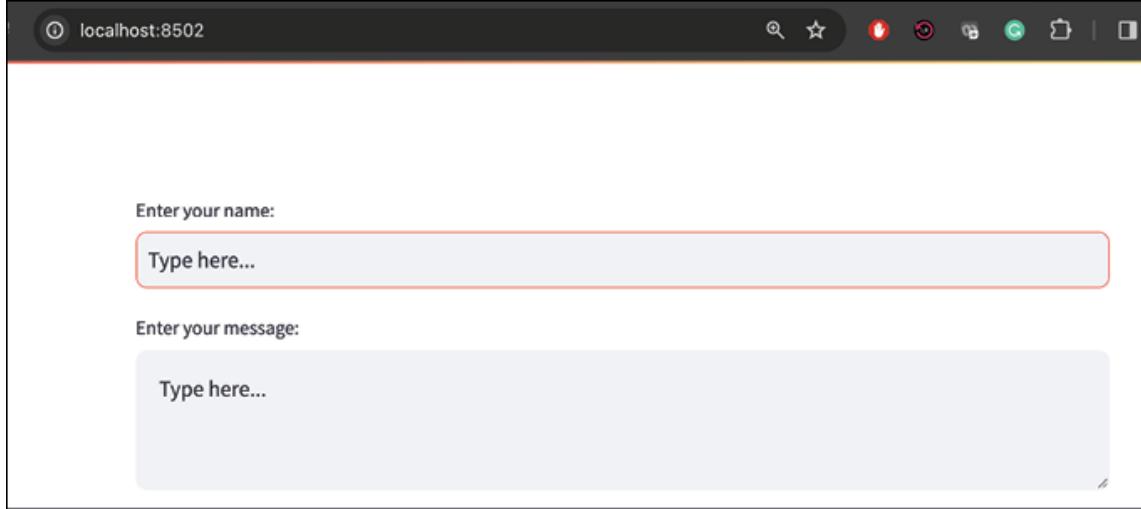


Figure 3.2: Streamlit app with text input widgets

Numerical input widgets

This widget allows the user to enter a numerical input. When the acceptable range is specified, Streamlit does a type check and the value check for the input entered. The numerical input widget include:

- `st.number_input`: Accept numeric inputs in the page, with capability to set checks.

The following code snippet is to test the numerical input widget:

```
import streamlit as st
```

```
st.number_input('Enter number', min_value=10,  
max_value=100)
```

The compiled page for the aforementioned code snippet looks as follows:

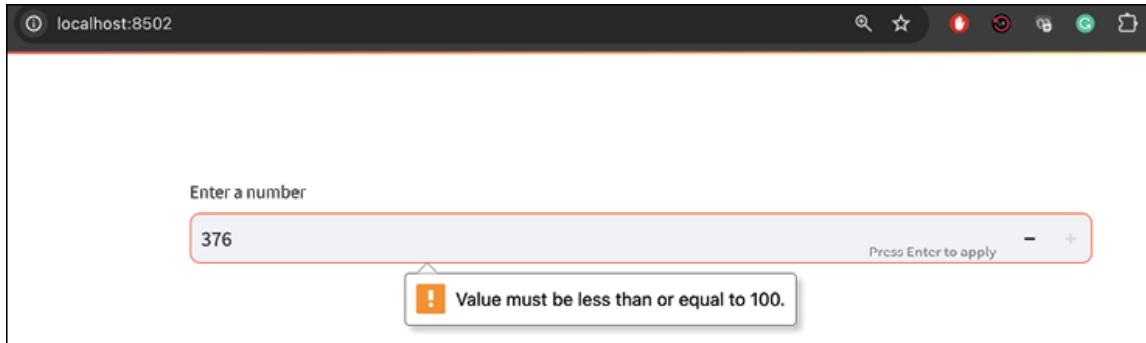


Figure 3.3: Streamlit app with numeric input widgets

Date and time widgets

These widgets allow you to select the date, date range from the calendar or time value. Date and time widgets include:

- **date_input:** Accepts date input in the page.
- **time_input:** Accepts time input in the page.

The following code builds a sample page for the date and time widget:

```
import datetime  
  
st.date_input('Enter date')  
  
  
st.time_input('Enter time')  
  
  
today = datetime.date.today()  
last_week = today - datetime.timedelta(days=7)  
st.date_input(  
    "Select date range ",  
    value = (last_week ,today),  
    min_value = datetime.date(2022, 12, 1),  
    max_value = datetime.date.today(),
```

```
    format="MM/DD/YYYY",  
)  
)
```

The following is the compiled Streamlit page for the aforementioned code:

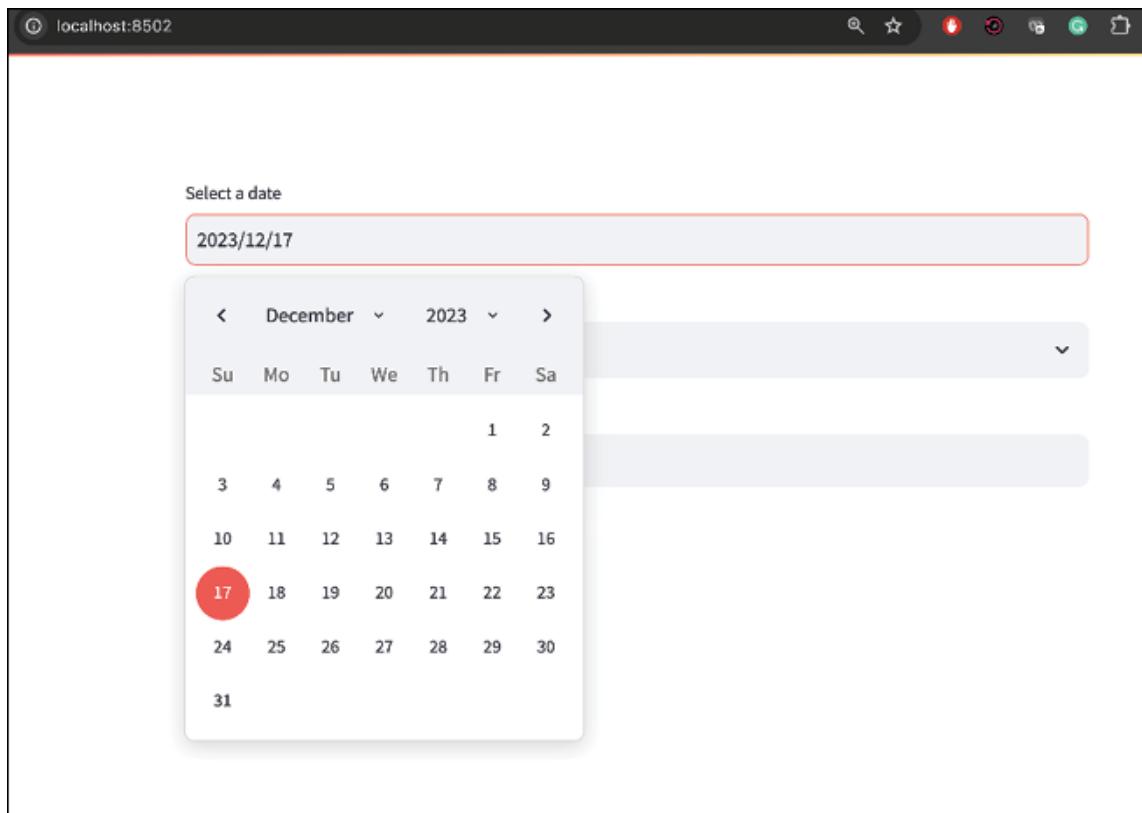


Figure 3.4: Streamlit app with date input widget

The following Streamlit page shows the behavior of the time widget:

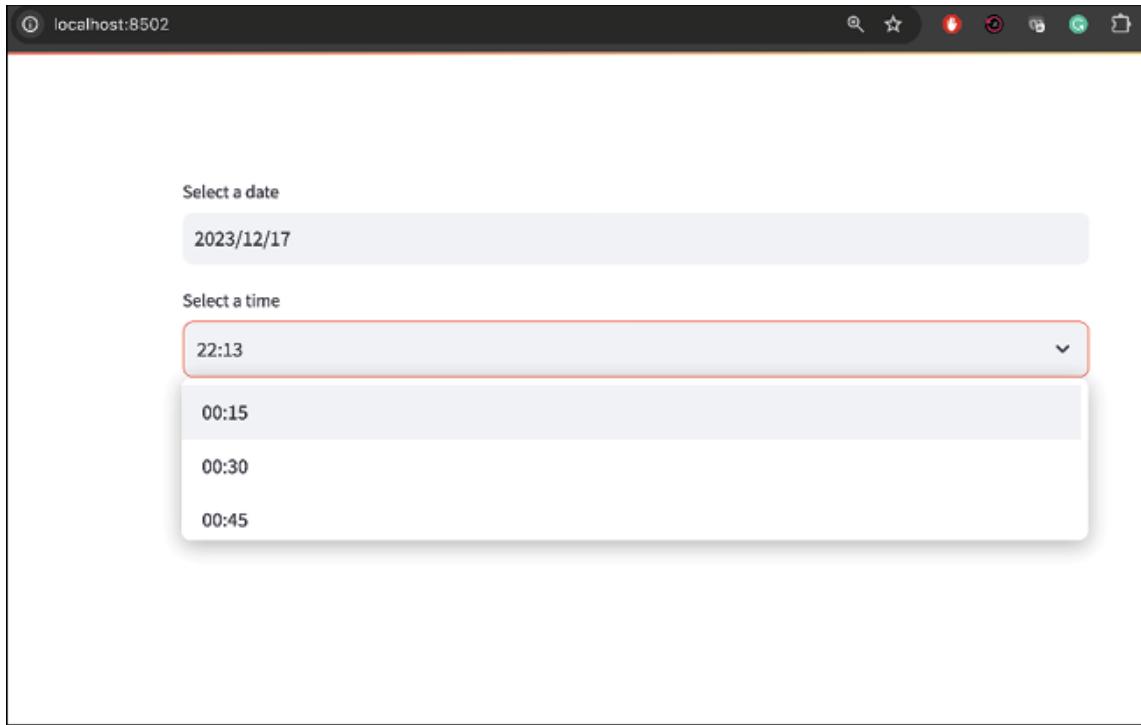


Figure 3.5: Streamlit app with time input widget

The following Streamlit page shows the behavior of the date range widget; this can be used to select the start and end date together:

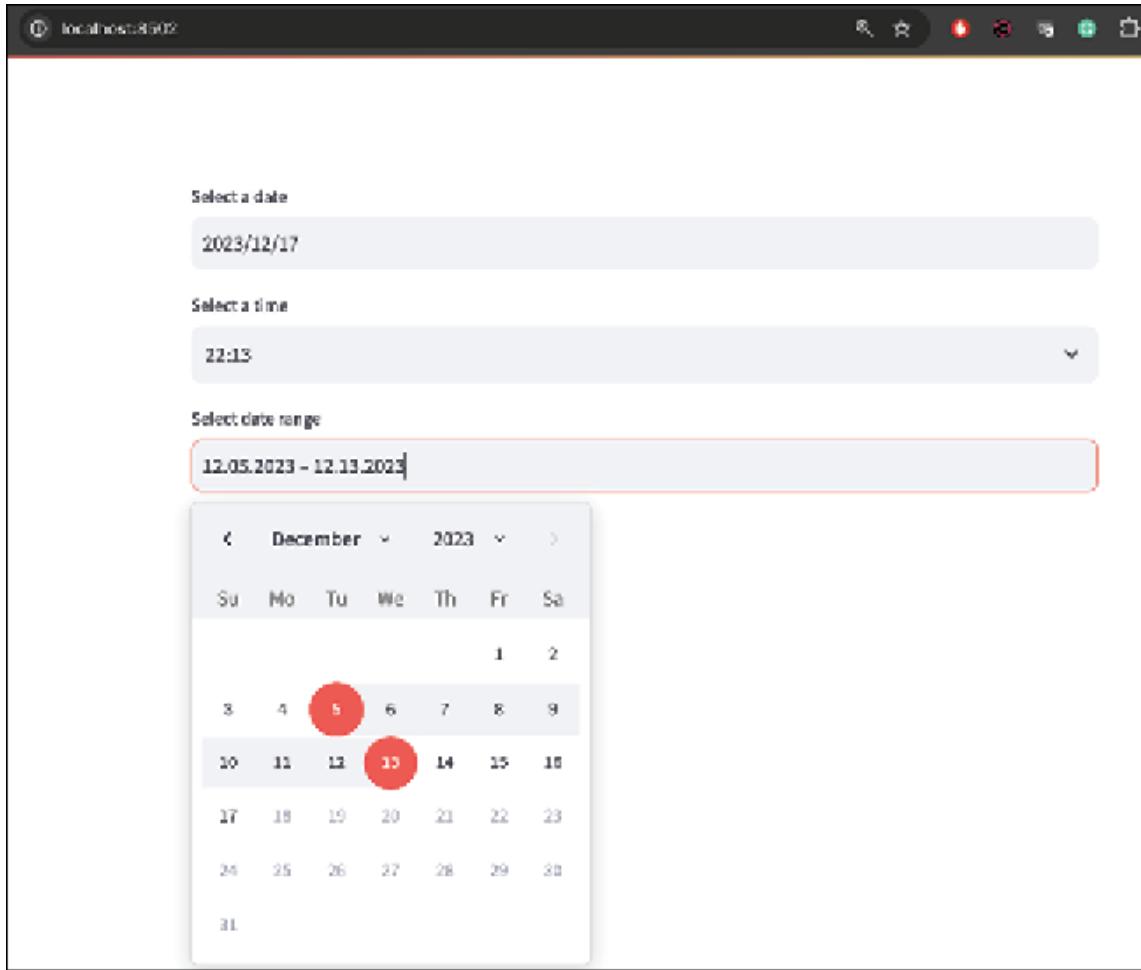


Figure 3.6: Streamlit app with date range input widget

Other input widgets

Streamlit offers other input widgets for capturing values of different formats like images, colors, or files. These widgets include the following:

- `st.camera_input`: Accept input from camera
- `st.color_picker`: Accept color code input
- `st.file_uploader`: Upload file in the app

The following code snippet is to test out the functionality of the file uploaded widget:

```
import streamlit as st
```

```
uploaded_file = st.file_uploader("Choose a file")
```

The compiled Streamlit page with file uploader. When clicking on the browse files button, a file explorer pane opens which lets you select the file from the local drive. The output is as follows:

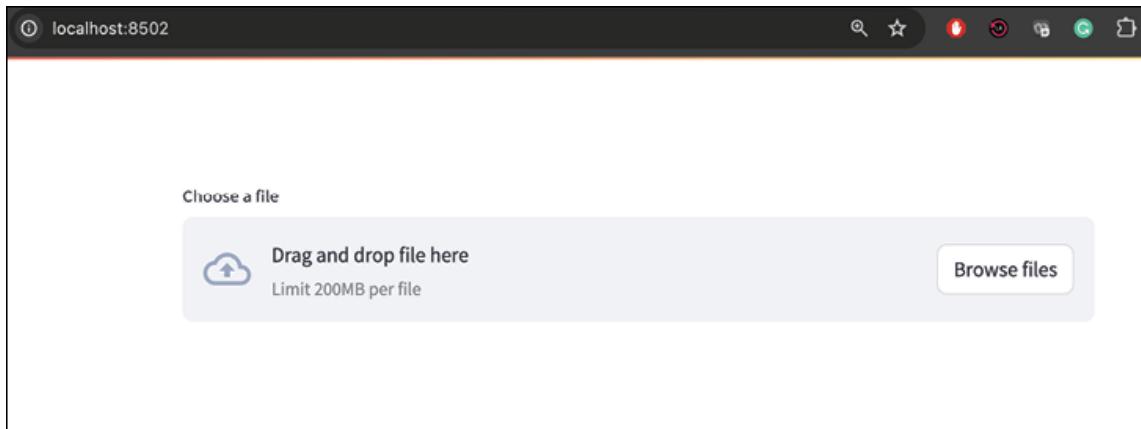


Figure 3.7: Streamlit app with file uploader widget

The following code snippet is to test the camera input widget:

```
import streamlit as st
```

```
picture = st.camera_input("Take a picture")
```

The following is the output for the camera input widget. The widget utilizes the default camera system setting to capture the input.

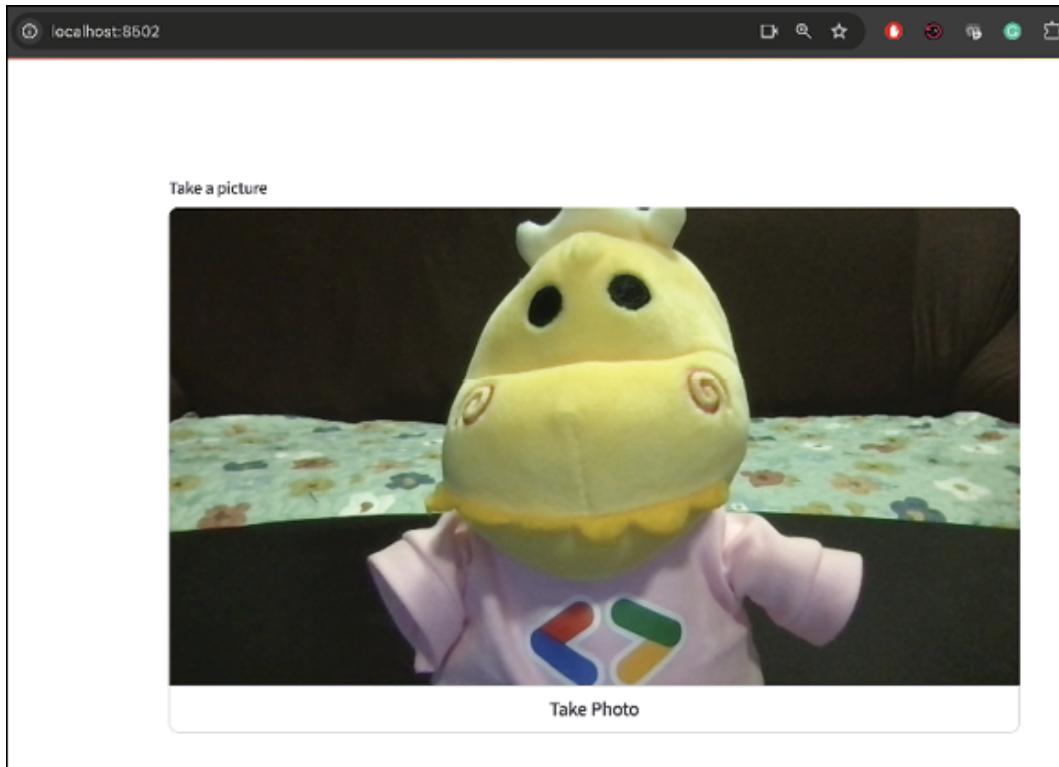


Figure 3.8: Streamlit app with camera input widget

Streamlit code snippet for color picker widget:

```
import streamlit as st  
  
color = st.color_picker('Pick A Color', '#00f900')
```

The compiled code for the Streamlit application. We have selected the default code to be green; any other color can be selected, and a default value can be set, as shown in the following figure:

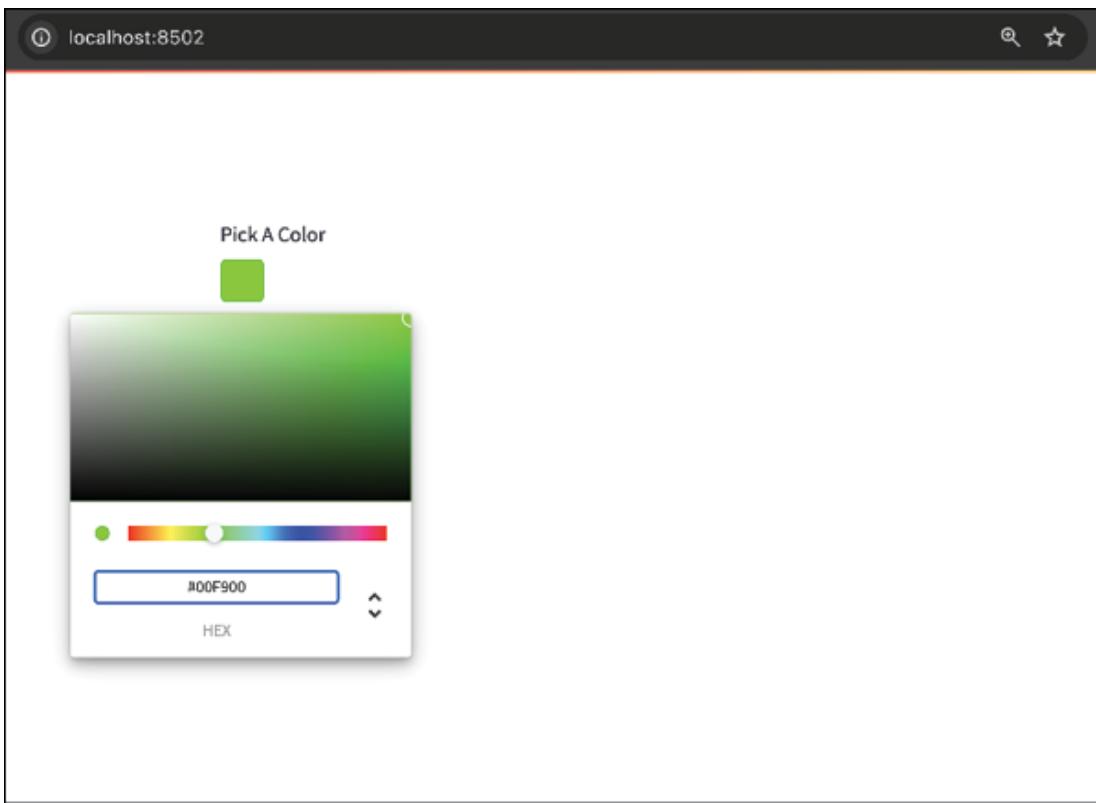


Figure 3.9: Streamlit app with color picker widget

Single choice widgets

For the scenarios where you want to limit the user input to a fixed set of responses, you can use choice widgets. Single-choice widgets allow passing only a single value to the application. These widgets include the following:

- `st.checkbox`: Checkbox selection input

Use case: This widget is useful when the user is required to select multiple options.

For example: Please select all the options that apply.

- `st.toggle`: Accepts Boolean input flag

Use case: This widget provides a switch to turn the value on/off.

For example: Switching from dark mode to light mode in an application.

- `st.radio`: Accepts input with radio button selection

Use case: This widget is useful when only one value needs to be selected

For example: Selecting a preferred payment method.

- `st.selectbox`: Accepts input with dropdown selection

Use case: This widget is useful when the user is required to choose a value from the set of values.

For example: Select the country of residence.

- `st.slider`: Accepts input with slider selection.

Use case: This widget is useful when the user wants to select the range as an input.

For example: Select the duration of stay

The following is the code snippet for single-choice widget:

```
import streamlit as st

#single input select

agree = st.checkbox('I agree')
# checkbox returns true/false
if agree:
    st.write('You have agreed!')

st.divider()
on = st.toggle('Activate feature')
# toggle returns true/false
if on:
    st.write('Feature activated!')
```

```
st.divider()

color = st.radio(
    "What's your favorite Color",
    [":rainbow[Rainbow]", "Black"],
    captions = ["All the colors in Rainbow", "Black
rules"])

# The application can behave as per selection made in
the radio button

if color == ':rainbow[Rainbow]':
    st.write('You selected rainbow.')
else:
    st.write("You didn't select rainbow.")

st.divider()

option = st.selectbox(
    "How would you like to be contacted?",
    ("Email", "Home phone", "Mobile phone"),
    index=None,
    placeholder="Select contact method...",
)

st.write('You selected:', option)

st.divider()
```

```
age = st.slider('How old are you?', 0, 130, 25)  
st.write("I'm ", age, 'years old')
```

The compiled code for the single input widget looks as follows:

The screenshot shows a Streamlit application interface. At the top, there is a checkbox labeled "I agree". Below it is a toggle switch labeled "Activate feature".

Underneath, there is a section titled "What's your favorite Color" with two options: "Rainbow" and "Black". The "Black" option is selected, indicated by a red dot next to the label. A note below says "You didn't select rainbow".

Next is a dropdown menu labeled "How would you like to be contacted?" with the placeholder "Select contact method...". A note below says "You selected: None".

Finally, there is a slider labeled "How old are you?" with a range from 0 to 130. The slider has a red dot at the value 0. A note below says "I'm 0 years old".

Figure 3.10: Streamlit app with single choice selection widgets (pre-selection)

For the single-choice widgets, only one option is allowed to be selected, as shown in the following figure:

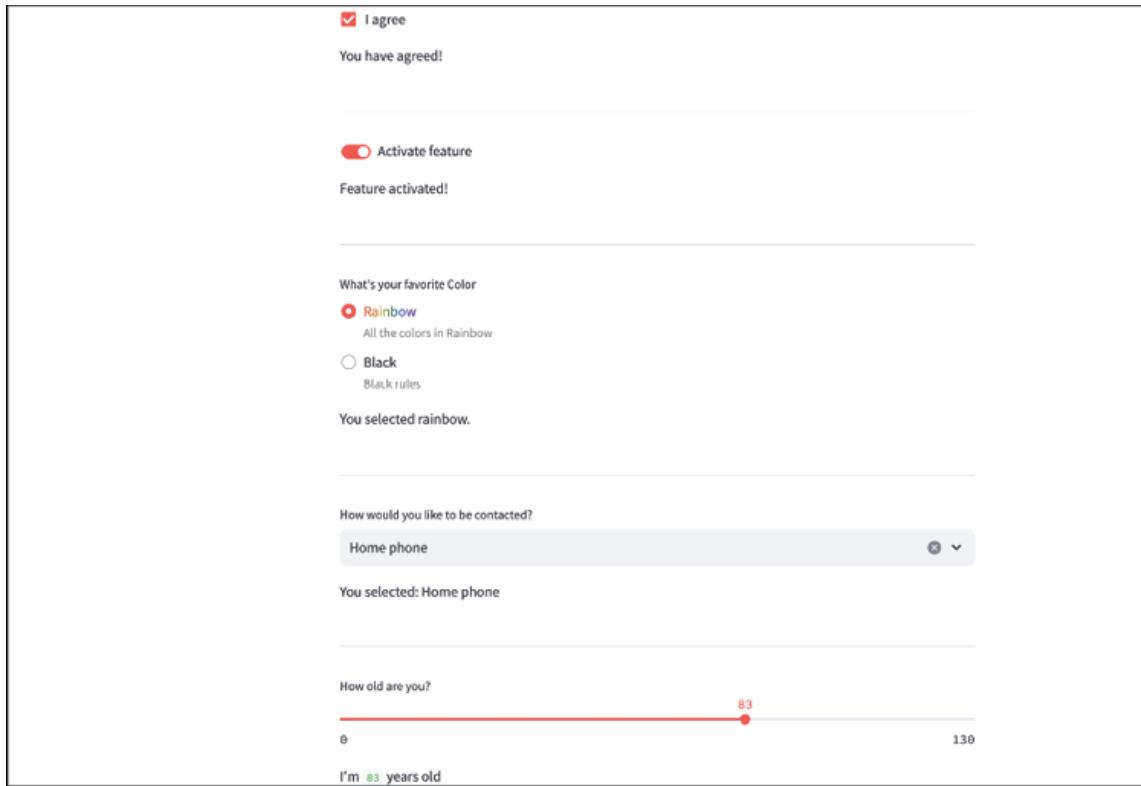


Figure 3.11: Streamlit app with single choice selection widgets (post-selection)

Multiple choice widgets

Similar to single-choice widgets, multiple-choice widgets can be used to restrict the user input to a filtered set of values. However, multiple-choice widgets allow the selection of multiple values, or a range of values, by the user. These widgets include:

- `st.multiselect`: Selects multiple options from the dropdown.
- `st.select_slider`: Accepts a range input from the slider.

The following code snippet is Streamlit code for multichoice widgets:

```
import streamlit as st

options = st.multiselect(
    'What are your favorite colors',
```

```

['Green', 'Yellow', 'Red', 'Blue'], #Options
available in the dropdown

['Yellow', 'Red']) #Default options selected

st.write('You selected:', options)

st.divider()

start_color, end_color = st.select_slider(
    'Select a range of color wavelength',
    options=['red', 'orange', 'yellow', 'green',
    'blue', 'indigo', 'violet'],
    value=('red', 'blue'))

st.write('You selected wavelengths between',
start_color, 'and', end_color)

```

The following figure is the compiled Streamlit page for the aforementioned code snippet:

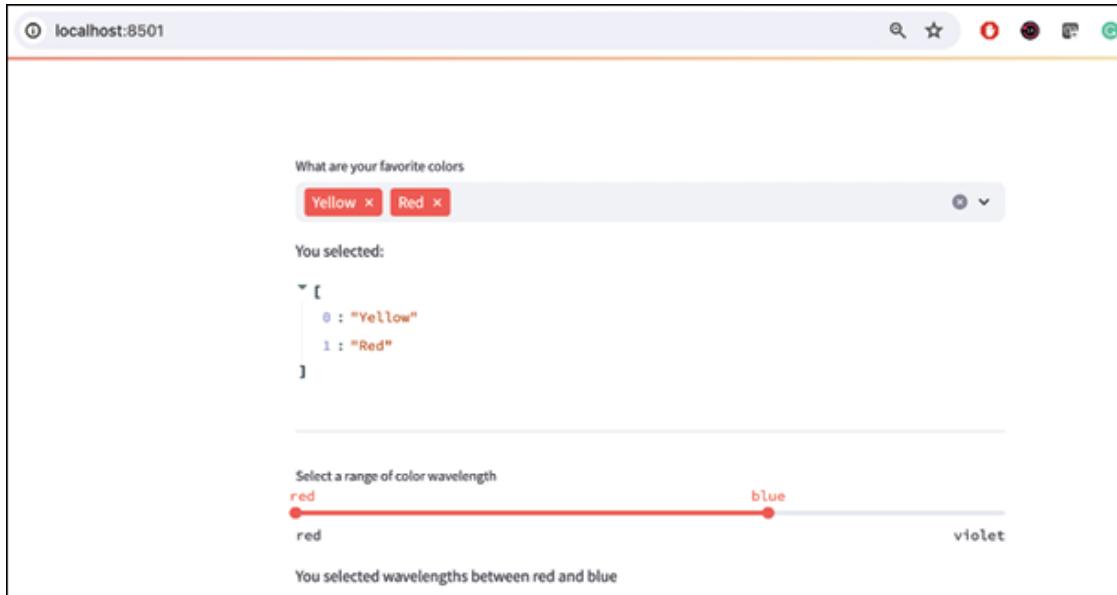


Figure 3.12: Streamlit app with multiple choice selection widgets

Button widgets

Button widgets help add dynamic processes and trigger event actions to your Streamlit pages. Buttons are one of the simplest input widgets but coupled with session caching, it can be used to implement advanced use cases. First, let us understand the basic functionality. These button widgets include the following:

- `st.button`: Button widget to click and perform actions like callback and submission.
- `st.download_button`: Button widget with function to download.
- `st.link_button`: Button widget tied to a hyperlink.

The following code snippet lets us try the button widgets:

```
import streamlit as st

st.button("Reset", type="primary")
if st.button('Say Good morning'):
    st.write('Good Morning ')
else:
    st.write('Good Evening')

st.divider()

#download the string as a file
text_contents = '''This is some text'''
st.download_button('Download some text', text_contents)

st.divider()

st.link_button("Go to gallery",
"https://streamlit.io/gallery")
```

Button widgets are generally tied to certain actions. The following Streamlit page shows the pre-click state of the page:

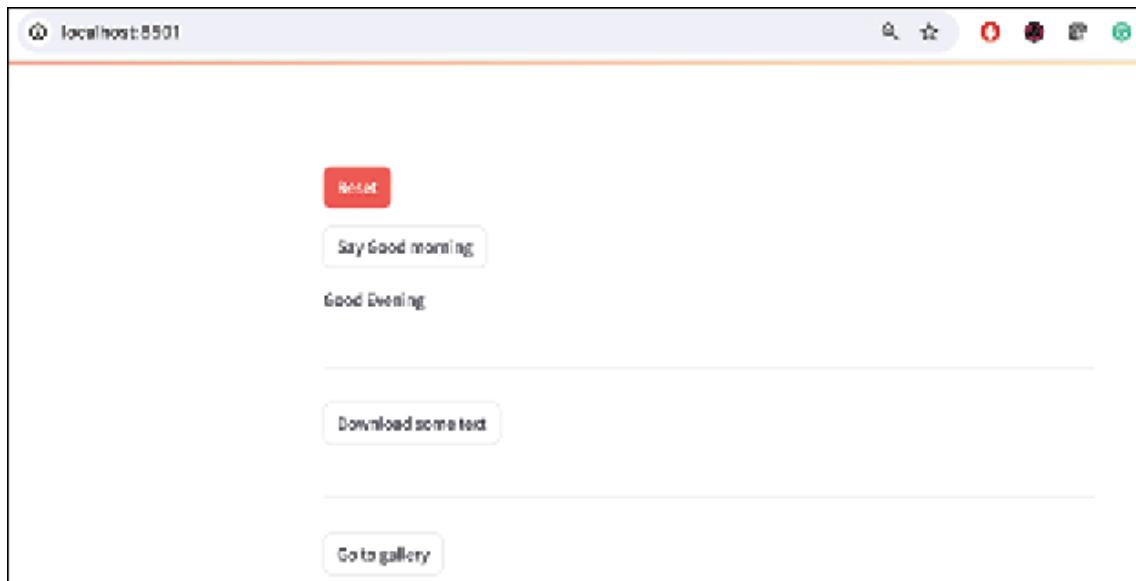


Figure 3.13: Streamlit app with button widgets (pre-click)

The following Streamlit page shows the post-click state of the page, with the downloaded file in the browser and display values reset as per the button actions:

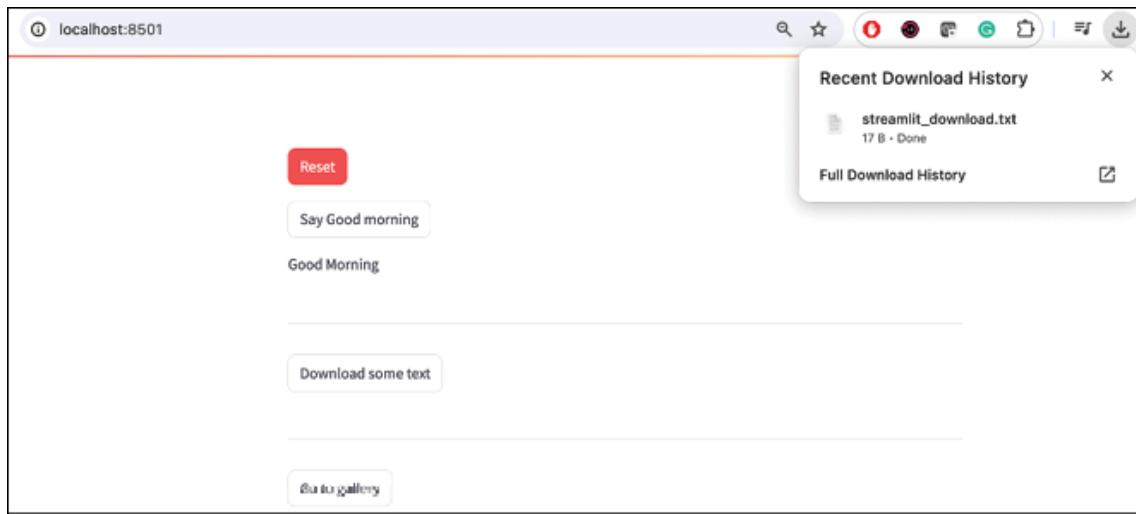


Figure 3.14: Streamlit app with button widgets (post-click)

Data widgets

Many times, the use case might require the data to be displayed in a tabular format. Streamlit provides data widgets that can display static tables and edit the cell values. These widgets include:

- `st.dataframe`: Displays dataframe in the app
- `st.table`: Displays table in the app
- `st.data_editor`: Displays a dataframe that can be edited from the app

The following code snippet is to explore the usage of Data widgets:

```
import streamlit as st
import pandas as pd

df = pd.DataFrame(
    [
        {"command": "st.selectbox", "rating": 4, "is_widget": True},
        {"command": "st.balloons", "rating": 5, "is_widget": False},
        {"command": "st.time_input", "rating": 3, "is_widget": True}
    ]
)

st.write("Table displayed using st.table")
st.table(df)
st.divider()
st.write("Table displayed using st.dataframe")
st.dataframe(df)
```

```

st.divider()

st.write("Table displayed and can be edited using
st.data_editor")

edited_df = st.data_editor(df, num_rows="dynamic")

favorite_command =
edited_df.loc[edited_df["rating"].idxmax()][["command"]]

st.markdown(f"Your favorite command is **{favorite_command}** ")

```

Some of the data widgets allow editing. The following Streamlit page is the complied first state of the data widget:

The screenshot shows a Streamlit application running at localhost:8501. It displays three tables:

- Table displayed using st.table:** A standard table showing the following data:

	command	rating	is_widget
0	st.selectbox	4	True
1	st.balloons	5	False
2	st.time_input	3	True

- Table displayed using st.dataframe:** A table with an edit mode, showing the same data as the first table but with checkboxes in the 'is_widget' column.
- Table displayed and can be edited using st.data_editor:** A table with a more detailed view, showing the data and checkboxes for each row.

At the bottom of the page, a message reads: "Your favorite command is st.time_input".

Figure 3.15: Streamlit app with data widgets (pre-editing)

The Streamlit page demonstrating how the values can be edited in the dataframe displayed by the data widgets is shown in [Figure 3.16](#). These changes can be captured and processed in the backend Streamlit application if required.

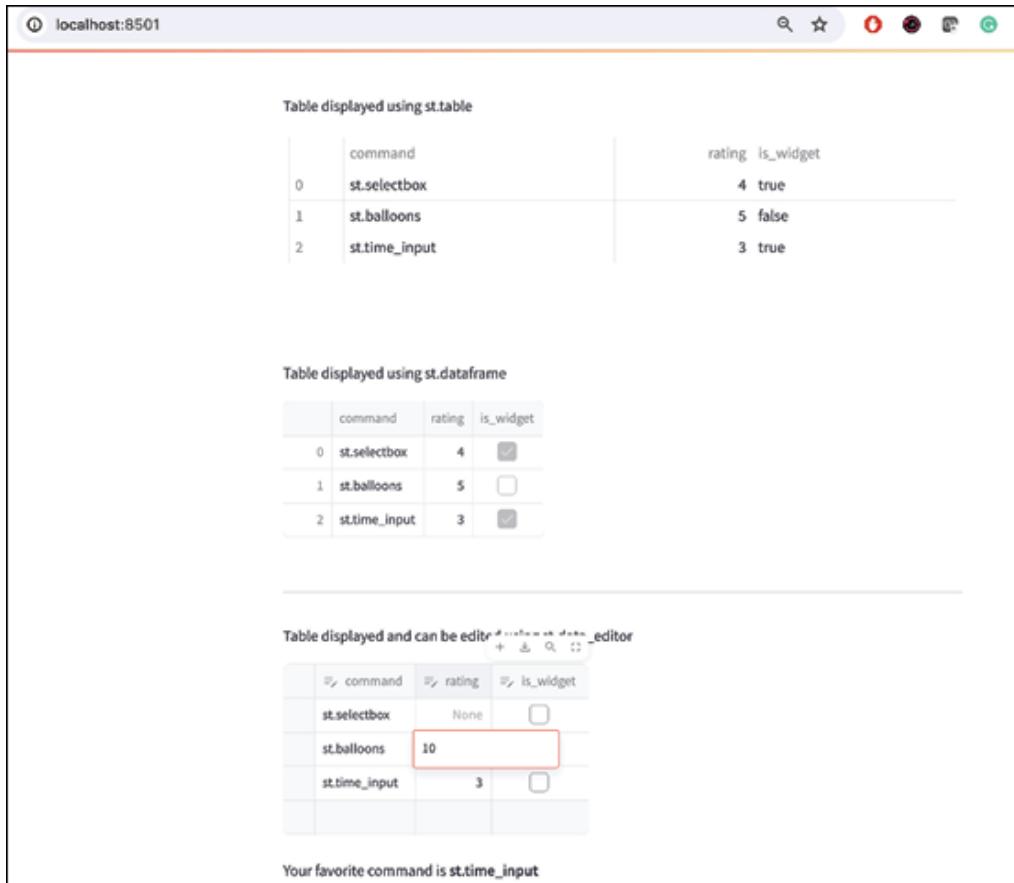


Figure 3.16: Streamlit app with data widgets (post-editing)

The following is the summary of the functionality offered by different Streamlit data widgets:

- `st.table` displays a static view of the dataframe.
- `st.dataframe` is a formatted view of the dataframe and can be downloaded.
- `st.data_editor` allows to edit the value and extract the updated value from the dataframe.

In Streamlit, the `st.column_config` class is a useful feature for setting up how data is shown and used. It is made for use with the `column_config` Option in `st.dataframe` and `st.data_editor`. This class methods lets you customize your data columns for different types of data, like text, numbers, lists, URLs, images, and more. Column configuration gives you the flexibility to display and interact with your data in the way you prefer.

The following are the class methods from `st.column_config`:

- `st.column_config.Column`
- `st.column_config.TextColumn`
- `st.column_config.NumberColumn`
- `st.column_config.CheckboxColumn`
- `st.column_config.SelectboxColumn`
- `st.column_config.DatetimeColumn`
- `st.column_config.DateColumn`
- `st.column_config.TimeColumn`
- `st.column_config.ListColumn`
- `st.column_config.LinkColumn`
- `st.column_config.ImageColumn`
- `st.column_config.LineChartColumn`
- `st.column_config.BarChartColumn`
- `st.column_config.ProgressColumn`

The following code snippet demonstrates different custom configuration options for the dataframe widget:

```
import streamlit as st
import pandas as pd

data_df = pd.DataFrame(
    {
        "widgets": ["st.selectbox", "st.number_input",
"st.text_area", "st.button"],

        "widgets_func": ["Let's you select a value ",
"Let's you input a number", "Let's you input text",
"Let's you click submission"],

        "widget_rank": [9, 950, 3, 1],
    }
)
```

```
        "favorite": [True, False, False, True],  
        "category": [  
            ". Data Exploration",  
            ". Data Visualization",  
            ". LLM",  
            ". Data Exploration",  
        ],  
    },  
)  
  
st.data_editor(  
    data_df,  
    column_config={  
        "widgets": st.column_config.Column(  
            "Streamlit Widgets",  
            help="Streamlit **widget** commands",  
            width="medium",  
            required=True,  
        ),  
        "widgets_func": st.column_config.TextColumn(  
            "What it does",  
            help="Streamlit **widget** function",  
            max_chars=50,  
            validate="^st\.[a-zA-Z_]+$"  
,  
    },
```

```
"widget_rank": st.column_config.NumberColumn(  
    "Global Widget rank",  
    help="Rank of **widget**",  
    min_value=0,  
    max_value=1000,  
    step=1,  
    format="%d",  
) ,  
"favorite": st.column_config.CheckboxColumn(  
    "Your favorite?",  
    help="Select your **favorite** widgets",  
    default=False,  
) ,  
"category": st.column_config.SelectboxColumn(  
    "Category of the widget",  
    help="Category of widget",  
    width="medium",  
    options=[  
        ". Data_Exploration",  
        ". Data_Visualization",  
        ". LLM",  
    ]  
)  
,  
hide_index=True,
```

)

The following is the compiled Streamlit page for the aforementioned code:

Streamlit Widgets	What it does	Global Widget rank	Your favorite?	Category of the widget
st.selectbox	Let's you select a value	9	<input checked="" type="checkbox"/>	Data Exploration
st.number_input	Let's you input a number	950	<input type="checkbox"/>	Data Visualization
st.text_area	Let's you input text	3	<input type="checkbox"/>	LLM
st.button	Let's you click submission	1	<input checked="" type="checkbox"/>	Data Exploration

Figure 3.17: Streamlit app using column_config class methods to define the column properties

Streamlit magic

As a testament to Streamlit's philosophy of being an intuitive, developer-friendly tool, streamlit can intuitively understand and execute Python commands to display data, create layouts, and add interactivity without requiring explicit calls to Streamlit's API functions. This is called Streamlit Magic.

With Streamlit Magic, you do not have to import `streamlit` library nor make any explicit calls to the APIs. Simply writing the Python code as if you were writing a script or a Jupyter Notebook will display outputs in the app. Python strings are automatically rendered as text in the Streamlit application, and the data frames, charts, or other visualizations are directly displayed without wrapping them in a Streamlit function.

The following code snippet is to demonstrate the Streamlit magic feature:

```
import pandas as pd

## This is a header

"And this is simple text"

df = pd.DataFrame({'A': [1, 2, 3], 'B': [4, 5, 6]})
```

df

The following is the compiled Streamlit page for the magic feature wherein Streamlit understands the commands and renders the widgets on the screen:

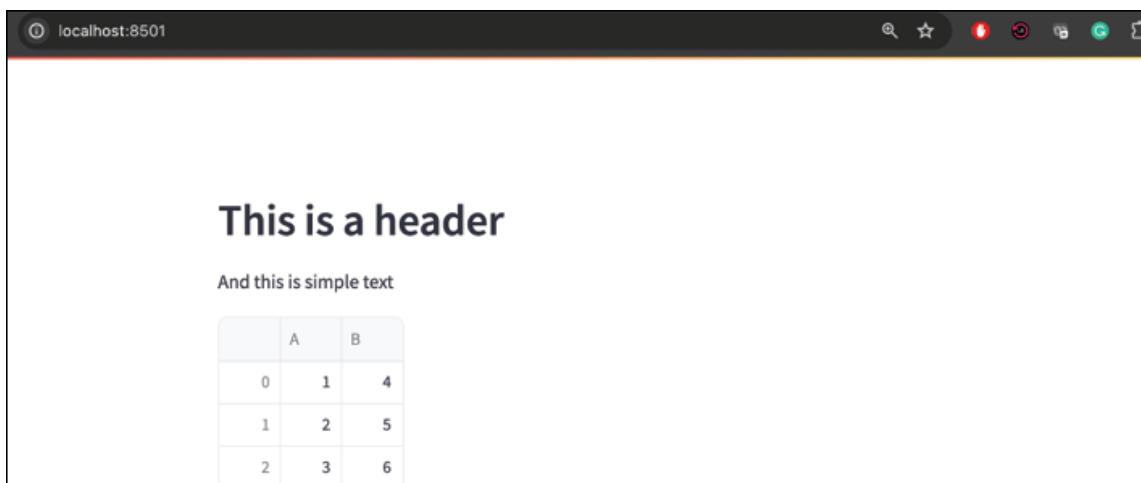


Figure 3.18: Streamlit app running with streamlit magic

While Streamlit magic is powerful and convenient for quick app development, it is important to know when to use it:

- **Ideal for prototyping:** It is perfect for creating prototypes or simple dashboards quickly.
- **Simple applications:** Best suited for applications where the layout and interactivity are straightforward.

When designing complex applications or applications that require more control on the data processed, it would be advisable to use the Streamlit API methods.

Hands-on: Adding widgets to enhance the app

Let us add widget controls to our Product Recommender application based on what we learned. The requirements are as follows:

- We want to have an input widget to take user inputs of the product bought

- We want to have a display widget to hold the recommended product based on the product bought by the user.

Note: Currently, we have not integrated machine learning or data processing capabilities in our application so it would be generating the recommendation randomly based on the fixed set of products provided.

Choosing the right widgets

With so many available options, picking the right widget for the application could be overwhelming. The following are principles that could be used to decide:

- **Clear visual hierarchy:** Establish a visual hierarchy in your application, ensuring that the most important elements stand out and guide the user's journey through the interface.
- **Minimalist design:** Adopt a minimalist design approach. Use whitespace effectively and avoid cluttering the UI with unnecessary elements.
- **Consistent navigation and familiar UI patterns:** Ensure that navigation is consistent throughout the application. Utilize familiar UI patterns and elements so users can quickly adapt to and navigate your application.

With the principles discussed in mind, let us start by adding widgets to our application based on our requirements. The following code snippet is to demonstrate how to build a cleaner design with Streamlit:

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

# Sample product data
products = ['Phone', 'Laptop', 'Headphones', 'Camera',
'Smartwatch', 'Tablet', 'Charger', 'Keyboard', 'Mouse',
'Speaker']
```

```
def generate_random_recommendations(selected_products,
products, num_recommendations=3):
    """Generate random product recommendations."""
    available_products = [p for p in products if p not
in selected_products]
    recommendations = random.sample(available_products,
min(num_recommendations, len(available_products)))
    return recommendations

def create_recommendation_df(recommendations):
    """Create a DataFrame with recommendations and
random likelihood percentages."""
    likelihoods = [random.randint(50, 99) for _ in
recommendations] # Random likelihood percentages
between 50% and 99%
    return pd.DataFrame({'Recommended Product':
recommendations, 'Likelihood of Purchase':
likelihoods})

# Streamlit application layout
st.title('Product Recommender')
st.write('Select one or more products to see
recommendations.')

# User selects multiple products
selected_products = st.multiselect('Choose products:',

products, default=None)

# Generate recommendations if any product is selected
if selected_products:
```

```

recommendations =
generate_random_recommendations(selected_products,
products)

# Create and display recommendations DataFrame

df_recommendations =
create_recommendation_df(recommendations)
st.write('Recommended Products:')
st.dataframe(df_recommendations,
            column_config={

                'Likelihood of Purchase':
st.column_config.NumberColumn(
                    'Likelihood of Purchase',
                    help="How likely is user to
buy this product **widget**",
                    min_value=0,
                    max_value=100,
                    step=1,
                    format="%d %%",
                )},
                hide_index=True,
            )
else:
    st.write('Please select at least one product to see
recommendations.')

```

The following figure is the Streamlit Product recommender app updated with the multi-selection widget, to select multiple

products purchased from the dropdown list:

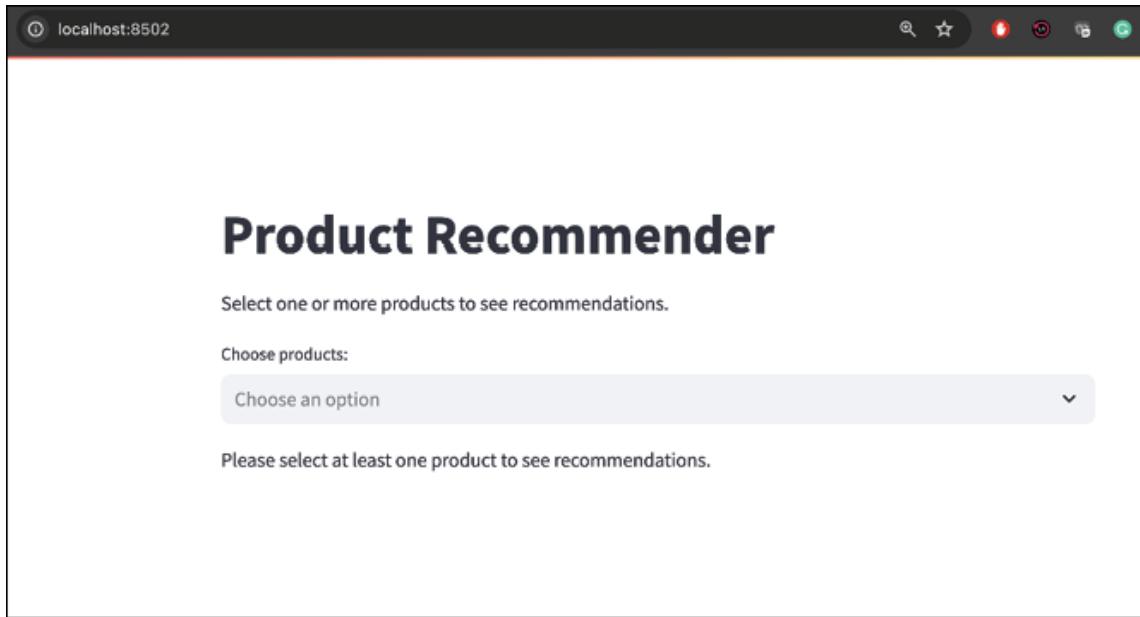


Figure 3.19: Streamlit Product recommender app with updated design

The following figure is the Streamlit page with the multi-select product drop down, with the user selecting Camera and keyboard as the products purchased:

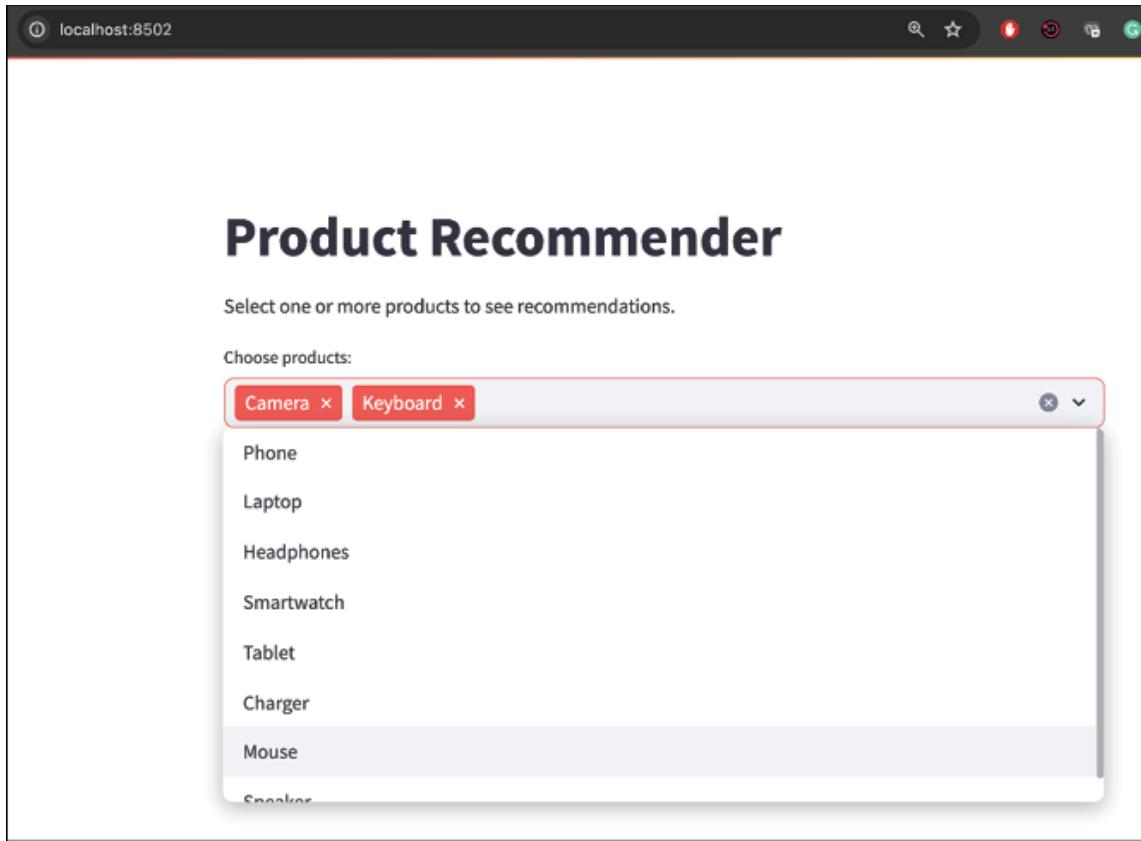


Figure 3.20: Streamlit Product recommender app multi-select dropdown

The following figure is the Streamlit Product recommender app displaying the output of recommended products based on the combination of input products selected from the multi-select dropdown:

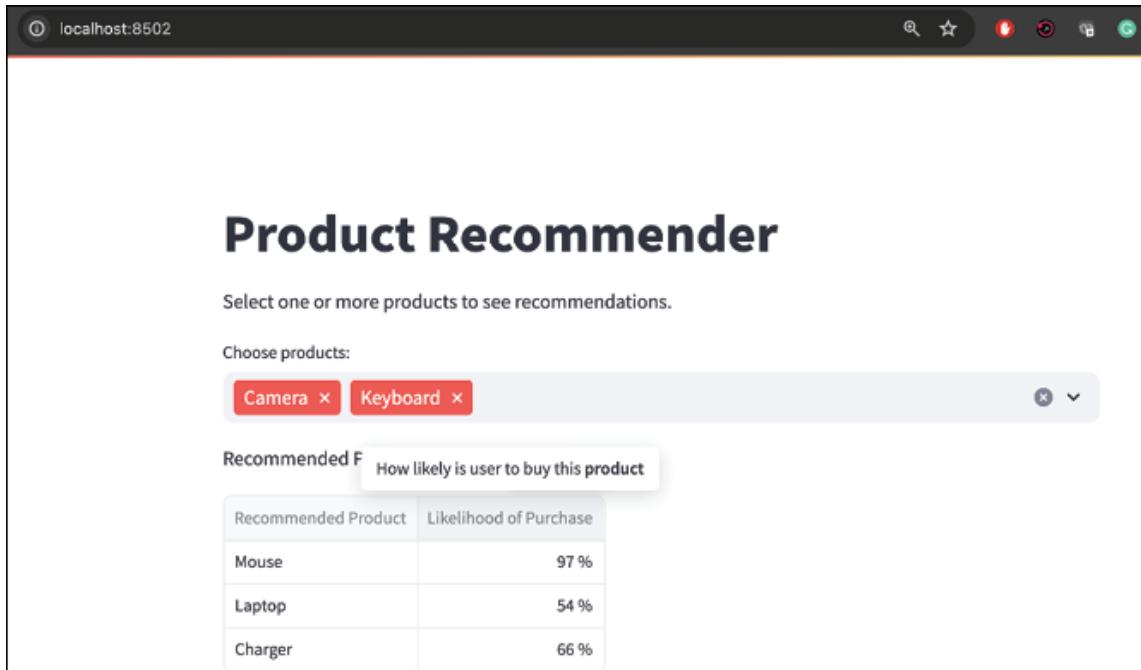


Figure 3.21: Streamlit Product recommender app displaying the output of recommended products

We used the multi-select box to add functionality to recommend products based on a combination of products bought. The results are displayed as a data frame formatted using the `column_config` class, making the dataframe more user-friendly.

Conclusion

This chapter covers basic core widgets provided by Streamlit. The strength of Streamlit lies in its simplicity and flexibility. With just a few lines of code, we can build intuitive and engaging interfaces. Streamlit widgets are designed to make your data applications more accessible and user-friendly. We will cover more advanced widgets and functionalities in upcoming chapters, but the best way to master these widgets is through practice and experimentation. Try to mix and match different widgets, explore their various configurations, and see how they can best serve your application's needs.

In the next chapter, we will explore how to build a cleaner and more intuitive application with streamlit. We will explore the layout and container options that Streamlit provides and learn

how to use them to design visually appealing applications. The next chapter will guide you through how to effectively organize your content and ensure your applications are not only functional but also aesthetically pleasing. We will look at practical examples, hands-on exercises, and best practices to help you master the art of creating professional-looking interfaces with ease.

Exercises

1. What does st.slider do in Streamlit?

- a. Creates a text input field
- b. Generates a dropdown menu
- c. Creates an interactive slider for numeric input
- d. Uploads files to the application

2. How can you display a button in Streamlit?

- a. st.button('Click Me')
- b. st.display_button('Click Me')
- c. st.show_button('Click Me')
- d. st.create_button('Click Me')

3. Which widget can be used for multi-line text input?

- a. st.text_input
- b. st.text_area
- c. st.text_box
- d. st.multi_text

4. Which Streamlit widget allows you to add a dropdown selection box?

- a. st.selectbox
- b. st.dropdown

- c. st.choice
- d. st.select

5. How do you display a checkbox in Streamlit?

- a. st.checkbox('Option')
- b. st.tickbox('Option')
- c. st.option('Option')
- d. st.select('Option')

Answers

- 1.** c
- 2.** a
- 3.** b
- 4.** a
- 5.** a

CHAPTER 4

Styling and Layouts in Streamlit

Introduction

In this chapter, we will learn how to build an aesthetically designed Streamlit application. Streamlit provides ways to enhance the design of your application by providing out-of-the-box features like layouts and containers. One does not need to be a web development expert to use them. Streamlit's simplicity and customization options allow you to build functional and attractive applications. By the end of this chapter, you will have the skills to create data applications that not only perform well but also provide a pleasant user experience.

Structure

The chapter discusses the following topics:

- Designing engaging and well-structured application
- User interface design with Streamlit
- State management
- Streamlit custom components
- Hands-on: Enhancing product recommender application

Objectives

In this chapter, we will look at the artistic side of building data applications with Streamlit, focusing on styling and layouts. While the functionality of an application is crucial, its visual appeal and **user interface (UI)** design play a significant role in enhancing **user experience (UX)**. A well-designed application is not just about aesthetics; it is about creating an environment where users can interact with your data intuitively and efficiently.

Designing engaging and well-structured application

Imagine walking into a well-organized room where everything is in its place versus a cluttered space where it is hard to find what you need. Similarly, a data application with a coherent theme, well-thought-out layout, and intuitive user experience makes it easier for users to navigate and understand the data being presented. It is about guiding the user's journey through the application in a way that feels natural and effortless. A consistent design reduces cognitive load, making the application not only visually appealing but also more user-friendly.

For example, consider an application that tracks fitness data. If the layout is cluttered and the theme inconsistent, users might find it difficult to locate their weekly progress report or compare their performance over time. However, a streamlined layout with well-organized sections and a consistent color scheme will make it much easier for users to find and understand their data, leading to a more satisfying and productive experience.

User interface design with Streamlit

Streamlit is not as complex as some advanced web development tools or libraries for creating fancy UI. However, it offers a range of tools that can help make your application intuitive and user-friendly. Even though you might not have the same level of detailed control as with more specialized UI libraries, Streamlit stands out for its simplicity and customization options.

Alongside the core APIs, theming options, layouts, and containers, Streamlit also has a vast library of components. These components are like building blocks that you can use to add different features and elements to your application without having to build everything from scratch. These tools help you to focus more on the data and less on the technicalities of UI development. Streamlit handles a lot of the heavy lifting for you, allowing you to create functional and attractive applications with less code and effort. Streamlit's styling, layout, and utility tools help to create applications that are not just functional but also aesthetically pleasing and user-centric. The key is to balance functionality with design, ensuring that your application is not only powerful in its data processing capabilities but also delightful to use.

Streamlit layouts and containers

Layouts and containers are fundamental concepts in UI design for creating an efficient, easy-to-use, and visually pleasing application. They help in structuring content in a way that makes sense to users, contributing significantly to a good user experience.

A layout determines where different parts of your app, like buttons, text, images, and charts, are placed. Layouts help in organizing information in a way that is easy to understand and navigate.

Containers help in keeping related items together, making the design organized and cleaner. They are used to hold and group together different elements or components. Let us illustrate and understand the impact of these design elements with an example.

Streamlit elements that help us achieve this are as follows:

- **st.sidebar**: Help to organize elements in the sidebar of the page.
- **st.columns**: Splits the page into columns.
- **st.tabs**: Splits the content into tabs inside the page.
- **st.expander**: Adds the content in the expandable section.

- `st.empty`: Creates an empty container, useful when a placeholder is required in the layout which can be later populated by dynamic actions.

The following code is the first iteration without the use of layouts and containers:

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

# Title
st.title('Streamlit App Without Layouts and
Containers')

# User Input
user_input = st.text_input("Enter some text")
number_input = st.number_input("Enter a number")

# Button
if st.button('Click Me'):
    st.write('Button clicked!')

# Plotting a simple line chart
data = pd.DataFrame(np.random.randn(20, 2), columns=
['A', 'B'])
st.line_chart(data)

# Displaying some text
```

```
st.write("Here's some text below the chart.")
```

The following is the Streamlit application without the use of layouts and containers:

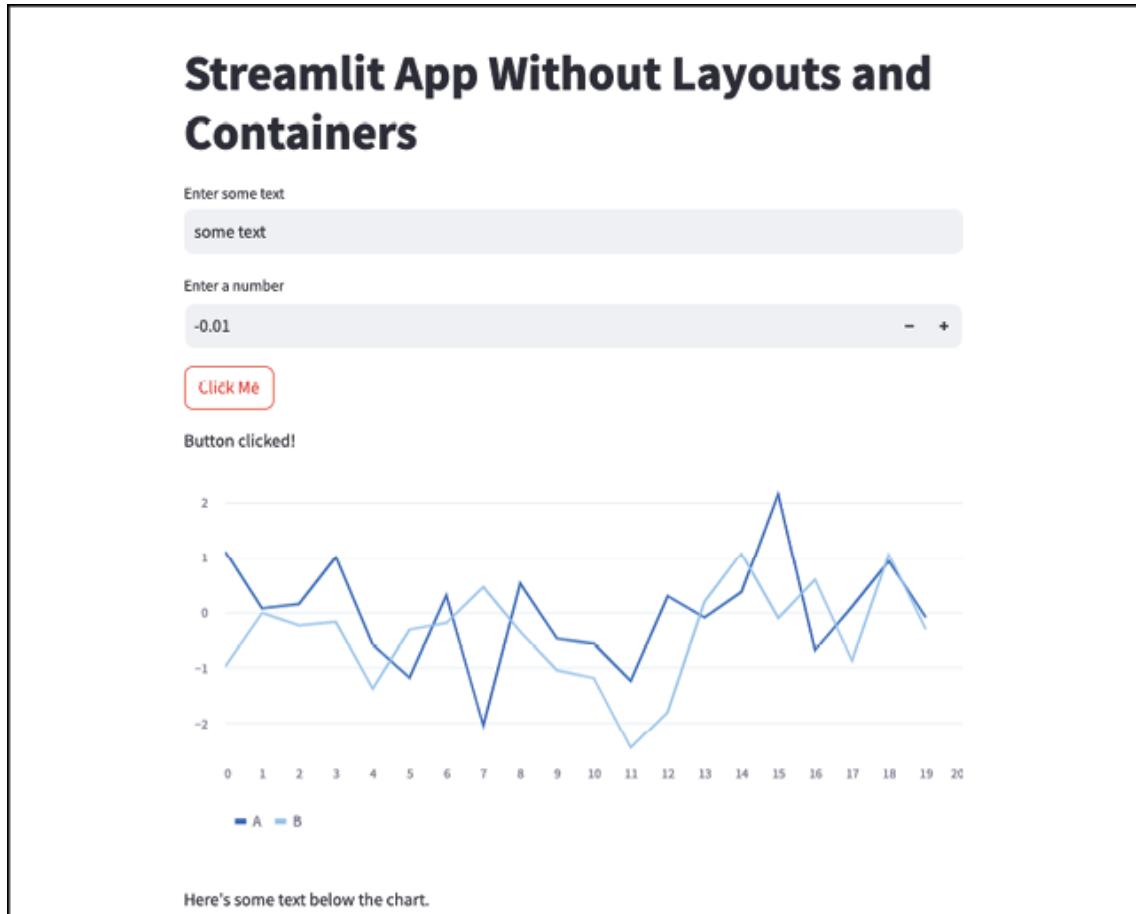


Figure 4.1: Streamlit app without layouts and containers

The following code is the next iteration. Add the layout and containers to organize our application elements as follows:

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

# Title
```

```
st.title('Streamlit App With Layouts and Containers')

# Using columns for input fields
col1, col2 = st.columns(2)
with col1:
    user_input = st.text_input("Enter some text")

with col2:
    number_input = st.number_input("Enter a number")

# Button in its own container
with st.container():
    if st.button('Click Me'):
        st.write('Button clicked!')

# Plotting a simple line chart in an expander
with st.expander("See Chart"):
    data = pd.DataFrame(np.random.randn(20, 2),
columns=['A', 'B'])
    st.line_chart(data)

# Footer text in a container
with st.container():
    st.write("Here's some organized text below the chart.")

The following is the Streamlit application with the use of layouts and containers:
```

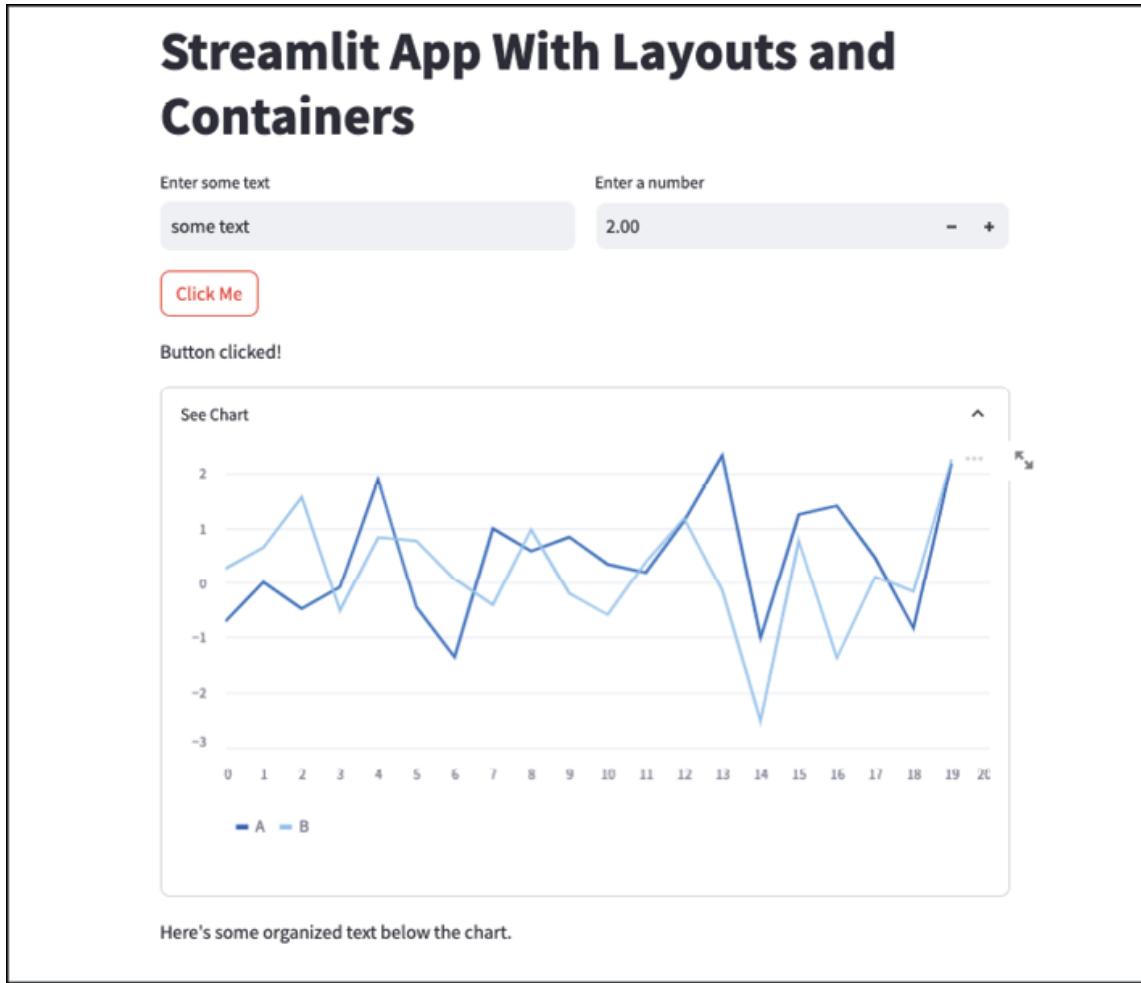


Figure 4.2: Streamlit app with layouts and containers

In summary, Streamlit containers and layouts can provide basic layouts and are very intuitive to use. They help in structuring content in a way that makes sense to users, contributing significantly to a good user experience.

Streamlit utility widgets

Apart from the prominent layout elements, a suite of utility functions plays a vital role in enhancing an application's functionality and user experience. While not strictly layout elements, these utilities are essential tools for creating more dynamic, informative, and user-friendly apps. In this subsection, we will explore three such utilities:

- **st.set_page_config**: Sets the page level settings, like page name, to be displayed on the browser tab.
- **st.echo**: Displays text as a code block.
- **st.help**: Prints help results for any object.

The following code builds a demo application to use the utility features of Streamlit:

```
import streamlit as st

# Setting the page config
st.set_page_config(page_title="My Awesome App",
                    page_icon=":tada:",
                    layout="wide")

# Displaying code and its output
with st.echo():
    # This code will be shown and executed
    st.write("This line will be printed and its code displayed.")

# Showing help information for a Streamlit function
st.help(st.sidebar)

The following is the Streamlit application with the utility features:
```

```

st.write("This line will be printed and its code displayed.")

This line will be printed and its code displayed.

st.sidebar DeltaGenerator DeltaGenerator(_root_container=1, _parent=DeltaGenerator())
root_container: BlockPath_pb2.BlockPath.ContainerValue or None
The root container for this DeltaGenerator. If None, this is a null
DeltaGenerator which doesn't print to the app at all (useful for
testing).

cursor: cursor.Cursor or None
This is either:
- None: if this is the running DeltaGenerator for a top-level
  container (MAIN or SIDEBAR)
- RunningCursor: if this is the running DeltaGenerator for a
  non-top-level container (created with dg.container())
- LockedCursor: if this is a locked DeltaGenerator returned by some
  other DeltaGenerator method. E.g., the dg returned in dg =
  st.text("foo").

parent: DeltaGenerator
To support the 'with dg' notation, DGs are arranged as a tree. Each DG
remembers its own parent, and the root of the tree is the main DG.

block_type: None or "vertical" or "horizontal" or "column" or "expandable"
If this is a block DG, we track its type to prevent nested columns/expanders

dg_property
id_property
add_rows method
altair_chart method
area_chart method
audio method
balloons method
bar_chart method

```

Figure 4.3: Streamlit app sample for streamlit utilities

Streamlit control flow widgets

Streamlit control flow widgets decide how your app responds when someone uses it. They are like traffic signals in a city; these widgets control what happens in your app when users interact with it. They play a crucial role in managing the execution flow of your Streamlit. By effectively utilizing these widgets, you can create more engaging and interactive experiences, guiding users through a seamless flow of operations and interactions within the app.

Streamlit control flow widgets include the following:

- **st.stop:** This is used to stop the execution of the Streamlit script.
- **st.form and st.form_submit_button:** These widgets allow grouping a set of input widgets into a form. This is helpful when complex input needs to be submitted collectively.

- `st.rerun`: This is used to rerun a Streamlit app programmatically.

The following code demonstrates the use of Streamlit control flow widgets:

```
import streamlit as st
import random

if st.checkbox("Stop the app"):
    st.write("App stopped.")
    st.stop()

st.write("This will not be displayed if the checkbox is checked.")

if st.button("Rerun the app"):
    st.rerun()

st.write(random.randint(1, 100)) # Display a random number

with st.form(key='my_form'):
    text_input = st.text_input(label='Enter some text')
    submit_button =
st.form_submit_button(label='Submit')

if submit_button:
    st.write(f'You entered: {text_input}')
```

The following is the build application with the control flow widgets:

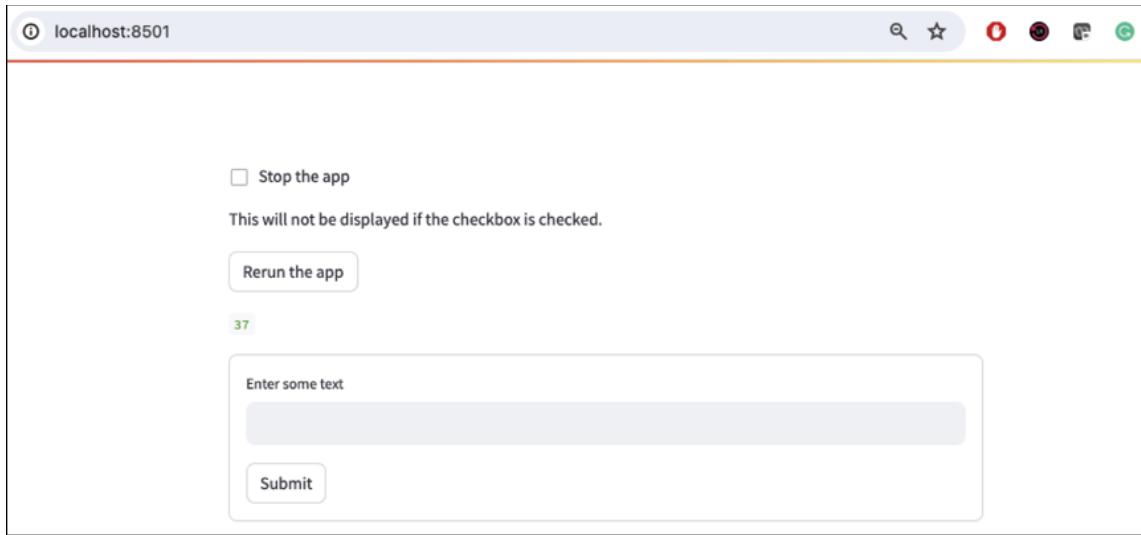


Figure 4.4: Streamlit app control flow widget sample

The following screen shows the control flow feature in action:

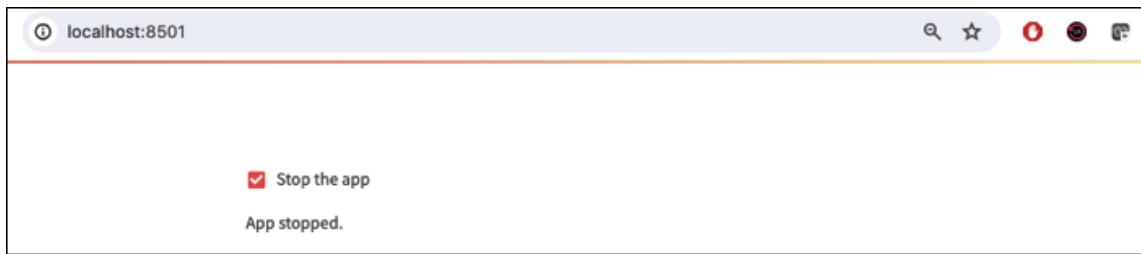


Figure 4.5: Streamlit app stopped running and rendering elements once the `st.stop` is run

The following screen shows the control flow when the rerun widget is called by the click of the button:

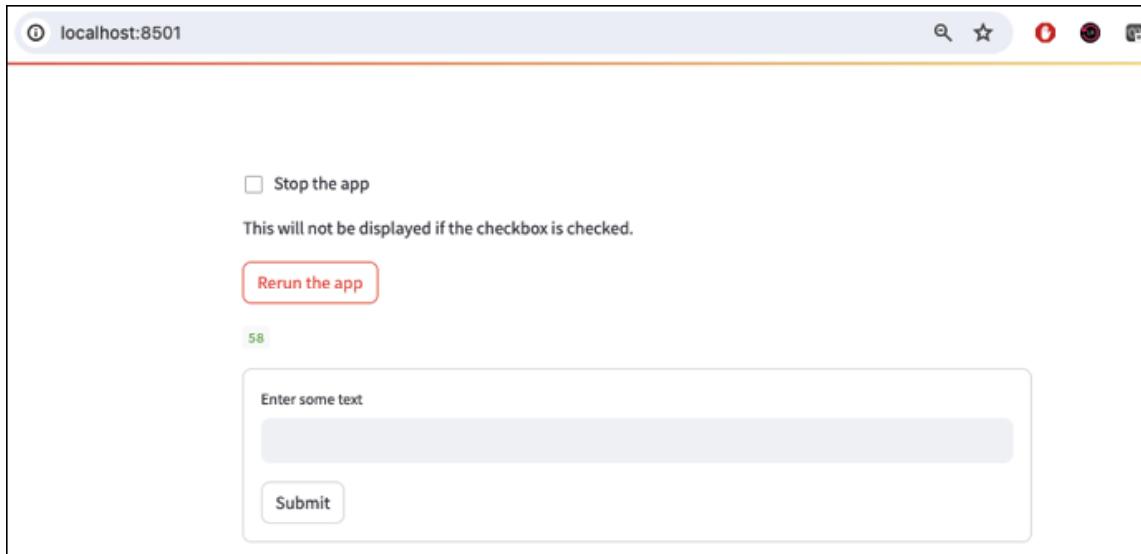


Figure 4.6: Streamlit app control flow widget sample- when the rerun app button is clicked, the counter increases

The following screen shows the action performed when the submit button is clicked:

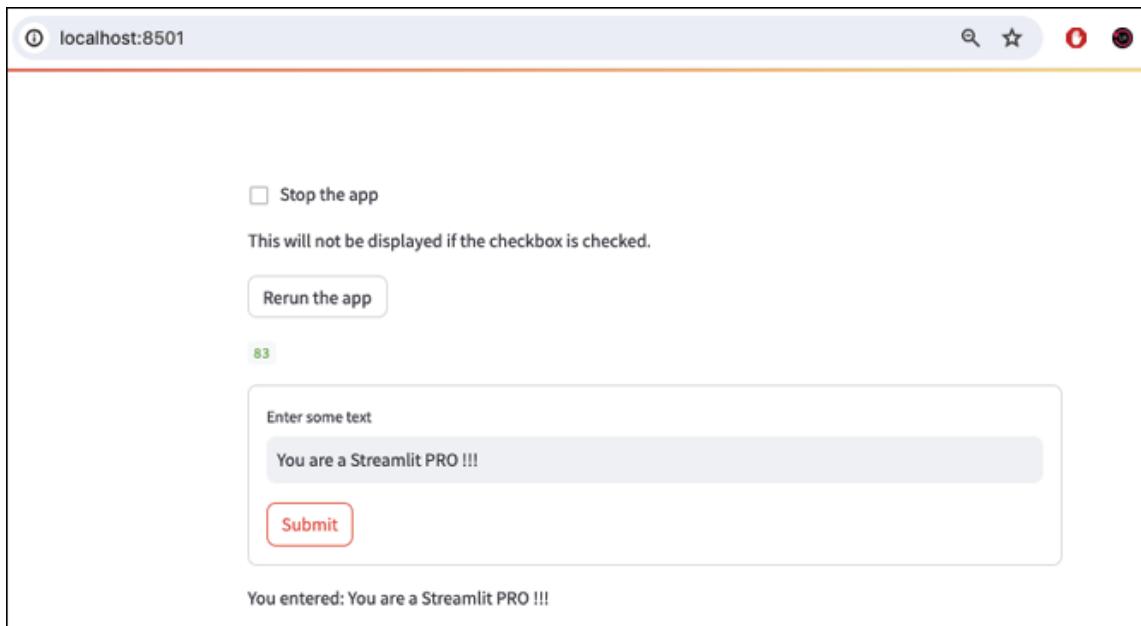


Figure 4.7: When the form submit button is clicked, the entered text is displayed

Streamlit theming

Streamlit provides flexible theming capabilities, allowing users to tailor the look and feel of their applications to match their

branding or personal preferences. While you can adjust the high-level theme settings using the settings hamburger menu in the top-right corner of the app, more granular control is achievable through the `.streamlit/config.toml` file. This configuration file enables you to set specific theme attributes to ensure a consistent visual experience.

The `config.toml` file allows you to define various aspects of your Streamlit app's theme. You can control the following six key options:

- **primaryColor**: Defines the main color or accent color used in the app (buttons, sliders, and other active elements).
- **backgroundColor**: Sets the color of the main background.
- **SecondaryBackgroundColor**: This is used for secondary areas, like the sidebar and background, for interactive widgets like charts.
- **textColor**: Determines the color of the text.
- **font**: Specifies the font used throughout the app.
- **base**: The base option, if defined, inherits the Streamlit preset theme light or dark; any additional configuration defined with the base will overwrite the respective configuration on top of the Streamlit preset theme.

The following is an example of how to set a custom theme:

```
[theme]
primaryColor="#333af6"
backgroundColor="#eea9a9"
secondaryBackgroundColor="#cad6e8"
textColor="#2030bb"

import streamlit as st

# App title
```

```

st.title("Streamlit Custom Theme Demo")

# Display text with different formatting
st.header("This is a header")
st.subheader("This is a subheader")
st.text("This is some text.")

st.write("This Streamlit app demonstrates the custom theme.")

# Sidebar with some widgets
st.sidebar.header("Sidebar")
st.sidebar.write("This is the sidebar.")
st.sidebar.checkbox("Check me out")
st.sidebar.radio("Choose an option", ['Option 1', 'Option 2', 'Option 3'])
st.sidebar.selectbox("Select a number", np.arange(1, 11))

```

The following is the Streamlit code for the custom theme set by the code:

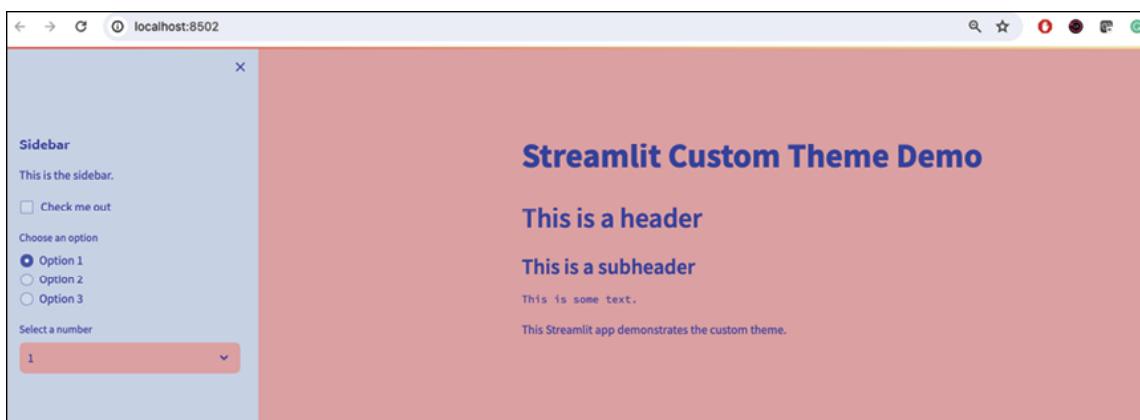


Figure 4.8: Streamlit app with custom theme set in config.toml

The following is a screenshot of the custom theme option available when you click the hamburger menu on the top right corner:

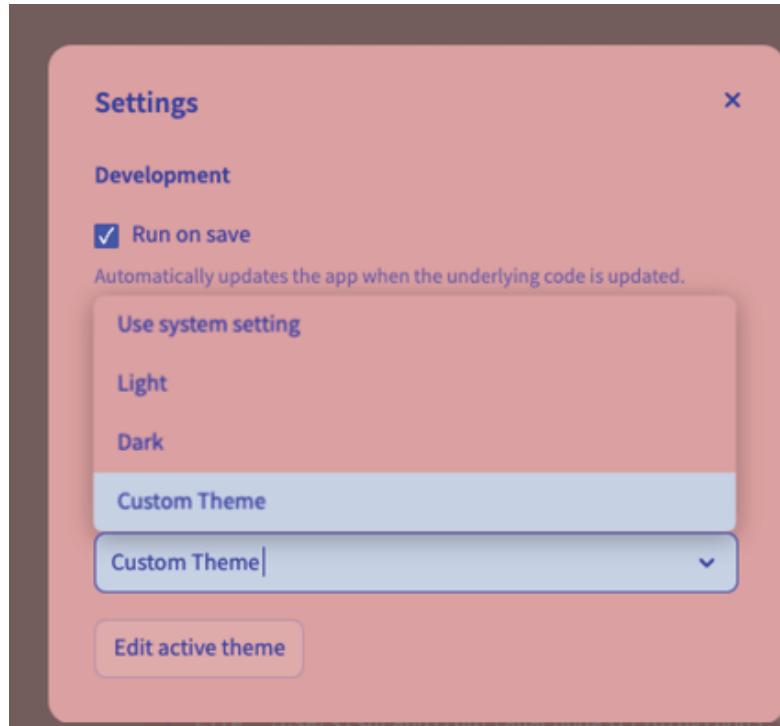


Figure 4.9: Streamlit app update theme using hamburger menu

The following is the screenshot for editing the theme for the Streamlit page using the option menu:

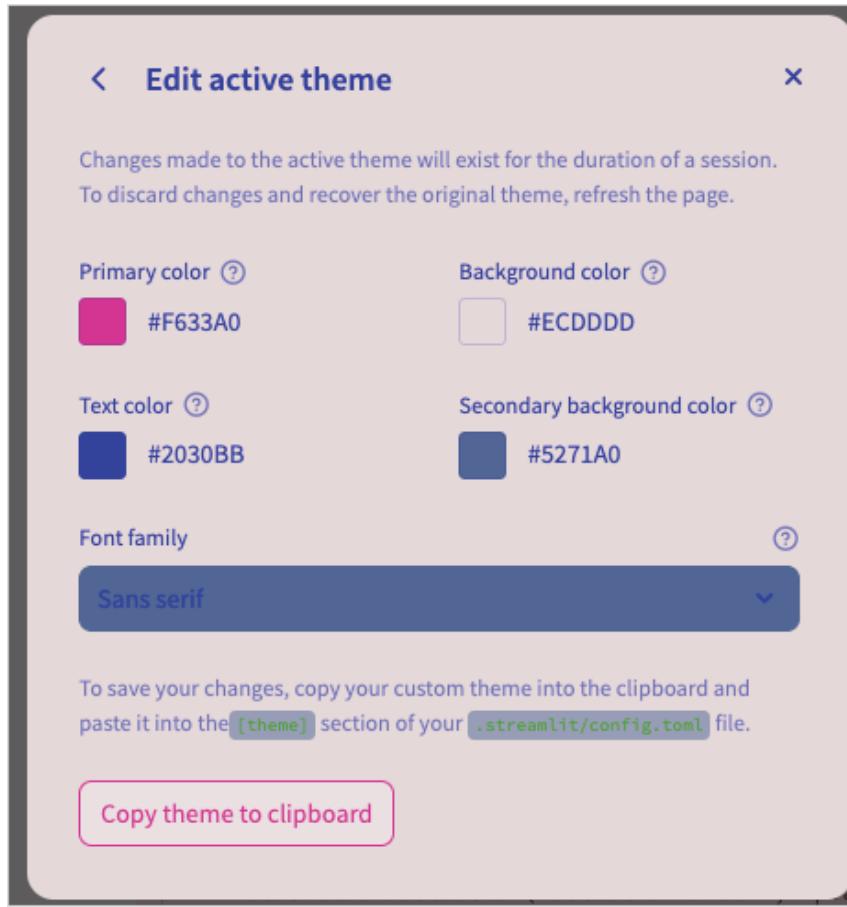


Figure 4.10: Editing custom theme options allows to set the theme from the front end and copy the configuration

Change the color or font options, then copy the theme on the clipboard and place it as `.streamlit/config.toml`.

Rendering HTML and CSS

Streamlit also allows you to refine your theme further by integrating custom CSS in your application. This allows you to customize the appearance of your Streamlit app beyond the built-in theming capabilities.

This can be achieved by using `st.markdown` with the flag `unsafe_allow_html`. This allows you to render raw HTML, including embedded styles and scripts, within a Streamlit application. While this feature offers greater flexibility and customization options, it also has certain trade-offs. The advantages and disadvantages are listed in [Table 4.1](#).

Advantages	Disadvantages
Customization: It allows for advanced customization of the app's UI, including custom styles, external content, or HTML features not natively supported by Streamlit.	Security risks: As the flag name suggests, allowing raw HTML can pose security risks, particularly from JavaScript injection attacks (XSS). This is a concern if the HTML content is dynamically generated from user input
Control over layout: With HTML, you gain more control over the layout and positioning of elements, which can be crucial for achieving specific design goals.	Maintainability: Custom HTML and CSS might make the app harder to maintain if the code becomes too complex or if it relies on specific behaviors that could change in future Streamlit updates.
Rich content integration: Enables the integration of rich content like frames, custom graphics, or interactive elements that can enhance the user experience.	Performance impact: Overuse of raw HTML and custom styles might impact the app's performance, especially if it leads to a large DOM or complex CSS rules.
Quick fixes and workarounds: Useful for quick fixes or temporary solutions where native Streamlit components may fall short.	Cross-browser compatibility: Custom HTML and CSS might not be consistent across all browsers, leading to compatibility issues.

Table 4.1: Comparing the Pros and Cons of using external HTML in streamlit applications

Thus, when opting to render HTML or CSS, always weigh the options and decide, considering the risks the implementation might pose. The following is a code snippet to demonstrate the usage of external CSS in a Streamlit application:

```
import streamlit as st

# Inject custom CSS with st.markdown
st.markdown("""
<style>
.main-title {
    color: blue;
    font-family: Arial, sans-serif;
}

```

```
.custom-text {  
    color: green;  
    font-size: 20px;  
}  
  
/* Style the Streamlit elements */  
.stTextInput > label, .stButton > button {  
    color: red;  
}  
</style>  
"""", unsafe_allow_html=True)  
  
# Use the custom CSS classes  
st.markdown('<h1 class="main-title">Custom CSS  
Title</h1>', unsafe_allow_html=True)  
st.markdown('<p class="custom-text">This text has  
custom CSS applied to it.</p>', unsafe_allow_html=True)  
  
# Standard Streamlit elements  
st.text_input("This input label is styled with custom  
CSS")  
st.button("This button's text is styled with custom  
CSS")  
The following is the build Streamlit application with custom CSS  
components:
```

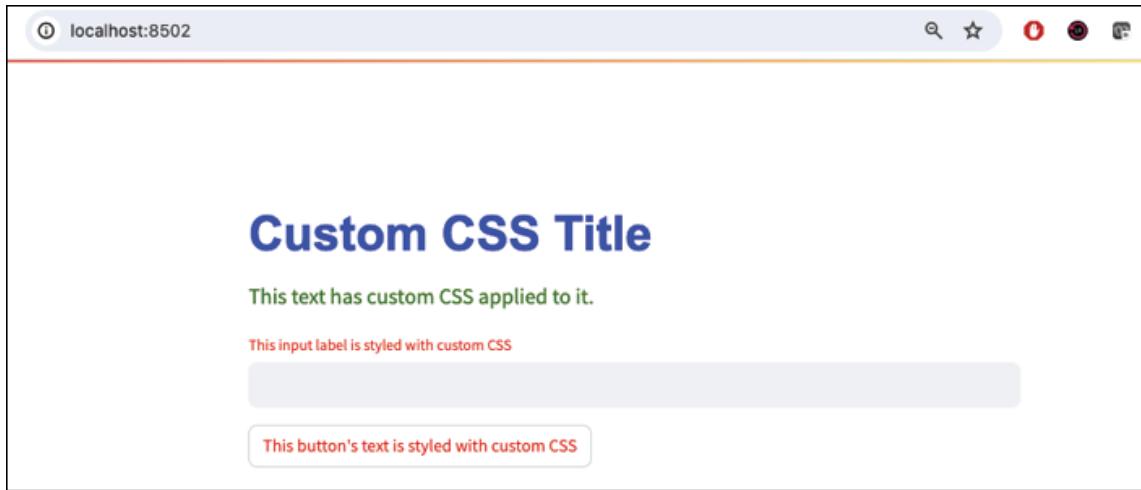


Figure 4.11: Streamlit app using custom CSS settings

State management

In the context of web development, state cookies are a common method used for state management. Cookies are small pieces of data stored on the user's computer. They are like memory notes that the app can read and write to remember things about the user's session. For example, when you log into a website, it keeps you logged in even when you close the browser, which is often done using cookies. While Streamlit does not use cookies in the traditional sense as in regular web development, it provides similar functionality through its session state management.

The following code snippet is a simple example of state management in Streamlit, which is creating a counter that increments with each button press:

```
import streamlit as st

# Initialize state
if 'count' not in st.session_state:
    st.session_state.count = 0

# Increment function
```

```

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

# Button to increment state
st.button('Increment', on_click=increment_counter)

# Display current state
st.write('Count:', st.session_state.count)

```

The following is the build Streamlit application demonstrating state tracking:



Figure 4.12: Streamlit app counter stored in the `session_state` object

In this example, `st.session_state` acts like memory, remembering the count of button presses across reruns of the app. This is similar to how cookies work in traditional web apps, keeping track of user interactions and states.

Streamlit's session state is a special memory space where your app can remember things. For example, if a user types something or clicks a button, Session State can be used to store these values. It can store any kind of Python object, like integers, floating-point numbers, complex numbers and Booleans, data frames, and even *lambdas* returned by functions. But in some cases, especially when you are running your app in different environments, like on different computers or servers, you need to be sure that everything you put in this memory can be safely stored and recovered. That is where the Serializable Session State comes in.

Streamlit has a setting (`runner.enforceSerializableSessionState`) that, when turned on, makes sure you only store things in Session State that can be safely pickled and unpickled. It is like ensuring everything you store in memory can be safely put in the box and removed without any problems.

To use this setting, you can add it to a configuration file for your Streamlit project or use a command when running your app. This helps ensure your app works smoothly, especially when running in different places or if you plan to change how you run your app.

The following is the snippet for configuring the Serializable Session state in `config.toml`:

```
# .streamlit/config.toml  
[runner]  
enforceSerializableSessionState = true
```

There are certain noteworthy limitations when working with a Streamlit session state. The following are the key points to remember when using session state:

- In a Streamlit form, only the submit button (`st.form_submit_button`) can have a callback function. Other widgets inside the form cannot use callbacks.

```
import streamlit as st  
  
with st.form("my_form"):  
    text_input = st.text_input("Enter some text")  
    submit_button = st.form_submit_button("Submit",  
                                         on_click=lambda: st.write("Form Submitted"))  
  
    # Note: The text_input widget does not have a  
    #        callback.  
  
    • Event handling options like on_change and on_click are only  
      supported by input widgets, such as st.text_input, st.button:
```

```
import streamlit as st

# on_change event in a text input widget
def handle_text_change():
    st.write("Text has changed")

st.text_input("Change the text",
on_change=handle_text_change)
```

```
# on_click event in a button widget
st.button("Click me", on_click=lambda:
st.write("Button clicked"))
```

- Once you create a widget, you cannot modify its value through the Session State API. Trying to do so will result in a **StreamlitAPIException**.

```
import streamlit as st
```

```
# Creating a text input widget
text = st.text_input(key="text_input", label =
"Enter text")
```

```
# Incorrect way: Attempting to modify the widget
value after creation
```

```
st.session_state.text_input = "New Value" # This
will raise StreamlitAPIException
```

The following is the sample Streamlit application when the session state is implemented incorrectly:

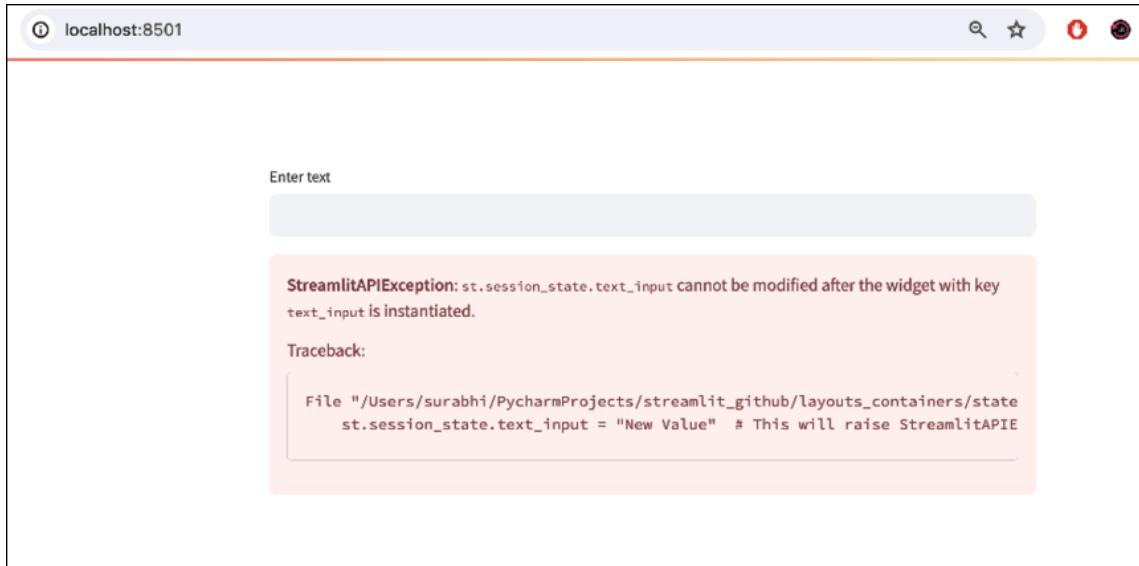


Figure 4.13: Streamlit app throws an Exception if the session state is used incorrectly

- It is not recommended to set a widget's state using both the Session State API and the widget's value parameter simultaneously. Doing so will trigger a warning on the first run.

```
import streamlit as st
```

```
# Incorrect way: Setting state via Session State
# API and value parameter
st.session_state.text_input = "New Value"
```

```
# Creating a text input widget
```

```
text = st.text_input(key="text_input", label =
"Enter text" ,
                     value =
st.session_state.text_input)
```

The following is the Streamlit page with a warning when the preceding code snippet is run:

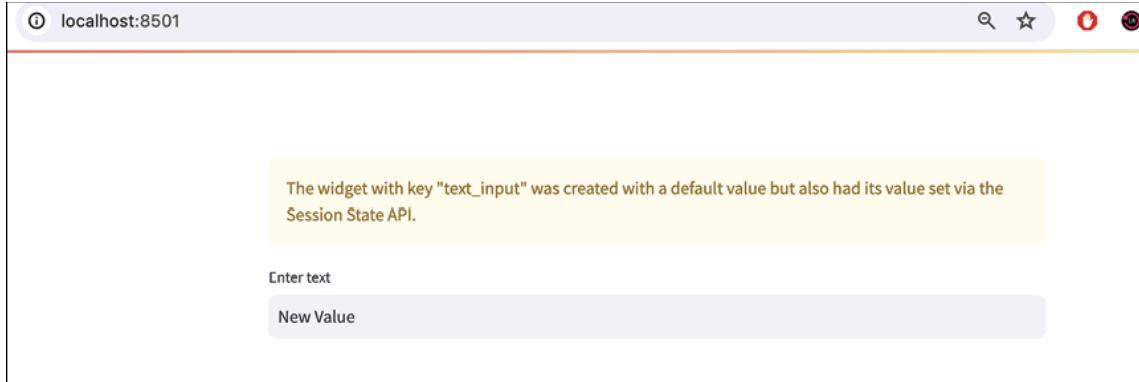


Figure 4.14: Streamlit app throws a warning if the session state is used incorrectly

- You cannot set the state of button-like widgets (`st.button`, `st.download_button`, and `st.file_uploader`) via the Session State API. These widgets have a default state of False and a temporary True state that is valid only for the current run.

```
import streamlit as st
```

```
# Incorrect way: Attempting to set the state of a
button
```

```
st.session_state.my_button = True # Not allowed
```

```
# Using a button
```

```
if st.button("Click me", key = 'my_button'):
    st.write("Button was clicked!")
```

The following is the Streamlit page with an exception when the value is initialized in the above way:

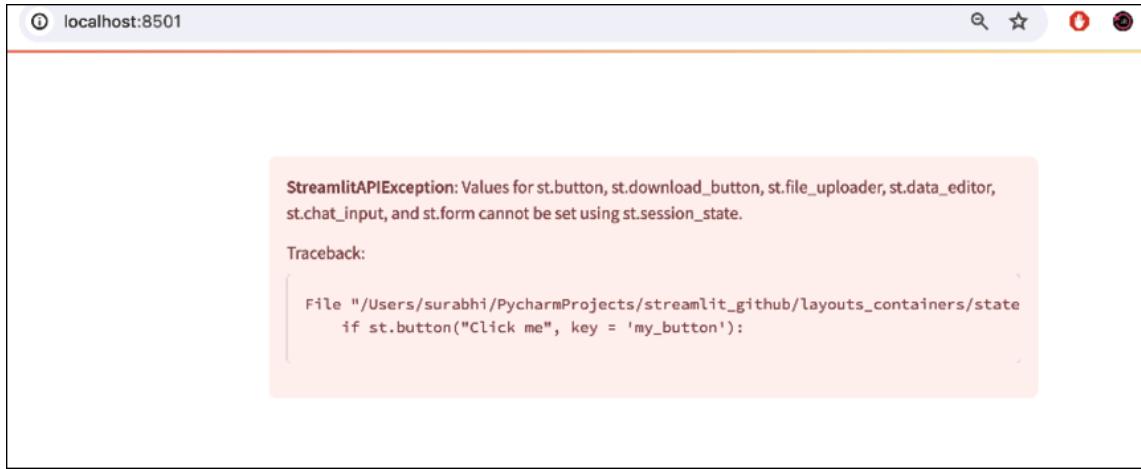


Figure 4.15: Streamlit app throws an Exception if the session state is used incorrectly

Streamlit custom components

An exciting aspect of Streamlit components is the vibrant community around it. Many developers share their custom components, which can be reused in other projects. These community components can offer novel functionalities, such as advanced data visualization, integration with machine learning tools, and so on. Custom components can be created using Streamlit's Component API and can be written in languages like JavaScript.

Understanding and utilizing Streamlit components, both built-in and custom, is crucial for expanding the capabilities of your applications. By integrating third-party libraries, you can bring an even greater level of sophistication and interactivity to your Streamlit apps.

To use a component in your app, follow these steps:

1. Select a component from the component gallery, as shown in the following figure:

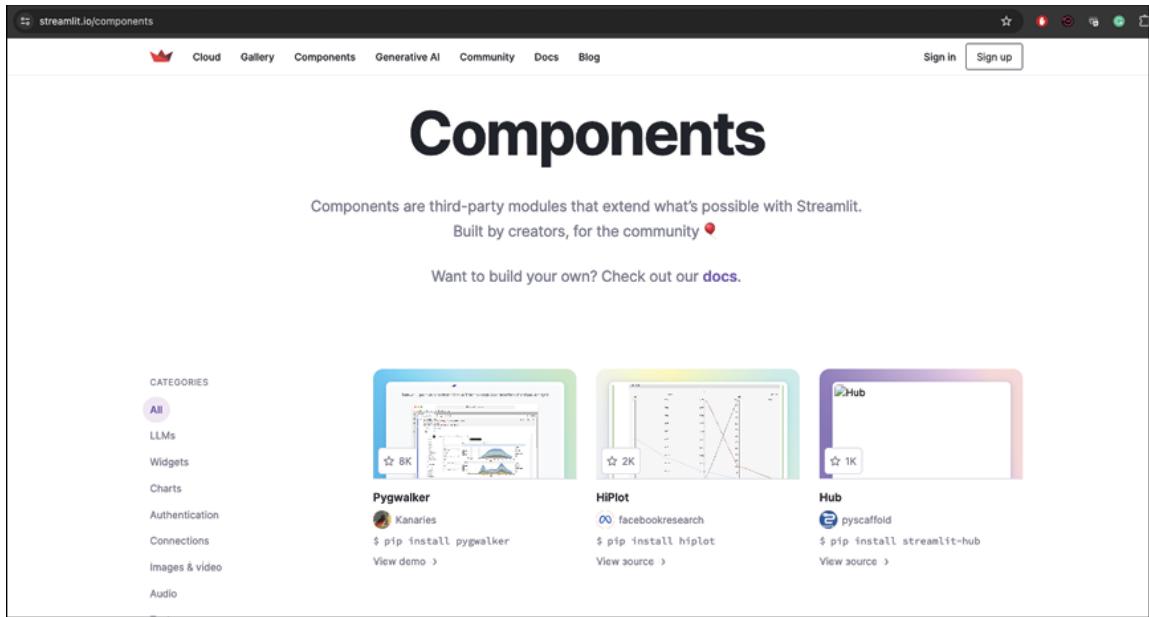


Figure 4.16: Streamlit official component library

2. Install component:

```
pip install streamlit-wordcloud
```

3. Import the component in your app:

```
import streamlit_wordcloud as wordcloud
```

4. Start using the component:

```
words = [  
    dict(text="Apple", value=16000),  
    dict(text="Banana", value=8500),  
    dict(text="Cherry", value=6700),  
    dict(text="Grapes", value=13400),  
    dict(text="Mango", value=8300)]  
  
wordcloud.visualize(words, per_word_coloring=True)
```

Hands-on enhancing product recommender application

Let us add the elements that we learned in this chapter to our product Recommender Application. The following is a list of enhancements that we would be adding to the data app.

- Add a sidebar, page title, and icon.
- Add different tabs for better segregation of the content (Introduction, Product Recommendation, and Top Product).
- Organize the content in a more readable and intuitive manner using containers and layouts.

With this in mind, let us make changes in our folder structure:

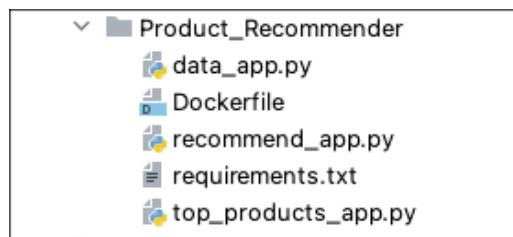


Figure 4.17: Streamlit project updated project structure

The `data_app.py` file is an entry page for our application, where we set the config for the entire application:

```
import streamlit as st
from recommend_app import *
from top_products_app import *

# Set page configuration
st.set_page_config(page_title="Product Recommender",
page_icon="..",
layout="wide",initial_sidebar_state="expanded")

def intro():
    st.title('Product Recommender')
    st.markdown("----")
    st.markdown(
```

```
"""
> What does this app do ?
    * This application helps to recommend products
based on the products selected by the user.

    * The application also shows the top trending
products for each category.

> How to use it ?
    * Select the products from the dropdown, you
can select multiple products too

    * The result will be displayed with the top
recommended products with the likelihood of purchase
""")\n\n# Sidebar for user input\nwith st.sidebar:\n    st.write("This is a demo app for Hands-on exercise\nfor building Product Recommender app")\n\nif __name__ == "__main__":\n    # main page\n    tab1, tab2, tab3 = st.tabs(["Intro", "Product\nRecommendation", "Top Products"])\n\n    with tab1:\n        intro()\n\n    with tab2:\n        recommend_main()\n\n    with tab3:\n        top_products_main()
```

The `recommend_app.py` file is where most of the recommendation logic resides:

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

# Sample product data
products = ['Phone', 'Laptop', 'Headphones', 'Camera',
'Smartwatch', 'Tablet', 'Charger', 'Keyboard', 'Mouse',
'Speaker']

# Product data with categories
product_categories = {
    'Educational': ['Book', 'Online Course',
'Educational Toy', 'Science Kit', 'Math Workbook',
'Language Learning Software'],
    'Travel': ['Luggage', 'Travel Pillow', 'World Map',
'Travel Guidebook', 'Portable Charger', 'Travel Backpack'],
    'Books': ['Fiction', 'Non-Fiction', 'Biography',
'Mystery', 'Science Fiction', 'Historical Fiction',
'Self-Help'],
    'Gadgets': ['Phone', 'Headset', 'Smartwatch',
'Laptop', 'Tablet', 'E-reader', 'Portable Speaker',
'Fitness Tracker'],
    'Home & Kitchen': ['Coffee Maker', 'Blender',
'Smart Home Device', 'Cookware Set', 'Decorative Vase',
'Bedding Set'],
    'Fashion': ['T-shirt', 'Jeans', 'Dress', 'Watch',
'Handbag', 'Sneakers', 'Sunglasses', 'Jacket'],
    'Health & Wellness': ['Yoga Mat', 'Fitness Band',
'Protein Powder', 'Skincare Product', 'Haircare Kit'],
```

```
'Vitamins', 'Aromatherapy Diffuser']  
}  
  
def generate_random_recommendations(selected_products,  
products, num_recommendations=3):  
    """Generate random product recommendations."""  
    available_products = [p for p in products if p not  
in selected_products]  
    recommendations = random.sample(available_products,  
min(num_recommendations, len(available_products)))  
    return recommendations  
  
def create_recommendation_df(recommendations):  
    """Create a DataFrame with recommendations and  
random likelihood percentages."""  
    likelihoods = [random.randint(50, 99) for _ in  
recommendations] # Random likelihood percentages  
between 50% and 99%  
    return pd.DataFrame({'Recommended Product':  
recommendations, 'Likelihood of Purchase':  
likelihoods})  
def recommend_main():  
  
    with st.container():  
        st.title('Product Recommender')  
  
        # Form for category and product selection  
        with st.expander(label="", expanded = True):  
            col1, col2 = st.columns(2)
```

```
        with col1 :  
            selected_category =  
st.selectbox('Choose category:', options=[''] +  
list(product_categories.keys()))  
  
        with col2:  
            # Enable product selection only if a  
category is selected  
            if selected_category:  
                selected_products =  
st.multiselect('Choose products:',  
product_categories[selected_category])  
            else:  
                selected_products =  
st.multiselect('Choose products:', [])  
  
            # Submit button for the form  
            submit_button = st.button(label='Get  
Recommendations')  
  
            if submit_button and selected_category and  
selected_products:  
                recommendations =  
generate_random_recommendations(selected_products,  
product_categories[selected_category])  
  
                df_recommendations =  
create_recommendation_df(recommendations)  
                st.write('Recommended Products:')  
  
                st.dataframe(df_recommendations,
```

```

        column_config={

            'Likelihood of Purchase': st.column_config.NumberColumn(
                'Likelihood of
Purchase',
                help="How likely is
user to buy this **product**",
                min_value=0,
                max_value=100,
                step=1,
                format="%d %%",
            )},
            hide_index=True,
        )

    elif submit_button:
        st.write('Please select a category and at
least one product to see recommendations.')
The top_products_app.py file is added for future use like to have a
visualization for the top products being requested:
import streamlit as st

def top_products_main():
    st.title('Work in progress')

```

The following is the build Streamlit updated home page for our Product Recommender application:

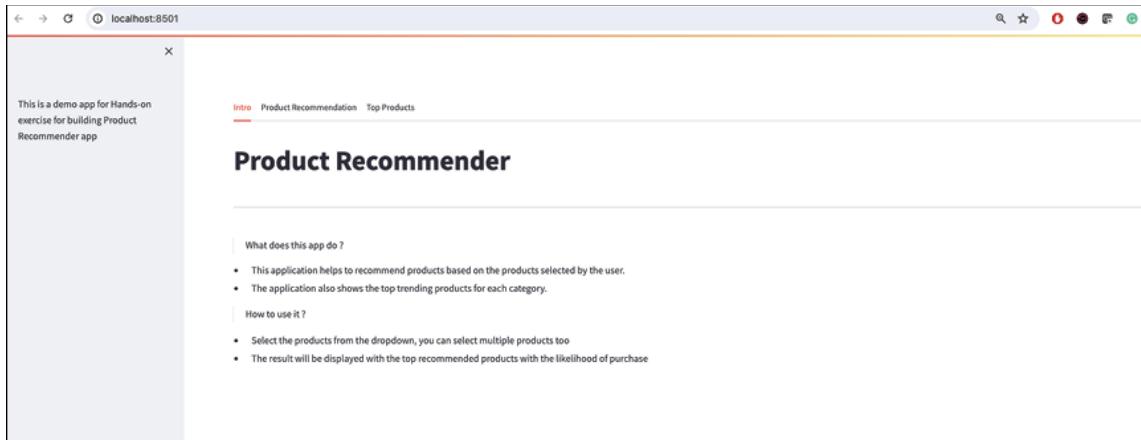


Figure 4.18: Streamlit app home page

The following is the second page and the next tab in the build Streamlit Product Recommender application:

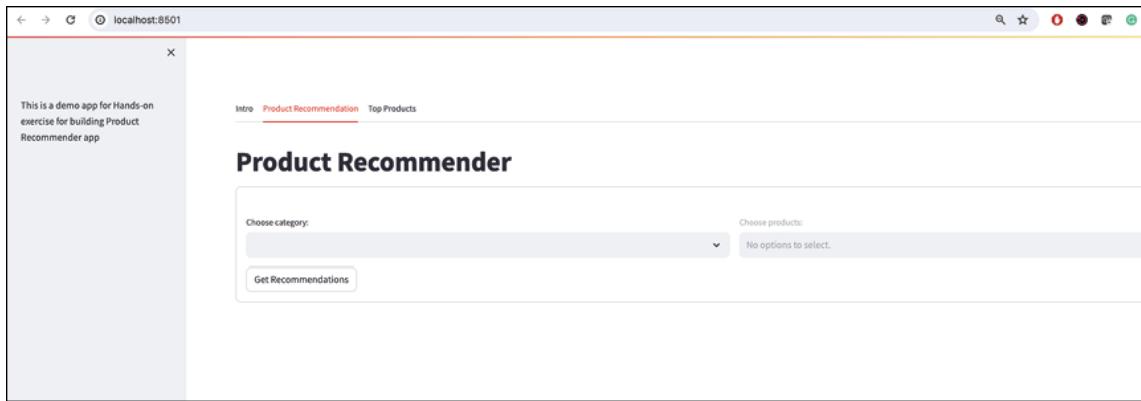


Figure 4.19: Streamlit app product recommendation page

As designed, the products are now divided into different categories. The user is first supposed to select the category, and then, based on the selection, the products are populated, as shown in the following figure:

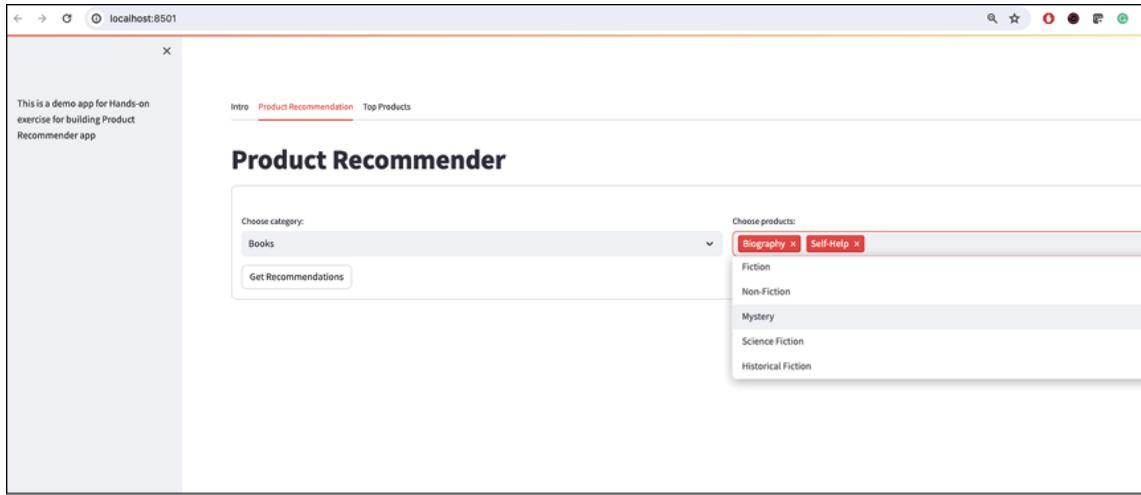


Figure 4.20: Streamlit app product recommendation selection flow

Once the selection is made for the product category and products, the **Get Recommendation** button is clicked. The app validates to check whether all the values are filled up and then generates the recommended products based on the functional logic, which would have been randomly recommending the products from the specific category.

The following is the screen with the returned results from the Product Recommender application:

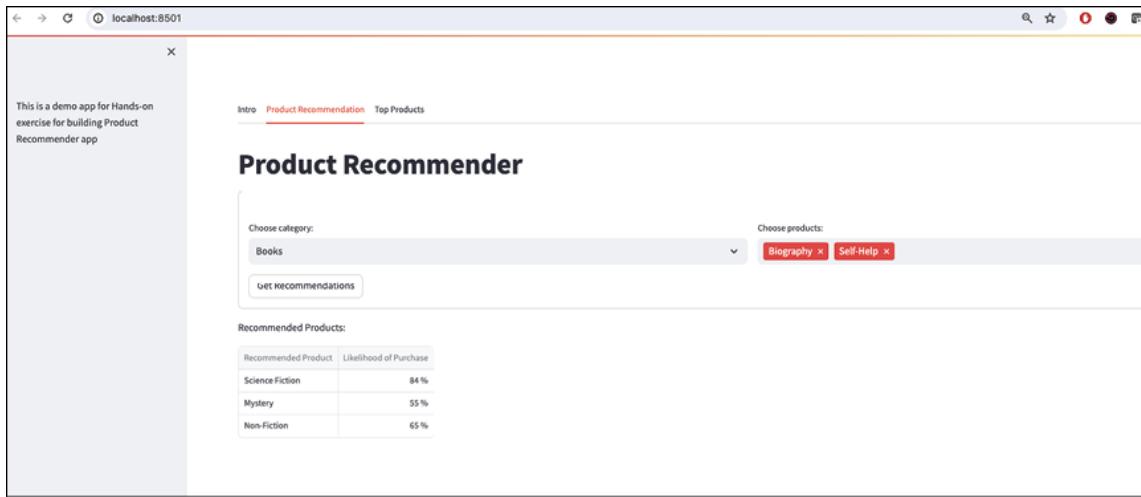


Figure 4.21: Streamlit app product recommendation result generation

The following screen is the last tab, top products, which is currently a placeholder. We will enhance this page in the next

chapter when we discuss the visualization options offered by Streamlit:

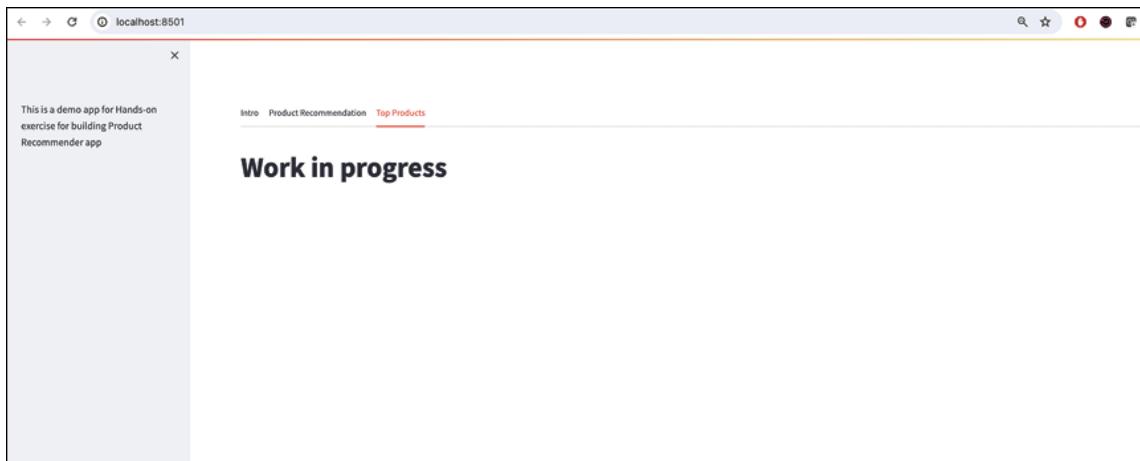


Figure 4.22: Streamlit app product recommendation Top products page

Conclusion

In this chapter, we have covered the diverse array of features and components that Streamlit offers to elevate the look, feel, and functionality of data applications. The contents covered range from the aesthetics of theming to the practicality of layouts and containers. Streamlit provides tools to design applications that are visually appealing as well as highly functional.

A key aspect we explored is the concept of state management in Streamlit. Like giving your application a memory, state management allows it to remember user interactions and data across sessions. Streamlit also provides the possibility of integrating third-party components to add advanced functionalities in a limited time. Also, the support for custom CSS and HTML can add a personal touch and unique flair to your data application. While these are the available options, it is essential to strike a balance; your application should be aesthetically pleasing without overwhelming the user. The goal should be to complement the functionality with design, not overshadow it. The sample code provided can be used as a foundation for exploration. Enhance, modify, and expand it to build applications

that seamlessly blend functionality with intuitive and engaging user interface.

Exercises

- 1. What function in Streamlit creates a sidebar for additional controls and information?**
 - a. st.sidebar
 - b. st.control
 - c. st.extra
 - d. st.side_panel

- 2. How can you set the page title and favicon in a Streamlit app?**
 - a. st.page_config
 - b. st.set_page_config
 - c. st.configure_page
 - d. st.title_config

- 3. What is the purpose of st.echo in Streamlit?**
 - a. To repeat user input
 - b. To print debug information
 - c. To show the source code of a block
 - d. To echo back server responses

- 4. How can you integrate custom HTML in a Streamlit app?**
 - a. st.html
 - b. st.embed_html
 - c. st.markdown with unsafe_allow_html=True

d. st.insert_html

5. Which of the following is NOT a Streamlit layout feature?

- a. st.columns
- b. st.tabs
- c. st.expander
- d. st.accordion

6. What is Streamlit's st.session_state used for?

- a. Managing user sessions and logins
- b. Storing and persisting user input for each session
- c. Configuring app settings
- d. Storing temporary files

7. Which of the following Custom Streamlit components can be created using Custom Streamlit?

- a. Python only
- b. HTML/CSS only
- c. Both Python and JavaScript
- d. JavaScript only

Answers

- 1.** a
- 2.** b
- 3.** c
- 4.** c
- 5.** d
- 6.** b

7. c

Join our book's Discord space

Join the book's Discord Workspace for Latest updates, Offers, Tech happenings around the world, New Release and Sessions with the Authors:

[https://discord.\(bpbonline.com](https://discord(bpbonline.com)



CHAPTER 5

Data Visualization with Streamlit

Introduction

In this chapter, we will discuss the importance of data visualization and dive into the visualization options offered by Streamlit. Streamlit allows you to create interactive and engaging visual elements in your data application. In addition to discussing the visualization options offered by Streamlit, we will also explore Streamlit integration with popular third-party libraries.

Structure

The chapter discusses the following topics:

- Importance of data visualization in data analysis
- Streamlit visualization options
- Streamlit display and aesthetics widgets
- Streamlit chat widgets
- Hands-on: Integrating visual elements in Product Recommender

Objectives

In the previous chapter, we covered Streamlit core widgets, and in this chapter, we will cover the widgets that enable the addition of visualization elements in the data application. We will also go over other media elements provided by Streamlit. Streamlit provides a wide range of chart elements; we will briefly discuss the chart types and which charts are more appropriate for which data.

Importance of data visualization in data analysis

As a data professional, you will often need to share datasets or statistical results with people who are not technical experts. In these situations, it is crucial to communicate in the simplest way possible, as you do not want to overwhelm your audience with complex data.

Data visualization is used in these scenarios. It is tough for most people to make sense of large sets of data or spot important trends just by looking at numbers. But with visualization, this becomes much easier. Think of visualization as a storytelling tool. Good storytelling is at the heart of effective communication for data scientists. It is about turning complex data into a clear, understandable story that anyone can grasp.

Streamlit visualization options

Streamlit offers a wide variety of visualization widgets, which can be used to add visual elements to your application. It is not just about what Streamlit offers; you can also use well-known libraries to make charts. First, we will go through the basic chart options that come with Streamlit. After that, we will explore how to include and set up charts using these extra libraries for even cooler visuals.

Streamlit core chart elements

Streamlit's core chart offerings include fundamental chart types like line charts, bar charts, and area charts, each serving a unique purpose in data storytelling. The following are the core chart elements supported by Streamlit:

- **Line charts:** Ideal for showcasing trends and progressions over time.
- **Bar charts:** Effective for comparing quantities across different categories.
- **Area charts:** Useful for illustrating cumulative totals and understanding part-to-whole relationships.
- **Scatter charts:** These are useful to show the relationship between two variables. They are powerful for visualizing patterns, concentrations, and outliers in data
- **Map charts:** Maps are used to visualize geospatial data. They help in visualizing locations, geographic distributions, or patterns related to physical locations.

Let us explore how to add them to your Streamlit applications. We will be using the following chart types:

- `st.bar_chart`: Display a bar chart
- `st.line_chart`: Display a line chart
- `st.area_chart`: Display an area chart
- `st.scatter_chart`: Display a scatter chart
- `st.map`: Display a map

The following code snippet explores the usage of Streamlit core chart elements:

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

st.divider()

#Bar chart
st.write('Bar chart')

bar_data = {'Categories': ['A', 'B', 'C'], 'Values': [10, 20, 30]}
```

```
df = pd.DataFrame(bar_data)  
st.bar_chart(df, x="Categories", y="Values")  
st.divider()
```

The following Streamlit application page demonstrates the use of a bar chart:

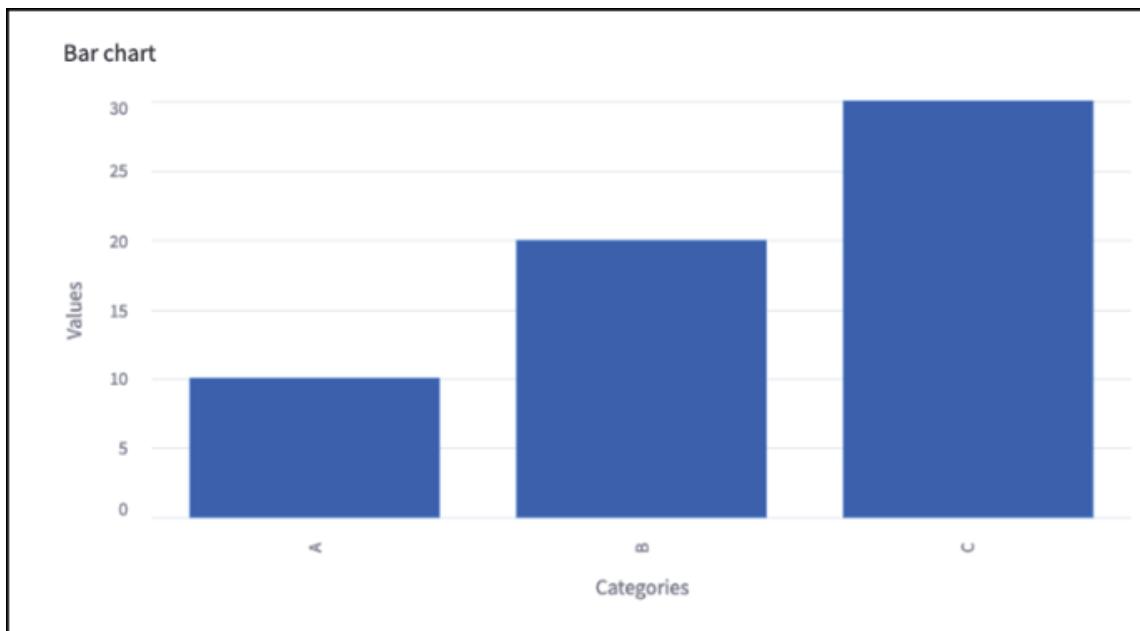


Figure 5.1: Streamlit app with bar charts

The following code snippet demonstrates an example of using a line chart:

```
import streamlit as st  
import pandas as pd  
import numpy as np  
  
st.divider()  
  
#Line chart  
st.write('Line chart')  
line_data = pd.DataFrame({'Date':  
pd.date_range(start='1/1/2020', periods=30), 'Value':
```

```
np.random.randn(30).cumsum()})  
st.line_chart(line_data.set_index('Date'))
```

```
st.divider()
```

The following Streamlit application is an example of using a line chart:



Figure 5.2: Streamlit app with line chart

The following code snippet demonstrates an example of using an area chart:

```
import streamlit as st  
import pandas as pd  
import numpy as np  
  
st.divider()  
#Area chart
```

```
st.write('Area chart')

area_data = pd.DataFrame({'Time': range(1, 11),
                           'Value': np.random.randn(10).cumsum()})

st.area_chart(area_data.set_index('Time'))

st.divider()
```

The following Streamlit application demonstrates the use of an area chart:

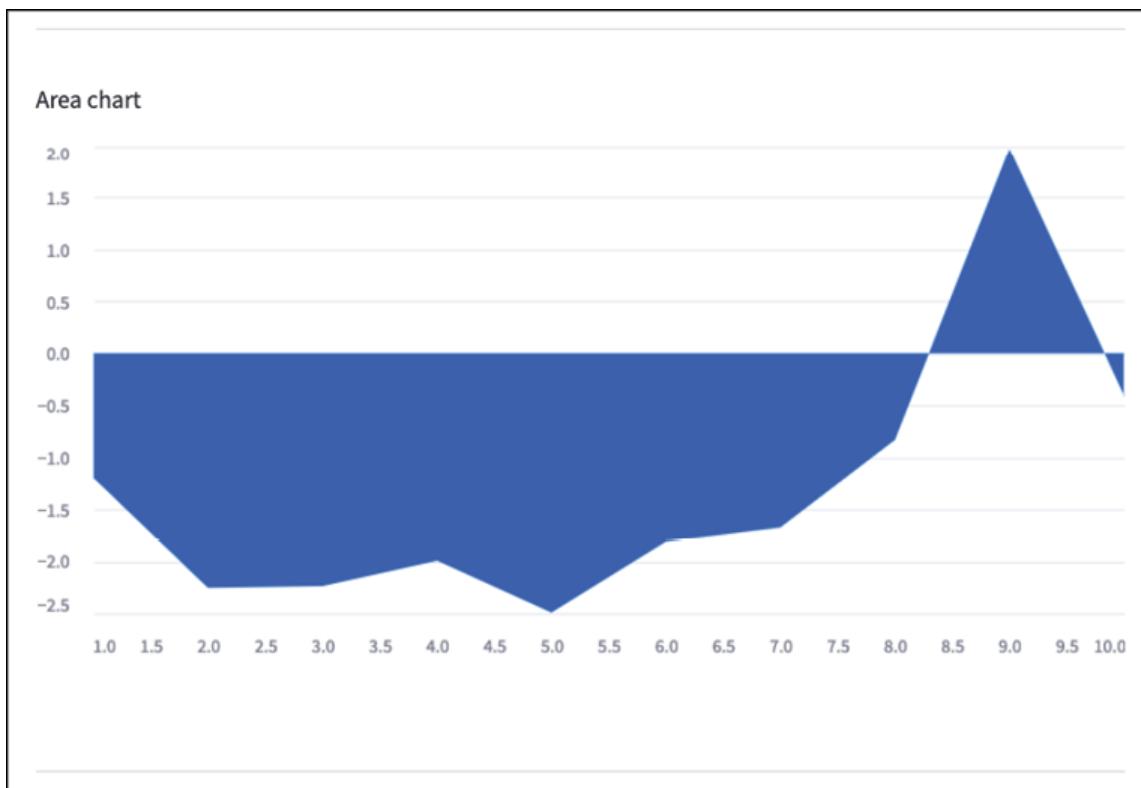


Figure 5.3: Streamlit app with area chart

The following code snippet demonstrates the use of a scatter chart:

```
import streamlit as st

import pandas as pd

import numpy as np
```

```

st.divider()
#Scatter chart
st.write('Scatter chart')
data = pd.DataFrame(np.random.randn(100, 2), columns=['A', 'B'])
st.scatter_chart(data)

```

st.divider()

The following Streamlit application demonstrates the use of a scatter chart:

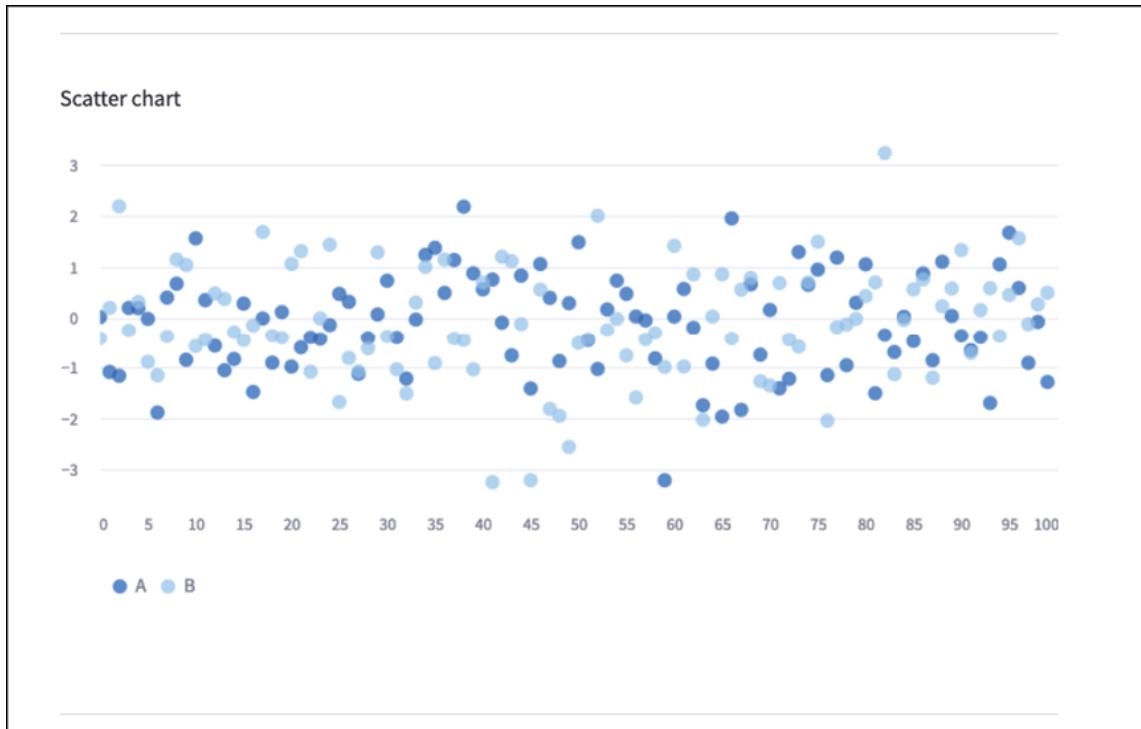


Figure 5.4: Streamlit app with scatter charts

The following code demonstrates the use of a map chart:

```

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

```

```
st.divider()  
#Map chart  
st.write('Map chart')  
  
data = pd.DataFrame({'lat': [37.77, 38.91], 'lon': [-122.41, -77.04]})  
  
st.map(data)  
st.divider()
```

The following Streamlit application demonstrates the use of a map chart:



Figure 5.5: Streamlit app with Map charts

Summarizing the chart types and comparing use cases

Now that we have understood the different core chart types that Streamlit offers let us compare the use cases for each type, as shown in [Table 5.1](#). This will help you decide the right chart type for any use case.

Chart type	Streamlit API	Strengths	Use case
Bar chart	<code>• st.bar_chart()</code>	<ul style="list-style-type: none">• Compare different groups or categories.• Ideal when you want to visualize categorical data like compare size, frequency, or other measures across different categories	Use Bar charts for scenarios such as: <ul style="list-style-type: none">• Comparing sales figures for different months.• Survey results across different demographic groups.
Line chart	<code>• st.line_chart()</code>	<ul style="list-style-type: none">• Best suited for showing trends over time.• Line charts help visualize the progression, speed, and direction of changes.	Use Line charts for scenarios such as: <ul style="list-style-type: none">• Stock prices over a month.• Temperature over a duration.• The volume of Daily website visitors.
Area chart	<code>• st.area_chart()</code>	<ul style="list-style-type: none">• Area charts are useful for showing cumulative totals.• Good for understanding how individual parts contribute to the whole.	Use Area charts for scenarios such as: <ul style="list-style-type: none">• Stock volume trend.
Scatter chart	<code>• st.scatter_chart()</code>	<ul style="list-style-type: none">• Scatter charts help to identify the relationship between two variables, like correlation.• They help to identify how one variable is	Use Scatter charts for scenarios such as: <ul style="list-style-type: none">• Analyzing the relationship between age and blood pressure.

Chart type	Streamlit API	Strengths	Use case
		affecting another.	
Map	• <code>st.map()</code>	Maps are used for geospatial analysis: <ul style="list-style-type: none"> • They help to identify patterns related to physical locations. 	Use maps for scenarios such as: <ul style="list-style-type: none"> • Tracking delivery routes. • Regional sales volume. • Monitor environmental changes.

Table 5.1: Comparing different chart types and their respective use cases

This concludes the core chart functionality provided by Streamlit. Although these charts can cover most of the use cases, they provide limited customization options. To build more advanced visualizations, Streamlit provides direct integrations with many advanced plotting libraries.

Streamlit supported charting libraries

Streamlit supports numerous charting libraries to enhance the user experience of Streamlit app users. The supported libraries spectrum ranges from basic matplotlib to Vega-lite for 2D charts and more advanced options for 3D charts and maps like `deck.gl`.

At the time this chapter was being written, Streamlit provided support for the following libraries:

- `st.pyplot`: Displays `matplotlib.pyplot` charts
- `st.vega_lite_chart`: Charts using Vega-lite library
- `st.plotly_chart`: Charts using plotly
- `st.pydeck_chart`: Charts using Pydeck library
- `st.graphviz_chart`: Charts using dagre-d3 library

Each of these libraries has a unique feature and excels in different scenarios. The choice of which one to use depends on the specific requirements of your data visualization task, such as the type of

data you are working with, the level of interactivity needed, and the complexity of the visualizations you intend to create.

With the following table, let us try to understand the strengths of each library for building charts:

Library	Unique feature	Pros and cons	Use case
Matplotlib	Versatility and wide adoption in the Python ecosystem.	<p>Pros:</p> <ul style="list-style-type: none"> Highly customizable. Wide range of chart types. Good for scientific plots. <p>Cons:</p> <ul style="list-style-type: none"> It can be verbose for complex charts. Less interactive. 	<ul style="list-style-type: none"> Ideal for statistical and scientific data visualizations. Scenarios where fine-grained control over the plot elements is required.
Vega-Lite	Compact JSON syntax and interactive visualizations.	<p>Pros:</p> <ul style="list-style-type: none"> Concise syntax. Good interactivity. Responsive and web-friendly. <p>Cons:</p> <ul style="list-style-type: none"> Requires json familiarity. 	<ul style="list-style-type: none"> Well suited for web-based interactive visualizations, where simplicity and responsiveness are required. Good for rapid prototyping data visualizations.
Plotly	Interactive web-based charts and dashboards.	<p>Pros:</p> <ul style="list-style-type: none"> Highly interactive and dynamic. Rich set of chart types Attractive and modern-looking. 	<ul style="list-style-type: none"> Suited for creating interactive dashboards with complex visualizations that require user engagement.
		<p>Cons:</p> <ul style="list-style-type: none"> It can be resource-intensive. The steeper learning curve for 	

Library	Unique feature	Pros and cons	Use case
		advanced features.	
Pydeck	Geospatial data visualizations	<p>Pros:</p> <ul style="list-style-type: none"> Powerful for 3D and large-scale geospatial plotting. Integration with deck.gl offers advanced features. <p>Cons:</p> <ul style="list-style-type: none"> Requires understanding of deck.gl. 	<ul style="list-style-type: none"> Geospatial visualization, such as city scale plotting, 3d landscapes, and large geographical datasets.
Dagre-D3	Graph and network visualizations.	<p>Pros:</p> <ul style="list-style-type: none"> Excellent for graph-like data structures. Good for flowcharts and process diagrams. <p>Cons:</p> <ul style="list-style-type: none"> Limited to graph-based visualizations. Less versatile for other types of data. 	<ul style="list-style-type: none"> Best for displaying network graphs, organizational charts, and workflow processes.

Table 5.2: Comparing popular charting libraries

Next, let us explore the basic functionality of these libraries as follows:

```

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

x = np.linspace(0, 10, 100)

```

```

y = np.sin(x)

plt.figure(figsize=(8, 4))
plt.plot(x, y, label='Sine Wave')
plt.title("Line Chart with Matplotlib")
plt.xlabel("X-axis")
plt.ylabel("Y-axis")
plt.legend()

st.pyplot(plt)

```

The following Streamlit application demonstrates a chart generated using Matplotlib:

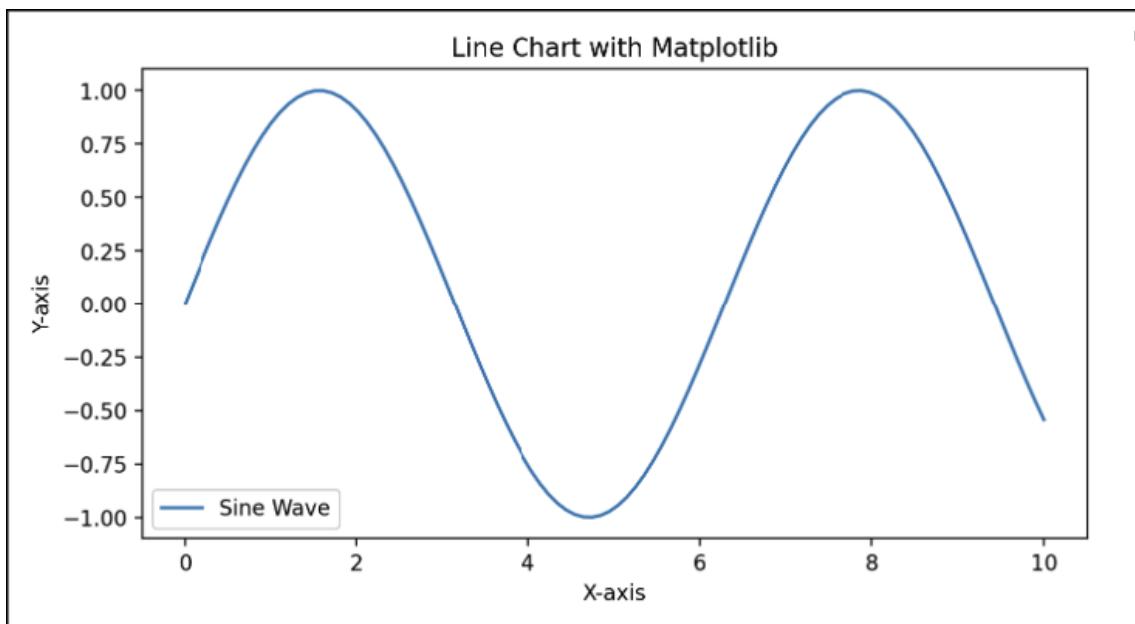


Figure 5.6: Streamlit app with line chart using matplotlib

The following code snippet demonstrates the use of the Vegalite library:

```
import streamlit as st
```

```
example_data = {
    "description": "A simple bar chart with embedded data.",
    "title": "Bar Chart with VegaLite",
    "data": {
        "values": [
            {"a": "A", "b": 28}, {"a": "B", "b": 55}, {"a": "C", "b": 43},
            {"a": "D", "b": 91}, {"a": "E", "b": 81}, {"a": "F", "b": 53},
            {"a": "G", "b": 19}, {"a": "H", "b": 87}, {"a": "I", "b": 52}
        ]
    },
    "mark": "bar",
    "encoding": {
        "x": {"field": "a", "type": "nominal", "axis": {"labelAngle": 0}},
        "y": {"field": "b", "type": "quantitative"}
    }
}
```

```
st.vega_lite_chart(example_data)
```

The following Streamlit application demonstrates the use of the Vega-Lite library:

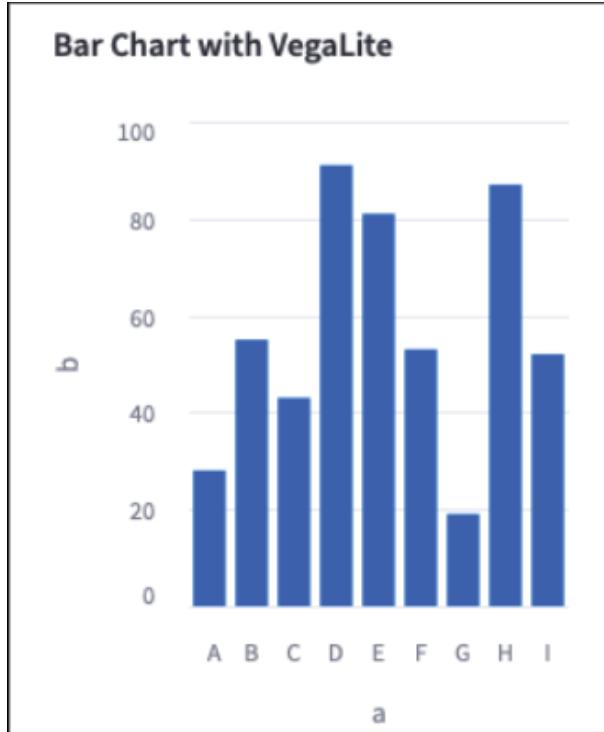


Figure 5.7: Streamlit app with bar chart using Vega-Lite

The following code snippet demonstrates the use of the `plotly` library:

```
import plotly.express as px
import streamlit as st

df = px.data.iris()

fig = px.scatter(df, x="sepal_width", y="sepal_length",
color="species")

st.plotly_chart(fig)
```

The following Streamlit application demonstrates the use of the Vega-Lite library:

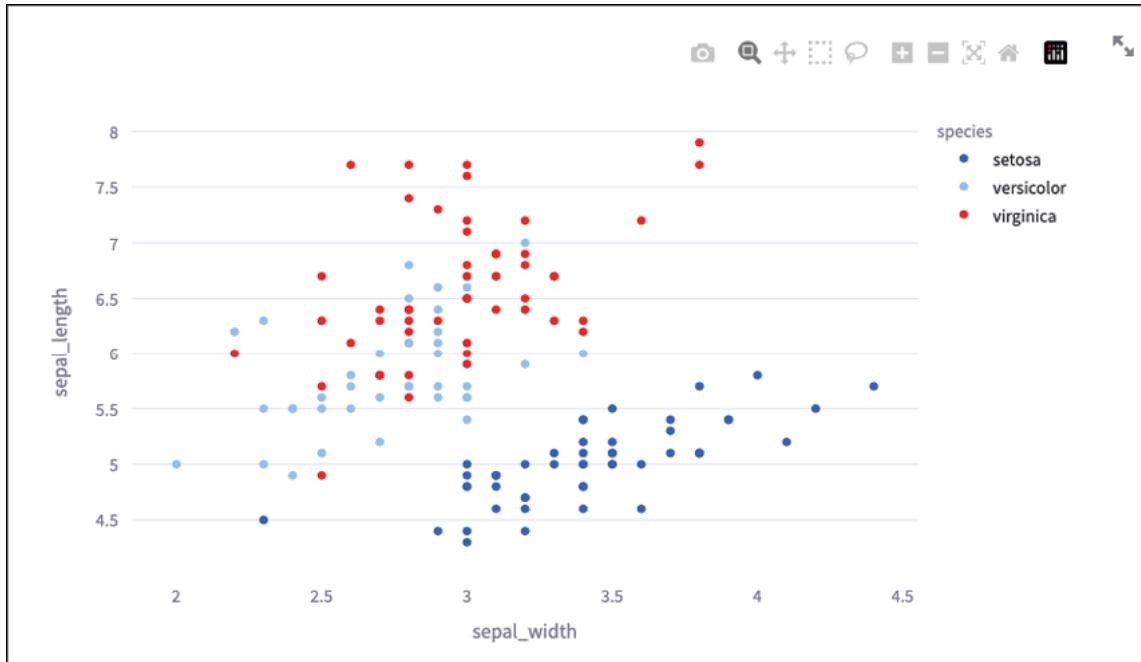


Figure 5.8: Streamlit app with plotly chart

The following code snippet demonstrates the use of the `pydeck` library:

```
import pydeck as pdk

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

# Generating some random data points
np.random.seed(42)
data = pd.DataFrame({
    'lat': np.random.normal(37.76, 0.05, 1000),
    'lon': np.random.normal(-122.4, 0.05, 1000)
})

# Define a layer to display on a map
```

```
layer = pdk.Layer(
    "HexagonLayer",
    data,
    get_position=['lon', 'lat'],
    auto_highlight=True,
    elevation_scale=50,
    pickable=True,
    elevation_range=[0, 3000],
    extruded=True,
    coverage=1
)

# Set the viewport location
view_state = pdk.ViewState(
    latitude=37.76,
    longitude=-122.4,
    zoom=11,
    pitch=40
)

# Render the deck.gl map in Streamlit
st.pydeck_chart(pdk.Deck(layers=[layer],
initial_view_state=view_state))

The following Streamlit application demonstrates the use of the
pydeck library:
```

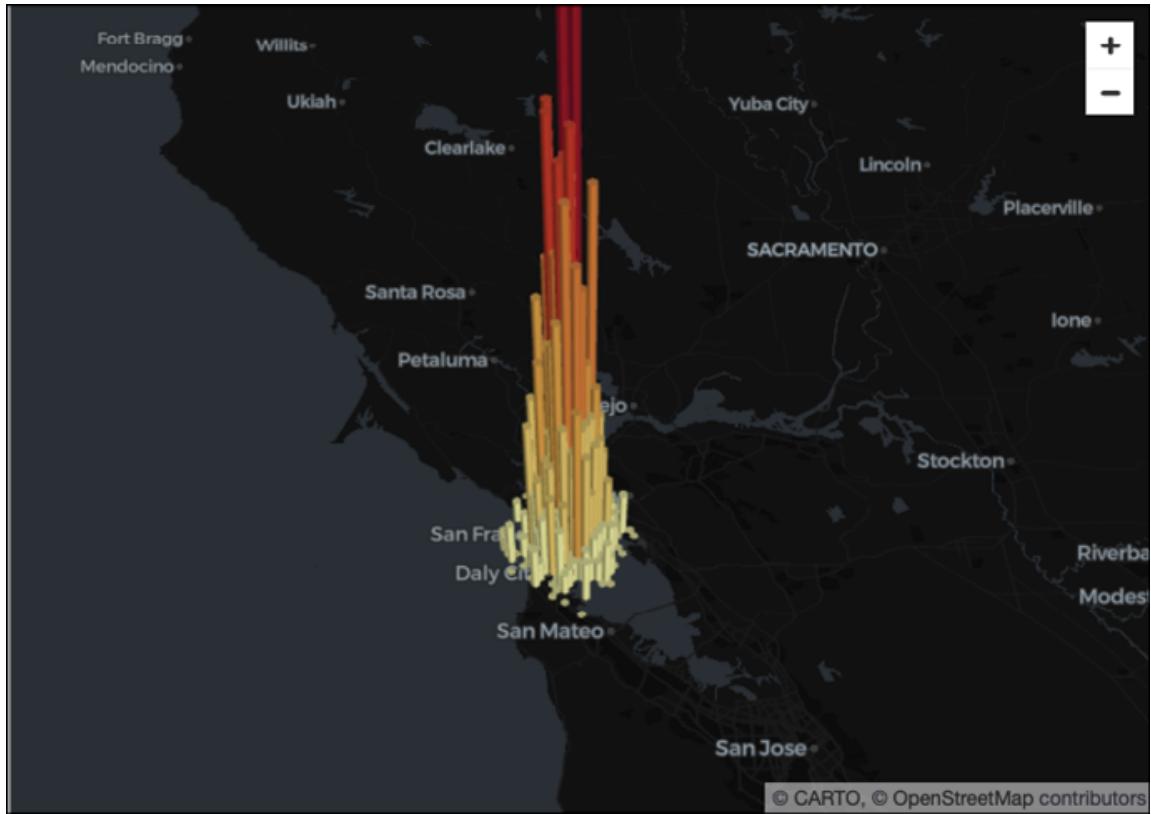


Figure 5.9: Streamlit app with 3d chart using pydeck

The following code snippet demonstrates the use of the Graphviz chart:

```
import streamlit as st

graph = """
digraph {
    a -> b;
    b -> c;
    c -> d;
    d -> a;
}
"""

if __name__ == "__main__":
    st.pydeck_chart(pydeck.from_json(graph))
```

```
st.graphviz_chart(graph)
```

The following Streamlit application demonstrates the use of the graphviz chart:

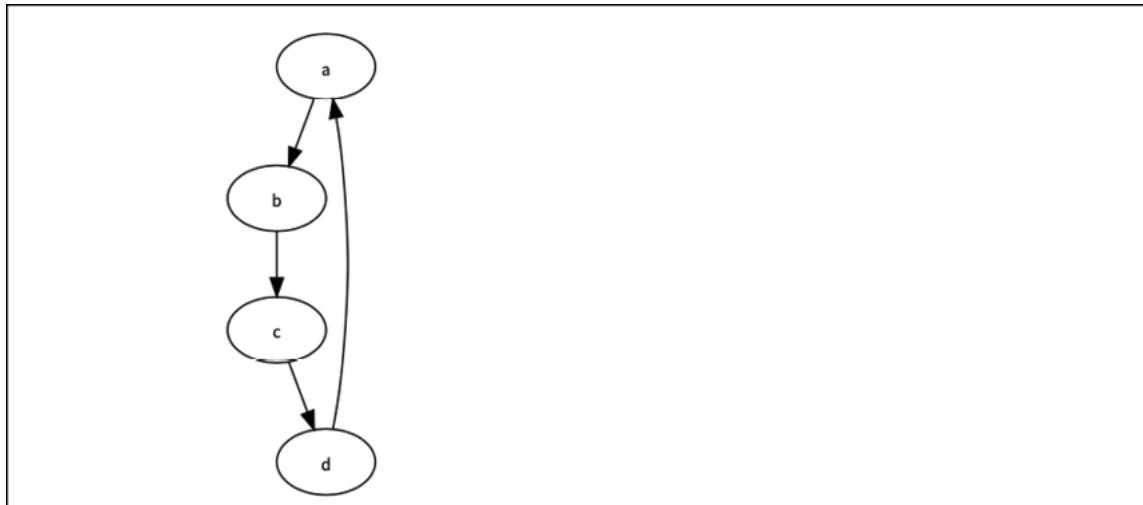


Figure 5.10: Streamlit app graphviz chart

This concludes the chart visualization options provided by Streamlit. Let us move to other graphic widgets:

Streamlit display and aesthetics widgets

This category of widgets plays a crucial role in making the data app more intuitive, informative, and engaging. We will explore a range of widgets that help display metrics, track progress, indicate status, and incorporate media elements. With these widgets, you can transform the way users interact with and perceive your Streamlit applications.

Display progress and status widgets

These widgets help to keep the users informed of task completion or background process progress. These widgets allow you to add animation to your app. The widgets include:

- **st.progress:** This widget displays a progress bar in your Streamlit app, which is useful for visualizing the progress of a long-running operation. It is useful in providing users with feedback on how much of the task has been completed.

- **st.spinner**: This widget shows a loading spinner while a specific operation or code block is running. It is a handy way to indicate to users that the app is processing something in the background.
- **st.status**: This widget is used to display the status of ongoing operations or tasks, typically in the form of notifications or alerts. This widget helps users stay informed about the current state of tasks, whether they are in progress, completed, or encountered errors.
- **st.balloons**: This widget adds a celebratory touch to your Streamlit app by triggering an animation of balloons floating up the screen. This widget is perfect for marking the completion of a task as a milestone achievement.
- **st.snow**: This widget creates a snowfall effect within your Streamlit app, adding a festive atmosphere. It's a nice way to bring seasonal cheer to your app or simply create a more engaging and visually appealing user experience.

The following code snippet demonstrates the use of progress bar widget:

```
import streamlit as st
import time

# Create a progress bar
progress_bar = st.progress(0)
for percent_complete in range(100):
    time.sleep(0.1)
    progress_bar.progress(percent_complete + 1)
```

The following Streamlit application demonstrates the use of the progress bar widget:



Figure 5.11: Streamlit app progress bar

The following code snippet demonstrates the use of the spinner widget:

```
import streamlit as st  
  
import time  
  
# Create a spinner  
with st.spinner('Wait for it...'):  
    time.sleep(5)
```

The following Streamlit application demonstrates the use of a spinner widget:

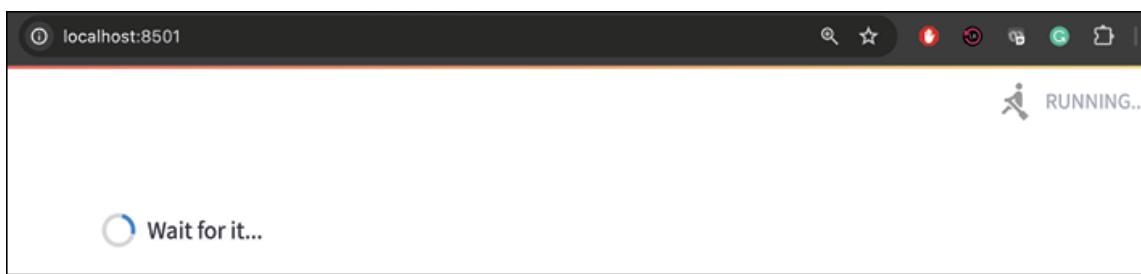


Figure 5.12: Streamlit app spinner

The following code snippet demonstrates the use of balloon widget:

```
import streamlit as st  
  
import time  
  
#balloons flying animation when the button is clicked  
if st.button('Celebrate with balloons!'):
```

```
st.balloons()
```

The following Streamlit application demonstrates the use of balloon widget:

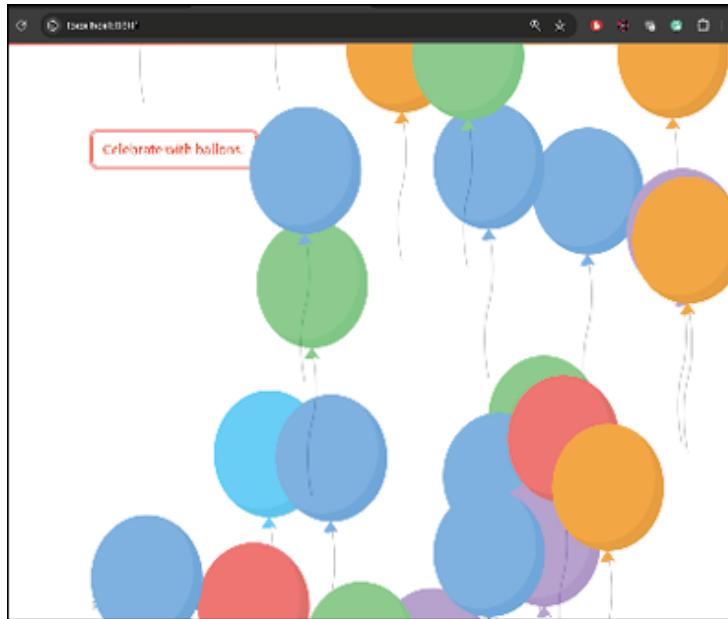


Figure 5.13: Streamlit app with balloons

The following code snippet demonstrates the use of a snow widget:

```
import streamlit as st  
  
import time  
  
#snow falling animation when the button is clicked  
if st.button('Celebrate with snow!'):  
    st.snow()
```

The following Streamlit application demonstrates the use of a snow widget:

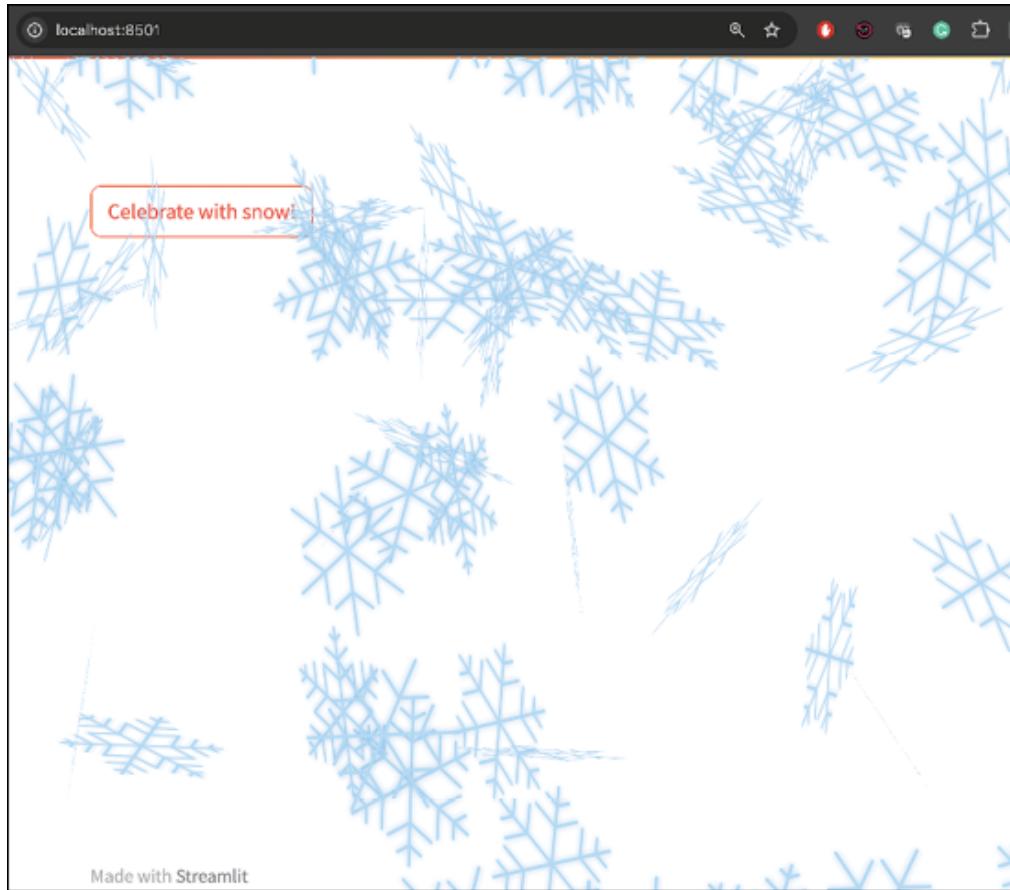


Figure 5.14: Streamlit app with snowflakes

Display message widgets

These widgets help display messages, which are generally the result of actions taken during user activity. Most of the time, these messages are preceded by progress and status activity. These widgets include:

- `st.info`
- `st.success`
- `st.error`
- `st.warning`
- `st.exception`
- `st.toast`

The following code snippet demonstrates the use of the display message widget:

```
import streamlit as st  
  
import time  
  
st.divider()  
  
# Display different status messages  
st.error('This is an error message')  
st.warning('This is a warning message')  
st.info('This is an informational message')  
st.success('This is a success message')
```

The following Streamlit application demonstrates the use of the display message widget:

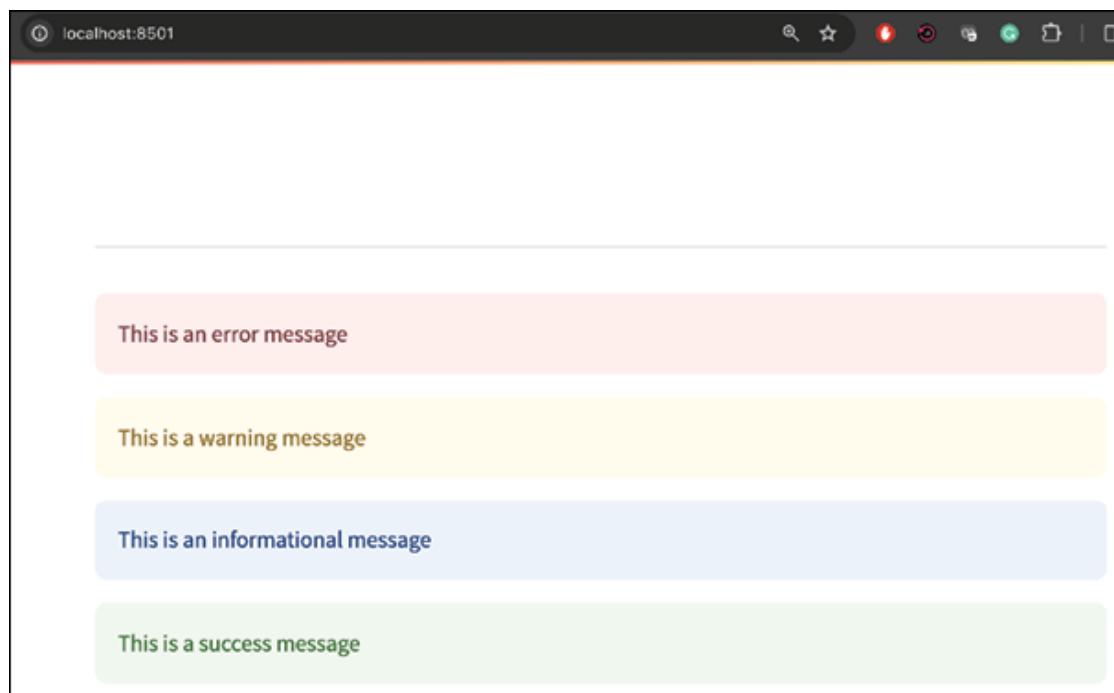


Figure 5.15: Streamlit app with different message types

Media widgets

These widgets are used to embed images, videos, or audio in your Streamlit data app. These widgets include:

- `st.image`
- `st.audio`
- `st.video`

The following code snippet demonstrates the use of the media widget:

```
import streamlit as st

image_url =
"https://unsplash.com/photos/WLUH09A_xik/download?
force=true&w=640"

st.image(image_url, caption='Beautiful Landscape')

audio_url = "https://freemusicarchive.org/track/love-
in-the-abstract/download"
st.audio(audio_url, format='audio/mp3')

video_url = "https://pixabay.com/videos/download/video-
211/download.mp4"
st.video(video_url)
```

The following Streamlit application demonstrates the use of the media widget:

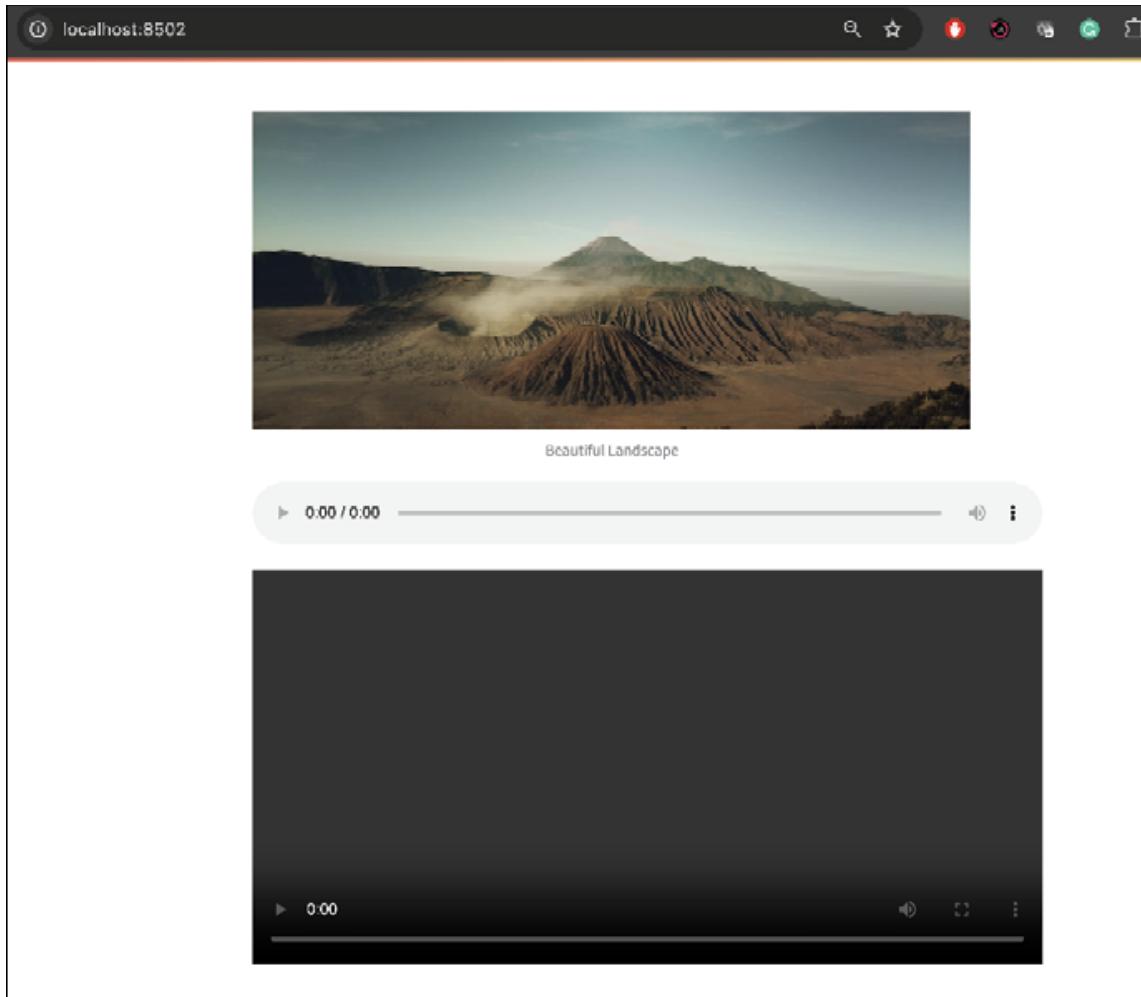


Figure 5.16: Streamlit app with media widgets

Metric widget

This widget is generally used while building data dashboards to maximize the impact of metrics in your application. It provides an effective way of displaying key figures and statistics.

- `st.metric`

The following code snippet demonstrates the use of the metric widget:

```
import streamlit as st
```

```
col1, col2, col3 , col4= st.columns(4)
```

```
col1.metric("Temperature", "26 °C", "1.2 °F")
col2.metric("Wind", "20 mph", "-8%")
col3.metric("Humidity", "16%", "4%")
col4.metric("Rain", "6%", "10%")
```

The following Streamlit application demonstrates the use of the metric widget; these are very useful when developing summary dashboards to display cumulative or summary statistics:

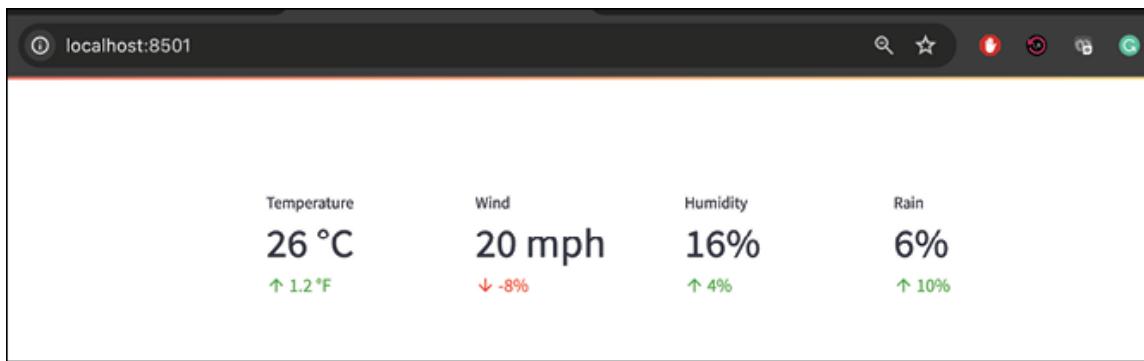


Figure 5.17: Streamlit app with metric widgets

Streamlit chat widgets

Streamlit's chat widgets offer a unique and interactive way to engage users in a conversational interface within your web applications. These widgets have the potential to build complex applications, such as chatbots powered by **Large Language Models (LLMs)**. To demonstrate its prowess, we will focus on creating a simple chat application that provides random quotes in response to user input. For this, we will use an online dataset of quotes and integrate it into our chat application.

The following code snippet demonstrates the use of the chat widget:

```
import streamlit as st
import requests
```

```
# Function to fetch random quotes from an online API
def fetch_random_quote():

    response =
requests.get('https://api.quotable.io/random')

    if response.status_code == 200:

        data = response.json()

        return f'{data['content']} - {data['author']}'"

    else:

        return "Unable to fetch quote at this time."

# Chat interface in Streamlit
st.title("Random Quote Chatbot")
user_input = st.chat_input("Send a message to the
chatbot")

if user_input:

    with st.chat_message('user'):

        st.write(user_input)

    with st.chat_message('assistant'):

        st.write(fetch_random_quote())
```

The following Streamlit code demonstrates the use of the chatbot build using the chat widget feature of Streamlit:

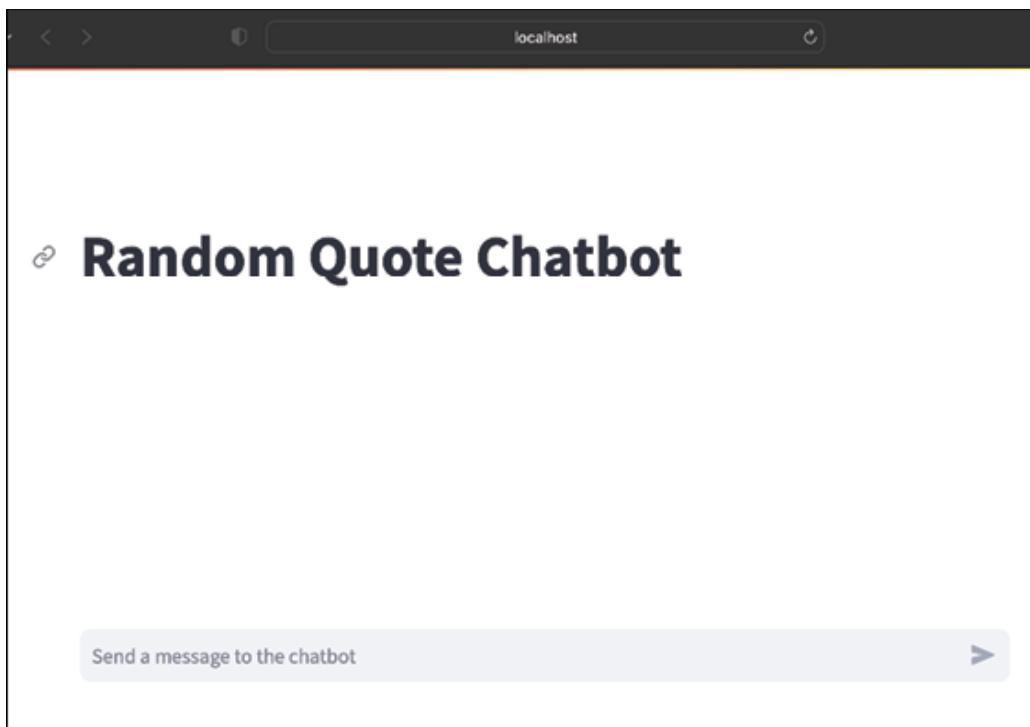


Figure 5.18: Streamlit app with chatbot feature

The following Streamlit application shows the entry box for the chatbot:

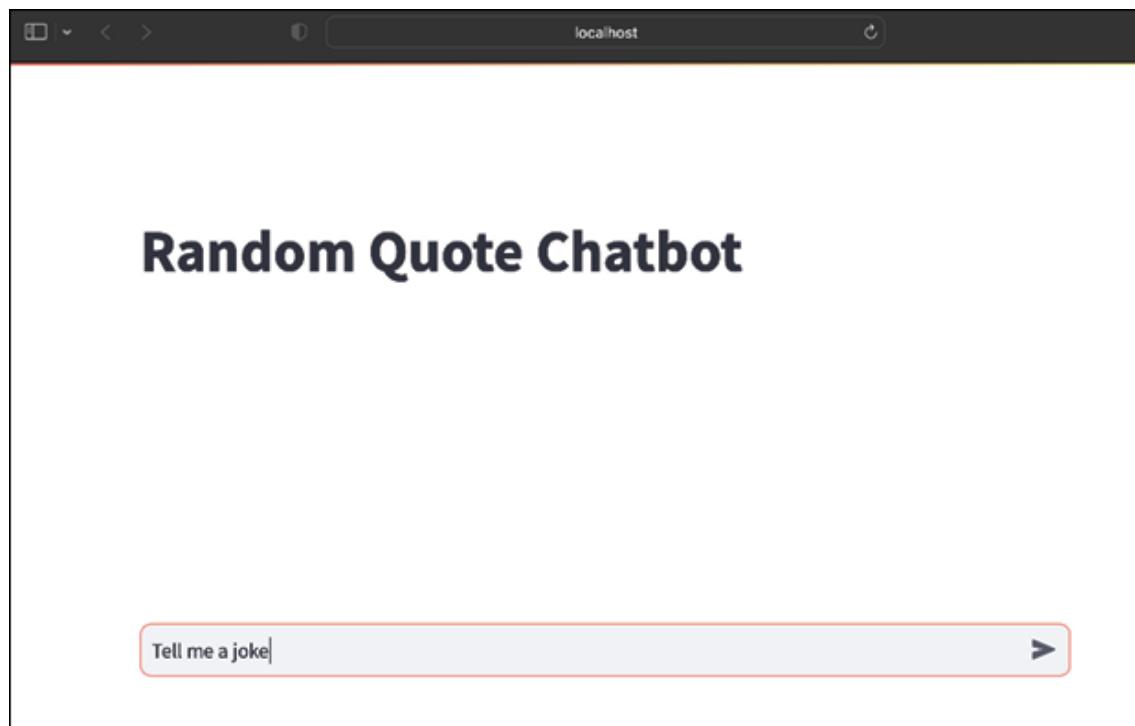


Figure 5.19: Streamlit app with chatbot feature and input string

The following Streamlit application shows the response received from the chatbot feature:

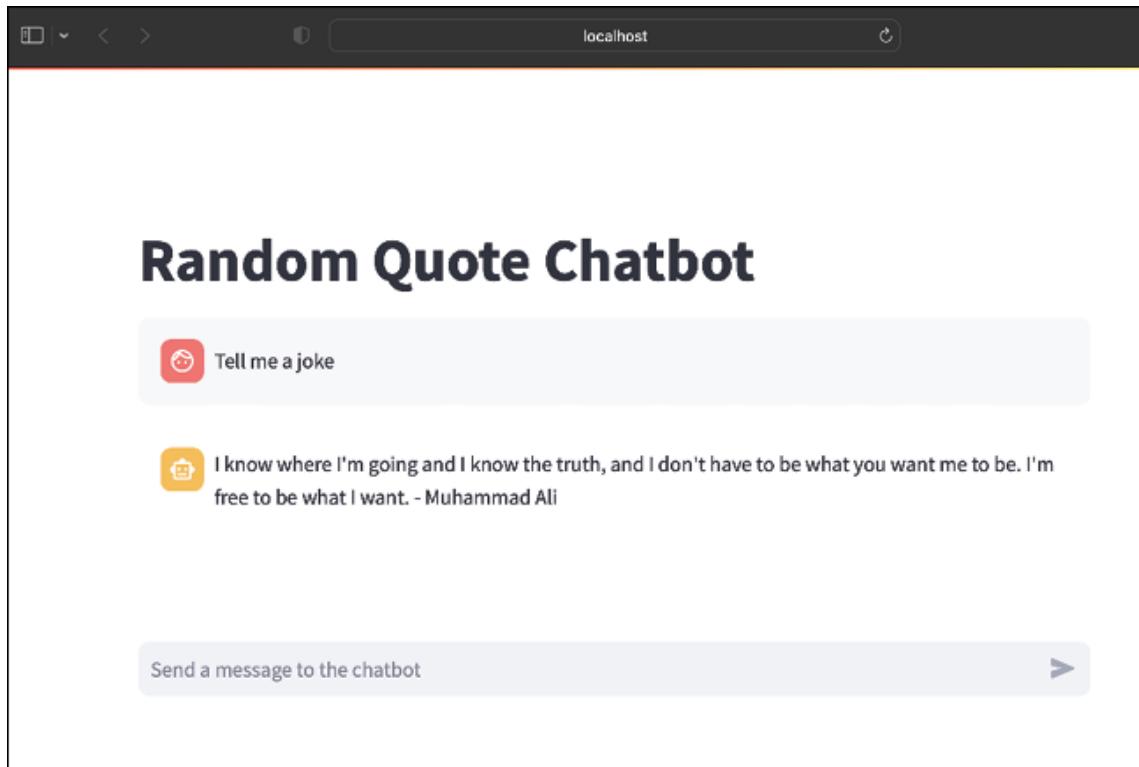


Figure 5.20: Streamlit app with chatbot feature and output response

The example provided here serves as a starting point for readers to explore more complex implementations.

Hands-on: Integrating visual elements in Product Recommender

Let us enhance our Product Recommender application with the visual elements we have learned in this chapter. We will be adding the following enhancements in our last iteration of Product Recommender:

- Add sales, customer, and product insight metrics and visualization in `top_products` file.
- On the recommendation page, add the bar chart to show the distribution of the products with the highest probability.

- Use product data from the CSV with additional features to enhance the insights and recommendations.

With these changes, the folder structure of the project is updated and looks as follows:

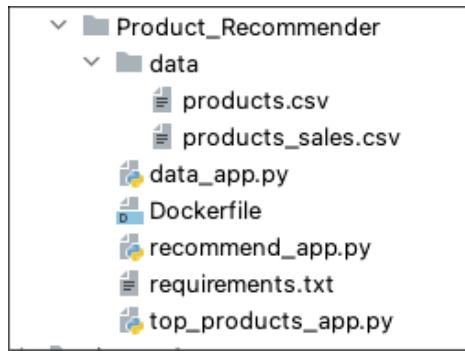


Figure 5.21: Updated project structure based on the new changes added

A new folder `data` is added, which contains the CSV files:

- `product.csv`: Contains the product and category information.
- `product_sales.csv`: Contains the sales data of the product.

All this data is synthetic and generated using ChatGPT. The data for `product.csv` is as follows:

Category,Product

Educational,Book

Educational,Online Course

Educational,Educational Toy

Educational,Science Kit

Educational,Math Workbook

Educational,Language Learning Software

Travel,Luggage

Travel,Travel Pillow

Travel,World Map

Travel,Travel Guidebook

Travel,Portable Charger
Travel,Travel Backpack
Books,Fiction
Books,Non-Fiction
Books,Biography
Books,Mystery
Books,Science Fiction
Books,Historical Fiction
Books,Self-Help
Gadgets,Phone
Gadgets,Headset
Gadgets,Smartwatch
Gadgets,Laptop
Gadgets,Tablet
Gadgets,E-reader
Gadgets,Portable Speaker
Gadgets,Fitness Tracker
Home & Kitchen,Coffee Maker
Home & Kitchen,Blender
Home & Kitchen,Smart Home Device
Home & Kitchen,Cookware Set
Home & Kitchen,Decorative Vase
Home & Kitchen,Bedding Set
Fashion,T-shirt
Fashion,Jeans

Fashion,Dress
Fashion,Watch
Fashion,Handbag
Fashion,Sneakers
Fashion,Sunglasses
Fashion,Jacket
Health & Wellness,Yoga Mat
Health & Wellness,Fitness Band
Health & Wellness,Protein Powder
Health & Wellness,Skincare Product
Health & Wellness,Haircare Kit
Health & Wellness,Vitamins
Health & Wellness,Aromatherapy Diffuser
Health & Wellness,Essential Oils
Toys & Games,Board Game
Toys & Games,Action Figure
Toys & Games,Stuffed Animals
Toys & Games,Outdoor Toy

The data for `product_sales.csv` is as follows:

Category	Product	Product Name	Order Date	Price	User Age	Country	Discount	Quantity	User Gender	User ID	Rating	Return Customer	Sales Channel
Fashion	Jeans	Jeans Standard	2023-12-17,144.76	32	Canada	0.37	9	Male	7912	1	True	Online	
Travel	Travel Pillow	Travel Pillow Pro	2023-10-23,23.0	30	Australia	0.32	9	Female	4611	4	False	Online	

Health & Wellness,Aromatherapy Diffuser,Aromatherapy Diffuser Standard,2023-11-30,217.08,35,Canada,0.11,6,Male,2519,4,True,In-Store

Health & Wellness,Protein Powder,Protein Powder Pro,2023-06-14,405.49,64,Germany,0.27,7,Male,5803,5,False,Third-Party Reseller

Travel,Travel Backpack,Travel Backpack Standard,2023-03-21,334.02,67,UK,0.49,4,Male,7227,3,False,Third-Party Reseller

Health & Wellness,Protein Powder,Protein Powder Standard,2023-07-25,184.08,60,UK,0.35,2,Non-Binary,3803,5,True,Online

Gadgets,Portable Speaker,Portable Speaker Deluxe,2024-02-26,323.6,53,Canada,0.34,1,Male,1525,3,False,In-Store

Educational,Online Course,Online Course Pro,2024-02-07,361.78,31,Australia,0.25,8,Male,5339,2,True,Third-Party Reseller

Home & Kitchen,Decorative Vase,Decorative Vase Deluxe,2023-12-19,296.45,55,Germany,0.18,3,Non-Binary,9085,1,True,Online

Travel,Travel Backpack,Travel Backpack Standard,2024-01-05,343.44,56,USA,0.19,10,Female,9669,3,True,Third-Party Reseller

Fashion,Jeans,Jeans Pro,2024-02-10,273.12,35,Australia,0.17,5,Female,3591,4,True,Third-Party Reseller

Fashion,Handbag,Handbag Pro,2023-12-25,97.54,24,Australia,0.15,9,Non-Binary,4258,2,False,Online

Home & Kitchen,Decorative Vase,Decorative Vase Standard,2023-10-22,168.84,19,USA,0.46,5,Male,1949,2,True,Online

Fashion,Jacket,Jacket Standard,2023-12-
24,271.03,26,Canada,0.33,9,Male,5342,5,False,Online

Home & Kitchen,Bedding Set,Bedding Set Pro,2023-05-
21,359.34,43,Australia,0.32,8,Non-
Binary,8397,1,True,Online

Educational,Educational Toy,Educational Toy
Standard,2023-10-
18,281.42,55,Canada,0.0,1,Male,2104,1,False,Online

Home & Kitchen,Blender,Blender Deluxe,2023-11-
18,247.85,52,Canada,0.36,10,Non-Binary,8744,2,False,In-
Store

Travel,Luggage,Luggage Standard,2023-11-
15,221.22,45,Germany,0.23,1,Non-
Binary,2612,1,False,Third-Party Reseller

Books,Self-Help,Self-Help Standard,2023-06-
08,103.88,52,Germany,0.07,3,Female,8579,2,True,In-Store

Health & Wellness,Aromatherapy Diffuser,Aromatherapy
Diffuser Pro,2023-04-
11,34.79,52,USA,0.48,4,Male,7658,4,False,Online

Health & Wellness,Skincare Product,Skincare Product
Standard,2023-05-
07,195.7,42,UK,0.46,8,Female,7930,5,False,Online

Travel,World Map,World Map Standard,2023-03-
26,293.8,52,USA,0.37,1,Male,8811,5,True,Online

Home & Kitchen,Coffee Maker,Coffee Maker Standard,2023-
03-30,301.57,61,Canada,0.2,10,Male,1651,5,True,In-Store

Fashion,Sneakers,Sneakers Deluxe,2023-05-
22,338.17,38,Canada,0.13,3,Non-
Binary,5915,4,False,Online

Educational,Science Kit,Science Kit Pro,2023-10-
08,498.09,22,France,0.11,5,Male,6718,1,True,In-Store

Books,Non-Fiction,Non-Fiction Deluxe,2024-01-21,276.18,37,France,0.49,9,Male,5905,1,True,In-Store Educational,Book,Book Pro,2023-10-05,86.17,36,France,0.11,6,Male,5325,5,False,In-Store Health & Wellness,Yoga Mat,Yoga Mat Standard,2023-11-05,217.55,35,USA,0.0,3,Non-Binary,5291,2,False,Third-Party Reseller Fashion,Sunglasses,Sunglasses Pro,2023-03-07,64.82,62,Canada,0.27,6,Non-Binary,3426,4,True,Online Books,Biography,Biography Standard,2024-02-16,185.33,61,Canada,0.33,6,Non-Binary,7658,5,True,Online Health & Wellness,Fitness Band,Fitness Band Standard,2024-02-09,212.02,29,Australia,0.46,7,Non-Binary,5065,3,True,Third-Party Reseller Educational,Science Kit,Science Kit Standard,2024-01-31,240.64,30,Germany,0.17,4,Male,1387,2,False,In-Store Books,Self-Help,Self-Help Standard,2023-12-29,146.77,59,France,0.2,9,Female,1452,1,False,Online Home & Kitchen,Smart Home Device,Smart Home Device Standard,2023-04-15,302.32,40,Australia,0.39,7,Non-Binary,9379,1,False,Third-Party Reseller Travel,World Map,World Map Standard,2023-12-04,223.67,51,France,0.34,4,Female,8066,1,False,Third-Party Reseller Books,Historical Fiction,Historical Fiction Standard,2023-12-08,451.02,50,UK,0.33,6,Female,5844,5,True,Online Gadgets,Portable Speaker,Portable Speaker Pro,2023-12-19,452.37,57,France,0.15,9,Male,5978,3,True,In-Store Health & Wellness,Haircare Kit,Haircare Kit Pro,2023-11-

13,167.9,46,Germany,0.34,9,Female,3780,1,False,Third-Party Reseller

Fashion,Sneakers,Sneakers Standard,2024-01-16,476.48,33,Australia,0.16,4,Male,1400,1,True,In-Store

Home & Kitchen,Coffee Maker,Coffee Maker Deluxe,2023-08-12,444.21,54,Canada,0.36,7,Female,7547,2,True,Third-Party Reseller

Fashion,T-shirt,T-shirt Standard,2023-12-31,218.32,29,Australia,0.26,1,Non-Binary,5082,1,False,Online

Health & Wellness,Skincare Product,Skincare Product Pro,2023-09-26,494.07,56,UK,0.48,8,Non-Binary,9270,4,False,Online

Fashion,Jacket,Jacket Deluxe,2023-06-12,378.37,58,UK,0.38,9,Female,4919,3,False,Online

Fashion,Handbag,Handbag Standard,2023-06-17,174.57,52,USA,0.07,4,Female,3503,2,True,In-Store

Gadgets,E-reader,E-reader Pro,2023-09-01,213.73,31,Germany,0.19,10,Non-Binary,1320,5,False,In-Store

Educational,Educational Toy,Educational Toy Deluxe,2023-12-21,201.09,44,France,0.37,9,Non-Binary,4613,4,True,In-Store

Gadgets,Fitness Tracker,Fitness Tracker Standard,2023-08-02,174.7,61,Germany,0.36,8,Male,9751,1,False,Third-Party Reseller

Home & Kitchen,Bedding Set,Bedding Set Standard,2023-04-05,324.95,26,Germany,0.09,5,Female,6363,2,False,In-Store

Books,Self-Help,Self-Help Deluxe,2023-06-20,378.48,44,UK,0.42,8,Male,9837,1,False,Online

Fashion,Jeans,Jeans Pro,2023-03-19,379.57,33,Canada,0.42,10,Male,4908,2,False,Third-Party Reseller

Educational,Math Workbook,Math Workbook Standard,2023-08-31,352.74,67,UK,0.08,10,Non-Binary,2876,2,False,Online

Home & Kitchen,Coffee Maker,Coffee Maker Deluxe,2023-10-13,341.9,42,Germany,0.47,4,Male,4978,1,False,Third-Party Reseller

Home & Kitchen,Coffee Maker,Coffee Maker Pro,2024-01-02,30.12,52,Germany,0.33,2,Non-Binary,6590,1,False,In-Store

Educational,Science Kit,Science Kit Deluxe,2023-11-06,446.87,47,Australia,0.08,3,Non-Binary,9548,3,False,In-Store

Gadgets,Tablet,Tablet Deluxe,2024-01-28,130.29,23,UK,0.44,4,Female,7209,3,True,In-Store

Health & Wellness,Protein Powder,Protein Powder Standard,2023-09-09,113.94,69,UK,0.17,10,Non-Binary,5526,5,True,Third-Party Reseller

Travel,Luggage,Luggage Standard,2023-12-08,209.15,53,Canada,0.35,8,Female,1282,1,False,Online

Gadgets,Laptop,Laptop Deluxe,2023-11-17,294.96,48,France,0.27,7,Non-Binary,6419,3,False,In-Store

Books,Biography,Biography Standard,2023-04-20,363.43,38,USA,0.37,3,Male,4545,4,False,Third-Party Reseller

Home & Kitchen,Decorative Vase,Decorative Vase Pro,2023-06-22,490.45,30,UK,0.11,3,Female,1231,5,True,In-Store

Educational,Book,Book Pro,2023-06-
25,351.73,26,Australia,0.43,8,Male,1200,5,False,In-
Store

Gadgets,Fitness Tracker,Fitness Tracker Deluxe,2023-05-
14,483.09,34,Germany,0.06,2,Female,9056,1,True,Online

Travel,Portable Charger,Portable Charger Deluxe,2023-
04-06,497.02,25,France,0.38,10,Non-
Binary,4697,5,False,In-Store

Gadgets,Tablet,Tablet Pro,2023-08-
17,159.64,57,USA,0.3,2,Male,4457,3,True,Online

Travel,Travel Pillow,Travel Pillow Pro,2023-04-
02,86.7,44,Germany,0.34,8,Female,1534,2,False,Third-
Party Reseller

Books,Historical Fiction,Historical Fiction
Deluxe,2023-03-
31,346.82,34,Australia,0.29,4,Female,2880,5,True,Third-
Party Reseller

Travel,World Map,World Map Standard,2023-03-
31,39.22,68,UK,0.3,10,Female,8195,1,False,Third-Party
Reseller

Books,Historical Fiction,Historical Fiction
Deluxe,2023-06-
17,255.23,49,Germany,0.04,1,Female,6280,5,False,Online

Educational,Online Course,Online Course Pro,2024-01-
22,431.65,55,USA,0.5,5,Non-Binary,1659,2,False,Third-
Party Reseller

Fashion,Jacket,Jacket Deluxe,2023-05-
13,498.5,45,Australia,0.41,2,Female,6700,4,False,In-
Store

Fashion,Jeans,Jeans Standard,2023-07-
10,211.71,49,UK,0.33,7,Non-Binary,1601,4,True,In-Store

Books,Biography,Biography Standard,2023-12-29,208.05,50,USA,0.33,9,Female,7770,1,True,Third-Party Reseller

Books,Science Fiction,Science Fiction Deluxe,2023-11-02,226.6,21,Canada,0.13,3,Female,8178,4,True,Online

Fashion,Watch,Watch Pro,2023-05-04,162.27,18,France,0.2,4,Male,8560,1,True,In-Store

Fashion,Handbag,Handbag Pro,2023-12-03,143.98,48,Germany,0.12,9,Male,7284,2,True,Online

Books,Self-Help,Self-Help Deluxe,2023-07-14,467.78,50,UK,0.41,5,Non-Binary,5891,5,False,Online

Gadgets,Fitness Tracker,Fitness Tracker Deluxe,2023-07-11,280.44,52,Germany,0.23,6,Male,4912,5,False,Online

Health & Wellness,Aromatherapy Diffuser,Aromatherapy Diffuser Deluxe,2023-03-21,165.91,48,Australia,0.46,7,Female,3492,4,True,Online

Home & Kitchen,Decorative Vase,Decorative Vase Deluxe,2024-02-04,59.18,46,USA,0.26,8,Male,3361,4,True,Online

Gadgets,Tablet,Tablet Deluxe,2023-11-01,349.43,59,USA,0.43,9,Female,6188,4,True,Third-Party Reseller

Educational,Online Course,Online Course Pro,2023-11-24,461.26,32,Australia,0.05,2,Non-Binary,2646,4,True,Third-Party Reseller

Books,Fiction,Fiction Standard,2023-07-08,400.11,36,UK,0.19,3,Male,9702,4,True,Online

Travel,Luggage,Luggage Pro,2024-02-05,197.45,61,Canada,0.25,10,Male,4804,4,False,In-Store

Books,Historical Fiction,Historical Fiction Standard,2024-02-

16,404.27,36,Australia,0.27,2,Female,6661,5,False,Third-Party Reseller

Gadgets,Tablet,Tablet Deluxe,2024-01-26,158.04,42,Germany,0.05,7,Non-Binary,6881,5,False,Third-Party Reseller

Books,Fiction,Fiction Pro,2023-08-04,144.5,54,Australia,0.39,1,Non-Binary,9138,3,True,Third-Party Reseller

Health & Wellness,Protein Powder,Protein Powder Standard,2023-11-06,103.16,34,Australia,0.38,3,Non-Binary,2591,1,False,In-Store

Educational,Math Workbook,Math Workbook Deluxe,2023-12-13,74.39,36,UK,0.37,3,Male,3165,5,False,Third-Party Reseller

Home & Kitchen,Smart Home Device,Smart Home Device Standard,2023-06-11,457.88,48,UK,0.37,6,Male,8673,1,True,Online

Health & Wellness,Vitamins,Vitamins Pro,2023-11-21,490.17,69,France,0.18,7,Male,5332,5,True,In-Store

Books,Historical Fiction,Historical Fiction Pro,2023-11-20,138.46,42,Australia,0.47,2,Non-Binary,4830,4,True,Third-Party Reseller

Home & Kitchen,Smart Home Device,Smart Home Device Pro,2023-05-29,327.35,58,Germany,0.45,5,Non-Binary,7689,1,True,Online

Educational,Educational Toy,Educational Toy Deluxe,2023-04-18,57.62,52,Canada,0.19,6,Non-Binary,9850,4,True,In-Store

Fashion,Jeans,Jeans Deluxe,2023-10-29,209.98,35,USA,0.35,4,Female,8285,2,False,Online

Fashion,Sneakers,Sneakers Pro,2024-02-17,489.14,40,USA,0.2,4,Male,8450,1,True,Third-Party

Reseller

Fashion,T-shirt,T-shirt Standard,2024-01-03,173.42,26,France,0.1,9,Male,4538,2,False,Online
Health & Wellness,Haircare Kit,Haircare Kit Standard,2023-06-19,430.62,27,Canada,0.27,3,Male,1422,2,True,In-Store
Health & Wellness,Aromatherapy Diffuser,Aromatherapy Diffuser Standard,2023-10-18,168.65,29,UK,0.03,7,Non-Binary,2862,1,False,In-Store
Health & Wellness,Protein Powder,Protein Powder Pro,2023-10-20,63.44,50,Canada,0.47,1,Non-Binary,9543,3,False,Third-Party Reseller

Educational,Book,Book Deluxe,2024-01-27,206.79,61,USA,0.25,8,Male,2323,3,True,Online

The updated code for `recommend_app.py` to use the `product.csv` is as follows:

```
import streamlit as st
import random
import pandas as pd
import matplotlib.pyplot as plt

@st.cache_data
def load_data(file_path):
    return pd.read_csv(file_path)

product_categories = load_data('data/products.csv')

def generate_random_recommendations(selected_products,
product_data, num_recommendations=3):
```

```

    available_products =
product_data[~product_data['Product'].isin(selected_products)]

    recommendations =
available_products.sample(n=min(num_recommendations,
len(available_products)))

    return recommendations

def create_recommendation_df(recommendations):
    recommendations['Likelihood of Purchase'] =
[random.randint(50, 99) for _ in
range(len(recommendations))]

    return recommendations[['Product', 'Likelihood of
Purchase']]

def plot_likelihood_of_purchase(df_recommendations):
    fig, ax = plt.subplots(figsize=(6, 4))

    df_sorted =
df_recommendations.sort_values('Likelihood of
Purchase', ascending=True)

    ax.barh(df_sorted['Product'], df_sorted['Likelihood
of Purchase'], color='skyblue')

    ax.set_xlabel('Likelihood of Purchase (%)')

    ax.set_title('Likelihood of Purchase for
Recommended Products')

def recommend_main():
    with st.container():

        st.title('Product Recommender')

        # Form for category and product selection

```

```
with st.expander(label="", expanded = True):
    col1, col2 = st.columns(2)
    with col1 :
        categories =
product_categories['Category'].unique()
        selected_category =
st.selectbox('Choose category:', options=[''] +
list(categories))
    with col2:
        # Enable product selection only if a
category is selected
        if selected_category:
            filtered_products =
product_categories[product_categories['Category'] ==
selected_category]['Product'].tolist()
            selected_products =
st.multiselect('Choose products:',
options=filtered_products)
        else:
            selected_products =
st.multiselect('Choose products:', [])
    submit_button = st.button(label='Get
Recommendations')

    if submit_button and selected_category and
selected_products:
        recommendations_df =
generate_random_recommendations(selected_products,
```

```
product_categories[product_categories['Category'] ==  
selected_category])  
  
        df_recommendations =  
create_recommendation_df(recommendations_df)  
  
        with st.container():  
            st.title('Recommendations')  
            with st.expander(label="",  
expanded=True):  
  
                col1, col2 = st.columns(2) # You  
can adjust the ratio as needed for your UI  
                with col1:  
  
  
st.dataframe(df_recommendations,  
            column_config={  
                'Likelihood of  
Purchase': st.column_config.NumberColumn(  
                    'Likelihood of  
Purchase',  
                    help="How  
likely is user to buy this **product**",  
                    min_value=0,  
                    max_value=100,  
                    step=1,  
                    format="%d  
%%",  
            )},
```

```

            hide_index=True,
        )

    with col2:

        fig =
plot_likelihood_of_purchase(df_recommendations)
        st.pyplot(fig)

    elif submit_button:
        st.write('Please select a category and at least one product to see recommendations.')
The new code added for the top_products_app.py is as follows:
import streamlit as st
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

def load_data(file_path):
    return pd.read_csv(file_path)

sales_df = load_data('data/products_sales.csv')
sales_df['Order Date'] = pd.to_datetime(sales_df['Order Date'])

def plot_sales_over_time(sales_df):
    sales_over_time = sales_df.groupby(sales_df['Order Date'].dt.to_period('M')).agg({'Price': 'sum'})
    sales_over_time.reset_index(inplace=True)

```

```
    sales_over_time['Order Date'] =
sales_over_time['Order Date'].dt.to_timestamp()

    plt.figure(figsize=(10, 5))
    plt.plot(sales_over_time['Order Date'],
sales_over_time['Price'], marker='o', linestyle='--')
    plt.title('Sales Over Time')
    plt.xlabel('Order Date')
    plt.ylabel('Total Sales ($)')
    plt.xticks(rotation=45)
    plt.tight_layout()
    plt.grid(True)
    st.pyplot(plt)

def plot_customer_demographics(sales_df):
    fig_size = (6, 4)
    fig_age, ax_age = plt.subplots(figsize=fig_size)
    ax_age.hist(sales_df['User Age'], bins=10,
edgecolor='black')
    ax_age.set_title('Age Distribution of Customers')
    ax_age.set_xlabel('Age')
    ax_age.set_ylabel('Frequency')
    fig_gender, ax_gender =
plt.subplots(figsize=fig_size)
    gender_counts = sales_df['User
Gender'].value_counts()
    ax_gender.pie(gender_counts,
labels=gender_counts.index, autopct='%.1f%%',
```

```

    startangle=90)

    plt.tight_layout()

    return fig_age, fig_gender

def plot_top_products(sales_df):
    top_products = sales_df.groupby('Product')
    ['Quantity'].sum().sort_values(ascending=False).head(10)

    fig_top, ax_top = plt.subplots()
    sns.barplot(x=top_products.values,
    y=top_products.index, ax=ax_top)
    ax_top.set_title('Top Products by Quantity Sold')
    st.pyplot(fig_top)

def calculate_clv(sales_df):
    revenue_per_user = sales_df.groupby('User ID')
    ['Price'].sum().mean()

    st.metric(label="Customer Lifetime Value (CLV)",
    value=f"${revenue_per_user:.2f}")

def calculate_aov(sales_df):
    total_revenue = sales_df['Price'].sum()
    total_orders = sales_df.shape[0]
    aov = total_revenue / total_orders
    st.metric("Average Order Value", f"${aov:.2f}")

def calculate_purchase_frequency(sales_df):
    total_orders = sales_df.shape[0]

```

```
unique_customers = sales_df['User ID'].nunique()
purchase_frequency = total_orders /
unique_customers
st.metric("Purchase Frequency(orders per
customer)", f"{purchase_frequency:.2f}")

def top_products_main():
    st.title('Product Recommender Insights')
    col1, col2, col3 = st.columns(3)
    with col1:
        with st.expander(label="", expanded=True):
            calculate_clv(sales_df)
    with col2:
        with st.expander(label="", expanded=True):
            calculate_aov(sales_df)
    with col3:
        with st.expander(label="", expanded=True):
            calculate_purchase_frequency(sales_df)
    st.header('Sales Insight')
    with st.expander(label="", expanded=True):
        plot_sales_over_time(sales_df)

    st.header('Customer Distribution Insights')
    col21, col22 = st.columns(2)
    fig_gender, fig_age =
plot_customer_demographics(sales_df)
```

```

with col21:
    with st.expander(label="", expanded=True):
        st.pyplot(fig_gender)

with col22:
    with st.expander(label="", expanded=True):
        st.pyplot(fig_age)

st.header('Overall Top Products')
plot_top_products(sales_df)

```

Based on the updates added, the recommendation page looks as shown in the following figure:

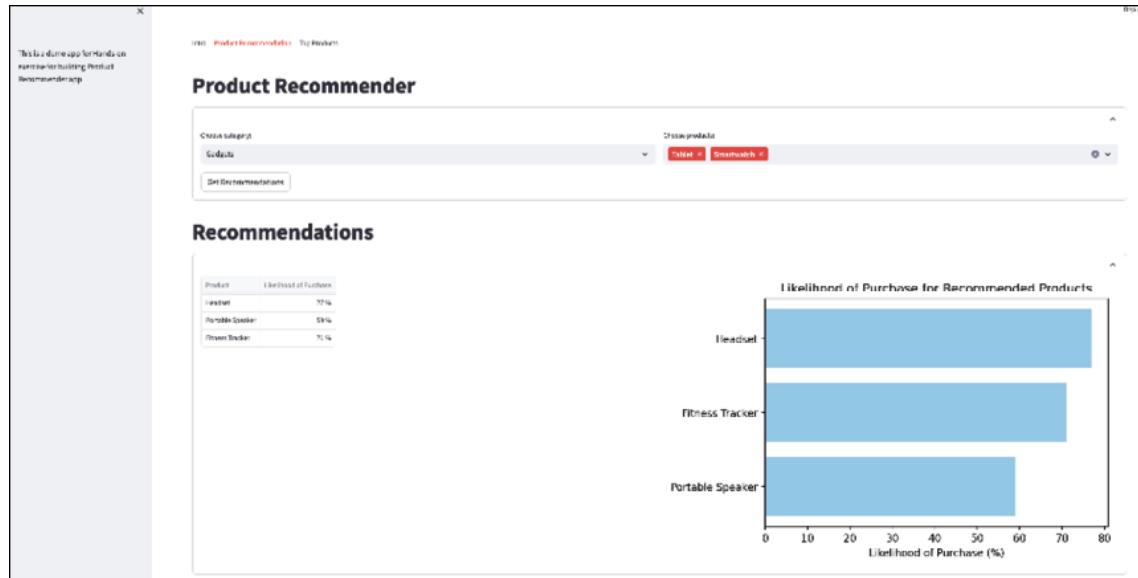


Figure 5.22: Streamlit app with enhanced product recommendation page with visual charts

The top product page has multiple metrics and insights being captured, as shown in the following figures:

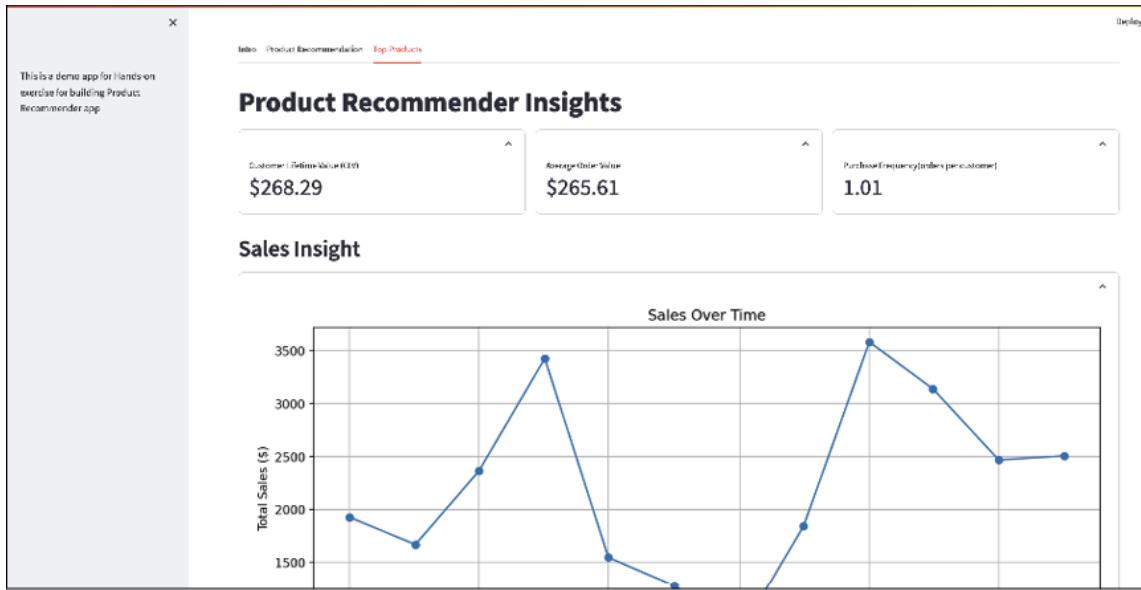


Figure 5.23: Streamlit app with top product sales metric and line chart with sales over time

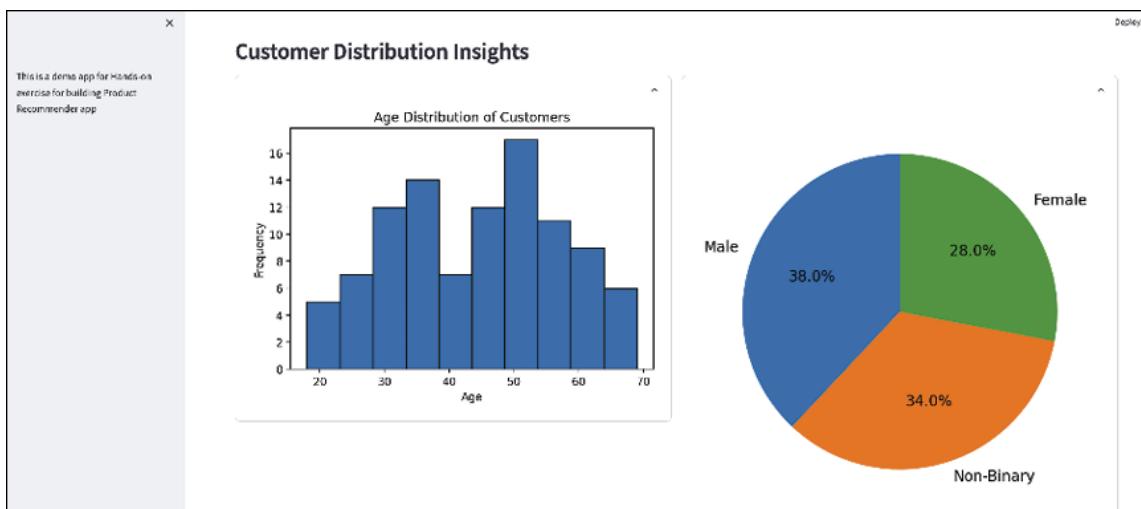


Figure 5.24: Streamlit app with customer insights, age distribution and gender distribution for the sales data

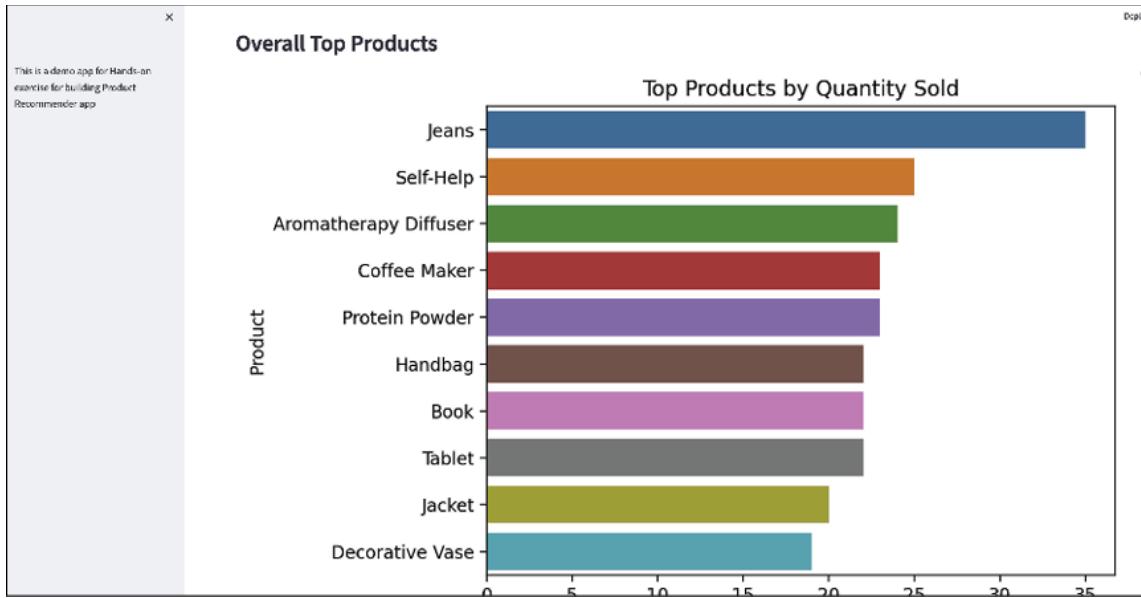


Figure 5.25: Streamlit app with the top product sold, based on the sales data.

These visualizations provide a clear insight into the data captured in `product_sales.csv`.

Conclusion

In this chapter, we have looked at the diverse and powerful visual elements that Streamlit offers to enhance the interactivity and appeal of your data applications. These features not only bring versatility to your toolbox but also enable rapid prototyping, allowing you to bring your ideas to life quickly.

A key takeaway from this chapter is the importance of selecting visual elements that complement the overall aesthetic of your application. While Streamlit offers extensive functionalities, it is crucial to avoid over-engineering. The visual components should enhance, not overpower, the user experience. Simplicity and clarity should always guide your design choices.

The comparison guide we provided is a valuable resource for choosing the right library and chart type for your specific needs. Each library has its strengths, and understanding these can help you make informed decisions about which one to use in different scenarios.

Remember, the key to mastering Streamlit's visual capabilities lies in practice and experimentation. Try out the different widgets and chart types provided, play around with their settings, and observe how they change the look and feel of your application. Through this exploratory process, you will find the right balance and appeal that will make your application visually engaging.

Exercises

- 1. Which Streamlit core API is used to add an interactive map?**
 - a. st.map
 - b. st.plot_map
 - c. st.geo_map
 - d. st.display_map

- 2. Streamlit's st.metric widget is primarily used for?**
 - a. Embedding videos
 - b. Displaying key metrics and KPIs
 - c. Uploading files
 - d. Creating interactive maps

- 3. Which function is used to embed plots from Matplotlib in a Streamlit app?**
 - a. st.pyplot
 - b. st.matplot
 - c. st.graph
 - d. st.plot

- 4. Which Streamlit function is used to display a progress bar?**
 - a. st.progress

- b. st.bar
 - c. st.loading
 - d. st.advance
5. **Which function should be used to create a scatter plot with Streamlit?**
- a. st.plot_scatter
 - b. st.scatter_chart
 - c. st.draw_scatter
 - d. st.scatter
6. **Which function is used in Streamlit to display error messages?**
- a. st.error
 - b. st.warning
 - c. st.info
 - d. st.alert
7. **What is a unique feature of `st.pydeck_chart` in Streamlit?**
- a. To create 3D visualizations
 - b. To upload files
 - c. To embed audio files
 - d. To generate bar charts

Answers

- 1.** a
- 2.** b
- 3.** a

4. a

5. b

6. a

7. a

CHAPTER 6

Streamlit and Machine Learning

Introduction

In this chapter, we will explore how to integrate machine learning models with Streamlit. We will begin with an introduction to machine learning within the Streamlit framework, exploring how this powerful combination can turn complex algorithms into user-friendly web applications. Streamlit's intuitive syntax and versatile widgets enable data professionals to create interactive applications quickly, allowing users to explore and interact with machine learning models effortlessly.

Structure

The chapter discusses the following topics:

- Introduction to machine learning in Streamlit
- Setting up your machine learning environment
- Dynamically adjusting model hyperparameters
- Hands-on: Integrating ML in Product Recommender

Objectives

Building on the foundations laid in the previous chapter, where we explored the capabilities of Streamlit for data visualization and the dynamic interaction it offers, this chapter introduces a pivotal extension of our journey, integrating machine learning with Streamlit. This chapter aims not only to instruct but also to inspire, showcasing how Streamlit can transform static machine-learning models into interactive experiences that encourage exploration, discovery, and insight.

Introduction to machine learning in Streamlit

Machine learning harnesses algorithms to parse data, learn from that data, and make informed decisions based on what it has learned. It is the science of getting computers to act without being explicitly programmed. In the context of data science, machine learning models serve as the engines of prediction and analysis, turning vast and complex datasets into actionable insights. However, the power of machine learning models is fully unleashed when these models become accessible to end-users, allowing for real-time interaction and exploration. This is where Streamlit steps in, offering a seamless bridge between complex machine learning algorithms and interactive, user-friendly web applications.

In this chapter, we will see how Streamlit, with its straightforward syntax and a wide array of widgets, empowers data professionals to rapidly develop applications that can showcase their machine-learning models and allow users to interact with these models in real-time. Whether it is adjusting input parameters to see how predictions change or visually exploring the impact of different data points on model outcomes, Streamlit can make it happen with minimal coding effort.

At a high level, the integration of machine learning models with Streamlit applications involves a few key steps:

- Loading the model
- Creating a user interface to accept inputs
- Displaying the model's predictions

This process transforms static models into dynamic tools for exploration and insight generation.

Consider a scenario where a data scientist has developed a product recommendation system. Traditionally, the model's insights might be shared through a static report or presentation, limiting the audience's ability to explore different scenarios. With Streamlit, however, the model can be brought to life. Users can specify their preferences, adjust their interests, and receive personalized recommendations on the fly. This not only enhances the user experience but also provides deeper insights into the model's behavior and potential applications.

In this chapter, we will also explore the practical aspects of bringing machine learning models into the Streamlit environment, focusing on making the process as intuitive and engaging as possible. From basic model integration to creating interactive visualizations that respond to user input, we will cover the essentials of Streamlit applications with machine learning capabilities.

Setting up your machine learning environment

Before we start the integration of machine-learning models with Streamlit, let us set up a machine-learning environment. This setup ensures that you have all the necessary tools and libraries at your disposal for developing, training, and deploying machine learning models within your Streamlit applications.

Essential Python libraries

The Python ecosystem is rich with libraries to facilitate and speed up various aspects of machine learning, from data manipulation and visualization to model development and evaluation. Here are some fundamental libraries and their respective use cases:

- **NumPy and pandas:**
 - This is for handling numerical data and data frames, respectively.
 - These libraries are essential for data preprocessing, manipulation, and analysis.
- **Scikit-learn:**
 - A comprehensive library that offers simple and efficient tools for data mining and data analysis.
 - It includes a wide range of machine learning models for classification, regression, clustering, and dimensionality reduction.
- **TensorFlow or PyTorch:**
 - These libraries are useful if your application involves deep learning models.
 - They provide flexible platforms for building and training neural networks with high-level capabilities for complex tasks such as image and text processing.
- **Matplotlib and seaborn:**
 - For data visualization. While Streamlit has its own visualization capabilities, these libraries are useful for exploratory data analysis during the model development phase.
 - Streamlit provides direct integration with both the libraries and many more third-party plugins and

libraries for enhanced visualization capabilities.

Preparing your machine learning models

This section aims to guide you through the process of preparing a machine learning model, from conception to deployment within a Streamlit application.

Starting with a simple machine learning model

Let us start with a simple but effective model: linear regression. This model attempts to predict a dependent variable value (y) based on the value of an independent variable (x) by fitting a linear equation to observed data. The simplicity of linear regression makes it a perfect starting point. The steps are as follows:

1. Dataset preparation: Before training the model, we need a dataset. The dataset used is synthetic but serves to illustrate the process well.

For this example, let us use a simple dataset with two columns:

- a. hours studied
- b. exam score

The code is as follows:

```
import pandas as pd  
import numpy as np  
  
data = {  
    'Hours_Studied': [1, 2, 3, 4, 5, 6, 7, 8,  
    9, 10],
```

```
        'Exam_Score': [51, 55, 60, 68, 72, 75, 78,  
82, 88, 90]  
    }  
  
df = pd.DataFrame(data)
```

2. **Model training:** For training, let us use scikit-learn's `LinearRegression` model. We can use this model to predict the value of exam scores based on hours studied.

```
from sklearn.model_selection import  
train_test_split  
  
from sklearn.linear_model import  
LinearRegression  
  
# Preparing the data  
X = df[['Hours_Studied']] # Predictor  
y = df['Exam_Score'] # Response  
  
# Splitting dataset into training and testing  
set  
X_train, X_test, y_train, y_test =  
train_test_split(X, y, test_size=0.2,  
random_state=0)  
  
# Training the model  
model = LinearRegression()  
model.fit(X_train, y_train)
```

3. **Saving the trained model:** The next step is to save the trained model; we will use Joblib for this. Joblib is a

serialization library that provides a simple way to save Python objects, making it especially useful for saving and loading machine learning models, which may include large arrays of data.

```
from joblib import dump  
dump(model, 'linear_regression_model.joblib')
```

Integrating machine learning trained model with Streamlit

Once the model is trained, let us integrate this model into our Streamlit application using the following code:

```
import streamlit as st  
  
from joblib import load  
  
# Load the model  
  
model = load('linear_regression_model.joblib')  
  
# Streamlit user interface  
  
st.title('Exam Score Prediction Based on Hours Studied')  
  
hours_studied = st.slider('Hours Studied',  
min_value=1, max_value=10, value=5)  
  
# Making predictions  
  
predicted_score = model.predict([[hours_studied]])  
[0]
```

```
st.write(f'Predicted Exam Score:  
{predicted_score:.2f}')
```

The following is a simple Streamlit app where users can input the number of hours they studied and receive a predicted exam score based on the linear regression model. It is a straightforward example of how machine learning models can be integrated into Streamlit applications to provide interactive and dynamic insights.

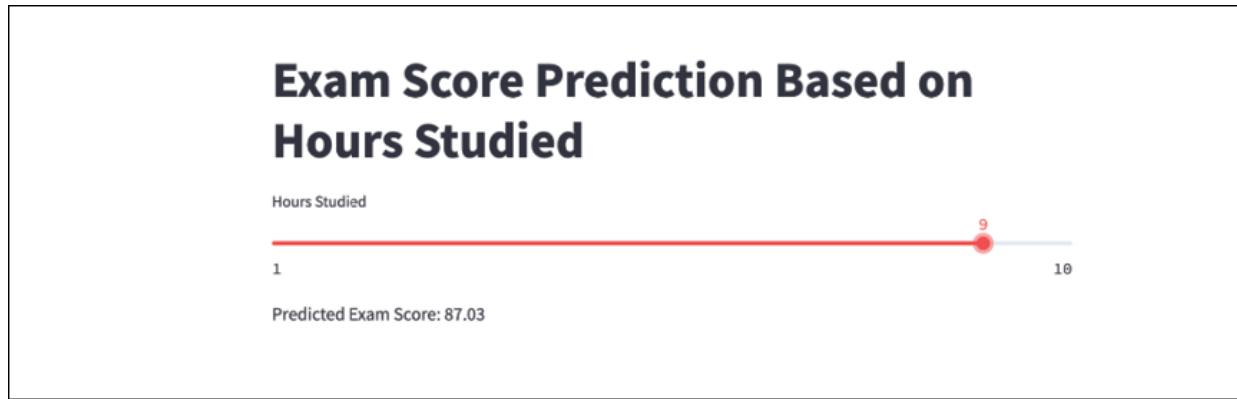


Figure 6.1: Streamlit screen for linear regression model

Dynamically adjusting model hyperparameters

In this section, we explore how to implement the functionality of adjusting model hyperparameters by turning the Streamlit app into a dynamic tool for exploring machine learning models.

Understanding model hyperparameters

Machine learning model hyperparameters are the configurations or settings that can be adjusted to alter the behavior or performance of a machine learning model. Hyperparameters control the behavior of the training process, affecting how the model parameters are adjusted and, consequently, the performance of the model. Proper tuning of hyperparameters is crucial for achieving optimal

model performance. Unlike parameters, they are not learned from the *data* but are used to *control* the learning process. These hyperparameters might include learning rates for optimization algorithms, the number of trees in a random forest model, or regularization strengths in logistic regression, or others. The ability to tweak these settings on the fly and immediately observe the outcomes can significantly aid in understanding the model's complexity and sensitivity to various inputs.

Implementing interactive controls for hyperparameters

Streamlit's interactive widgets can be used to create controls for adjusting the machine learning model's hyperparameters.

Let us consider a scenario with a logistic regression model for binary classification. Important parameters for this model might include the regularization strength (C) and the type of penalty (penalty).

In Streamlit, users can adjust the regularization strength through a slider and choose the penalty type via a select box. The model is then retrained using these hyperparameters, allowing users to explore how different configurations affect the model.

The following code snippet demonstrates how Streamlit can add interactive control for model tuning for end-users:

```
import streamlit as st  
from sklearn.linear_model import LogisticRegression  
from sklearn.datasets import make_classification  
import numpy as np  
  
# Generate a synthetic dataset
```

```

X, y = make_classification(n_samples=100,
n_features=2, n_informative=2, n_redundant=0,
random_state=42)

# Create interactive controls

C = st.slider("Regularization strength",
min_value=0.01, max_value=1.0, value=0.5,
step=0.01)

penalty = st.selectbox("Penalty", options=["l2",
"l1", "elasticnet", "none"])

# Train the model based on user-selected
hyperparameters

model = LogisticRegression(C=C, penalty=penalty,
solver='saga', l1_ratio=0.5 if penalty ==
'elasticnet' else None)

model.fit(X, y)

```

The following is the Streamlit application, which is built by using the preceding code:

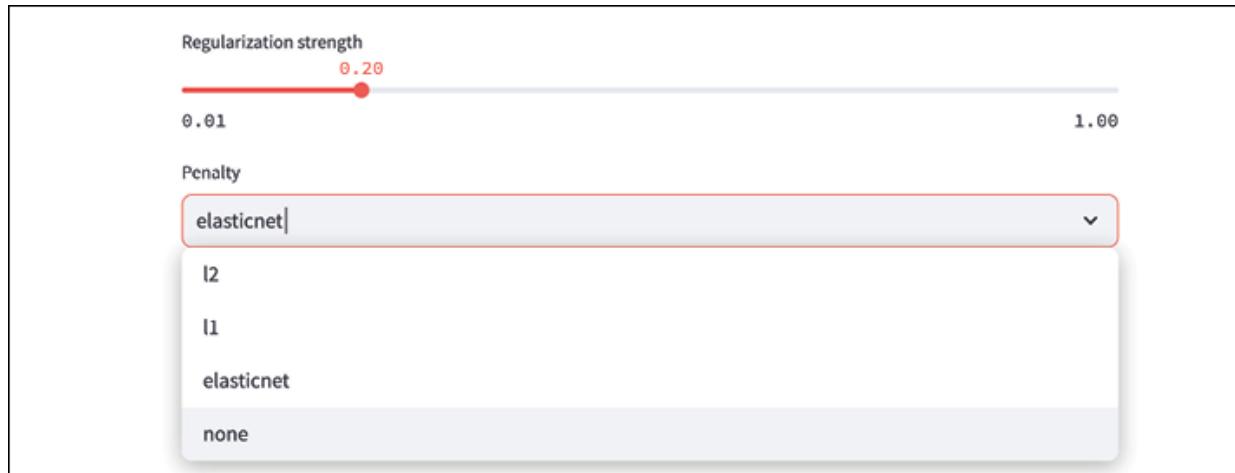


Figure 6.2: Streamlit application with parameter tuning sample

To maximize the exploratory value of dynamically adjusting model parameters, it is beneficial to provide immediate feedback on the effects of these adjustments. This feedback can take many forms, from simple text displaying accuracy or loss metrics to more complex visualizations such as decision boundaries or ROC curves for classification tasks.

For example, a simple approach could be to display the model's accuracy on the training dataset after retraining with the new parameters.

The following is the Streamlit application with the proposed simple approach:



Figure 6.3: Streamlit application with the variation of accuracy results as the parameters are modified

The following is the Streamlit application to show how the hyperparameter values selected by users can impact the model accuracy:

Logistic Regression Model

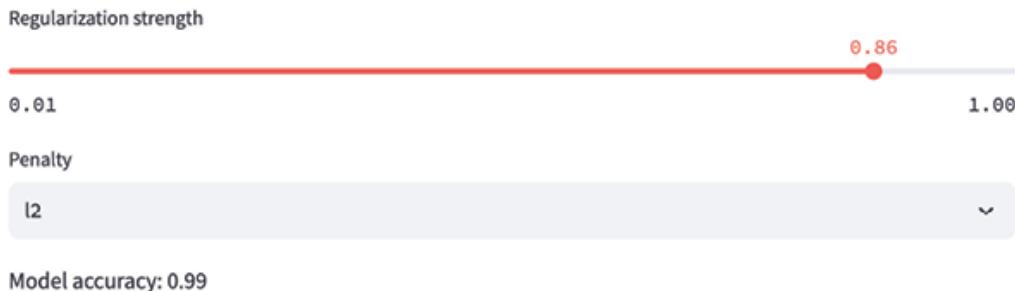


Figure 6.4: Demonstrating how the model accuracy does not change with an increase in regularization strength

Tips to improve the performance

When implementing dynamic parameter adjustment, consider the computational cost of retraining models and the user experience.

For computationally intensive models, you might opt for a **Recalculate** button using `st.button()` to trigger retraining and updating rather than doing it automatically upon every parameter change. This approach can prevent your app from becoming unresponsive or slow if the model takes a significant amount of time to train.

Hands-on: Integrating ML in Product Recommender

Now, let us enhance our Product Recommender application by integrating the machine learning model to see real-time change. As our focus is on integrating the model with Streamlit, we will not focus on the accuracy and complexity of the machine learning model. The following are the changes that we will proceed to make in the data application:

- Add a machine-learning `models` folder in our product structure to hold the model code and the compiled model dump to be used in the Streamlit file.
- Update `products_sales.csv` to add new features and increase the user purchase granularity.
- Update the `recommend_app.py`, update the user inputs, and start using the machine-learning model to predict the likelihood of purchase.

With these changes, the folder structure of our project looks as follows:

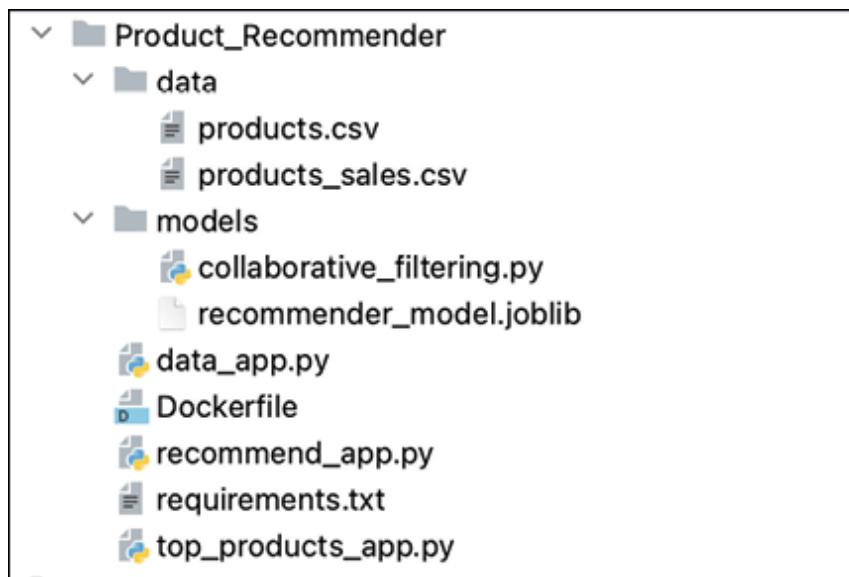


Figure 6.5: Updated Project structure with machine learning model integration

For our model, we will be taking a simplistic approach to using collaborative filtering to get the recommendation for our selected user and products in the selected category.

Collaborative filtering is a technique used in recommendation systems to predict a user's preferences by collecting preferences or taste information from many users (collaborating). The underlying assumption is that if user A has the same opinion as user B on an issue, A is more likely

to have B's opinion on a different issue than that of a random user.

Let us start by creating a simple user-item rating matrix and then applying a collaborative filtering algorithm. For simplicity and effectiveness, let us use the surprise library, a Python scikit for building and analyzing recommender systems that deal with explicit rating data.

In our dataset, we have user ratings on various products, which we would use to build the ratings matrix. But let us also diversify the ratings to get more range.

The updated dataset for product sales is as follows:

Category,Product,Product Name,Order Date,Price,User Age,Country,User Gender,User ID,Sales Channel,Rating,Purchase Frequency,Average Rating,Number of Ratings,Discount,Quantity

Fashion,Jeans,Jeans Standard,2023-12-17,144.76,32,Canada,Male,7912,Online,3,16,2.5,2,0.3
7,9

Travel,Travel Pillow,Travel Pillow Pro,2023-10-23,23,32,Canada,Male,7912,Online,5,17,3,2,0.32,9

Health & Wellness,Aromatherapy Diffuser,Aromatherapy Diffuser Standard,2023-11-30,217.08,32,Canada,Male,7912,In-Store,5,13,2.5,2,0.11,6

Health & Wellness,Protein Powder,Protein Powder Pro,2023-06-14,405.49,32,Canada,Male,7912,Third-Party Reseller,5,8,4,2,0.27,7

Travel,Travel Backpack,Travel Backpack Standard,2023-03-21,334.02,32,Canada,Male,7912,Third-Party Reseller,4,14,3,2,0.49,4

Health & Wellness, Protein Powder, Protein Powder Standard, 2023-07-25, 184.08, 32, Canada, Male, 7912, Online, 5, 15, 3.6666666667, 3, 0.35, 2

Gadgets, Portable Speaker, Portable Speaker Deluxe, 2024-02-26, 323.6, 32, Canada, Male, 7912, In-Store, 5, 1, 3, 1, 0.34, 1

Educational, Online Course, Online Course Pro, 2024-02-07, 361.78, 32, Canada, Male, 7912, Third-Party Reseller, 3, 15, 2.666666667, 3, 0.25, 8

Home & Kitchen, Decorative Vase, Decorative Vase Deluxe, 2023-12-19, 296.45, 55, Germany, Non-Binary, 9085, Online, 3, 11, 2.5, 2, 0.18, 3

Travel, Travel Backpack, Travel Backpack Standard, 2024-01-05, 343.44, 55, Germany, Non-Binary, 9085, Third-Party Reseller, 3, 14, 3, 2, 0.19, 10

Fashion, Jeans, Jeans Pro, 2024-02-10, 273.12, 55, Germany, Non-Binary, 9085, Third-Party Reseller, 5, 15, 3, 2, 0.17, 5

Fashion, Handbag, Handbag Pro, 2023-12-25, 97.54, 55, Germany, Non-Binary, 9085, Online, 3, 18, 2, 2, 0.15, 9

Home & Kitchen, Decorative Vase, Decorative Vase Standard, 2023-10-22, 168.84, 55, Germany, Non-Binary, 9085, Online, 3, 5, 2, 1, 0.46, 5

Fashion, Jacket, Jacket Standard, 2023-12-24, 271.03, 55, Germany, Non-Binary, 9085, Online, 5, 9, 5, 1, 0.33, 9

Home & Kitchen, Bedding Set, Bedding Set Pro, 2023-05-21, 359.34, 55, Germany, Non-Binary, 9085, Online, 3, 8, 1, 1, 0.32, 8

Educational, Educational Toy, Educational Toy Standard, 2023-10-18, 281.42, 55, Germany, Non-Binary, 9085, Online, 3, 1, 1, 1, 0, 1

Home & Kitchen, Blender, Blender Deluxe, 2023-11-18, 247.85, 55, Germany, Non-Binary, 9085, In-Store, 2, 10, 2, 1, 0.36, 10

Travel, Luggage, Luggage Standard, 2023-11-15, 221.22, 55, Germany, Non-Binary, 9085, Third-Party Reseller, 2, 9, 1, 2, 0.23, 1

Books, Self-Help, Self-Help Standard, 2023-06-08, 103.88, 52, Germany, Female, 8579, In-Store, 2, 12, 1.5, 2, 0.07, 3

Health & Wellness, Aromatherapy Diffuser, Aromatherapy Diffuser Pro, 2023-04-11, 34.79, 52, USA, Male, 7658, Online, 4, 4, 4, 1, 0.48, 4

Health & Wellness, Skincare Product, Skincare Product Standard, 2023-05-07, 195.7, 42, UK, Female, 7930, Online, 5, 8, 5, 1, 0.46, 8

Travel, World Map, World Map Standard, 2023-03-26, 293.8, 52, USA, Male, 8811, Online, 5, 15, 2.3333333333, 3, 0.37, 1

Home & Kitchen, Coffee Maker, Coffee Maker Standard, 2023-03-30, 301.57, 61, Canada, Male, 1651, In-Store, 5, 10, 5, 1, 0.2, 10

Fashion, Sneakers, Sneakers Deluxe, 2023-05-22, 338.17, 38, Canada, Non-Binary, 5915, Online, 5, 3, 4, 1, 0.13, 3

Educational, Science Kit, Science Kit Pro, 2023-10-08, 498.09, 22, France, Male, 6718, In-Store, 1, 5, 1, 1, 0.11, 5

Books,Non-Fiction,Non-Fiction Deluxe,2024-01-21,276.18,37,France,Male,5905,In-Store,3,9,1,1,0.49,9

Educational,Book,Book Pro,2023-10-05,86.17,36,France,Male,5325,In-Store,5,14,5,2,0.11,6

Health & Wellness,Yoga Mat,Yoga Mat Standard,2023-11-05,217.55,36,France,Male,5325,Third-Party Reseller,3,3,2,1,0,3

Fashion,Sunglasses,Sunglasses Pro,2023-03-07,64.82,36,France,Male,5325,Online,5,6,4,1,0.27,6

Books,Biography,Biography Standard,2024-02-16,185.33,36,France,Male,5325,Online,5,18,3.33333333,3,0.33,6

Health & Wellness,Fitness Band,Fitness Band Standard,2024-02-09,212.02,36,France,Male,5325,Third-Party Reseller,5,7,3,1,0.46,7

Educational,Science Kit,Science Kit Standard,2024-01-31,240.64,36,France,Male,5325,In-Store,2,4,2,1,0.17,4

Books,Self-Help,Self-Help Standard,2023-12-29,146.77,36,France,Male,5325,Online,2,12,1.5,2,0.2,9

Home & Kitchen,Smart Home Device,Smart Home Device Standard,2023-04-15,302.32,36,France,Male,5325,Third-Party Reseller,2,13,1,2,0.39,7

Travel,World Map,World Map Standard,2023-12-04,223.67,36,France,Male,5325,Third-Party Reseller,2,15,2.33333333,3,0.34,4

Books, Historical Fiction, Historical Fiction Standard, 2023-12-08, 451.02, 50, UK, Female, 5844, Online, 5, 8, 5, 2, 0.33, 6

Gadgets, Portable Speaker, Portable Speaker Pro, 2023-12-19, 452.37, 57, France, Male, 5978, In-Store, 5, 9, 3, 1, 0.15, 9

Health & Wellness, Haircare Kit, Haircare Kit Pro, 2023-11-13, 167.9, 46, Germany, Female, 3780, Third-Party Reseller, 2, 9, 1, 1, 0.34, 9

Fashion, Sneakers, Sneakers Standard, 2024-01-16, 476.48, 33, Australia, Male, 1400, In-Store, 3, 4, 1, 1, 0.16, 4

Home & Kitchen, Coffee Maker, Coffee Maker Deluxe, 2023-08-12, 444.21, 54, Canada, Female, 7547, Third-Party Reseller, 2, 11, 1.5, 2, 0.36, 7

Fashion, T-shirt, T-shirt Standard, 2023-12-31, 218.32, 29, Australia, Non-Binary, 5082, Online, 3, 10, 1.5, 2, 0.26, 1

Health & Wellness, Skincare Product, Skincare Product Pro, 2023-09-26, 494.07, 29, Australia, Non-Binary, 5082, Online, 4, 8, 4, 1, 0.48, 8

Fashion, Jacket, Jacket Deluxe, 2023-06-12, 378.37, 29, Australia, Non-Binary, 5082, Online, 4, 11, 3.5, 2, 0.38, 9

Fashion, Handbag, Handbag Standard, 2023-06-17, 174.57, 29, Australia, Non-Binary, 5082, In-Store, 4, 4, 2, 1, 0.07, 4

Gadgets, E-reader, E-reader Pro, 2023-09-01, 213.73, 29, Australia, Non-Binary, 5082, In-Store, 5, 10, 5, 1, 0.19, 10

Educational, Educational Toy, Educational Toy Deluxe, 2023-12-21, 201.09, 29, Australia, Non-Binary, 5082, In-Store, 5, 15, 4, 2, 0.37, 9

Gadgets, Fitness Tracker, Fitness Tracker Standard, 2023-08-02, 174.7, 29, Australia, Non-Binary, 5082, Third-Party Reseller, 3, 8, 1, 1, 0.36, 8

Home & Kitchen, Bedding Set, Bedding Set Standard, 2023-04-05, 324.95, 29, Australia, Non-Binary, 5082, In-Store, 2, 5, 2, 1, 0.09, 5

Books, Self-Help, Self-Help Deluxe, 2023-06-20, 378.48, 44, UK, Male, 9837, Online, 2, 13, 3, 2, 0.42, 8

Fashion, Jeans, Jeans Pro, 2023-03-19, 379.57, 33, Canada, Male, 4908, Third-Party Reseller, 4, 15, 3, 2, 0.42, 10

Educational, Math Workbook, Math Workbook Standard, 2023-08-31, 352.74, 67, UK, Non-Binary, 2876, Online, 3, 10, 2, 1, 0.08, 10

Home & Kitchen, Coffee Maker, Coffee Maker Deluxe, 2023-10-13, 341.9, 42, Germany, Male, 4978, Third-Party Reseller, 1, 11, 1.5, 2, 0.47, 4

Home & Kitchen, Coffee Maker, Coffee Maker Pro, 2024-01-02, 30.12, 52, Germany, Non-Binary, 6590, In-Store, 1, 2, 1, 1, 0.33, 2

Educational, Science Kit, Science Kit Deluxe, 2023-11-06, 446.87, 47, Australia, Non-Binary, 9548, In-Store, 4, 3, 3, 1, 0.08, 3

Gadgets, Tablet, Tablet Deluxe, 2024-01-28, 130.29, 47, Australia, Non-Binary, 9548, In-Store, 5, 20, 4, 3, 0.44, 4

Health & Wellness, Protein Powder, Protein Powder Standard, 2023-09-09, 113.94, 47, Australia, Non-Binary, 9548, Third-Party Reseller, 5, 15, 3.666666667, 3, 0.17, 10

Travel, Luggage, Luggage Standard, 2023-12-08, 209.15, 53, Canada, Female, 1282, Online, 2, 9, 1, 2, 0.35, 8

Gadgets, Laptop, Laptop Deluxe, 2023-11-17, 294.96, 48, France, Non-Binary, 6419, In-Store, 4, 7, 3, 1, 0.27, 7

Books, Biography, Biography Standard, 2023-04-20, 363.43, 38, USA, Male, 4545, Third-Party Reseller, 4, 18, 3.333333333, 3, 0.37, 3

Home & Kitchen, Decorative Vase, Decorative Vase Pro, 2023-06-22, 490.45, 30, UK, Female, 1231, In-Store, 5, 3, 5, 1, 0.11, 3

Educational, Book, Book Pro, 2023-06-25, 351.73, 26, Australia, Male, 1200, In-Store, 5, 14, 5, 2, 0.43, 8

Gadgets, Fitness Tracker, Fitness Tracker Deluxe, 2023-05-14, 483.09, 26, Australia, Male, 1200, Online, 2, 8, 3, 2, 0.06, 2

Travel, Portable Charger, Portable Charger Deluxe, 2023-04-06, 497.02, 26, Australia, Male, 1200, In-Store, 5, 10, 5, 1, 0.38, 10

Gadgets, Tablet, Tablet Pro, 2023-08-17, 159.64, 26, Australia, Male, 1200, Online, 5, 2, 3, 1, 0.3, 2

Travel, Travel Pillow, Travel Pillow Pro, 2023-04-02, 86.7, 26, Australia, Male, 1200, Third-Party

Reseller,3,17,3,2,0.34,8

Books, Historical Fiction, Historical Fiction

Deluxe,2023-03-

31,346.82,26,Australia,Male,1200,Third-Party
Reseller,5,5,5,2,0.29,4

Travel, World Map, World Map Standard, 2023-03-

31,39.22,26,Australia,Male,1200,Third-Party

Reseller,1,15,2.333333333,3,0.3,10

Books, Historical Fiction, Historical Fiction

Deluxe,2023-06-

17,255.23,26,Australia,Male,1200,Online,5,5,5,2,0.0
4,1

Educational, Online Course, Online Course Pro, 2024-

01-22,431.65,26,Australia,Male,1200,Third-Party

Reseller,3,15,2.666666667,3,0.5,5

Fashion, Jacket, Jacket Deluxe, 2023-05-

13,498.5,26,Australia,Male,1200,In-

Store,5,11,3.5,2,0.41,2

Fashion, Jeans, Jeans Standard, 2023-07-

10,211.71,26,Australia,Male,1200,In-

Store,5,16,2.5,2,0.33,7

Books, Biography, Biography Standard, 2023-12-

29,208.05,50,USA,Female,7770,Third-Party

Reseller,1,18,3.333333333,3,0.33,9

Books, Science Fiction, Science Fiction Deluxe, 2023-

11-

02,226.6,21,Canada,Female,8178,Online,5,3,4,1,0.13,

3

Fashion, Watch, Watch Pro, 2023-05-

04,162.27,18,France,Male,8560,In-

Store,2,4,1,1,0.2,4

Fashion,Handbag,Handbag Pro,2023-12-03,143.98,48,Germany,Male,7284,Online,3,18,2,2,0.12,9

Books,Self-Help,Self-Help Deluxe,2023-07-14,467.78,50,UK,Non-Binary,5891,Online,5,13,3,2,0.41,5

Gadgets,Fitness Tracker,Fitness Tracker Deluxe,2023-07-11,280.44,52,Germany,Male,4912,Online,5,8,3,2,0.23,6

Health & Wellness,Aromatherapy Diffuser,Aromatherapy Diffuser Deluxe,2023-03-21,165.91,48,Australia,Female,3492,Online,4,7,4,1,0.46,7

Home & Kitchen,Decorative Vase,Decorative Vase Deluxe,2024-02-04,59.18,46,USA,Male,3361,Online,5,11,2.5,2,0.26,8

Gadgets,Tablet,Tablet Deluxe,2023-11-01,349.43,59,USA,Female,6188,Third-Party Reseller,5,20,4,3,0.43,9

Educational,Online Course,Online Course Pro,2023-11-24,461.26,32,Australia,Non-Binary,2646,Third-Party Reseller,4,15,2.666666667,3,0.05,2

Books,Fiction,Fiction Standard,2023-07-08,400.11,36,UK,Male,9702,Online,4,3,4,1,0.19,3

Travel,Luggage,Luggage Pro,2024-02-05,197.45,61,Canada,Male,4804,In-Store,5,10,4,1,0.25,10

Books,Historical Fiction,Historical Fiction Standard,2024-02-

16,404.27,36,Australia,Female,6661,Third-Party Reseller,5,8,5,2,0.27,2

Gadgets,Tablet,Tablet Deluxe,2024-01-26,158.04,42,Germany,Non-Binary,6881,Third-Party Reseller,5,20,4,3,0.05,7

Books,Fiction,Fiction Pro,2023-08-04,144.5,54,Australia,Non-Binary,9138,Third-Party Reseller,4,1,3,1,0.39,1

Health & Wellness,Protein Powder,Protein Powder Standard,2023-11-06,103.16,34,Australia,Non-Binary,2591,In-Store,1,15,3.666666667,3,0.38,3

Educational,Math Workbook,Math Workbook Deluxe,2023-12-13,74.39,36,UK,Male,3165,Third-Party Reseller,5,3,5,1,0.37,3

Home & Kitchen,Smart Home Device,Smart Home Device Standard,2023-06-11,457.88,48,UK,Male,8673,Online,2,13,1,2,0.37,6

Health & Wellness,Vitamins,Vitamins Pro,2023-11-21,490.17,69,France,Male,5332,In-Store,5,7,5,1,0.18,7

Books,Historical Fiction,Historical Fiction Pro,2023-11-20,138.46,69,France,Male,5332,Third-Party Reseller,4,2,4,1,0.47,2

Home & Kitchen,Smart Home Device,Smart Home Device Pro,2023-05-29,327.35,69,France,Male,5332,Online,1,5,1,1,0.45,5

Educational,Educational Toy,Educational Toy Deluxe,2023-04-18,57.62,69,France,Male,5332,In-Store,5,15,4,2,0.19,6

Fashion,Jeans,Jeans Deluxe,2023-10-29,209.98,69,France,Male,5332,Online,5,4,2,1,0.35,4

Fashion,Sneakers,Sneakers Pro,2024-02-17,489.14,40,USA,Male,8450,Third-Party Reseller,2,4,1,1,0.2,4

Fashion,T-shirt,T-shirt Standard,2024-01-03,173.42,26,France,Male,4538,Online,3,10,1.5,2,0.1,9

Health & Wellness,Haircare Kit,Haircare Kit Standard,2023-06-19,430.62,27,Canada,Male,1422,In-Store,4,3,2,1,0.27,3

Health & Wellness,Aromatherapy Diffuser,Aromatherapy Diffuser Standard,2023-10-18,168.65,29,UK,Non-Binary,2862,In-Store,2,13,2.5,2,0.03,7

Health & Wellness,Protein Powder,Protein Powder Pro,2023-10-20,63.44,50,Canada,Non-Binary,9543,Third-Party Reseller,3,8,4,2,0.47,1

Educational,Book,Book Deluxe,2024-01-27,206.79,61,USA,Male,2323,Online,4,8,3,1,0.25,8

Next, we move to the collaborative filtering model as follows:

```
from surprise import Dataset, Reader, SVD
from surprise.model_selection import GridSearchCV
import joblib
import pandas as pd

df = pd.read_csv('../data/products_sales.csv')
reader = Reader(rating_scale=(1, 5))
```

```
data = Dataset.load_from_df(df[['User ID',
 'Product', 'Rating']], reader)

param_grid = {

    'n_epochs': [5, 10, 20], # Number of
    iterations of the SGD procedure

    'lr_all': [0.002, 0.005, 0.01], # Learning
    rate for all parameters

    'reg_all': [0.02, 0.1, 0.4] # Regularization
    term for all parameters

}

gs = GridSearchCV(SVD, param_grid, measures=
['rmse', 'mae'], cv=3)

gs.fit(data)

print(gs.best_params['rmse'])

algo = SVD(n_epochs=gs.best_params['rmse'],
            ['n_epochs'], lr_all=gs.best_params['rmse'],
            ['lr_all'], reg_all=gs.best_params['rmse'],
            ['reg_all'])

trainset = data.build_full_trainset()

algo.fit(trainset)
```

```
joblib_file = "recommender_model.joblib"
joblib.dump(algo, joblib_file)
```

The next step is to run the machine learning model and get the dump, which can be used to deploy in our Streamlit.

```
python collaborative_filtering.py
```

Once you run the model successfully, you can see the model dump in the root directory, as shown in the following figure:

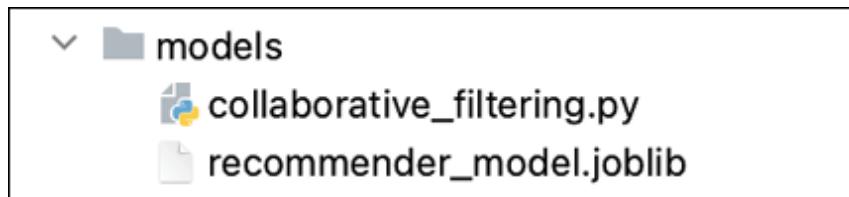


Figure 6.6: Dump for the machine learning model generated using joblib library

The next step is to update our Streamlit code, to take the user input for the user_id and interested category that the user is looking to buy from. Then, our model can propose the top five products the product user is most likely to buy from the interested category. We will also add basic validations to ensure that the user input value is as expected. The code is as follows:

```
import streamlit as st
import random
import pandas as pd
import matplotlib.pyplot as plt
from joblib import load

st.set_option('deprecation.showPyplotGlobalUse',
False)

@st.cache_data
def load_data(file_path):
    return pd.read_csv(file_path)
```

```
product_sales =
load_data('data/products_sales.csv')

def predict_and_rank(user_id, category,
product_sales):

    if user_id not in product_sales['User
ID'].unique():
        return f"No User"

    model_pipeline =
load('models/recommender_model.joblib')

    all_products_info = product_sales[['Category',
'Product Name']].drop_duplicates()

    filtered_products =
all_products_info[all_products_info['Category'] ==
category]['Product Name'].unique()

    predictions = []
    for product_name in filtered_products:
        prediction =
model_pipeline.predict(str(user_id),
str(product_name))

        predictions.append((product_name,
prediction.est))

    predictions_df = pd.DataFrame(predictions,
columns=['Product Name', 'Likelihood'])
```

```
    predictions_df.sort_values(by='Likelihood',
                                 ascending=False, inplace=True)

    return predictions_df.head(5)

def plot_likelihood_of_purchase(df_recommendations):
    fig, ax = plt.subplots(figsize=(6, 4))

    df_sorted =
df_recommendations.sort_values('Likelihood',
                                 ascending=True)

    ax.barh(df_sorted['Product Name'],
            df_sorted['Likelihood'], color='skyblue')

    ax.set_xlabel('Likelihood of Purchase (%)')
    ax.set_title('Likelihood of Purchase for
Recommended Products')

def recommend_main():
    with st.form("recommendation_form"):

        st.title('Product Recommender')

        user_id = st.number_input("Enter User ID",
                                 value=0, step=1)

        interested_category = st.selectbox("Select
Interested Category",
                                             options=
[''] + list(product_sales['Category'].unique()))
```

```
    submit_button =
st.form_submit_button(label='Get Recommendations')

    if submit_button and interested_category:
        ranked_products =
predict_and_rank(user_id, interested_category,
product_sales)

    with st.container():
        st.title('Recommendations')
        if isinstance(ranked_products,
str):
            st.warning("User ID not found
in the database.")
        else:
            with st.expander(label="" ,
expanded=True):
                col1, col2 = st.columns(2)
# You can adjust the ratio as needed for your UI
                with col1:
st.dataframe(ranked_products,
column_config={

'Likelihood of Purchase':
st.column_config.NumberColumn(
```

```
'Likelihood' ,  
  
help="How likely is user to buy this **product**" ,  
  
min_value=0 ,  
  
max_value=100 ,  
  
step=1 ,  
  
format="%d %%" ,  
 )} ,  
  
hide_index=True ,  
 )  
  
with col2:  
    fig =  
plot_likelihood_of_purchase(ranked_products)  
    st.pyplot(fig)  
  
elif submit_button:  
    st.write('Please select a category and  
at least one product to see recommendations.')  
The following is the updated Streamlit page for the Product  
Recommendation Tab:
```

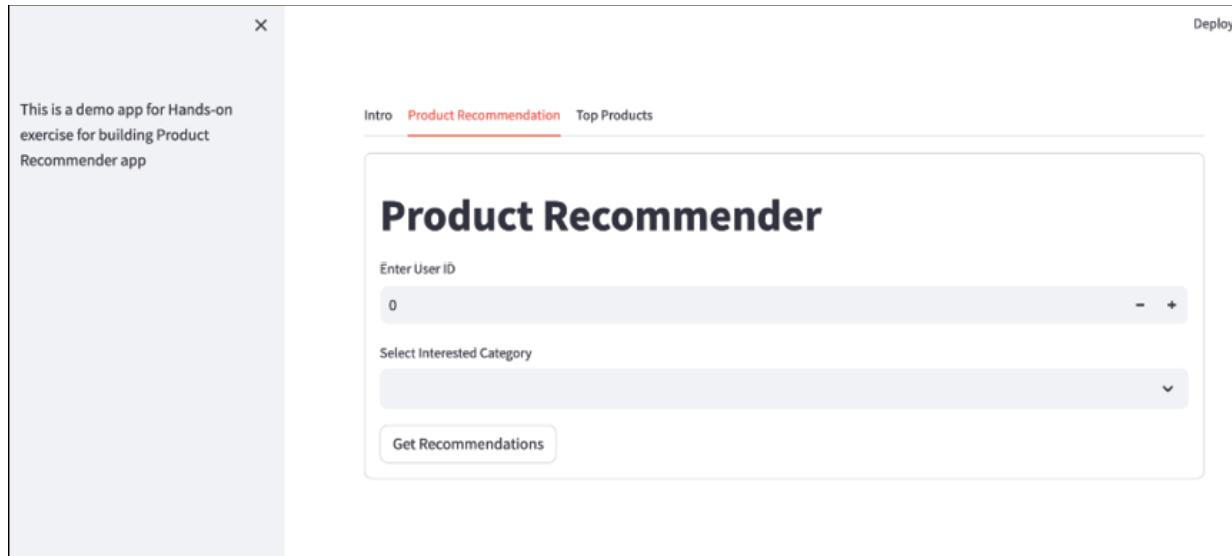


Figure 6.7: Updated Product recommender screen with user inputs

The following screen shows the validation checks with a warning if the expected input value is not provided by the user:

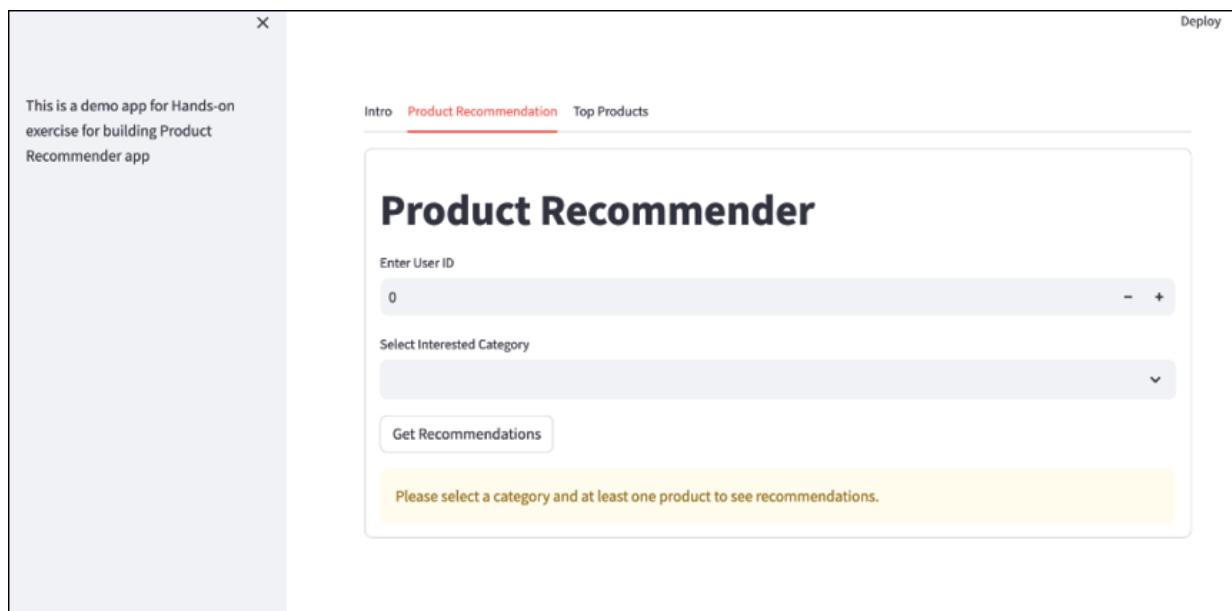


Figure 6.8: Validation screen with a warning when the category is not selected

The following screen shows the input drop-down for the user selection:

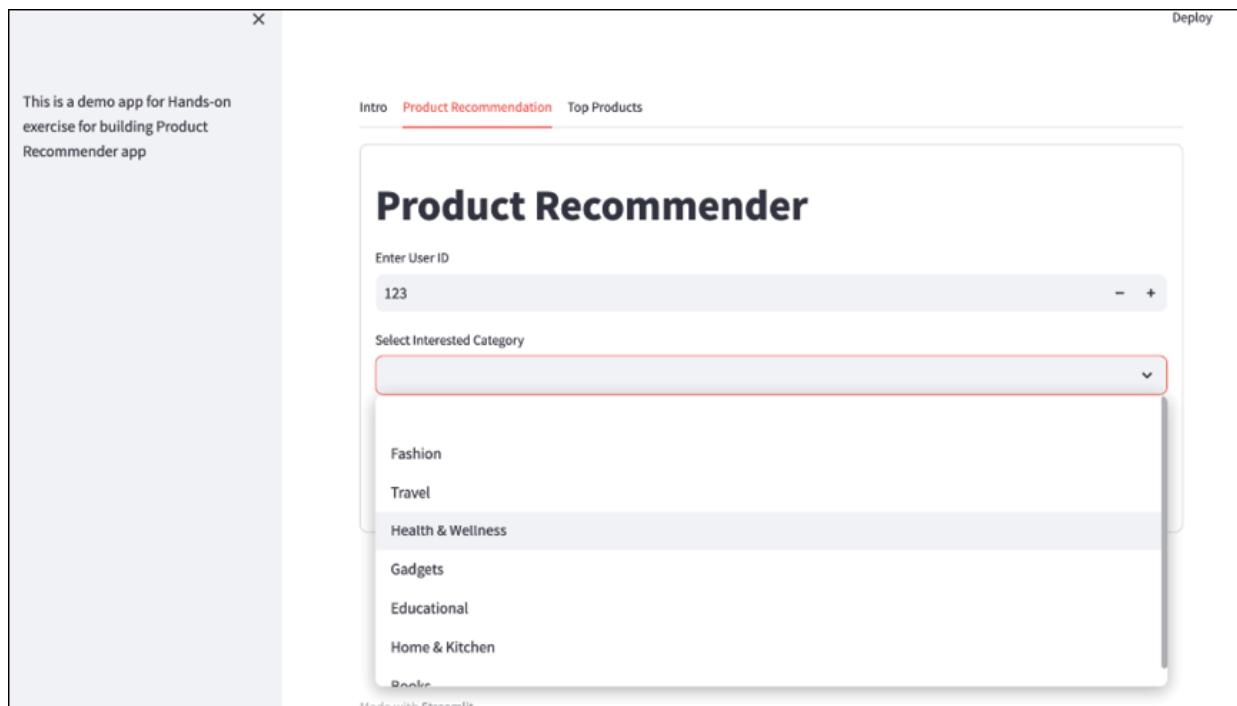


Figure 6.9: Entering the user ID and selecting the category from the drop-down

The following screen shows the validation checks with a warning if the expected user ID does not exist in the dataset:

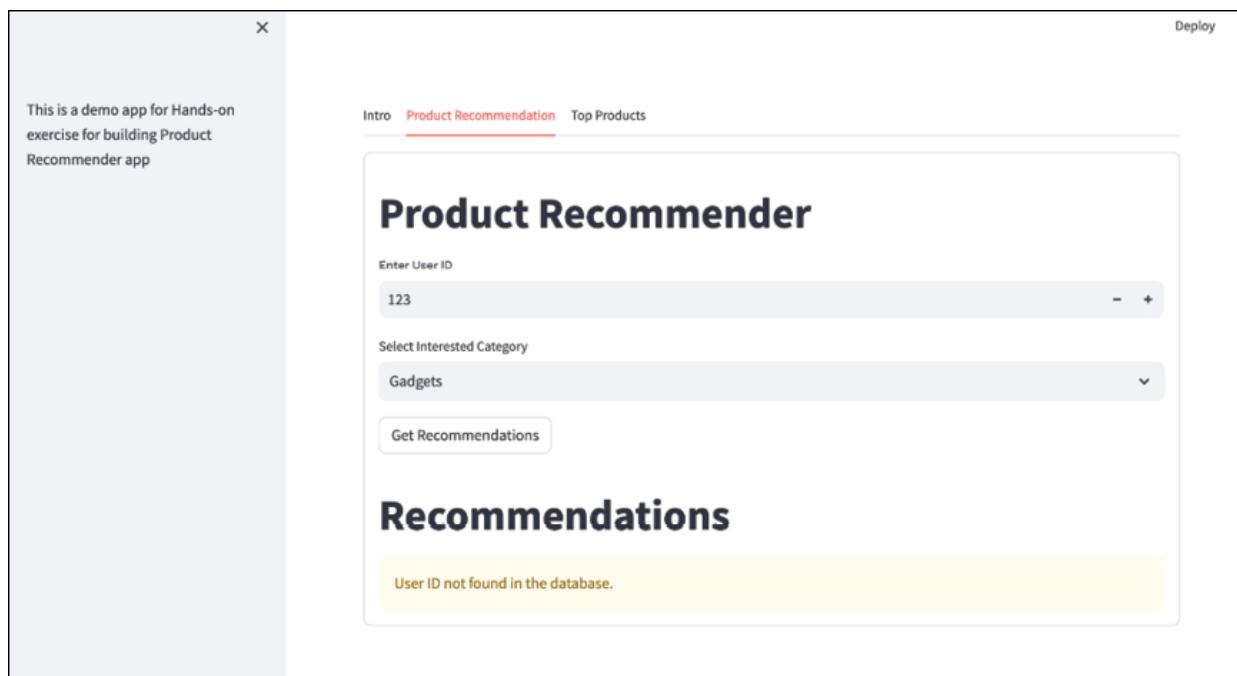


Figure 6.10: Validation response when user ID does not match the user IDs present in the product_sales.csv

Now, let us enter the correct values for our user ID and category. The result is shown in the following figure:

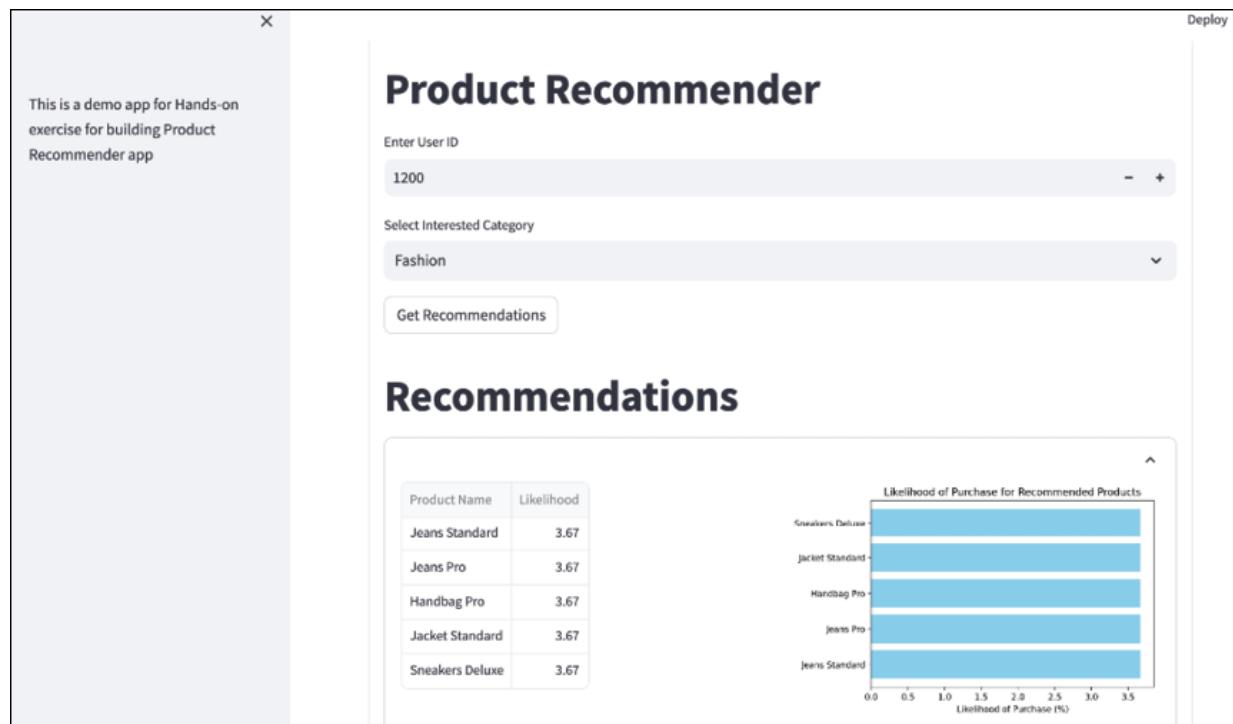


Figure 6.11: Streamlit data application screen with the recommendation of products based on the user ID and the interested category

As per our data diversity and model accuracy, the top five products in the fashion category for users with `user_id` 1200 are shown in *Figure 6.12*.

Furthermore, to build a more diversified result, the following things must be considered:

- Model enhancements:
 - More complex models, such as SVD++ or NMF, might capture nuances better.
 - Hybrid models that combine collaborative and content-based filtering.
- Hyperparameter tuning and validation:

- Tune the hyperparameters of the model.
- Validate the model's performance using appropriate metrics.
- Add temporal features in the `product_sales` dataset:
 - Seasonality effects (for example, products related to holidays or events).
 - Trends or fads that may influence short-term popularity.

Conclusion

In this chapter, we learned how to integrate machine learning models with Streamlit. Using Streamlit can provide the front-end to machine learning models, which can be very helpful if the models are to be presented and handled by non-technical stakeholders and partners. Streamlit can also help highlight the complexity of the models and the way each parameter might impact the accuracy and precision score.

In the next chapter, we will discuss the advanced concepts of Streamlit, which will help to add more sophistication to the data application. Streamlit offers various features like session tracking and caching, which can elevate your data application.

Exercises

1. **What is the first step in integrating a machine-learning model with Streamlit?**
 - a. Designing a user-friendly UI
 - b. Training the model on a dataset
 - c. Deploying the model to a cloud service

- d. Writing an API for the model
- 2. Which of the following is essential for displaying model predictions in a Streamlit app?**
- a. A high bandwidth internet connection
 - b. Converting the model to a specific format
 - c. Using Streamlit's built-in functions like `st.write()` or `st.dataframe()`
 - d. A subscription to Streamlit sharing
- 3. How can you save a trained ML model for use in a Streamlit app?**
- a. Using the pickle or joblib libraries to serialize the model
 - b. Converting the model to JSON format
 - c. Printing the model's parameters to a text file
 - d. Manually coding the model's decision rules
- 4. Which Streamlit function can be used to display a plot showing model performance metrics?**
- a. `st.plot()`
 - b. `st.pyplot()`
 - c. `st.graph()`
 - d. `st.chart()`
- 5. How can model predictions be updated in real-time within a Streamlit app?**
- a. By refreshing the webpage
 - b. Through callback functions linked to input widgets

- c. By scheduling a cron job
 - d. Using AJAX calls within the app
6. **Which component would you use to allow users to upload a dataset for prediction in Streamlit?**
- a. st.file_uploader()
 - b. st.dataset_uploader()
 - c. st.upload()
 - d. st.data_uploader()
7. **How can you ensure that a Streamlit app efficiently uses a trained ML model?**
- a. By retraining the model each time the app is used
 - b. Using @st.cache to load the model only once
 - c. Limiting the app's access to the internet
 - d. Storing the model in a database

Answers

- 1.** b
- 2.** c
- 3.** a
- 4.** b
- 5.** b
- 6.** a
- 7.** b

Join our book's Discord space

Join the book's Discord Workspace for Latest updates, Offers, Tech happenings around the world, New Release and Sessions with the Authors:

[https://discord\(bpbonline\).com](https://discord(bpbonline).com)



CHAPTER 7

Advanced Streamlit Concepts

Introduction

In this chapter, we will explore a range of features and techniques designed to enhance the functionality, security, efficiency, and user experience of your Streamlit applications. By the end of this chapter, you will have a comprehensive understanding of Streamlit's advanced features and best practices, enabling you to build secure, efficient, and user-friendly web applications.

Structure

The chapter discusses the following topics:

- Streamlit advanced topics
- Callbacks
- Secret management
- Working with timezones
- Building mutating charts with Streamlit

- Connecting to databases with Streamlit
- App testing with Streamlit
- Design patterns pitfalls to avoid for Streamlit

Objectives

This chapter aims to cover the advanced features of Streamlit, enabling us to build a great app. Topics covered in this chapter range from the streamlet backend working model to importing or using third-party applications and libraries to enhance the data application. It addresses the need for secure, efficient, and user-friendly web applications, focusing on advanced features, performance optimization, and best practices.

Streamlit advanced topics

Streamlit has revolutionized data application development by simplifying the process. However, creating applications that are not only functional but also efficient, secure, and user-friendly requires a deep dive into Streamlit's advanced features and best practices. Let us look at these advanced concepts, providing readers with the tools to take their Streamlit applications to the next level.

Callbacks

Callbacks enable developers to create highly interactive and dynamic web applications by reacting to user inputs or actions. Streamlit's approach to callbacks allows for direct association of functions with user interface elements, such as buttons, sliders, or text inputs. This interaction design enables developers to design apps that respond immediately to user inputs, making applications more engaging and intuitive.

Let us try to implement callbacks with buttons. The code is as follows:

```
import streamlit as st

# Define the callback function

def toggle_message():

    if 'show_message' not in st.session_state:
        st.session_state.show_message = False

        st.session_state.show_message = not
st.session_state.show_message

# Create a button and associate the callback

st.button("Toggle Message",
on_click=toggle_message)

# Conditional display based on the callback's
effect on session state

if st.session_state.get('show_message', False):
    st.write("Hello, Streamlit!")
```

The following Streamlit application demonstrates the first state of the application:



Figure 7.1: First run screen of the application

When you press the toggle button, the `toggle_message` function is being called, as shown in the following figure:



Figure 7.2: Second run screen of the application

Streamlit also supports complex interactions, such as dynamically updating content based on form submission. The following example shows how to use a form with a callback to update the app's state based on user input:

```
import streamlit as st

# Callback to process form submission
def process_form():
    user_name = st.session_state.user_name
    user_age = st.session_state.user_age
    st.write(f"Name: {user_name}, Age: {user_age}")

# Define a form with inputs and a submit button
with st.form("user_form", clear_on_submit=True):
    st.text_input("Enter your name",
key="user_name")
    st.number_input("Enter your age", min_value=0,
max_value=100, step=1, key="user_age")
```

```
submit_button = st.form_submit_button("Submit",  
on_click=process_form)
```

The following Streamlit application is the first state of the preceding code:

A screenshot of a Streamlit application interface. At the top right, there are 'Deploy' and three-dot menu icons. The main area contains a form with two text input fields and a submit button. The first field is labeled 'Enter your name' and contains the value 'superman'. The second field is labeled 'Enter your age' and contains the value '28', with a minus and plus button for adjustment. A red-bordered 'Submit' button is at the bottom of the form.

Figure 7.3: Enter the name and the age in the form

Once you click the **Submit** button, the form callback function is called, as shown in the following figure:

A screenshot of the same Streamlit application after the 'Submit' button was clicked. At the top right, there are 'Deploy' and three-dot menu icons. The main area shows the submitted data: 'Name: superman, Age: 28'. Below this, the form fields are empty again, showing the initial state before the submission. The 'Submit' button is now greyed out.

Figure 7.4: When using a form, the page is not refreshed till the callback function is called

Callbacks are a powerful feature in Streamlit that allows developers to build interactive and dynamic applications. By using callbacks, one can significantly enhance the user

experience of your Streamlit apps, making them engaging and responsive to user inputs.

The best practices to be kept in mind when using callbacks are as follows:

- **Use session state judiciously:** Leverage session state to manage app state across callbacks, but be mindful of its scope and lifecycle.
- **Keep callbacks lightweight:** Since callbacks can be triggered frequently, ensure they perform efficiently to maintain a responsive user experience.
- **Debugging:** Utilize `st.write(st.session_state)` to debug and understand how the session state changes with user interactions.

Secret management

At times, your data application will interact with external APIs, databases, or other services that require authentication keys to be used by your application. In the context of this chapter, secret management refers to securely handling sensitive information such as API keys, database credentials, and other secrets within Streamlit applications. Streamlit provides built-in capabilities for secret management, ensuring that developers can maintain the security of their applications without compromising sensitive data.

Streamlit provides a streamlined approach to managing secrets via the `.streamlit/secrets.toml` file in your project directory. Secrets stored in this file are automatically encrypted when you deploy your app using Streamlit Sharing or Streamlit for Teams. For self-deployment, you can either use the external secret managers or authentication libraries.

Let us look at the usage of secret management recommended by Streamlit. The folder structure for this

demo is as follows:



Figure 7.5: Folder structure of the streamlit app with secret management

```
# .streamlit/secrets.toml
weather_api_key = "YOUR_API_KEY"

For this demonstration, you can get your API key from
https://www.weatherapi.com/. The following code
demonstrates how to use secrets.toml to store the API key
secret:

import streamlit as st
import requests

# Access secrets
api_key = st.secrets["weather_api_key"]

# Use the secret to make an API request
def get_weather_data(api_key, city):
    url =
    f"https://api.weatherapi.com/v1/current.json?key={api_key}&q={city}&aqi=no"
    response = requests.get(url)
    return response.json()
```

```
city = st.text_input("Enter a city name",  
value="London")  
  
if city:  
    weather_data = get_weather_data(api_key, city)  
    st.write(weather_data)
```

The following is the build Streamlit application from the preceding code:

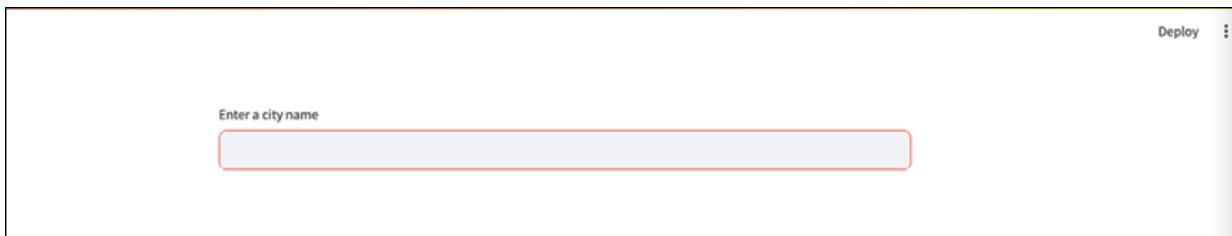


Figure 7.6: Enter the city name to get the weather information

The screenshot shows a Streamlit application interface. At the top right, there are buttons for 'Deploy' and three vertical dots. Below the header is a search bar containing the text 'kuala lumpur'. The main area displays a JSON response from a weather API. The JSON structure includes a location object with details like name, region, country, coordinates, and time zones. It also includes a current weather object with temperature in Celsius and Fahrenheit, a day indicator, a condition object with text ('Partly cloudy'), an icon URL, and a code ('1003'). A 'wind_mph' field is also present.

```
Deploy :  
kuala lumpur  
{"location": {"name": "Kuala Lumpur", "region": "Kuala Lumpur", "country": "Malaysia", "lat": 3.17, "lon": 101.7, "tz_id": "Asia/Kuala_Lumpur", "localtime_epoch": 1711758072, "localtime": "2024-03-30 08:21"}, "current": {"last_updated_epoch": 1711757700, "last_updated": "2024-03-30 08:15", "temp_c": 25, "temp_f": 77, "is_day": 1, "condition": {"text": "Partly cloudy", "icon": "//cdn.weatherapi.com/weather/64x64/day/116.png", "code": 1003}, "wind_mph": 2.2}}
```

Figure 7.7: Weather information returned from the weather API

Best practices for Secret Management using Streamlit

The following are some best practices to be mindful of when using secret management with Streamlit:

- **Do not commit secrets to version control:** Always exclude the `.streamlit/secrets.toml` file from your version control system (for example., by adding it to your `.gitignore` file) when working locally.
- **Use environment-specific secrets:** For different deployment environments (development, staging, production), you can configure environment-specific

secrets through the Streamlit sharing or Streamlit for Teams settings panel.

- **Limit access to secrets:** Only store necessary secrets and limit access to the `secrets.toml` file to authorized personnel.

Working with timezones

When building a data application that deals with global data, timezones could impose a certain challenge. Most data warehouses and cloud services provide the functionality to handle time zones by default, but if you are ingesting or using data from multiple sources, the strategy to handle time zones might vary from system to system. Thus, it is important to ensure that your application correctly manages and presents time-sensitive data across different geographical regions.

Streamlit does not add any timezone transformation on its own. By default, it displays the same result as extracted from the backend, so it is the responsibility of the developer to ensure that the application handles the time zones correctly. One way to ensure this is to build support for time zones using Python libraries.

Best practices when using different timezones in application

The following are some best practices to be mindful about:

- **Always store time in UTC:** When working with databases or persistent storage, always store datetime values in UTC. Convert to the user's local timezone only when displaying times.
- **Be explicit about timezone:** Make it clear to the user which timezone a given time refers to, especially if your application handles users from multiple regions.

- **Use library support:** Leverage Python's `datetime` and `pytz` libraries for timezone conversions. Consider using `dateutil` to parse dates with timezones.

Building mutating charts with Streamlit

Mutating charts enhance the interactivity and effectiveness of data visualization in web applications. By providing users with the ability to explore data dynamically, these charts serve as powerful tools for analysis, monitoring, and storytelling.

In Streamlit, mutating charts are typically implemented by rerunning the script or parts of it in response to user interactions. Streamlit's design automatically updates the app's output whenever the user interacts with widgets that are linked to data visualizations. This reactive programming model simplifies the process of creating interactive and dynamic applications.

For instance, when a user moves a slider that controls the range of data displayed in a plot, Streamlit reruns the script, which then updates the plot based on the new slider value. This process is seamless from the user's perspective, providing a smooth and interactive experience.

In this example, let us try a simple mutating chart:

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

# Generate a simple plot function
def generate_plot(sin_frequency=1.0):
    plt.figure(figsize=(10, 4))
```

```

t = np.linspace(0, 1, 500)
plt.plot(t, np.sin(2 * np.pi * sin_frequency * t))
plt.title(f"Sin Wave - Frequency: {sin_frequency} Hz")
plt.xlabel("Time")
plt.ylabel("Amplitude")
plt.grid(True)
return plt

# Sidebar widget for user input
frequency = st.sidebar.slider("Frequency",
min_value=0.1, max_value=5.0, value=1.0, step=0.1)

# Display the chart
st.pyplot(generate_plot(sin_frequency=frequency))

```

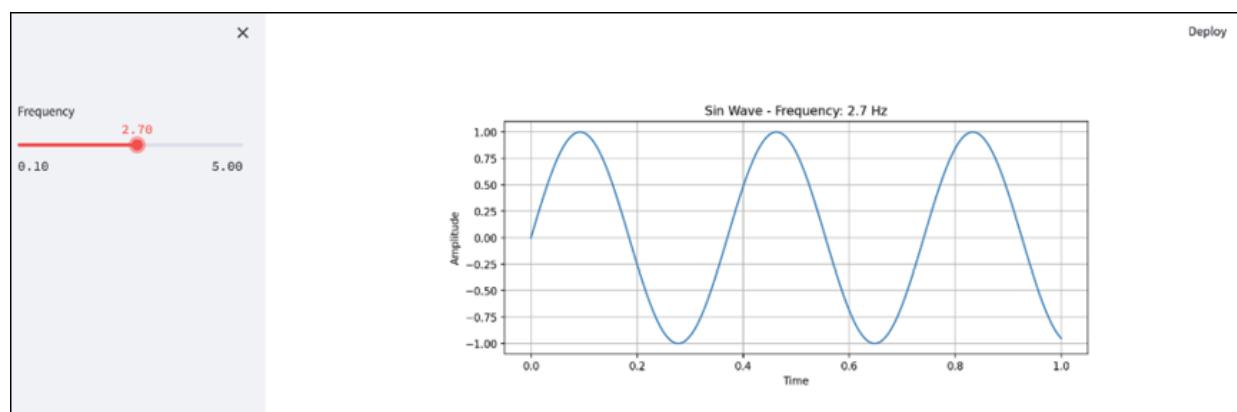


Figure 7.8: Mutating charts based on the slider value

[Figure 7.8](#) is a simplistic example; more advanced mutating chart can help update the data distribution from the API or

external database in specified time intervals for the end users.

Best practices when using mutating charts

The following are some best practices to be mindful about:

- **Responsive design:** Ensure the charts are responsive and fit well within the layout of the Streamlit app. Utilize `use_container_width=True`, and set up the layout using `cols` and expanders.
- **Optimize for performance:** For real-time data or complex charts, consider optimizing data processing and plotting code to improve performance and responsiveness.
- **User feedback:** To improve user experience, add elements like spinners with immediate visual feedback when interacting with controls that affect the chart.

Connecting to databases with Streamlit

It is important for a complex data application to take input and be able to write back to databases, transactional or analytical. The choice of database (SQL databases like PostgreSQL, MySQL, or NoSQL databases like MongoDB) depends on the application's requirements. Python offers several libraries to facilitate these connections, such as `sqlite3` for SQLite databases, `psycopg2` for PostgreSQL, and `pymongo` for MongoDB.

The following code is a simple example of using SQLite in a database. Other databases and warehouses provide their own Python APIs and connectors in Python.

```
import sqlite3  
import streamlit as st
```

```
# Connect to SQLite database (or create it if it
# doesn't exist)

conn = sqlite3.connect('example.db')
c = conn.cursor()

# Create a table

c.execute('''CREATE TABLE IF NOT EXISTS users
(username TEXT, age INTEGER)''')

conn.commit()

# Function to add a new user

def add_user(username, age):
    c.execute("INSERT INTO users VALUES (?,?)",
(username, age))

    conn.commit()

# Function to fetch all users

def get_all_users():
    c.execute("SELECT * FROM users")

    return c.fetchall()

# User input

username = st.text_input("Username")

age = st.number_input("Age", min_value=1,
max_value=100, step=1)

if st.button('Add User'):
```

```

add_user(username, age)

# Display users
st.write("Users:")
users = get_all_users()
for user in users:
    st.write(user)

# Close the connection to the database
conn.close()

```

The output screen is as follows:

The screenshot shows a Streamlit application window. At the top, there are two input fields: 'Username' containing 'wonderwoman' and 'Age' containing '10000'. Below these is a red-bordered button labeled 'Add User'. Underneath the button, the text 'Users:' is followed by a list of tuples: ('superman', 1), ('batman', 1), and ('aquaman', 100). The entire application has a light gray background.

Figure 7.9: Enter the username and age to be persisted and fetched from SQLite database

Best practices for database connections

The following are some best practices to be mindful about:

- **Use connection pools:** To manage multiple connections efficiently, especially in production, consider using connection pooling.
- **Secure your database:** Ensure that your database credentials are stored securely using environment variables Streamlit's secret management, or other external secret management tools.
- **Close connections:** Always close database connections when no longer needed to free up resources.
- **Input sanitization:** When executing queries that include user input, ensure that inputs are sanitized to prevent SQL injection attacks.

App testing with Streamlit

Streamlit's testing framework simplifies automating app tests, allowing for simulations of user actions and checking outputs without needing a browser. This system works seamlessly with test automation tools, making it easy to test app functionality through a straightforward API. It is designed to work well with testing suites like `pytest`, making sure your app works as expected through development changes.

Testing a Streamlit app can be achieved by following a simple pattern using Streamlit's testing framework along with `pytest`. To test this, ensure that `pytest` is installed in your environment

A test flow includes the following:

- Initializing the app test environment.
- Simulating user actions like incrementing input values or clicking buttons.
- Asserting the expected state of the app's output.

This process allows you to ensure your app behaves correctly in response to user inputs. Let us test a Streamlit app that counts beans.

- The user enters a number
- The user clicks a button
- The app displays the total count

Python folder structure for `pytest` looks as follows:

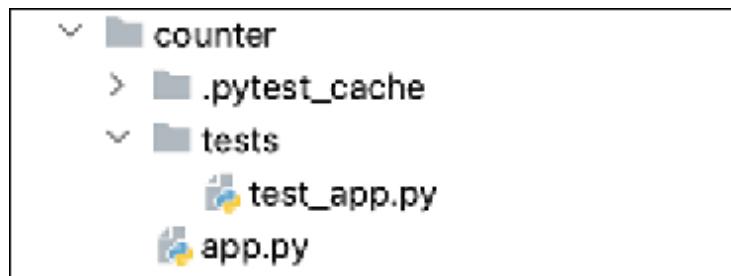


Figure 7.10: Folder structure for Streamlit app with `pytest` test cases.

```
import streamlit as st

st.title('Data Counter')

number = st.number_input('Enter the count',
value=0, step=1)

if st.button('Add More Count'):

    st.write('Total count:', number)
```

The screen displayed will be as follows:

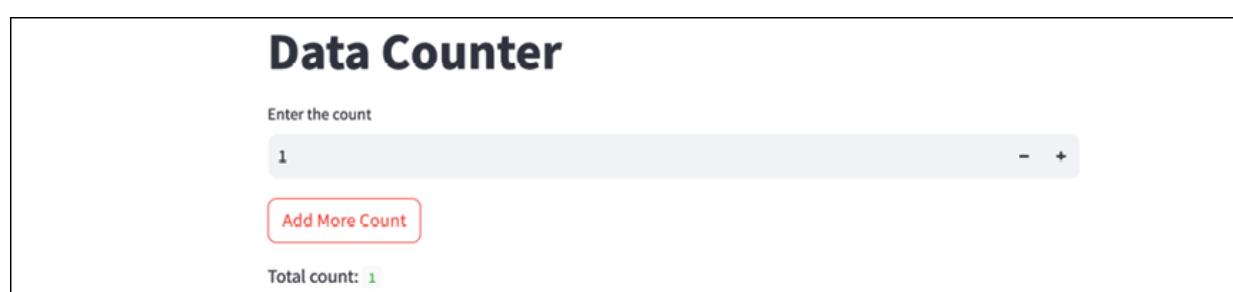
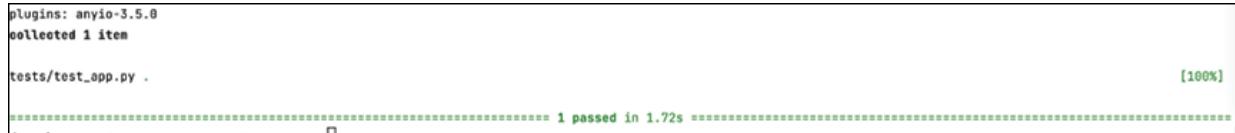


Figure 7.11: Data application with counter and returns the entered count

```
from streamlit.testing.v1 import AppTest
```

```
def test_beans_counter():
    app_test = AppTest.from_file("app.py").run()
    app_test.number_input[0].increment().run()
    app_test.button[0].click().run()
    assert app_test.markdown[0].value == 'Total
count: `1`'
```



A screenshot of a terminal window displaying Pytest test results. The output shows:

```
plugins: anyio-3.5.0
collected 1 item

tests/test_app.py .                                         [100%]

=====
1 passed in 1.72s =====
```

Figure 7.12: Pytest test results run.

This is a simplistic example of app testing; more complex cases can be built using Streamlit's AppTest API and `pytest`. These tests can be automated and integrated into continuous integration and deployment pipelines. Some of the most preferred tools to achieve this would be GitHub actions, Jenkins, and GCP cloud build.

Best practices for app testing with Streamlit

The following are some best practices to be mindful about:

- **Write tests early and often:** Incorporate testing early into the development cycle and update the tests as the app evolves; this is to ensure that every iteration of delivery follows the right standards.
- **Test for different user interactions:** Simulate them to ensure your app can handle them gracefully.

- **Use mocks for external services:** When testing functions that rely on external services, use mocks to simulate those services.
- **Integrate testing into CI/CD:** Automatically run your tests as part of your Continuous Integration/Continuous Deployment pipeline to catch issues early.

Design patterns pitfalls to avoid for Streamlit

Streamlit, as a Python library for creating web applications, does not enforce specific design patterns, such as class-based design, directly within its API. However, Python's flexibility allows developers to apply various software design patterns, including custom class design patterns, to organize their Streamlit code better, enhance maintainability, and facilitate reuse across different parts of an application or multiple projects.

Defining custom classes in Streamlit apps requires mindful strategies to prevent issues related to Streamlit's rerun mechanism. Streamlit's design to rerun your script after each user action, your custom classes could be redefined repeatedly in a single session. This situation might lead to unexpected behaviors, particularly when comparing classes and instances. By organizing classes in separate modules and focusing comparisons on internal values, developers can ensure that their applications remain robust and consistent across user interactions.

Strategies for defining custom classes

The following are some strategies and design patterns recommended by Streamlit to avoid unusual behavior of the application:

- **Move class definition is a separate module:** The most recommended approach involves moving your class definitions to a separate Python file and importing them into your Streamlit script. This method ensures that, unless you are actively editing the class file, Streamlit does not reload and redefine the class with each script rerun.
- **Internal value comparison in classes:** For classes primarily storing data (for example, data classes), comparing the stored values may be more relevant than comparing class types directly. By implementing a custom `__eq__` method, you can ensure comparisons focus on internal values.
- **Serialization of class instances:** Another effective technique for managing classes that encapsulate data involves implementing serialization and deserialization methods, such as `to_str` and `from_str`. This approach enables you to store the data of a class instance in `st.session_state` as a string rather than directly storing the instance. This method also focuses on comparing the internal data, sidestepping the issue of varying in-memory IDs.
- **Caching to preserve class instances:** For classes serving as resources (for example, for database connections, state management, or API interactions), applying the cached singleton pattern might be beneficial. This involves using a decorator like `@st.cache_resource` on a class method to produce and cache a singleton class instance.

These strategies demonstrate how to effectively manage custom class instances in Streamlit apps, addressing common issues related to the rerunning script mechanism. By employing serialization for data classes or caching for

resource classes, you can ensure your Streamlit apps remain efficient and robust.

Conclusion

In this chapter, we looked at the advanced concepts of Streamlit, focusing on features and methodologies that transition a good app into a great one. The topics discussed ranged from understanding the details of Streamlit's execution model to utilizing third-party libraries and applications for enhanced functionality. We discussed callbacks, secret management, timezone handling, mutating charts, database connections, app testing, and avoiding pitfalls in custom class design, which has laid a foundation for building secure, efficient, and user-friendly web applications. These practices not only streamline development but also ensure that the applications are scalable, maintainable, and ready to tackle complex challenges.

Exercises

- 1. What is the purpose of using callbacks in Streamlit applications?**
 - a. To schedule tasks to run at specific times.
 - b. To create dynamic and interactive applications by reacting to user inputs.
 - c. To call external APIs periodically.
 - d. To optimize app performance automatically.

- 2. How does Streamlit handle secret management?**
 - a. By using environment variables exclusively.

- b. Through the `.streamlit/secrets.toml` file for secure credential storage.
 - c. By encrypting all data stored in `st.session_state`.
 - d. Using third-party secret management services only.
3. **What is the recommended strategy to avoid class redefinitions in Streamlit?**
- a. Always define classes within the Streamlit callback functions.
 - b. Define your class in a separate module and import it.
 - c. Serialize all class instances before using them.
 - d. Use only built-in Python classes.
4. **Which of the following is NOT a recommended best practice for working with timezones in Streamlit applications?**
- a. Always store datetime values in UTC.
 - b. Be explicit about which timezone a given time refers to.
 - c. Ignore daylight saving time changes as they are automatically adjusted.
 - d. Use library support for timezone conversions.
5. **What is the benefit of mutating charts in Streamlit?**
- a. They reduce the amount of code needed for chart creation.
 - b. They are only used for static data visualization.

- c. They provide an interactive and dynamic way to visualize data based on user input or live data.
 - d. They automatically cache data for faster loading.
6. **Which of the following best practices is crucial for database connections in Streamlit applications?**
- a. Store database credentials in plaintext within the script.
 - b. Close database connections when they're no longer needed.
 - c. Connect to the database at the beginning of each user session.
 - d. Use a single database connection for all users.
7. **During app testing for Streamlit, what is the primary benefit of using Streamlit's testing framework along with pytest?**
- a. It eliminates the need to write any test cases.
 - b. It allows for simulating user actions and checking outputs without a browser.
 - c. It automatically fixes any bugs found during testing.
 - d. It only tests the visual aspects of the app.
8. **What does serialization of class instances allow you to do in Streamlit apps?**
- a. Increases the speed of app reruns.
 - b. Store class instance data in `st.session_state` rather than the instance itself.

- c. Automatically encrypts class data.
- d. Prevents any form of data comparison.

9. Why should custom classes be defined in a separate module when working with Streamlit?

- a. To increase the app load time.
- b. To prevent the class from being redefined on each app rerun.
- c. Classes cannot be defined within Streamlit scripts.
- d. It is a requirement by Python, not Streamlit.

10. What does using `@st.cache_resource` do for class instances in Streamlit?

- a. Decreases the security of cached data.
- b. Preserves the instance across reruns by caching it.
- c. Makes the class instance immutable.
- d. Automatically serializes the class instance.

Answers

- 1.** b
- 2.** b
- 3.** b
- 4.** c
- 5.** c
- 6.** b
- 7.** b
- 8.** b

9. b

10. b

CHAPTER 8

Deployment of Streamlit Apps

Introduction

Deploying your Streamlit application is the final step in making your data-driven insights accessible to users around the world. In this chapter, we will explore the various aspects of hosting and deploying Streamlit applications, ensuring that they run smoothly and efficiently in different environments. By the end of this chapter, you will have a thorough understanding of how to deploy your Streamlit applications effectively. Whether you are hosting your application on a cloud service or setting it up in an on-premises environment, you will be equipped with the knowledge and tools to make your application accessible to your audience.

Structure

The chapter discusses the following topics:

- Hosting considerations for Streamlit applications
- Hands-on hosting application using Streamlit share
- Hands-on hosting application using GCP cloud run

Objectives

In this chapter, we aim to cover the deployment of Streamlit applications and various cloud service providers to host and serve your applications. Streamlit simplifies the creation of interactive

applications; in this chapter, we will explore various considerations for deploying applications in different environments.

Hosting considerations for Streamlit applications

Once the Streamlit application is ready to be served, it is important to decide on the right hosting platform to serve your application. It is like picking the right stage for your performance.

When the Streamlit applications are run in the local environment with the following command:

```
Streamlit run app.py
```

The application runs on your local web browser as follows:

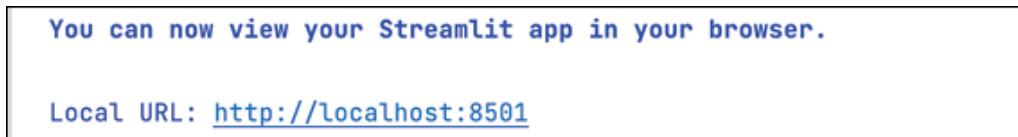


Figure 8.1: Run result for Streamlit local application

The following figure shows the output:



Figure 8.2: Streamlit application served locally

Streamlit applications use the client-server application, wherein the client's side is the front end (which appears on the browser), and the server side is the Python backend. For any action performed on the front end, the request is sent to the backend. In the local setup, both the client and server side of the application are in the local computer. When we want to serve the application to the public, we need to ensure that the backend can receive the requests from the front end and that communication flows seamlessly.

Deciding on a hosting service is deciding where the application's backend will be running and how the backend will be exposed to the public. If the application needs to store secrets or use additional storage, then these factors should also be evaluated.

Streamlit official recommendation can be found here:

<https://docs.Streamlit.io/develop/concepts/architecture/arc>

hitecture.

Streamlit provides guidance on how to avoid the common pitfalls, as shown in the following figure:

The screenshot shows a documentation page with a sidebar on the left containing navigation links for 'Get started', 'Fundamentals', 'First steps', 'Develop', 'Concepts', 'CORE', 'Architecture & execution', 'Running your app', 'Streamlit's architecture', 'The app chrome', 'Caching', 'Session State', 'Forms', 'Fragments', and 'Widget behavior'. The main content area is titled 'Server-client impact on app design' and contains a list of considerations for building Streamlit apps. At the bottom of the content area, there is a 'Still have questions?' section with a link to forums. Navigation arrows point to 'Previous: Running your app' and 'Next: The app chrome'.

Figure 8.3: Streamlit recommended app design principles

This decision is made while considering several factors surrounding your application type, audience, future use case, and cost. The following are key considerations for deciding on the hosting services:

- **Scalability:** Scalability refers to the application's ability to handle increases and decreases in the load by adjusting its resources. A scalable system can automatically scale up (add more resources like servers, storage, or processing power) during high-demand peak periods and scale down (reduce resources) during low-demand periods, optimizing both performance and cost efficiency. For peak load handling, the hosting platform should be able to scale up and scale down based on the load requirement.
- **Security:** Security involves protecting the data application from various threats and vulnerabilities that could compromise data integrity, confidentiality, and availability. Most of the Cloud service providers offer robust in-house

security options, including measures or services to prevent common security issues like **Distributed Denial of Service (DDoS)** attacks, which can overwhelm a service with traffic and render it unusable, and IP whitelisting, which restricts access to the application to specific, pre-approved IP addresses. If the data application needs to be accessed from the public network, it is crucial to ensure that security measures are considered when deciding on the hosting provider.

- **Ease of deployment:** Ease of deployment refers to how straightforward and efficient it is to deploy new updates or releases of a data application to the hosting environment. A smooth, **continuous integration and continuous deployment (CI/CD)** pipeline reduces the risk of errors during deployment, ensures that new features and bug fixes are delivered rapidly, and allows for quick rollback if something goes wrong.
- **Cost:** The most important factor to consider is budgeting. It is important to plan and estimate the cost and decide on the hosting service.

Hosting Streamlit applications

Streamlit applications are Python applications that can work with only two dependencies: python and Streamlit. When deciding on the hosting service for Streamlit, the first thing to take note of is to cater to the dependencies for the Streamlit application. The following are some of the popular options and combinations of services. We will try packaging and deploying our Product Recommender application using the options discussed in this section.

Hands-on hosting application with Streamlit Share

Streamlit offers options for deploying Streamlit applications using public and private cloud services called Streamlit sharing. The public service lets you host your application in a public space, wherein the application is available for viewing in the Streamlit application gallery and is searchable by search engines.

The deployment option appears on the top-right corner of the Streamlit application local run(*Figure 8.4*):

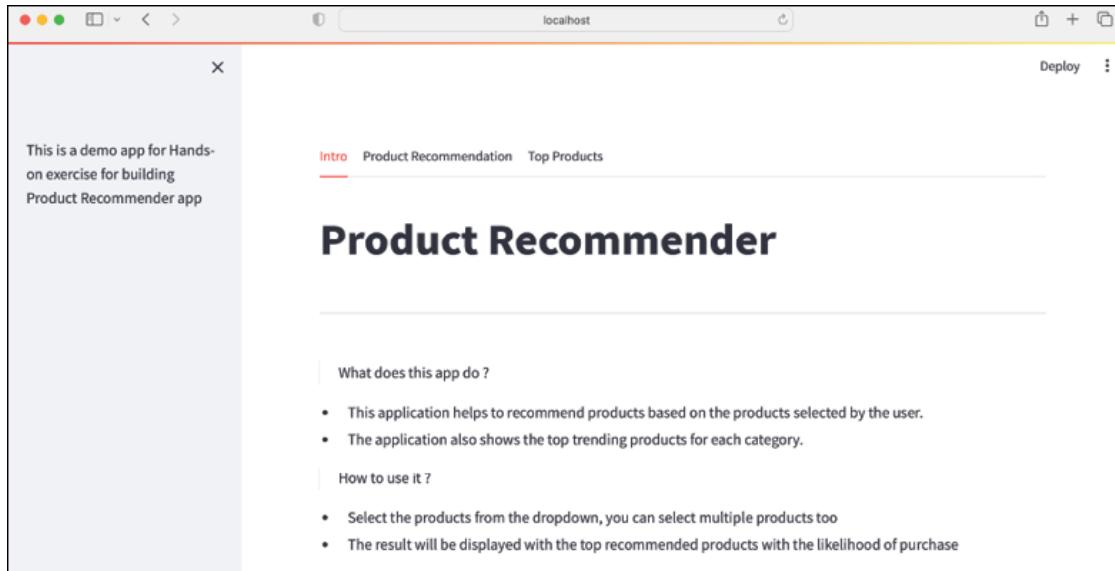


Figure 8.4: Streamlit share deploy option

The following screen shows the deploy option provided by Streamlit for your data application:

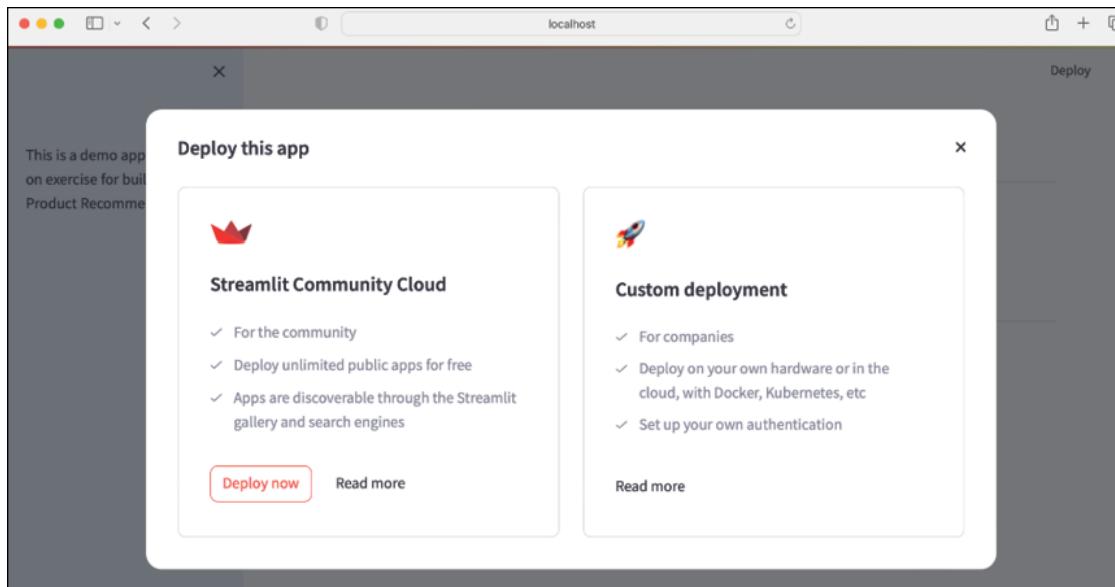


Figure 8.5: Streamlit deploy option window

The easiest way to deploy on the community cloud is by connecting the GitHub repository, as shown in the following figure:

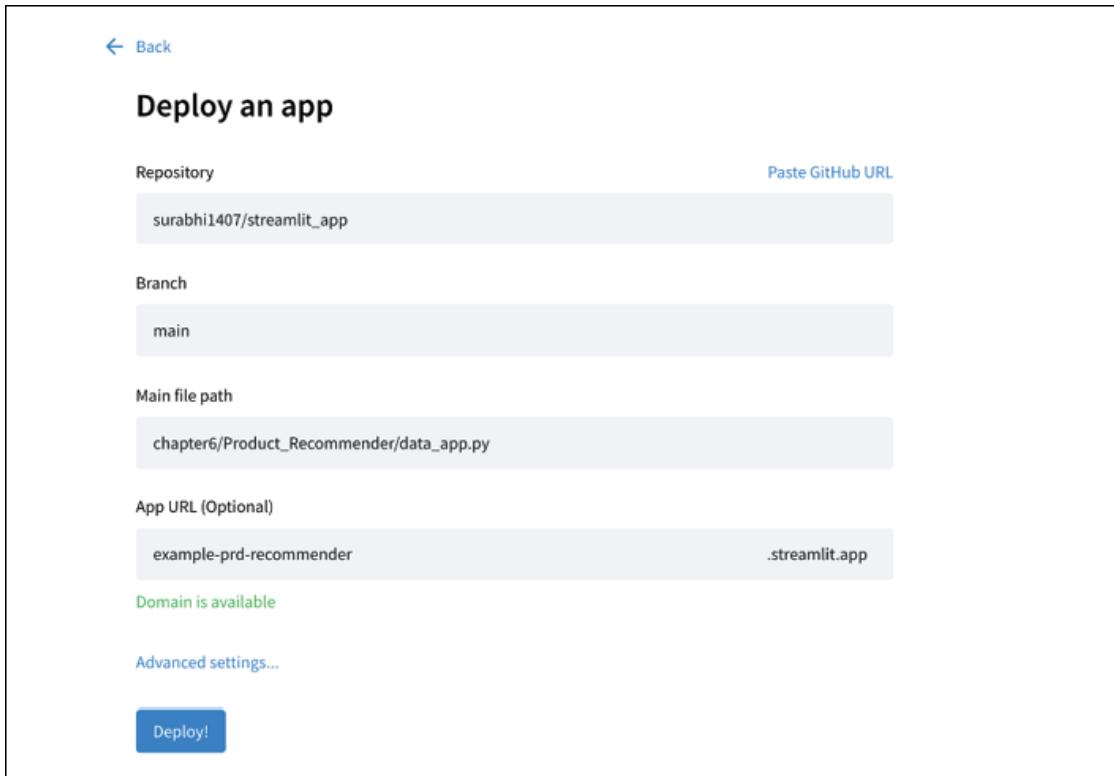


Figure 8.6: Setting up the Streamlit share option for your application

For applications that use secrets, it can be configured in the advanced settings as follows:

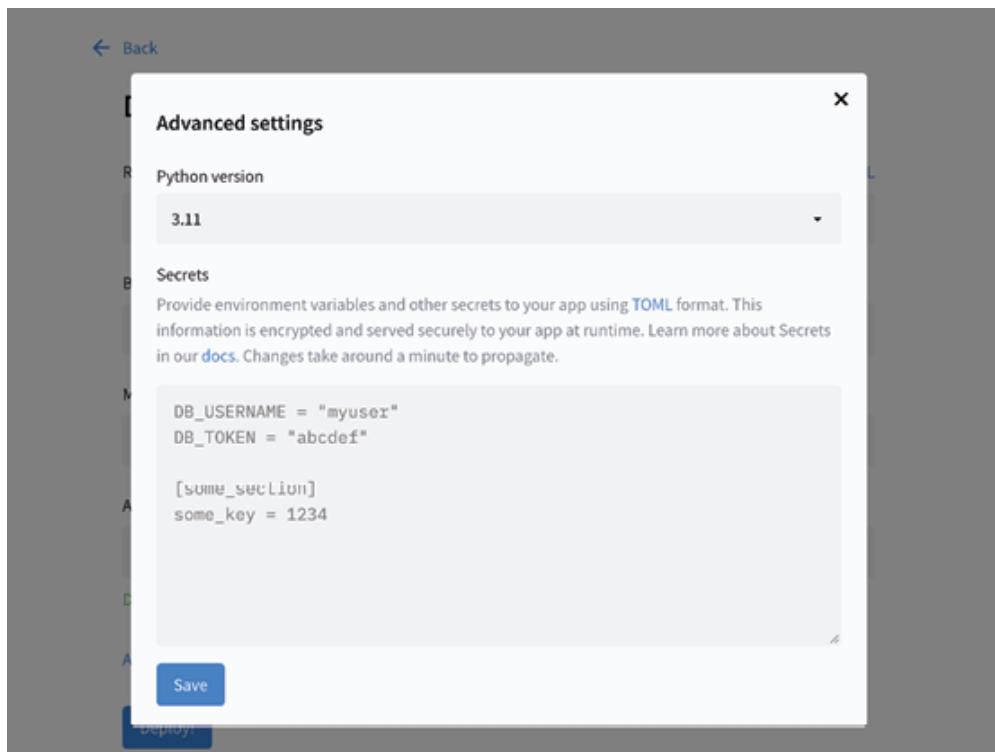


Figure 8.7: Advanced setting for secret management in Streamlit sharing

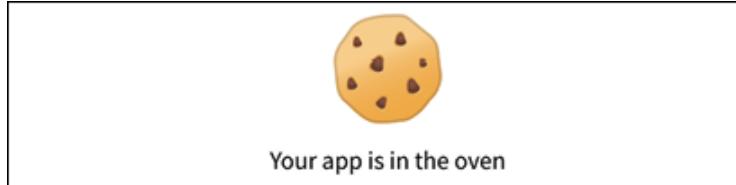


Figure 8.8: Streamlit share flash screen for ongoing build and deployment of the application

Once the data application is set up on Streamlit Cloud, you can access it using your custom URL:

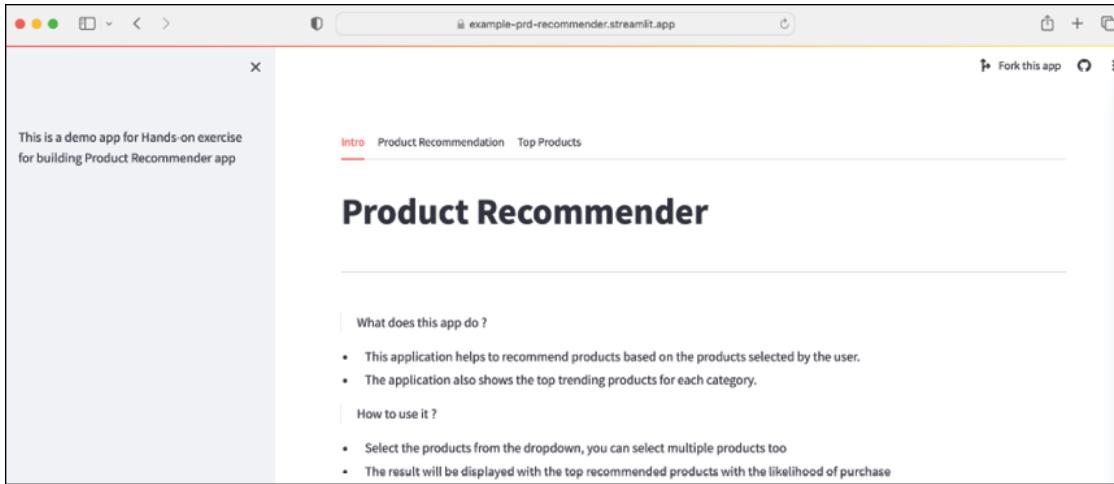


Figure 8.9: ¹ Streamlit application successfully deployed and served on Streamlit share.

Every time you push a commit in your GitHub branch, the application can be rerun (just like local installation). This deployment is simple, fast, and requires the least maintenance. On the downside, because the application is hosted on the Streamlit cloud, it does not provide privacy and data protection.

Overall, this is a very good option for self-hosted projects, but for more robust security features and data protection, Streamlit recommends custom deployment.

Debugging tips

The following are some common errors and issues:

- **Missing dependencies:** Ensure that requirements.txt is updated correctly in the root folder of the base repository.
- **Path inconsistencies:** Use the relative path or the Python OS module to get the right path when trying to access any file in the local tree.
- **Missing module imports:** If using base classes, ensure that the imports are not broken.

Hands-on hosting application using GCP cloud run

The next deployment option to explore is using Google Cloud Platform and its serverless service called Cloud Run. This is one of

the most promising options after Streamlit cloud, requiring very little maintenance and cost.

Google Cloud Platform

Google Cloud Platform (GCP) is a collection of cloud computing services that runs on the same infrastructure that Google uses internally for its end-user products, like Google Search and YouTube. It offers a range of tools for building and managing applications on the web. Just like other cloud providers, GCP offers a service level agreement for the service uptime and provides additional services to add a layer of protection for the applications, such as IP whitelisting and secret manager.

One of the most notable features of GCP is the simplicity of its services.

Google Cloud Run

Cloud Run is Google's serverless service that lets you run your applications in the cloud without worrying about the underlying infrastructure. It focuses on making it easy for developers to deploy applications that can start quickly and scale automatically based on the traffic they receive.

It is designed for applications that need to respond to web requests, like web apps or APIs. One simply needs to provide the code, and Cloud Run handles everything from server management to scaling. It is especially useful if you want your app to scale down to zero when not in use, which can help reduce costs. This makes it a great choice for projects where traffic load can be unpredictable.

The simplicity and ease of use of Cloud Run make it one of the best services that Google offers and a popular industry option for serverless deployments.

Deploying the application

Let us get started with deploying our Product Recommender application on a cloud run. The following are the steps to be taken to get started:

1. **Create a GCP account:** Go to the Google Cloud Platform home page and create a Google Cloud account. This account lets you access the Google Cloud services from a web console, terminal or using Google's developer SDK (*Figure 8.10*):

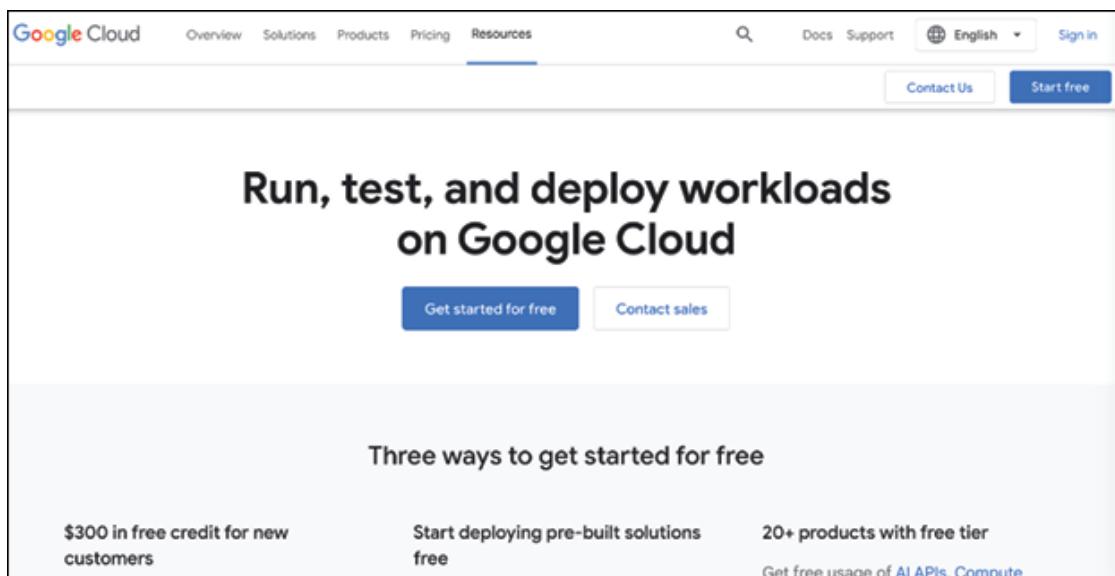


Figure 8.10: Google Cloud Platform home screen

2. **Create a docker container for your application:** To create the docker container for our updated application, let us first update the dockerfile. Note that we do not need to copy the models to our container as we would be using the code:
 - i. The code is as follows:

```
FROM python:3.9-slim
```

```
EXPOSE 8501
```

```
RUN pip install --upgrade pip
```

```
WORKDIR /app
```

```
# Copying requirements file first to leverage  
Docker cache
```

```
COPY requirements.txt .
```

```
RUN pip install --no-cache-dir -r requirements.txt

# Copying your Streamlit applications
COPY ./data_app.py .
COPY ./recommend_app.py .
COPY ./top_products_app.py .

# Copying the data directory
COPY ./data ./data

# Copying the models directory
COPY ./models ./models

CMD Streamlit run data_app.py --server.port=8501 \
    --browser.serverAddress="0.0.0.0"
```

- ii. Go to the root folder of the project and run the docker build command to generate the docker image.

```
docker build -t Streamlit-prod-recommend .
```

The following is the output for the docker build command.

```

streamlit@gcp $ docker build -t streamlit-prod-recommend .
[+] Building 85.3s (15/15) FINISHED
=> [internal] load build definition from Dockerfile
=> => transferring dockerfile: 574B
=> [internal] load .dockerignore
=> => transferring context: 2B
=> [internal] load metadata for docker.io/library/python:3.9-slim
=> [internal] load build context
=> => transferring context: 97.85kB
=> [ 1/10] FROM docker.io/library/python:3.9-slim
=> CACHED [ 2/10] RUN pip install --upgrade pip
=> CACHED [ 3/10] WORKDIR /app
=> [ 4/10] COPY requirements.txt .
=> [ 5/10] RUN pip install --no-cache-dir -r requirements.txt
=> [ 6/10] COPY ./data_app.py .
=> [ 7/10] COPY ./recommend_app.py .
=> [ 8/10] COPY ./top_products_app.py .
=> [ 9/10] COPY ./data ./data
=> [10/10] COPY ./models ./models
=> exporting to image
=> => exporting layers
=> => writing image sha256:6b43f1aee751ff99d8952fe80ca7651a7297fe9ef0d068ae3aab347c25aab8a
=> => naming to docker.io/library/streamlit-prod-recommend

```

Figure 8.11: Docker image successful build log

3. Upload the container to the GCP container registry:

Google Cloud container registry is a managed and secure service that allows storage, version, management, and deployment of docker images. Google Cloud infrastructure provides a streamlined means to connect and integrate with other Google services, such as the compute engine, Kubernetes engine, and cloud run.

For more details on the docker usage and commands, refer to Appendix C.

- We tag our docker image and push it into the GCP container registry.

The command syntax is as follows:

```
docker tag my-image-name gcr.io/my-project-id/my-image-name
```

```
docker tag Streamlit-prod-recommend
gcr.io/psychic-raceway-406207/Streamlit-prod-recommend
```

We will use cloud CLI to push the container from our local filesystem to the GCP project.

```
gcloud auth configure-docker
```

- ii. Once configured correctly, we push the container:

```
docker push gcr.io/psychic-raceway-  
406207/Streamlit-prod-recommend
```

The following is the output of the docker `push` command:

```
streamlit@gcp $docker push gcr.io/psychic-raceway-406207/streamlit-prod-recommend  
Using default tag: latest  
WARNING: Python 3.5-3.7 will be deprecated on August 8th, 2023. Please use Python version 3.8 and up.  
  
If you have a compatible Python interpreter installed, you can use it by setting  
the CLOUDSDK_PYTHON environment variable to point to it.  
  
The push refers to repository [gcr.io/psychic-raceway-406207/streamlit-prod-recommend]  
2fcca0ed8546: Pushed  
3cea90235625: Pushed  
a3c29304e4a1: Pushed  
4fa22500fb5f4: Pushed  
16a2208acba9: Pushed  
902087bdbf96: Pushed  
8468b1747ad5: Pushed  
cae6af1d0d79: Pushed  
7d8b83ea1f73: Pushed  
1d4bb68dc30b: Layer already exists  
da0d3aec36bf: Layer already exists  
0aedf96a6779: Layer already exists  
ec3c4788435a: Layer already exists  
8a70d251b653: Layer already exists  
latest: digest: sha256:057b3427909d60f9ed43489d8e1ea952223eb801010f5d65d2f29b9310e6619a size: 3247
```

Figure 8.12: Push the docker container to the GCP repo

- iii. Once successfully pushed, validate the result in the web console search for the artifact registry in the search bar on top of the GCP console, as shown in [Figure 8.13](#) and [Figure 8.14](#).

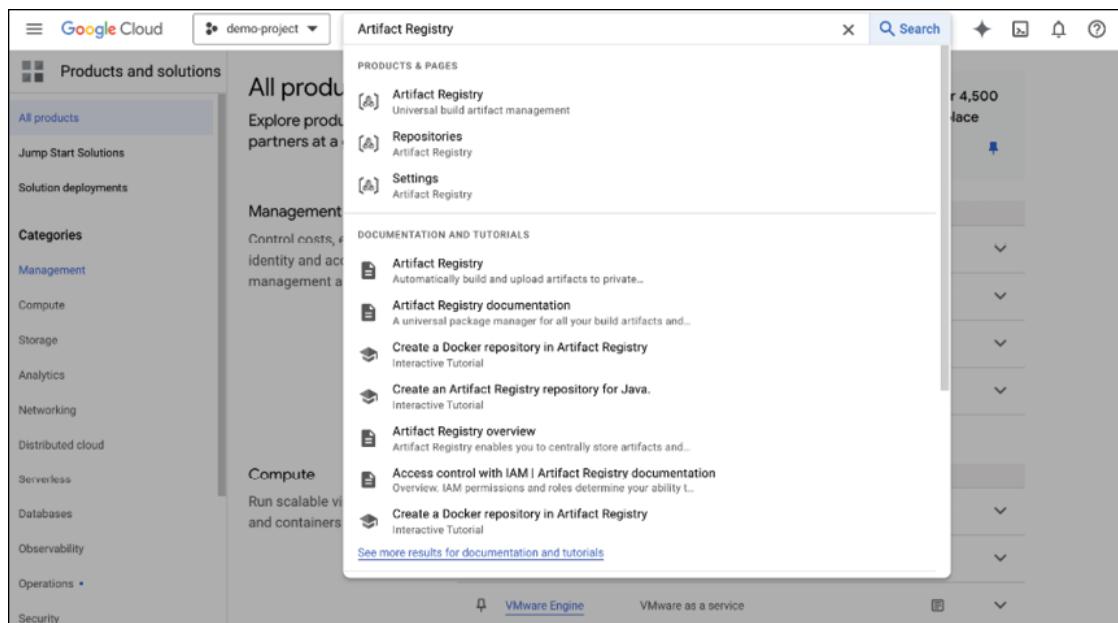


Figure 8.13: Search for artifact registry in the Google console

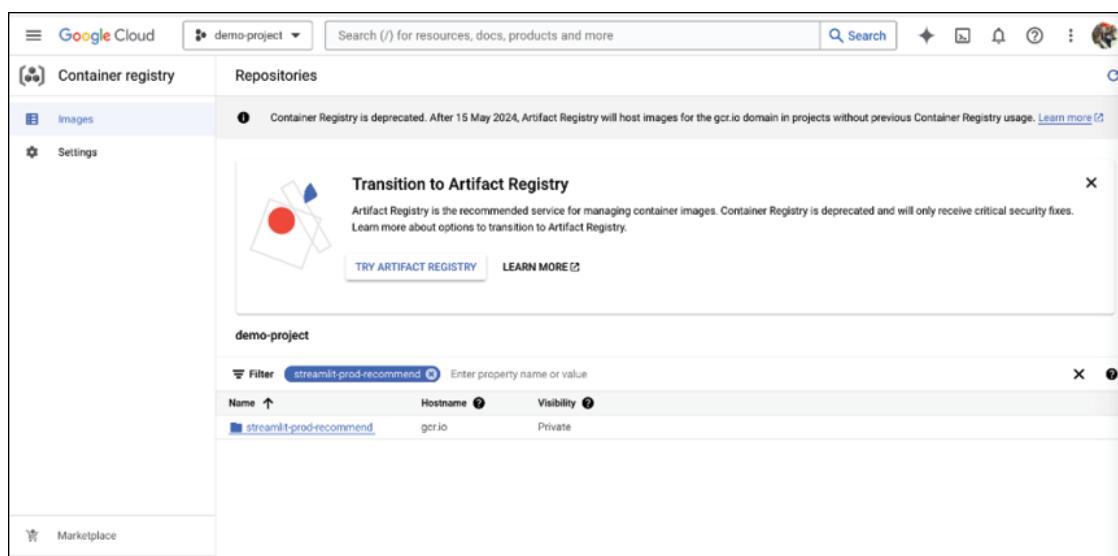


Figure 8.14: GCP container registry images

- iv. Once the Docker image is pushed successfully, the image will appear in the registry list.

4. **Create a cloud-run service to deploy the docker container:** Next, we would deploy the docker image using a cloud-run service. In the web console, search for cloud run service in the GCP console:

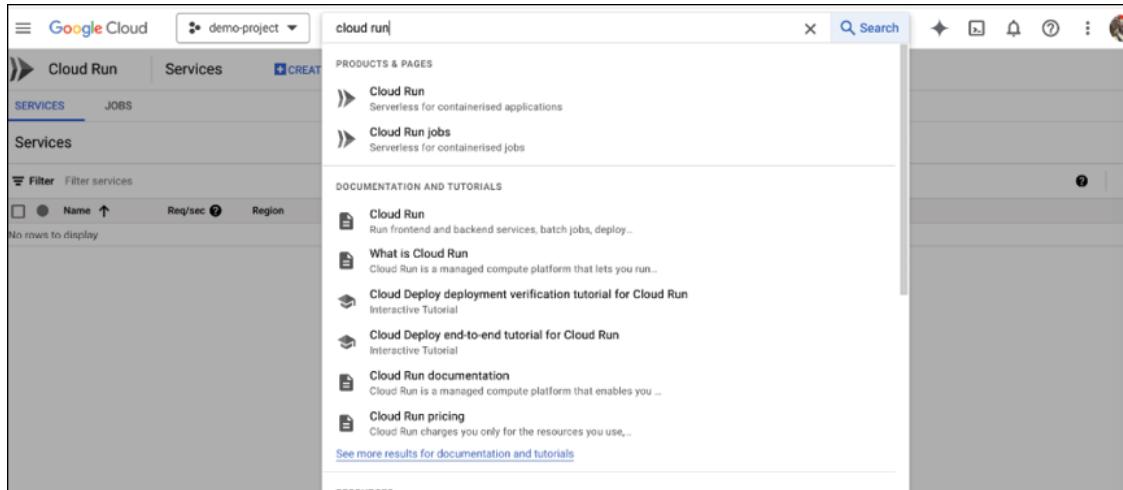


Figure 8.15: Search for cloud run in GCP search

- i. Create a new service using the top tab button as follows:

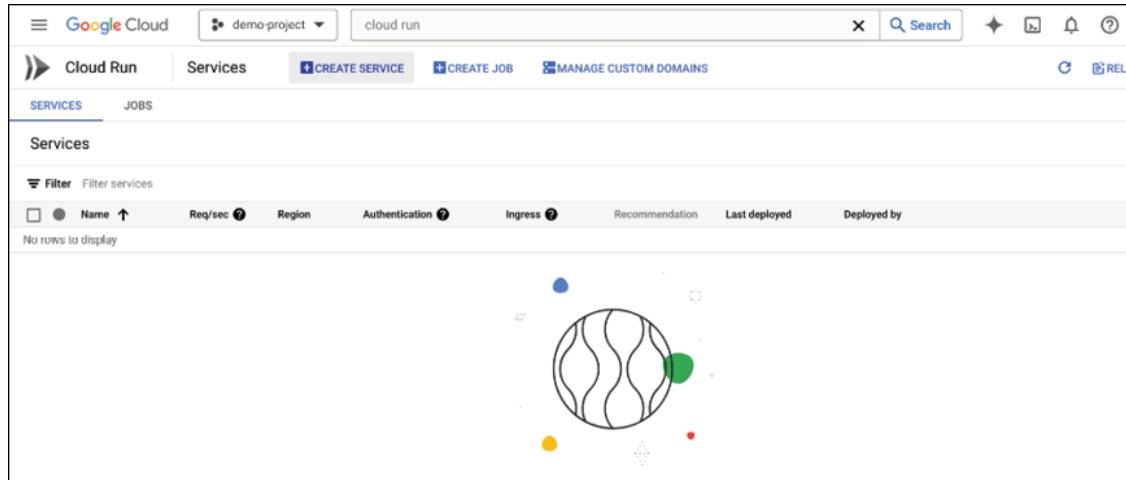


Figure 8.16: Cloud Run home page

- ii. There are multiple options to set up the cloud-run service. First, let us try to deploy our service using the artifact registry.

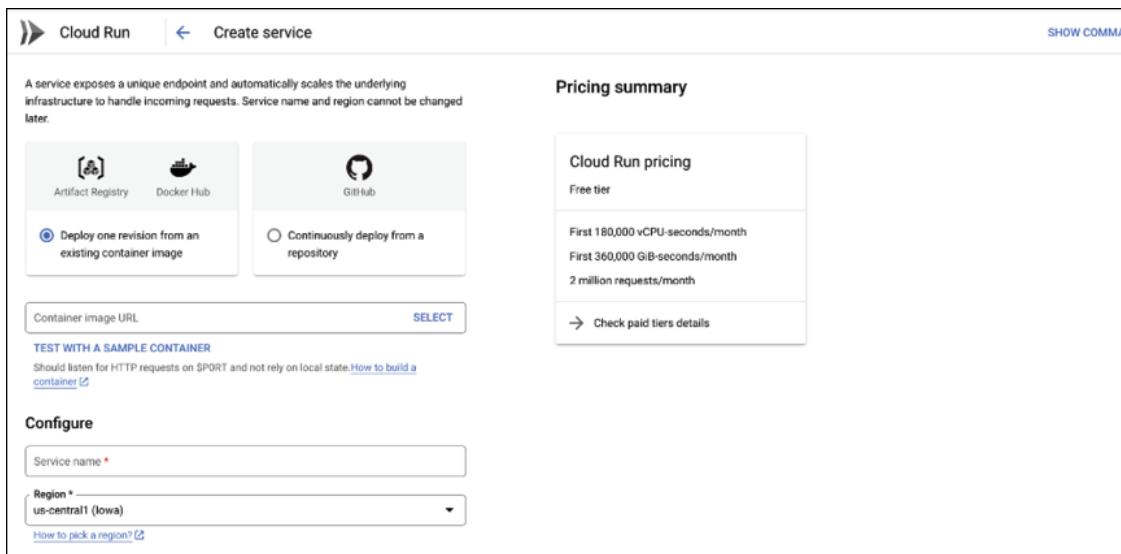


Figure 8.17: Create cloud run service configuration options

- Select the container image from the container registry, which we uploaded in the last step, as shown in the following figure:

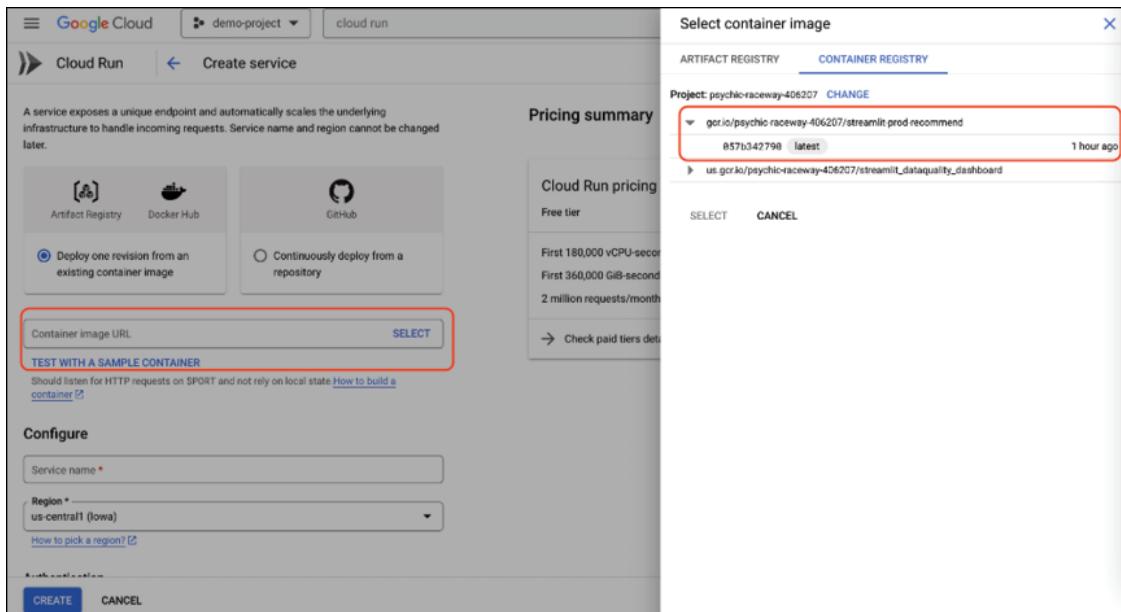


Figure 8.18: Select the docker image from the container registry to be used in the cloud-run service creation

- Once the container image is selected, check the pricing summary options as follows:

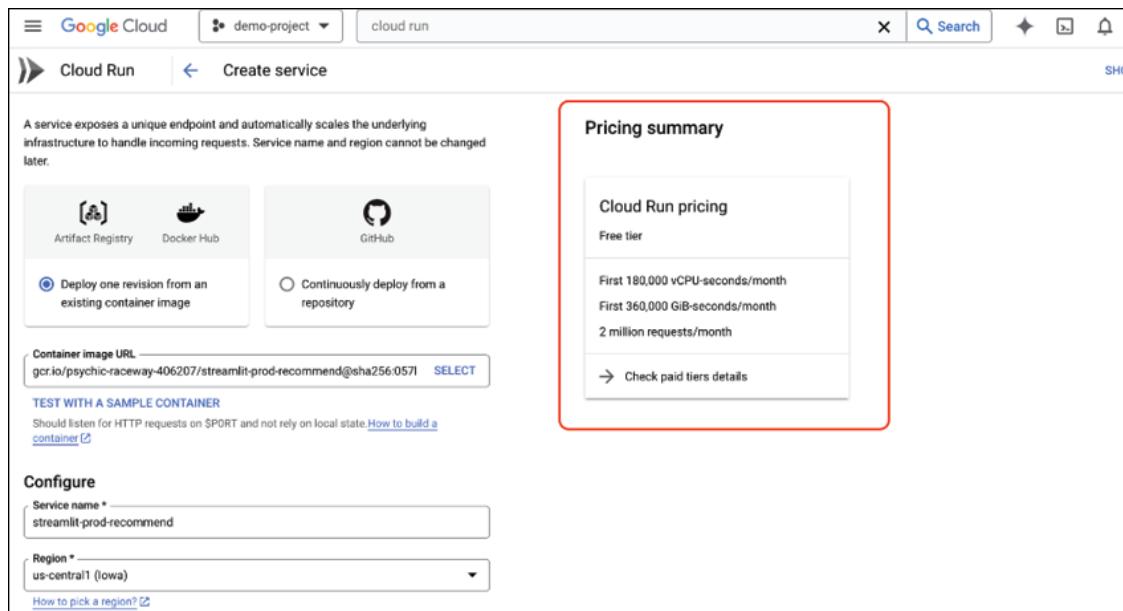


Figure 8.19: Cloud Run pricing options

v. The free-tiring costing can cover most of the use cases, as shown in *Figure 8.20*:

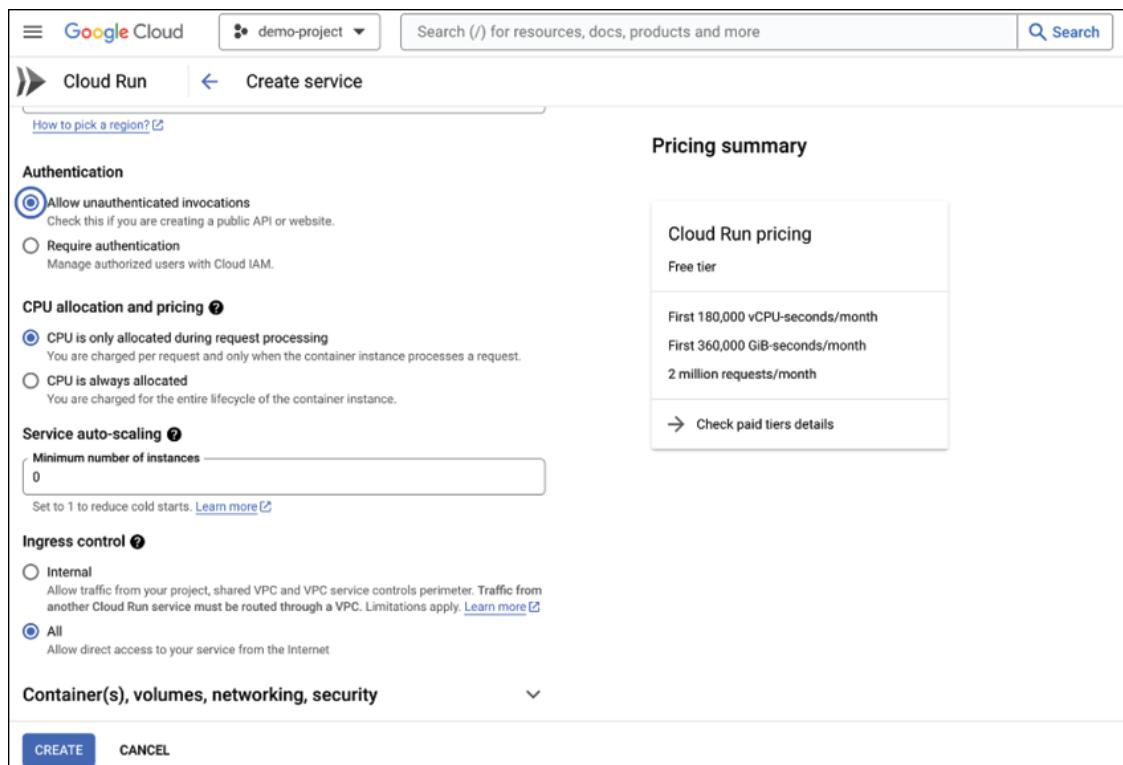


Figure 8.20: Cloud run service configuration options

vi. Note for the following selected options:

- **Authentication:** Allow unauthenticated invocations; this is because we want our Streamlit application to be accessible as a website without the need for an authenticated token or service account
- **Ingress control:** This ensures that the URL accepts traffic outside the Google VPC; anyone with the URL can access the service.

The following is the screen for the cloud-run service dashboard with deployed but failed service:

The screenshot shows the Google Cloud Cloud Run service dashboard. At the top, there's a navigation bar with 'Google Cloud' and a dropdown for 'demo-project'. The search bar contains 'cloud run'. Below the navigation, there are tabs for 'Cloud Run', 'Services', '+CREATE SERVICE', '+CREATE JOB', and 'MANAGE CUSTOM DOMAINS'. The 'SERVICES' tab is selected. Under 'Services', there's a table with one row. The table columns are 'Name', 'Req/sec', 'Region', 'Authentication', 'Ingress', 'Recommendation', and 'Last deployed'. The single row shows a service named 'streamlit-prod-recommend' with 0 req/sec, in the 'us-central1' region, requiring authentication, and ingress set to 'All'. It was last deployed 3 minutes ago. There's also a 'Filter' button and a 'Services' link.

Figure 8.21: Cloud Run service dashboard with failed service

- vii. Once the service is deployed with the configuration, you can see that the build for the container fails. Let us try to see the error and debug the issue, as shown in the following figure:

The screenshot shows the 'Service details' page for the 'streamlit-prod-recommend' service. At the top, it says 'Deploying revision' and has a 'HIDE STATUS' link. A red box highlights the 'Creating revision' status, which shows an error message: 'Revision 'streamlit-prod-recommend-00001-qx4' is not ready and cannot serve traffic. The user-provided container failed to start and listen on the port defined provided by the PORT=8080 environment variable. Logs for this revision might contain more information. Logs URL: Open Cloud Logging' and 'Logs for this revision might contain more information. Logs URL: Open Cloud Logging'. Below this, under 'streamlit-prod-recommend', it shows 'Region: us-central1', 'URL:(missing)', and 'Min. instances: 0'. There are tabs for 'METRICS', 'SLOS', 'LOGS', 'REVISIONS', 'NETWORKING', 'SECURITY', 'TRIGGERS', 'INTEGRATION', 'PREVIEW', and 'YAML'. The 'LOGS' tab is selected. It shows a log entry: 'No errors found during this interval.' and a link 'See more in Error reporting'.

Figure 8.22: Cloud Run service details

viii. When you click on the failed build, you can check the error in the image deployment, which states that the container port is not mapped correctly. As in our docker file, we exposed port **8501** from the container, so this needs to be correctly specified in the container configuration of the cloud-run service as follows:

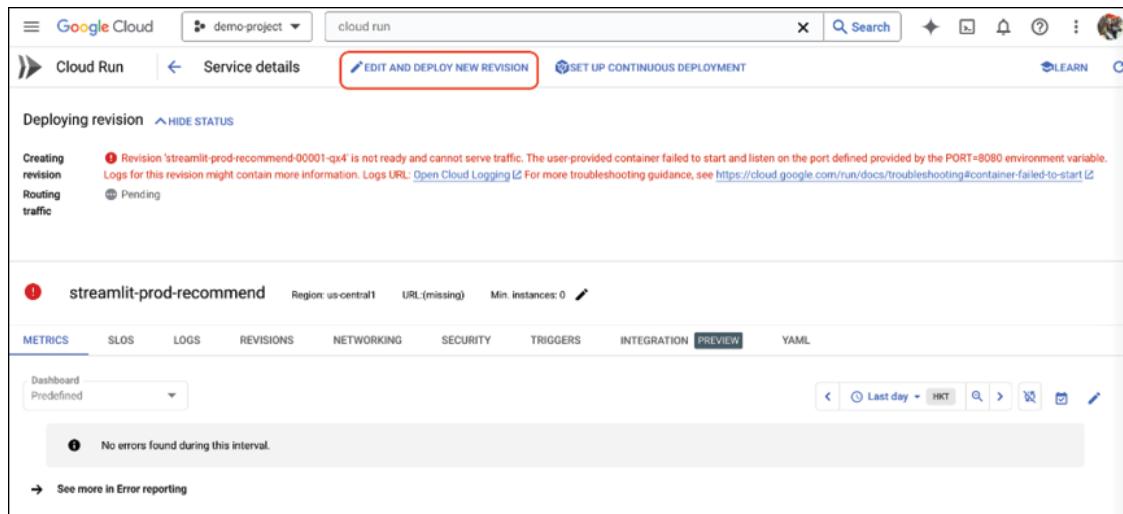


Figure 8.23: Cloud Run service edit screen option

Let us edit the deployment configuration and deploy the new revision as follows:

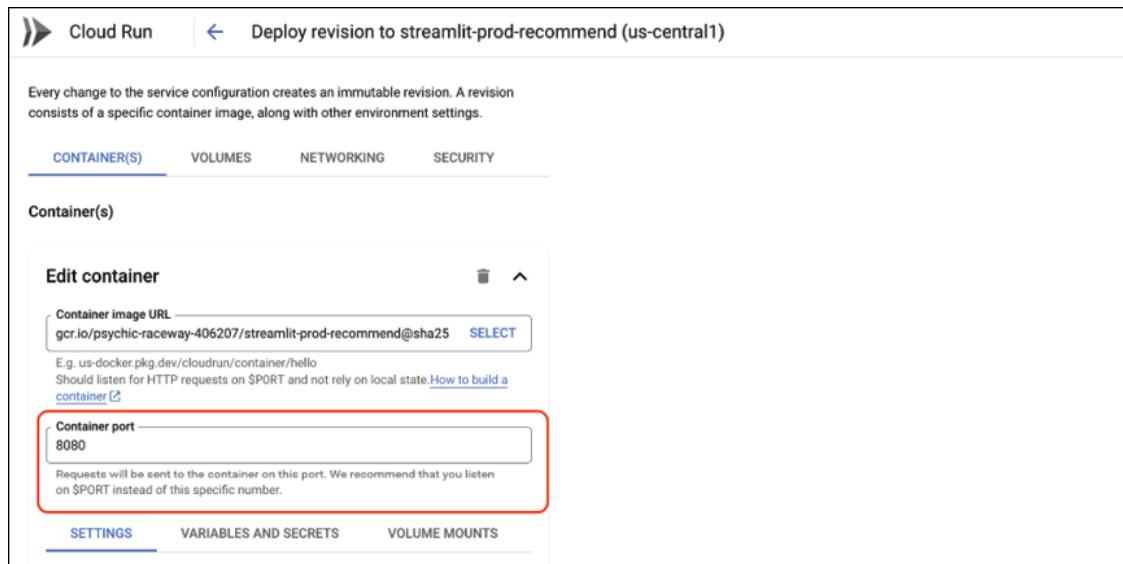


Figure 8.24: Cloud Run service container configuration options

- ix. Update the container port to map the correct port exposed in the docker image, as shown in the following figure:

The screenshot shows the Cloud Run service revision deployment screen for a service named 'streamlit-prod-recommend'. At the top, it says 'Deploying revision' and 'HIDE STATUS' (Assigning traffic to latest specified targets). Below that, 'Creating revision' is completed (green checkmark) and 'Routing traffic' is pending (grey circle). The service details are: Region: us-central1, URL:(loading), Min. instances: 0. The navigation tabs are METRICS, SLOS, LOGS, REVISIONS (underlined), NETWORKING, SECURITY, and TRIGGERS. Under REVISIONS, there's a 'Revisions' section with a 'MANAGE TRAFFIC' button. A 'Filter' dropdown is set to 'Name'. The table lists two revisions: 'streamlit-prod-recommend-00002-t9r' (status: Just now, Actions: three-dot menu) and 'streamlit-prod-recommend-00001-qx4' (status: 10 minutes ago, Actions: three-dot menu).

Name	Traffic	Deployed	Revision URL	Actions
streamlit-prod-recommend-00002-t9r	0%	Just now	+	⋮
streamlit-prod-recommend-00001-qx4	0%	10 minutes ago		⋮

Figure 8.25: Cloud Run service revision deployment screen

- x. Once the revision is deployed, we can observe that the build is successful. For any other detailed errors, you can check the cloud logging for more information on the error, as shown in the following figure:

The screenshot shows the Google Cloud Platform Cloud Run Service Details page for a service named "streamlit-prod-recommend". The service is deployed to the "us-central1" region and has a URL of <https://streamlit-prod-recommend-k0jkpilzy2a-uc.a.run.app>. The "REVISIONS" tab is selected, displaying two revisions:

Name	Traffic	Deployed	Revision	Actions
streamlit-prod-recommend-00002-t9r	100% (to latest)	Just now	+ ⋮	
streamlit-prod-recommend-00001-qx4	0%	11 minutes ago	⋮	

Figure 8.26: Cloud Run service updated dashboard with successful revision

Post successful deployment, the cloud-run service URL can be extracted from the application as follows:

The screenshot shows the Google Cloud Platform Cloud Run Service Details page for the same service. The "URL" field is highlighted with a red box, showing the value <https://streamlit-prod-recommend-k0jkpilzy2a-uc.a.run.app>. The "REVISIONS" tab is selected, showing the same two revisions as Figure 8.26.

Figure 8.27: Cloud Run service URL

- xi. After successful deployment, we can try accessing the URL to see if the application is accessible, as shown in the following figure:

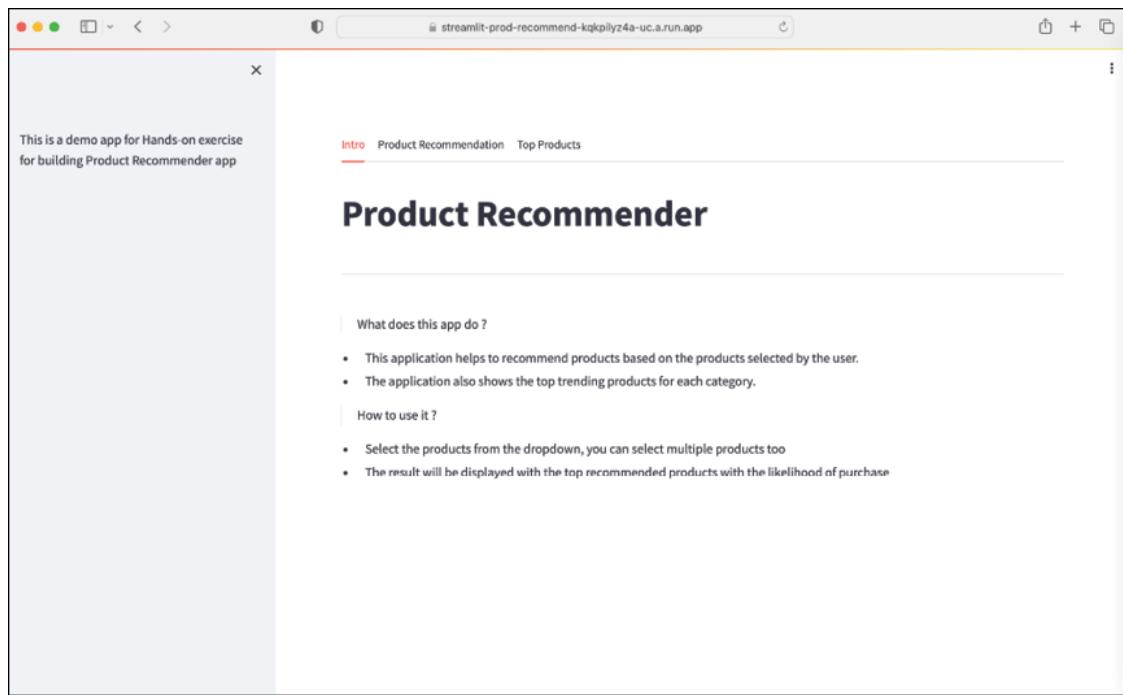


Figure 8.28: Streamlit application on cloud run Intro Page

The next page of the Product recommender application is as follows:

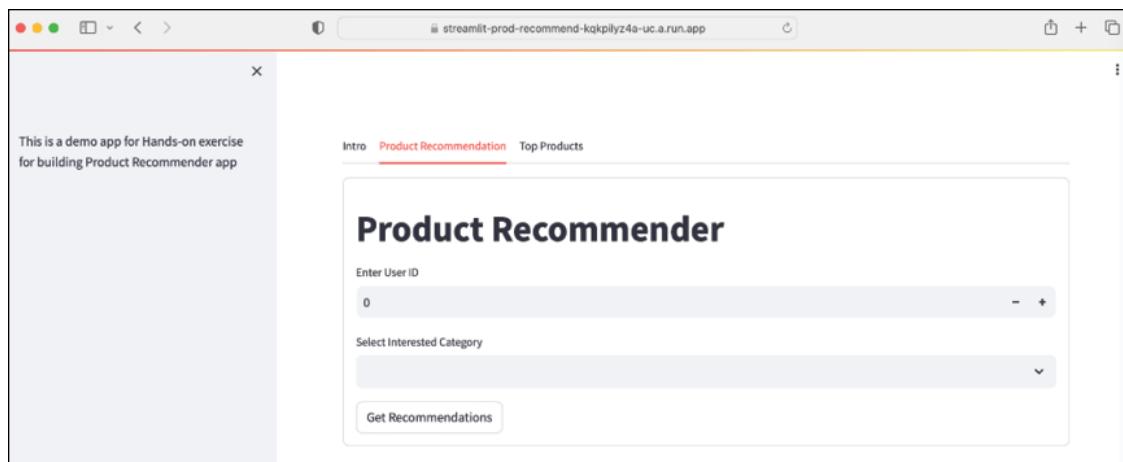


Figure 8.29: Streamlit application on cloud run Product Recommender Page

The top product page of the Product Recommender application is as follows:

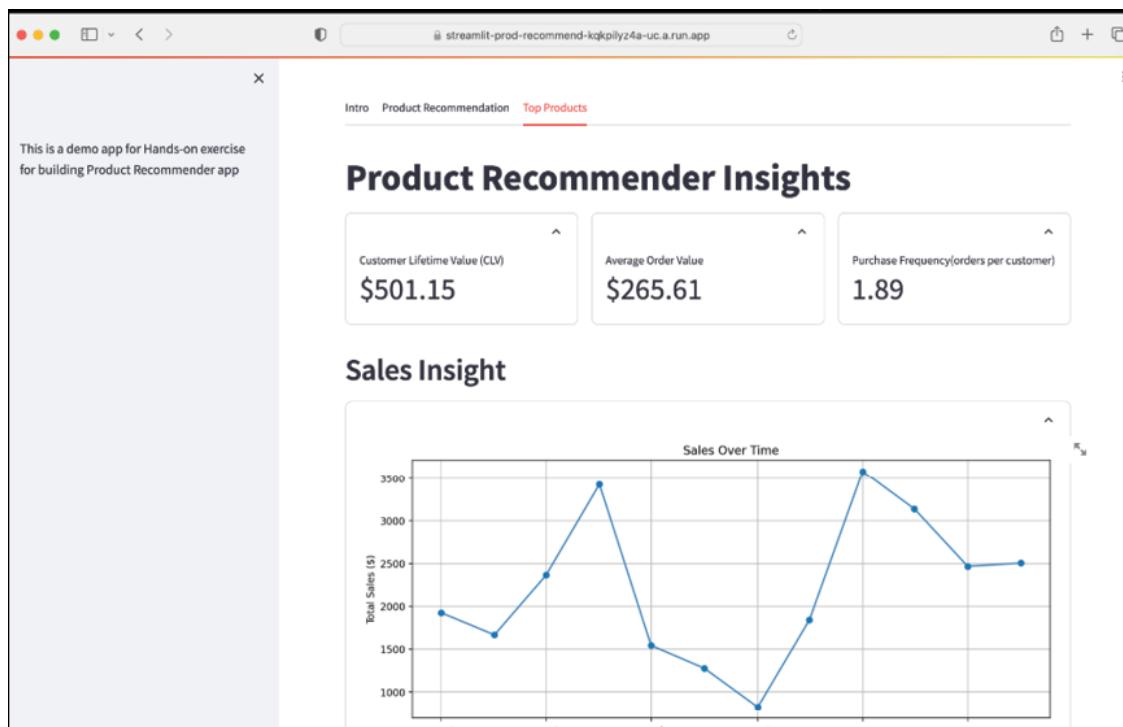


Figure 8.30: Streamlit application on cloud run Top Products Page

You can implement one of the following solutions to add a layer of security for your application or when the Cloud Run service needs to be private and accessed by authenticated users/services while avoiding the need to manage authentication tokens or service accounts yourself:

- **Identity-aware proxy:** This allows you to control access to your application by securing it with OAuth 2.0. IAP can authenticate users without having to handle tokens directly in your application code.
- **API Gateway:** Using Google Cloud API Gateway before your Cloud Run service. The API Gateway can handle OAuth2 tokens, API keys, or Firebase authentication and pass the traffic to your Cloud Run service only after successful authentication.
- **Custom front-end:** A custom front-end that handles user authentication using Google sign-in or a similar OAuth service can also be implemented. The front end would be responsible for obtaining and managing the authentication tokens when calling the Cloud Run service.

- **Cloud endpoints:** Google Cloud Endpoints can be used to add authentication to your service. It supports multiple authentication methods, including API keys and JWTs. It acts as a gateway that handles authentication for you and then forwards the request to your Cloud Run service.

Setting up continuous integration and deployment for the data application

To set up the continuous deployment with Cloud Run, we need to configure the repository and the branch name that triggers the deployment build process is shown in the following figure:

The screenshot shows the Google Cloud Platform interface for a Cloud Run service named 'streamlit-prod-recommend'. The 'Logs' tab is selected, displaying a list of log entries. At the top of the logs section, there is a red box highlighting the 'SET UP CONTINUOUS DEPLOYMENT' button. The logs themselves show several requests from different browsers and devices, with timestamps ranging from April 15, 2024, at 18:22 to 18:26. The requests are primarily GET requests for static files like chunk.js and media files, with some POST requests for health checks and scores.

Severity	Timestamp	Summary
INFO	Scanned up to 15/04/2024, 15:52. Scanned 129.5 KB.	
INFO	2024-04-15 18:22:14.751 HKT	GET 200 1.82 KB 36 ms Safari 16 https://streamlit-prod-recommend-kokpilyz4a-uc-a.run.app/static/js/4319.a6745434.chunk.js
INFO	2024-04-15 18:22:15.645 HKT	GET 200 1.74 KB 18 ms Safari 16 https://streamlit-prod-recommend-kokpilyz4a-uc-a.run.app/static/js/1307.8ea0f33f1.chunk.js
INFO	2024-04-15 18:22:15.959 HKT	GET 200 105.39 KB 31 ms Safari 16 https://streamlit-prod-recommend-kokpilyz4a-uc-a.run.app/media/137a4a0e8fb7bf99cb149215ff79233cf0bea257124c957d2bb890..
INFO	2024-04-15 18:22:16.019 HKT	GET 200 37.74 KB 14 ms Safari 16 https://streamlit-prod-recommend-kokpilyz4a-uc-a.run.app/app/media/e0cc633d0933ef73e115209a398c4ed71b7ccae8869ebd37be9aa27..
INFO	2024-04-15 18:22:16.196 HKT	GET 200 37.19 KB 24 ms Safari 16 https://streamlit-prod-recommend-kokpilyz4a-uc-a.run.app/media/88c173bd25deba97c0843a55a15498d7a72893ad81905caa3b69ff90..
INFO	2024-04-15 18:22:17.072 HKT	GET 200 57.88 KB 3 ms Safari 16 https://streamlit-prod-recommend-kokpilyz4a-uc-a.run.app/media/e6b950cb3bb11ed55a165b0949b5d25324dc9e26d156e62f89a3f54.p..
INFO	2024-04-15 18:26:43.037 HKT	GET 304 696 B 3 ms Chrome 123 https://streamlit-prod-recommend-kokpilyz4a-uc-a.run.app/_stcore/hsmt-config
INFO	2024-04-15 18:26:43.045 HKT	GET 304 782 B 3 ms Chrome 123 https://streamlit-prod-recommend-kokpilyz4a-uc-a.run.app/_stcore/health
INFO	2024-04-15 18:26:43.443 HKT	GET 101 7.34 KB 301 s Chrome 123 https://streamlit-prod-recommend-kokpilyz4a-uc-a.run.app/_score/stream
INFO	2024-04-15 18:26:44.029 HKT	GET 200 4.64 KB 23 ms Chrome 123 https://streamlit-prod-recommend-kokpilyz4a-uc-a.run.app/static/js/1479.6789ab03.chunk.js
INFO	2024-04-15 18:26:44.834 HKT	GET 200 6.77 KB 89 ms Chrome 123 https://streamlit-prod-recommend-kokpilyz4a-uc-a.run.app/static/js/9656.8c935274.chunk.js
INFO	2024-04-15 18:26:44.846 HKT	GET 200 4.36 KB 107 ms Chrome 123 https://streamlit-prod-recommend-kokpilyz4a-uc-a.run.app/static/js/7175.bea076c.chunk.js
INFO	2024-04-15 18:26:45.133 HKT	GET 200 1.82 KB 22 ms Chrome 123 https://streamlit-prod-recommend-kokpilyz4a-uc-a.run.app/static/js/4319.a6745434.chunk.js

Figure 8.31: Cloud Run service logs

The following screen is the screen to set up continuous deployment for the cloud build application:

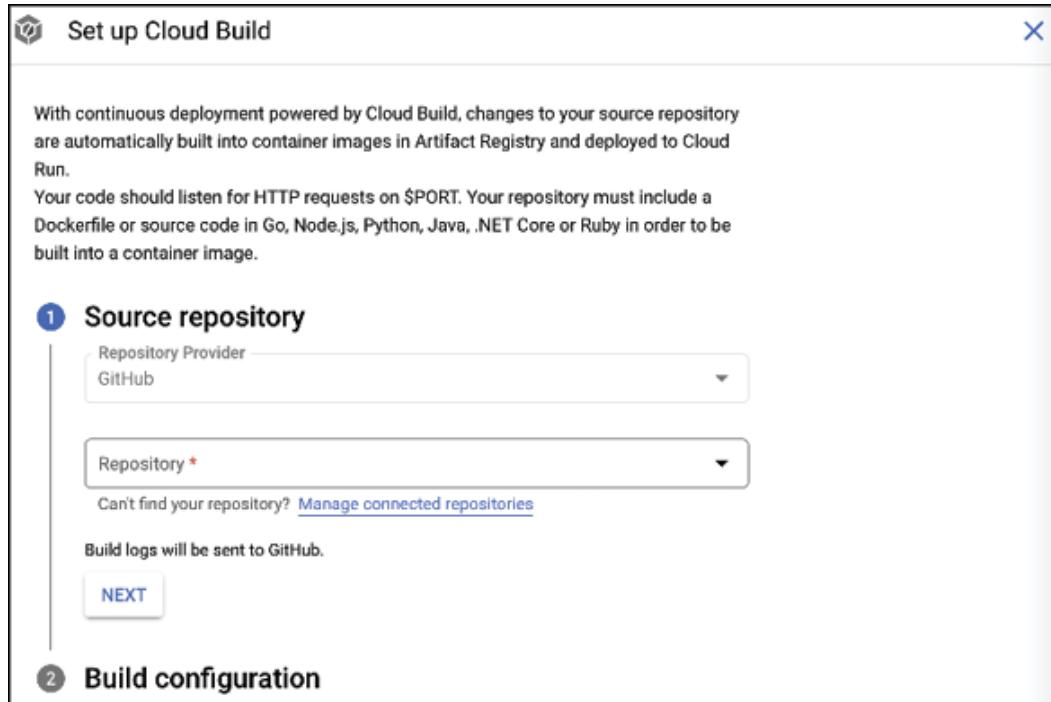


Figure 8.32: Set up cloud build deployment for cloud run

Add your repository in the manage connected repositories by giving GCP access to the repo files and information. For the example, we will use GitHub repo as our base, as follows:

 Set up Cloud Build X

With continuous deployment powered by Cloud Build, changes to your source repository are automatically built into container images in Artifact Registry and deployed to Cloud Run.

Your code should listen for HTTP requests on \$PORT. Your repository must include a Dockerfile or source code in Go, Node.js, Python, Java, .NET Core or Ruby in order to be built into a container image.

1 Source repository

Repository Provider GitHub

Repository * surabhi1407/streamlit_app

Can't find your repository? [Manage connected repositories](#)

I understand that GitHub content for the selected repositories will be transferred to this GCP project to provide the connected service. Principals with access to this GCP project with sufficient permissions will be able to create and run triggers on these repositories, based on transferred GitHub content. I also understand that content from all GitHub app triggers in this GCP project may be transferred to GitHub in order to provide functionality like showing trigger names in GitHub build results. This will apply to all existing and future GitHub App triggers in this project. [Learn more](#)

Build logs will be sent to GitHub.

2 Build configuration

NEXT

Figure 8.33: Cloud build deployment repo configuration

The following screen is the setup for the branch configuration for the cloud build:

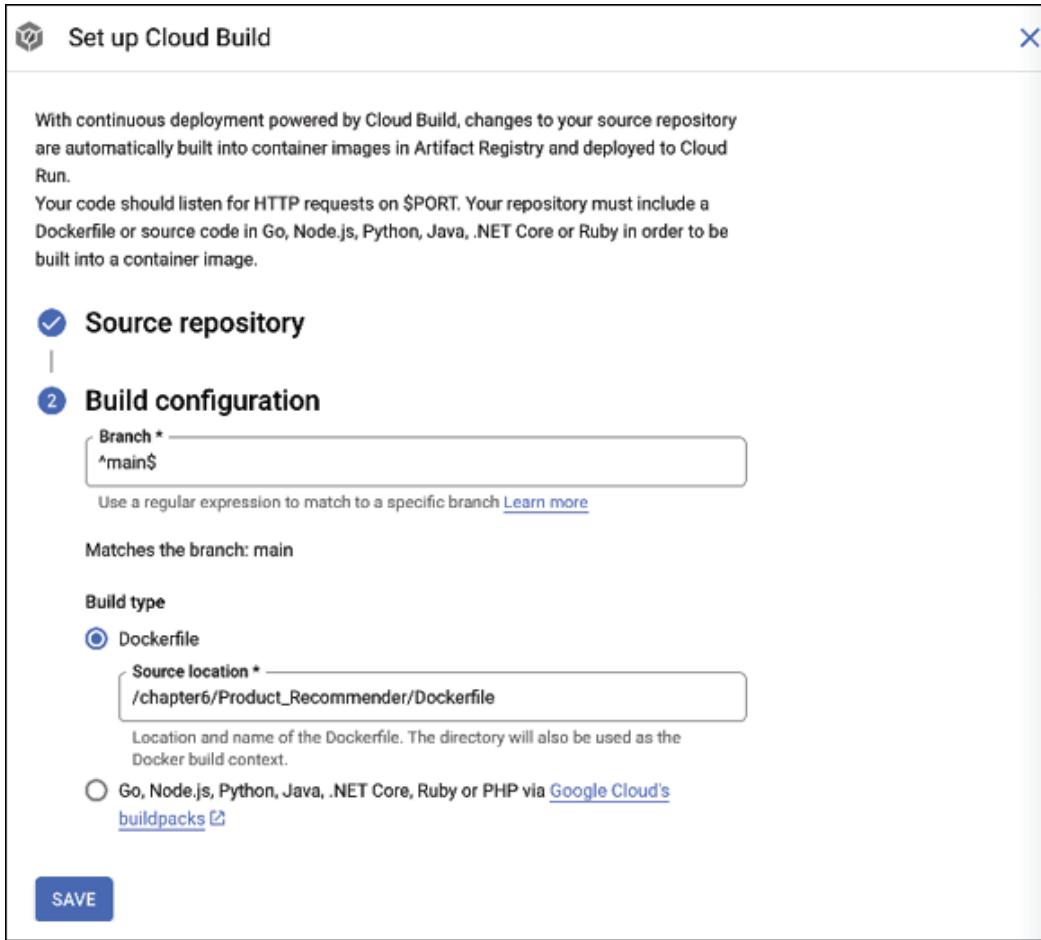


Figure 8.34: Cloud build deployment branch configuration

Once the repository is configured, add the branch name and build file to be used. Since we are using dockerfile, add the path of your dockerfile from the repository home. The branch name refers to the branch for which the push will trigger the new build.

Once the configuration is saved, cloud run will create a trigger for the respective repository and the branch name, as shown in the following figure:

The screenshot shows the Google Cloud Build dashboard for a project named 'demo-project'. The main area displays a successful build for the repository 'surabhi1407/streamlit_app' triggered by 'rmgpab-streamlit-prod-recommend-us-central1-surabhi1407-strnaj'. The build was triggered on '15/04/2024, 22:15' and completed in '00:02:51'. The trigger description is 'Build and deploy to Cloud Run service'. The source is 'surabhi1407/streamlit_app' and the commit is '1e5207f'. Below this, there's a summary of the build history with an average duration of '00:02:51' and a pass-fail percentage of '100% - 0%'. On the left sidebar, the 'Triggers' option is selected.

Figure 8.35: Cloud build deployment dashboard

Validate your build from the cloud build dashboard or history tab as follows:

The screenshot shows the 'Triggers' tab in the Google Cloud Build interface. It lists a single trigger named 'rmgpab-streamlit-prod-recommend-us-central1-surabhi1407-strnaj'. The trigger is configured to build a branch ('Build a...') from the repository 'surabhi1407/streamlit_app' ('Push to branch') using an in-line build configuration. The sidebar on the left shows the 'Triggers' option is also selected.

Figure 8.36: Cloud build deployment triggers

Check the trigger details from the trigger tab, as shown in the following figure:

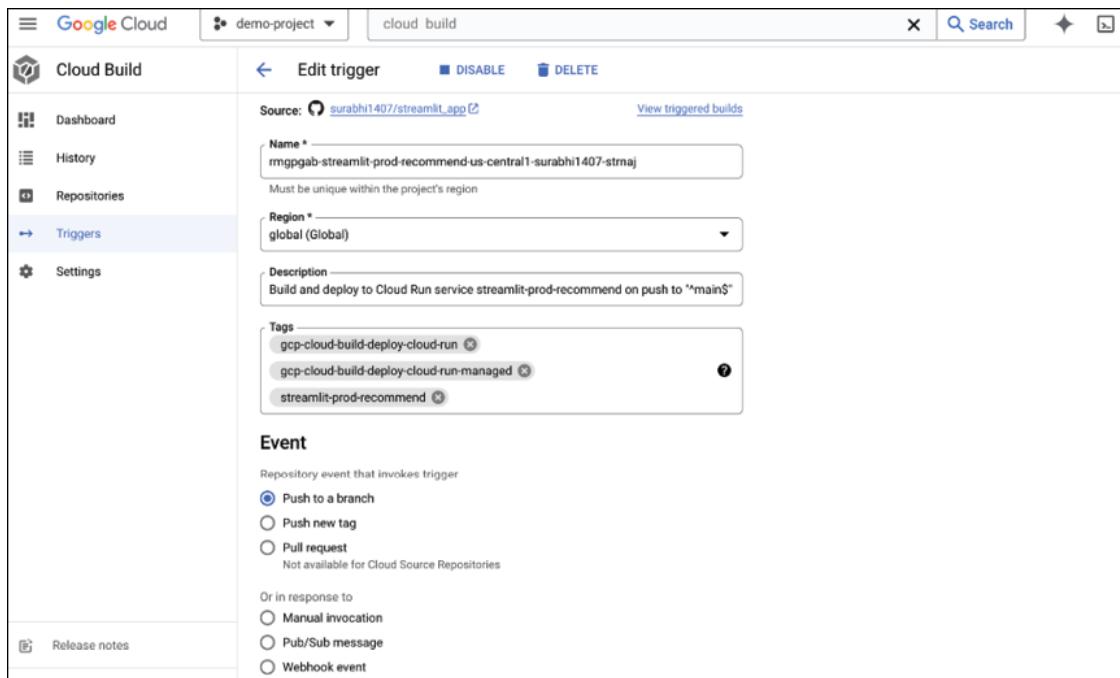


Figure 8.37: Cloud build deployment trigger configuration

The configuration of the trigger can be edited manually as well, if required.

Once the CICD process is set, every merge in the specified branch (main branch in the above example) will trigger a build of the new docker image from the updated code in the repository; the new container will be stored in the GCP container registry and revision deployed on the cloud run service.

Getting the metrics and alert information from Cloudrun

Cloud Run provides various metrics in the metric dashboard, as shown in the following figure:

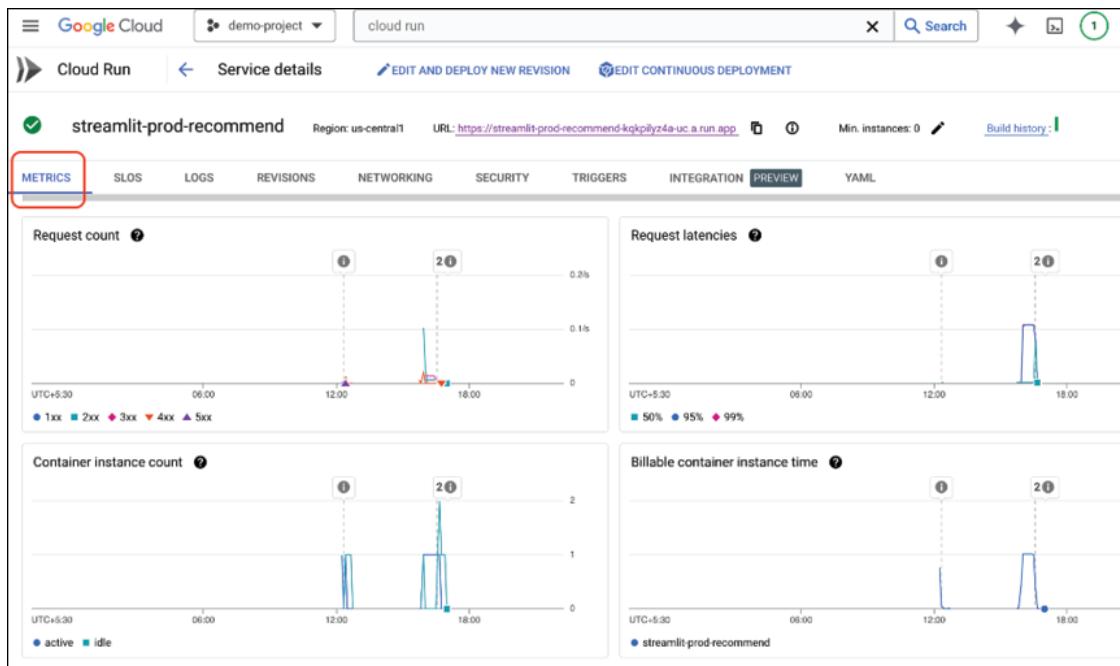


Figure 8.38: Cloud run metrics dashboard

The following screen is the **Metrics** tab where the alerting policy for the cloud-run service can be set:

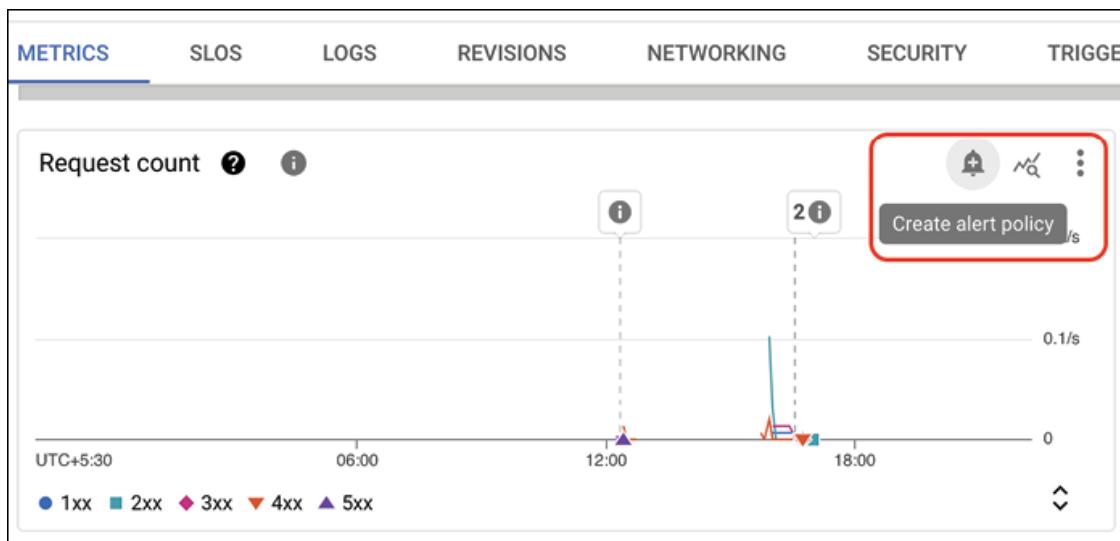


Figure 8.39: Cloud run metric alert policy setup options

For each of the out-of-box metrics provided, an alerting policy can be set up for monitoring purposes. The custom trigger rule set can be configured; for example, if the request count for the Streamlit application exceeds 200, we want to get an email notification or a

Slack channel alert. This helps to track the application performance and avoid any bottleneck issues.

Other Streamlit deployment options

Streamlit community offers a wide variety of options for deploying Streamlit applications. Some of them can be found in the community wiki.

[**https://discuss.Streamlit.io/t/Streamlit-deployment-guide-wiki/5099**](https://discuss.Streamlit.io/t/Streamlit-deployment-guide-wiki/5099)

It has a separate thread for discussing the Streamlit deployment option and issues related to various tools or services used for Streamlit deployment.

[**https://discuss.Streamlit.io/c/deployment/14**](https://discuss.Streamlit.io/c/deployment/14)

When deciding on any new mode for Streamlit applications, it is advisable to refer to the wiki guide to get ideas or to see what others use and recommend for a given use case.

Conclusion

In this chapter, we had a walkthrough of deploying the Streamlit application using Streamlit sharing or GCP cloud run. We discussed the nitty gritty details of deployment, common issues, and debugging options faced during the building and application service. We covered the details about setting up the serverless service Cloud Run on GCP to serve the Streamlit application, setting up the CI/CD pipeline, and configuring monitoring and alerting for various important metrics. Also, Streamlit has a rich and active community of developers who work on various use cases and industries. They share their challenges and solutions in the Streamlit community wiki. Thus, it is an important source of information and guide when choosing which deployment option should be selected for a given scenario.

In the next chapter, we explore more Streamlit projects; these projects cover a wide range of domains and Streamlit features. Implementing them would help to reinforce what we have learned and also build a strong portfolio.

Exercises

- 1. What command is used to run a Streamlit application in a local environment?**
 - a. Streamlit run local.py
 - b. Streamlit start app.py
 - C. Streamlit run app.py
 - d. Streamlit execute app.py
- 2. What type of application architecture does Streamlit use?**
 - a. Peer-to-peer
 - b. Client-server
 - c. Monolithic
 - d. Microservices
- 3. What is NOT a key consideration when choosing a hosting service for Streamlit applications?**
 - a. Scalability
 - b. Color scheme of the interface
 - c. Security
 - d. Cost
- 4. What does Streamlit Sharing provide for deploying applications?**
 - a. Private cloud services
 - b. Public cloud services
 - c. Local hosting services
 - d. None of the above
- 5. How can Streamlit Sharing be accessed for deployment?**

- a. Through a local terminal command
- b. Via the top-right corner of the Streamlit application local run or Streamlit sharing webpage
- c. By sending an email request to Streamlit
- d. Through an FTP server

6. Where should you configure secrets for an application deployed with Streamlit Sharing?

- a. In the `app.py` file
- b. In the Streamlit cloud settings
- c. In the `requirements.txt` file
- d. Directly in the source code

7. Which of the following statements is true about Streamlit Sharing?

- a. It is the most secure deployment option available.
- b. It provides complete privacy and data protection.
- c. It is recommended for self-hosted projects, without strict data protection considerations.
- d. It does not allow for the use of secrets.

8. What platform does the chapter suggest using after Streamlit Sharing for deployment?

- a. Microsoft Azure
- b. **Google Cloud Platform (GCP)**
- c. IBM Cloud
- d. Oracle Cloud

9. What GCP service is recommended for serverless deployment of Streamlit apps?

- a. Google Compute Engine

- b. Google Kubernetes Engine
- c. Google App Engine
- d. Google Cloud Run

10. What should you do first before deploying a Streamlit app on GCP Cloud Run?

- a. Create a GCP account
- b. Buy a new domain name
- c. Optimize your local network settings
- d. Install additional RAM on your

Answers

- 1.** c
- 2.** b
- 3.** b
- 4.** b
- 5.** b
- 6.** b
- 7.** c
- 8.** b
- 9.** d
- 10.** a

1 Source: <https://example-prd-recommender.streamlit.app/>

Join our book's Discord space

Join the book's Discord Workspace for Latest updates, Offers, Tech happenings around the world, New Release and Sessions with the Authors:

<https://discord.bpbonline.com>



CHAPTER 9

Hands-On Projects: Easy

Introduction

In this chapter, we will look at beginner-friendly Streamlit projects that will help you apply the concepts and techniques you have learned so far. These projects are designed to be straightforward, allowing you to practice and build confidence with Streamlit. By the end of this chapter, you will have completed several small projects that not only enhance your understanding of Streamlit but also provide tangible examples of your skills. These projects will be valuable additions to your portfolio, demonstrating your ability to create interactive, data-driven applications.

Structure

The chapter discusses the following topics:

- Streamlit projects abstract: Easy
- Hands-on: Implementing sample project on the Weather dashboard

Objectives

We have covered the Streamlit framework's basic and advanced concepts. We have also discussed the best practices for the Streamlit project and explored various deployment options while building a hands-on project for product recommendation. In this chapter, we will explore and implement some easy Streamlit project ideas that can help you understand and practice the concepts. These projects will also help you build your GitHub portfolio.

Streamlit projects abstract: Easy

We will discuss Streamlit project ideas and implement a few of them. These projects will help you understand Streamlit's capabilities in detail. Easy projects are mostly single-page applications designed to be straightforward and minimalistic. They use a maximum of five Streamlit elements and are ideal for beginners. The projects might include basic visualizations, images, or file upload or download components. The focus is on simple data presentation and interaction, with limited interactivity or custom functionalities.

Project 1: Weather dashboard

A single-page weather dashboard that displays current weather information based on a user-provided city name. Users can input a city name, and the application will fetch and display real-time weather conditions, temperature, and other relevant details from a public API.

Implementation guide

This outlines how to build the project at a higher level:

1. Use Streamlit's text input to get the city name.
2. Fetch weather data from a public API.

3. Display the current weather conditions, temperature, and other relevant details using Streamlit widgets.

The Streamlit components used are as follows:

- `st.text_input`
- `st.write`
- `st.button`
- `st.json`

Project 2: Basic stock price viewer

The Basic Stock Price Viewer is an application to display and analyze historical trends of stock prices for a selected company. Users can input a stock symbol, and the application retrieves and visualizes the historical price data using a line chart, allowing for analysis of stock trends over time.

Implementation guide

This outlines how to build the project at a higher level:

1. Use text input to receive a stock symbol.
2. Fetch historical stock price data from an API.
3. Plot the stock price data over time using a basic line chart.

The Streamlit components used are as follows:

- `st.text_input`
- `st.line_chart`

Project 3: To-do list

This project is a to-do list application that enables users to manage their tasks efficiently. The application allows users to add, update, and delete tasks through an intuitive interface.

It keeps track of tasks and provides options to mark them as complete.

Implementation guide

This outlines how to build the project at a higher level:

1. Create a list to store tasks.
2. Allow users to add new tasks using text input and a button.
3. Display the tasks with options to mark them as complete or delete them.

The Streamlit components used are as follows:

- `st.text_input`
- `st.button`
- `st.checkbox`

Project 4: Image gallery

The Image Gallery project allows users to upload and view images in a dynamic gallery. Users can upload images through a file uploader, and the application will display the images on the page, providing an interactive and user-friendly platform for managing visual content.

Implementation guide

This outlines how to build the project at a higher level:

1. Use a file uploader to allow users to upload images.
2. Store uploaded images in a list.
3. Display the images on the page.

The Streamlit components used are as follows:

- `st.file_uploader`

- `st.image`

Project 5: Movie recommendation system

A movie recommendation system involves building a movie recommendation system that suggests movies based on user-selected genres. Users can select their preferred genres using checkboxes, and the system will recommend relevant movies, offering a personalized and engaging movie discovery experience.

Implementation guide

This outlines how to build the project at a higher level:

1. Provide a list of movie genres.
2. Use checkboxes to allow users to select genres.
3. Recommend movies based on the selected genres.

The Streamlit concepts used are as follows:

- `st.checkbox`
- `st.write`

Project 6: BMI calculator

The BMI calculator is an application that calculates and displays a user's **Body Mass Index (BMI)** based on their height and weight. Users input their physical details, and the application computes the BMI and briefly explains it.

Implementation guide

This outlines how to build the project at a higher level:

1. Use text inputs to receive height and weight.
2. Calculate BMI based on the provided values.
3. Display the calculated BMI and a brief explanation.

The Streamlit concepts used are as follows:

- `st.text_input`
- `st.write`
- `st.button`

Project 7: Simple expense tracker

This project is an expense tracker that allows users to record and monitor their daily expenses. Users can input details about their expenses, and the application will store and display a summary of their spending.

Implementation guide

This outlines how to build the project at a higher level:

1. Use text inputs to get expense details (item, cost).
2. Store the expenses in a list.
3. Display a summary of expenses and total costs.

The Streamlit concepts used are as follows:

- `st.text_input`
- `st.write`
- `st.button`

Project 8: Currency converter

The currency converter project is a tool that enables users to convert amounts between different currencies. By inputting the amount and selecting the desired currencies, users can quickly obtain the converted amount using real-time conversion rates fetched from an API.

Implementation guide

This outlines how to build the project at a higher level:

1. Use text inputs to get the amount and convert the currencies.
2. Fetch conversion rates from an API.
3. Display the converted amount.

The Streamlit concepts used are as follows:

- `st.text_input`
- `st.button`
- `st.write`

Project 9: Interactive quiz

The Interactive Quiz project provides an engaging way for users to test their knowledge through multiple-choice questions. The application displays a series of questions, and users can select their answers using radio buttons. Upon completion, the correct answers are revealed, making it a fun experience.

Implementation guide

This outlines how to build the project at a higher level:

1. Display a set of multiple-choice questions.
2. Allow users to select answers using radio buttons.
3. Show the correct answers after the quiz is completed.
4. Add celebration widgets like balloon/snowflake when the right answer is selected.

The Streamlit Concepts Used are as follows:

- `st.radio`
- `st.button`
- `st.write`
- `st.balloon`

Project 10: Random quote generator

This project involves creating a random quote generator that displays a new quote each time the user clicks a button. The application fetches quotes from a list and presents a different one each time, offering inspiration and motivation.

Implementation guide

This outlines how to build the project at a higher level:

1. Fetch a list of quotes.
2. Use a button to generate a new random quote.
3. Display the selected quote on the page.

The Streamlit concepts used are as follows:

- `st.button`
- `st.write`

Hands-on: Implementing sample project on the Weather dashboard

Let us implement *Project 1* and *Project 8* from the aforementioned easy Streamlit projects and see it in action. Essentially, all the easy-level projects are of low complexity.

Project structure for Weather data app

The project structure for the Weather data app is shown in the following figure:

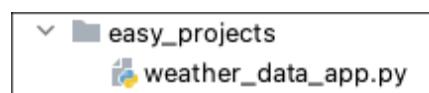


Figure 9.1: Project structure for weather data application

The code for the Streamlit `weather_data_app` is as follows:

```
import Streamlit as st
import requests

def get_weather(city_name):
    api_key = '<Enter your API key>'
    url =
f'http://api.openweathermap.org/data/2.5/weather?q={city_name}&appid={api_key}&units=metric'
    response = requests.get(url)
    data = response.json()
    return data

def main():
    st.title('Weather Dashboard')

    # Get user input for city name
    city_name = st.text_input('Enter City Name')

    # Display weather data when button is clicked
    if st.button('Get Weather'):
        weather_data = get_weather(city_name)
        if weather_data:
            st.write(f"**Location:**\n{weather_data['name']}, {weather_data['sys']['country']}")
```

```

        st.write(f"**Temperature:**\n{weather_data['main']['temp']}°C")
        
        st.write(f"**Weather:**\n{weather_data['weather'][0]['main']}")
        
        st.write(f"**Description:**\n{weather_data['weather'][0]['description']}")
        
        st.write(f"**Wind Speed:**\n{weather_data['wind']['speed']} m/s")
        
        st.write(f"**Humidity:**\n{weather_data['main']['humidity']}%")
        
    else:
        
        st.write('City not found. Please enter\na valid city name.')
        
if __name__ == '__main__':
    main()

```

Get your API key from the <https://openweathermap.org/>, as shown in the following figures:

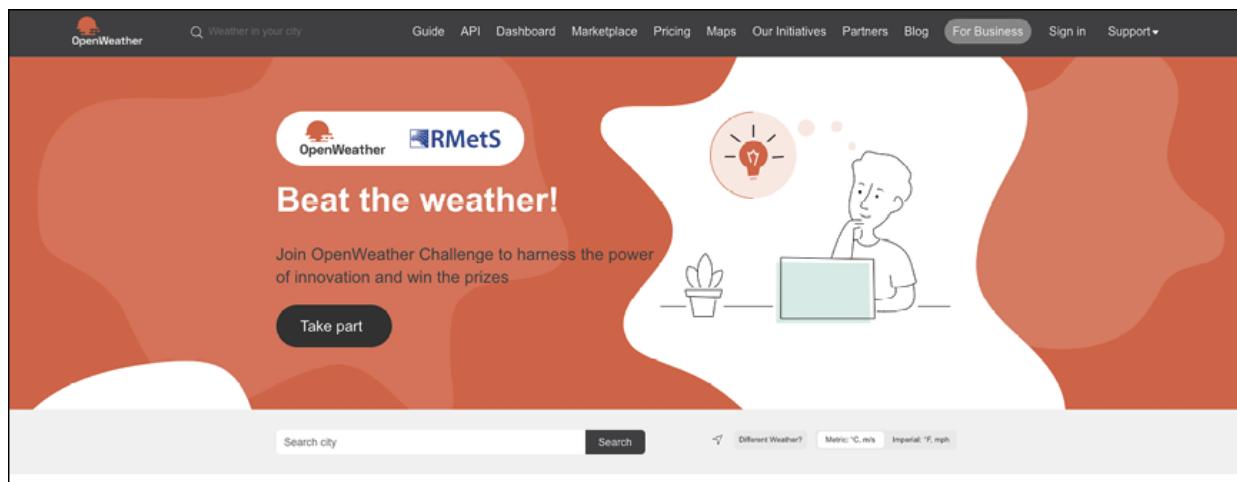


Figure 9.2: Open weather website openweathermap.org

Weather Dashboard

Enter City Name

Mumbai

Get Weather

Location: Mumbai, IN

Temperature: 33.99°C

Weather: Smoke

Description: smoke

Wind Speed: 3.09 m/s

Humidity: 46%

Figure 9.3: Weather application dashboard.

The dashboard lets you enter the name of the city and parses the information returned from the weather API on the Streamlit page.

Project structure for currency converter data app

The project structure for the currency converter data app is shown in the following figure:



Figure 9.4: Currency converter application.

The code for the currency converter application is as follows:

```
import requests  
import Streamlit as st
```

```
def convert_currency(amount, from_currency,
to_currency, api_key):

    url = f"https://v6.exchangerate-
api.com/v6/{api_key}/pair/{from_currency}/{to_curre
ncy}/{amount}"

    response = requests.get(url)
    data = response.json()
    if data['result'] == 'success':
        return data['conversion_result']
    else:
        return "Error: " + data['error-type']

st.title("Currency Converter")

api_key = '<Enter your API key>'

amount = st.text_input("Amount to convert", "1")
from_currency = st.text_input("From Currency (e.g.,
USD)", "USD")
to_currency = st.text_input("To Currency (e.g.,
EUR)", "EUR")

if st.button('Convert'):

    result = convert_currency(amount,
from_currency.upper(), to_currency.upper(),
api_key)
```

```

        if isinstance(result, str) and
result.startswith("Error"):

    st.error(result)

else:

    st.success(f"{amount} {from_currency} is
{result} {to_currency}")

```

Get your API key <https://www.exchangerate-api.com/>, as shown in the following figures:

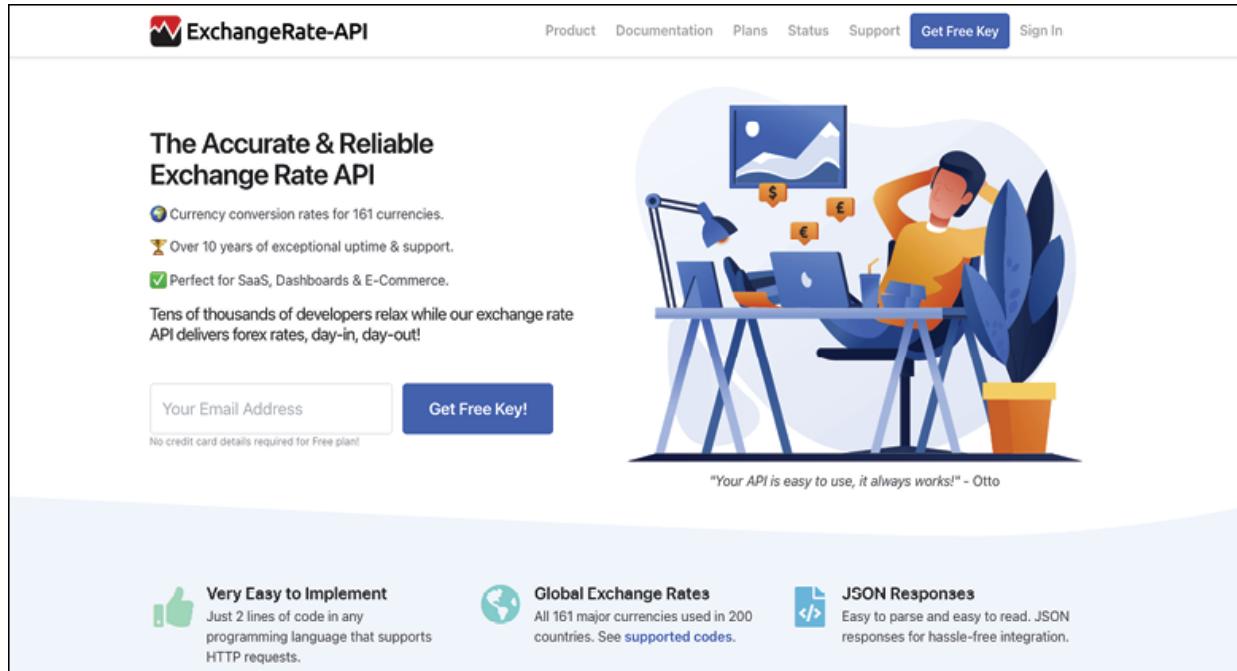


Figure 9.5: Exchange Rate API can be generated from [exchangerate-api.com](https://www.exchangerate-api.com/)

Currency Converter

Amount to convert
2000

From Currency (e.g., USD)
USD

To Currency (e.g., EUR)
INR

Convert

2000 USD is 166943.2 INR

Figure 9.6: Currency Converter Data application

Conclusion

In this chapter, we discussed and implemented the easy Streamlit projects. These projects are mostly simple single-page applications that are implemented with basic Streamlit components. They provide a good starting point to start experimenting with Streamlit components. The project abstracts shared in this chapter should serve as a starting point and can be built upon as complete applications by adding complex Streamlit components.

In the next chapter, we will raise the bar of complexity and explore intermediate Streamlit projects, which would use more Streamlit features than basic projects.

CHAPTER 10

Hands-On Projects:

Intermediate

Introduction

In this chapter, we will explore and implement projects that require a deeper understanding of Streamlit's capabilities. These intermediate projects are designed to help you practice more complex functionalities and enhance your portfolio with more sophisticated applications. These projects will be excellent additions to your GitHub portfolio, highlighting your growing expertise in building interactive, data-driven web applications.

Structure

The chapter discusses the following topics:

- Streamlit project abstracts: Intermediate
- Hands-on: Implementing sample project

Objectives

In this chapter, we will explore and implement some intermediate Streamlit project ideas, which can help you understand and practice the concepts. These projects will also help you build your GitHub portfolio and develop your understanding of the depth of usage of Streamlit components.

Streamlit project abstracts: Intermediate

We will discuss Streamlit project ideas and implement a few of them. These projects will help to understand the Streamlit's capabilities in more detail. Intermediate projects are a little more advanced than the easy projects that we covered in the last chapter; these applications will use more interactions and components. The projects might include basic visualizations, images, or file upload/download components. The focus is building streamlined data applications with enhanced interaction and complex operability.

Project 1: Multi-page data analysis dashboard

A data dashboard with multiple pages to analyze and visualize datasets. Each page could focus on a specific metric that is useful for the user.

Implementation guide

1. Use Streamlit's page navigation features to create multiple pages.
2. On each page, present a different dataset and create interactive charts and graphs.
3. Add filters to allow users to customize the data view.

The Streamlit concepts used are as follows:

- `st.sidebar.selectbox`

- `st.line_chart`
- `st.bar_chart`
- `st.pyplot`
- `st.write`

Project 2: Job application tracker

A multi-page job application tracker to manage job applications and their status.

Implementation guide

The implementation guide is as follows:

1. Create pages for different stages of the job application process (for example, application, interviewing, offers).
2. Use input forms to add job applications and update their status.
3. Include advanced UI/UX elements for intuitive tracking and data visualization.

The Streamlit concepts used are as follows:

- `st.sidebar`
- `st.text_input`
- `st.selectbox`
- `st.write`
- `st.bar_chart`

Project 3: Budget planner with visualizations

A budget planner with separate pages for income, expenses, and summary visualizations. The application could be connected to the backend file-based database to track the budgets and user profiles.

Implementation guide

The implementation guide is as follows:

1. Use multiple pages to input income and expense data.
2. Implement features to categorize expenses and set budget goals.
3. Create visualizations to display income versus expenses and savings.

The Streamlit concepts used are as follows:

- `st.sidebar.selectbox`
- `st.text_input`
- `st.write`
- `st.bar_chart`
- `st.pie_chart`

Project 4: Personal fitness tracker

A multi-page fitness tracker to record workouts, track progress, and set goals. The application could be linked to personal tracker devices like Fitbit to gather daily health statistics.

Implementation guide

The implementation guide is as follows:

1. Create pages for workout routines, progress tracking, and goal setting.
2. Allow users to log workouts with details like type, duration, and calories burned.
3. Use charts to visualize fitness progress over time.

The Streamlit concepts used are as follows:

- `st.sidebar.radio`

- `st.text_input`
- `st.write`
- `st.line_chart`

Project 5: Multi-page portfolio website

A personal portfolio website with separate pages for projects, experience, and contact information. This application could serve as a resume or curriculum vitae and be shared with future collaborators.

Implementation guide

The implementation guide is as follows:

1. Create pages for different sections of the portfolio.
2. Include options to add project details, work experience, and contact information.
3. Add interactive elements to enhance user engagement.

The Streamlit concepts used are as follows:

- `st.sidebar`
- `st.write`
- `st.file_uploader`
- `st.button`

Project 6: Task management application

A multi-page task management application to create, update, and organize tasks.

Implementation guide

The implementation guide is as follows:

1. Create pages for different task categories (for example,

personal, work, urgent).

2. Allow users to add tasks with various details, including priority and deadlines.
3. Implement features for task filtering and sorting.

The Streamlit concepts used are as follows:

- `st.sidebar.selectbox`
- `st.text_input`
- `st.button`
- `st.write`
- `st.checkbox`

Project 7: Multi-page product catalog

A product catalog application with separate pages for different product categories.

Implementation guide

The implementation guide is as follows:

1. Create pages for various product categories.
2. Allow users to upload product images and details.
3. Implement search functionality to find products by name or category.

The Streamlit concepts used are as follows:

- `st.sidebar`
- `st.text_input`
- `st.file_uploader`
- `st.button`

Project 8: Sports team management application

A multi-page application to manage sports teams, track player statistics, and organize matches.

Implementation guide

The implementation guide is as follows:

1. Create pages for team details, player statistics, and match schedules.
2. Allow users to add players and update their stats.
3. Include features for tracking match results and team performance.

The Streamlit concepts used are as follows:

- `st.sidebar.selectbox`
- `st.text_input`
- `st.write`
- `st.button`

Project 9: Multi-page event planner

An event planning application with separate pages for event details, guest lists, and budget planning.

Implementation guide

The implementation guide is as follows:

1. Create pages for different aspects of event planning.
2. Allow users to add event details, create guest lists, and set budgets.
3. Implement visualizations for budget tracking and event timelines.

The Streamlit concepts used are as follows:

- `st.sidebar`

- `st.text_input`
- `st.write`
- `st.bar_chart`

Project 10: Multi-page recipe organizer

A recipe organizer with separate pages for different recipe categories and a feature to create shopping lists.

Implementation guide

The implementation guide is as follows:

1. Create pages for various recipe categories (for example, appetizers, main courses, and desserts).
2. Allow users to add and categorize recipes.
3. Implement a shopping list feature to generate lists based on selected recipes.

The Streamlit concepts used are as follows:

- `st.sidebar`
- `st.text_input`
- `st.write`
- `st.file_uploader`
- `st.button`

Hands-on: Implementing sample project

Let us implement Project 5 from the aforementioned Intermediate Streamlit projects and see it in action. This project is a portfolio website that highlights the experience, skills, and the project. It can be used to share your skillset with potential recruiters or project sponsors.

Streamlit focuses on providing simple formatting and design elements. We will use the available Streamlit components to build this minimalist portfolio application.

This is a multipage application, and it contains three pages:

- **Home page**: This page is the landing page of the portfolio.
- **Portfolio page**: This page contains the skills, experience, education, certification, and detailed curriculum vitae.
- **Contact page**: This page is to reach out to the person owning the portfolio.

We also have additional assets that would be used, like the professional headshot for the introduction and the detailed curriculum vitae, which can be downloaded from the portfolio page.

The project structure for the portfolio app is shown in the following figure:

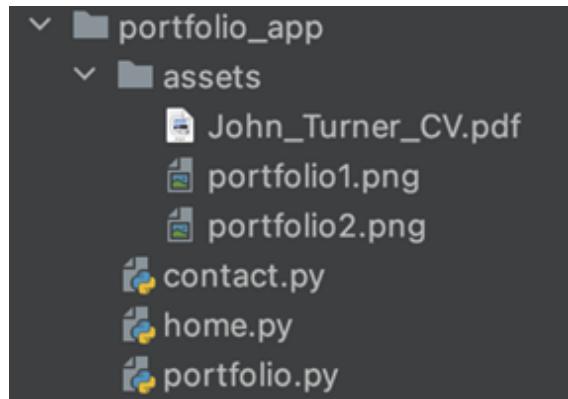


Figure 10.1: Project structure for Portfolio application

Let us start with the home page for the application:

```
import Streamlit as st  
from portfolio import *
```

```
from contact import *

def home_show():
    col1, col2, col3, col4 = st.columns((1,1,1,1))

    with col2:
        st.title("HELLO !")
        st.header("I am John Turner")
        st.write("""
                    I am a passionate data
                    professional focused on building stories from the
                    data. I love analyzing the data, cleaning it and
                    looking for patterns.

                    I have considerable experience
                    in building complex pipelines for Data ingestion
                    and modeling. I have worked with numerous business
                    and Creative stakeholders.

                    I believe every Dataset has a
                    story to be told
                """
)
        st.write("""
                    - **My Key Skills are**
                    - Data Exploration
                    - Data Modeling
                    - Building Ingestion
                    pipeline
                """
```

```
- Business Intelligence  
  
- **I love to do**  
  - Photography  
  - Travel  
  - Reading  
"""")  
  
st.markdown("[LinkedIn]  
(https://www.linkedin.com/in/johnturner)[|Twitter|]  
(https://www.twitter.com/johnturner) [GitHub]  
(https://www.github.com/johnturner) [|Instagram]  
(https://www.instagram.com/johnturner)")  
  
with col3:  
    base_dir = os.path.dirname(__file__)  
    image_path = os.path.join(base_dir,  
'assets', 'portfoliol.png')  
    st.image(image_path, width=500)  
  
st.set_page_config(page_title='John Turner  
Portfolio', layout='wide')  
  
if __name__ == "__main__":  
    tab1, tab2, tab3 = st.tabs(["Home", "Portfolio",  
"Contact"])  
  
    with tab1:
```

```
home_show()

with tab2:

    portfolio_show()

with tab3:

    contact_show()
```

We have added the tabs to other pages to add interactivity from the home page. Please note that Streamlit, by default, does not provide text alignment options, so we have used a column to align the text and other elements.

Please note the portfolio images are taken from Canva stock photos and are to be used for understanding and educational purposes.

Next, we will build the portfolio page. The following is the sample code for the portfolio page.

```
import Streamlit as st

import os

def portfolio_show():

    st.title('Portfolio')

    col1, col2 = st.columns(2)

    with col1:

        with st.expander("Experience & Awards"):

            st.write("""
                - 5+ Years of experience
            """)
```

```
        - Specialized in optimizing and  
deploying robust prediction models  
Specialization  
        - Consumer Marketing  
cohort 2020  
        - Top Candidate in Eucley's  
        """")
```

```
with col2:  
    with st.expander("Education"):  
        st.write("""  
        - Post graduation in Data  
Science  
        - Visualization degree from  
Courera  
        - Advance Data modeling course  
in Udemy  
        - Cloud Computing Certification  
        """")
```

```
col1, col2, col3 = st.columns((1,2,1))  
  
with col2 :  
    with st.expander("My favorite Projects"):  
        st.write("""
```

```
        - Building a Master Data Layer  
for Analytics and Data Science  
  
        - Building a self-serve data  
analytics platform for the organization  
  
        - Building customer acquisition  
and churn prediction model  
  
        - On-going contributor to open-  
source library  
    """)  
  
    col1, col2, col3 = st.columns((1,2,1))  
  
    with col2:  
  
        base_dir =  
os.path.dirname(__file__)  
  
        cv_path = os.path.join(base_dir,  
'assets', 'John_Turner_CV.pdf')  
  
        if os.path.exists(cv_path):  
  
            with open(cv_path, "rb") as  
file:  
  
                btn = st.download_button(  
                    label="Download CV",  
                    data=file,  
  
file_name="John_Doe_CV.pdf",  
                    mime="application/pdf"  
                )  
  
            else:
```

```
    st.write("Reach out to me to  
get my CV")
```

This page contains the downloaded link to the CV. Please note the CV is generated using ChatGPT and is only for sample and understanding purposes.

Let us move on to the next page, which is the contact page. This page contains the contact form, with a callback action calling the `send_email` function. This function can be implemented using the `MIMEText` and `MIMEMultipart` libraries.

This is a sample code for the contact page:

```
import Streamlit as st  
  
def contact_show():  
  
    st.title('Contact Me')  
  
    with st.form(key='contact_form'):  
        name = st.text_input('Name')  
        email = st.text_input('Email')  
        message = st.text_area('Message')  
        submit_button =  
        st.form_submit_button('Send')  
  
        if submit_button:  
            if len(name) < 2:  
                st.warning('Name too short.')  
            elif '@' not in email:
```

```
        st.warning('Please enter a valid  
email address.')  
  
    else:  
  
        st.success('Thanks for your  
message!')  
  
    col1, col2, col3,col4 , col5 = st.columns(5)  
  
    with col3:  
  
        st.markdown(  
            "[LinkedIn]  
(https://www.linkedin.com/in/johnturner) [|Twitter|]  
(https://www.twitter.com/johnturner) [GitHub]  
(https://www.github.com/johnturner) [|Instagram]  
(https://www.instagram.com/johnturner)")
```

Let us run the application and check our results. The following is the sample run for the home page for the portfolio application:

Home Portfolio Contact

HELLO !

I am John Turner

I am a passionate data professional focused on building stories from the data. I love analyzing the data, cleaning it and looking for patterns. I have considerable experience in building complex pipelines for Data ingestion and modeling. I have worked with numerous business and Creative stakeholders. I believe every Dataset has a story to be told

- My Key Skills are
 - Data Exploration
 - Data Modeling
 - Building Ingestion pipeline
 - Business Intelligence
- I love to do
 - Photography
 - Travel
 - Reading

[LinkedIn](#) | [Twitter](#) | [GitHub](#) | [Instagram](#)



Figure 10.2: Home page of introduction

The following is the sample page for the next tab in the portfolio application:

Home Portfolio Contact

Portfolio

Experience & Awards

- 5+ Years of experience
- Specialized in optimizing and deploying robust prediction models
- Consumer Marketing Specialization
- Top Candidate in Euclay's cohort 2020

Education

- Post graduation in Data Science
- Visualization degree from Courera
- Advance Data modeling course in Udemy
- Cloud Computing Certification

My favorite Projects

- Building a Master Data Layer for Analytics and Data Science
- Building a self serve data analytics platform for the organization
- Building customer acquisition and churn prediction model
- On-going contributor to open-source library

[Download CV](#)

Figure 10.3: Portfolio Page with experience, education, projects, and the option to download the CV

The following figure shows the sample page for the contact page for the portfolio application:

The screenshot shows a Streamlit application interface. At the top, there is a navigation bar with three items: 'Home', 'Portfolio', and 'Contact'. The 'Contact' item is underlined, indicating it is the active page. Below the navigation bar, the title 'Contact Me' is displayed in a bold, black font. Underneath the title are three input fields: 'Name', 'Email', and 'Message', each with a corresponding placeholder text. A 'Send' button is located at the bottom left of the form area. At the very bottom of the page, there is a row of four small blue hyperlinks: LinkedIn, Twitter, GitHub, and Instagram.

Figure 10.4: Contact Page with contact form and links to the external media accounts

The sample applications are for reference and as the baseline. These applications serve as a boilerplate code and should be used as a starting point for your application. Experiment with different available options, layouts, and containers to further expand your Streamlit expertise.

Conclusion

In this chapter, we discussed and implemented an intermediate Streamlit project. These projects are a little more complicated than easy projects but more satisfying. They are more interactive and explore slightly advanced features of the framework. It is important to use your creativity and add enhancements using Streamlit.

In the next chapter, we will take our exploration to the next level with advanced projects. These projects will challenge you with more sophisticated features, including integration with external APIs and custom components. By looking at

these advanced topics, you will push the boundaries of what can be achieved with Streamlit, equipping you with the skills to create highly interactive and functional applications.

Join our book's Discord space

Join the book's Discord Workspace for Latest updates, Offers, Tech happenings around the world, New Release and Sessions with the Authors:

<https://discord.bpbonline.com>



CHAPTER 11

Hands-On Projects:

Advanced

Introduction

In the final hands-on projects chapter, we will discuss projects with more complex use cases and structures than the previous chapters. These advanced projects are designed to deepen your understanding of Streamlit, enabling you to build sophisticated, multi-faceted applications that can handle real-time data, secure sensitive information, and deploy machine learning models.

These projects will require you to use a range of Streamlit components and discover creative ways to combine them to solve real-world problems.

Structure

The chapter discusses the following topics:

- Streamlit projects abstract: Advanced
- Hands-on: Implementing sample project on News Aggregator

Objectives

In this chapter, we will explore and implement some advanced Streamlit project ideas, which can help you understand and practice the concepts we have learned in the previous chapters in depth. These projects will also help you build your GitHub portfolio and develop your understanding of the depth of usage of Streamlit components.

Streamlit projects abstract: Advanced

We will discuss Streamlit project ideas and implement a few of them. Advanced projects are more advanced than the intermediate and easy projects that we covered in the last chapter. These applications use more interactions and components. The focus is on building streamlined data applications with enhanced interaction and complex operability.

Project 1: Real-time data analytics dashboard

This project involves creating a real-time data analytics dashboard that continuously updates and displays complex visualizations. The application connects to a data source providing real-time updates, such as stock market data or IoT device outputs, and uses caching to optimize data retrieval. Advanced charts and visualizations present the data dynamically.

Implementation guide

The implementation guide is as follows:

1. Implement caching to optimize data retrieval.
2. Connect to a data source that provides real-time updates (for example, stock market, IoT devices).

3. Use advanced charts and visualizations to display real-time data.

The Streamlit concepts used are as follows:

- `st.cache`
- `st.pyplot`
- `st.line_chart`
- `st.bar_chart`
- `st.sidebar`
- `st.selectbox`

Project 2: Secure financial portfolio manager

The Secure Financial Portfolio Manager is a multi-page application designed to manage a financial portfolio while ensuring data security through secret management. Users can securely store sensitive information and connect to external financial services to retrieve and display data across various portfolio aspects, including stocks, bonds, and cryptocurrencies.

Implementation guide

The implementation guide is as follows:

1. Use secret management to store sensitive information securely.
2. Connect to external financial services for data retrieval.
3. Implement multiple tabs for different portfolio aspects (for example, stocks, bonds, crypto).

The Streamlit concepts used are as follows:

- `st.secrets`
- `st.sidebar`

- `st.multiselect`
- `st.write`
- `st.line_chart`

Project 3: Machine learning model deployment

This project involves deploying machine learning models within a multi-page application, allowing users to make predictions based on input data. The application supports pre-trained models, uses caching for faster model loading, and provides visualizations to display predictions and performance metrics. This project could have similar implementation like Product Recommender application.

Implementation guide

The implementation guide is as follows:

1. Deploy pre-trained machine learning models to make predictions based on user input.
2. Use caching for faster model loading.
3. Implement visualizations to display model predictions and metrics.

The Streamlit concepts used are as follows:

- `st.cache`
- `st.sidebar`
- `st.file_uploader`
- `st.write`
- `st.pyplot`

Project 4: Customizable news aggregator

The Customizable news aggregator is a multi-page application that fetches and displays news from various

sources according to user-defined preferences. Users can customize their news feed by selecting preferred sources and topics, while caching optimizes data retrieval. We will explore the in-depth implementation in the next section.

Implementation guide

The implementation guide is as follows:

1. Implement multiple pages for news categories (e.g., technology, sports, entertainment).
2. Allow users to customize news sources and topics.
3. Use caching to optimize news data retrieval and storage.

The Streamlit concepts used are as follows:

- `st.cache`
- `st.sidebar`
- `st.selectbox`
- `st.write`
- `st.file_uploader`
- `st.button`

Project 5: Advanced time series forecasting application

This project involves building an advanced time series forecasting application with multi-page navigation for different forecasting aspects, including data input, model training, and forecasting results. The application connects to external data sources to handle large datasets and uses caching to optimize data processing.

Implementation guide

The implementation guide is as follows:

1. Implement multi-page navigation for different forecasting aspects (e.g., data input, model training, forecasting).
2. Use external data connectors to retrieve large datasets for forecasting.
3. Implement caching to optimize data retrieval and model training.

The Streamlit concepts used are as follows:

- `st.cache`
- `st.sidebar`
- `st.file_uploader`
- `st.write`
- `st.pyplot`
- `st.line_chart`

Project 6: Interactive Geographic Information System

The Interactive Geographic Information System (GIS) project is a multi-page application that presents advanced interactive maps and integrates external geographic data sources. Users can explore different geographic data layers across multiple pages and interact with maps to gain insights into geographic information.

Implementation guide

The implementation guide is as follows:

1. Implement multiple pages for different geographic data layers.

2. Use external data connectors to retrieve geographic data.
3. Implement advanced interactive maps to display geographic information.

The Streamlit concepts used are as follows:

- `st.sidebar`
- `st.selectbox`
- `st.write`
- `st.pyplot`
- `st.map`
- `st.cache`

Hands-on: Implementing sample project on news aggregator

Let us implement *Project 4* from the above advanced Streamlit projects and see it in action. This project is a news aggregator, and we will use Streamlit session state and caching to make the interaction more dynamic and enhance user experience. In the previously explained intermediate project, we focused on building a clean, minimalist design; for this project, we will focus on interactivity.

A news aggregator application lets the user select the category and language for the news. The summary of the news will be displayed on the main page, with an option to click and read the original article on the news website. We will use newsapi.org to get the news dump. This API provides a free developer limit and is easy to configure.

This is a multi-page application. It contains the following two pages:

- **News page:** This page is the landing page of the news aggregator
- **Read Again page:** On this page, the user can read articles that have been marked as favorites and later remove them from the favorite list.

We will use Streamlit caching to ensure the data is not loaded for every action performed. We will use the `session_state` to track the favorite article list, so for every session the user uses the application, his favorite list persists in the session state and can be referred to again. We will also use the Streamlit rerun function to load the application page every time the favorite article is read.

The next step to enhance this could be to use a file database like SQLite to create and store the user profile with login details. This would enable a user to log in when starting the application, and their past state could be retrieved from the file storage. This could make the user experience more streamlined and consistent throughout their usage.

The project structure for the portfolio app is shown in the following figure:

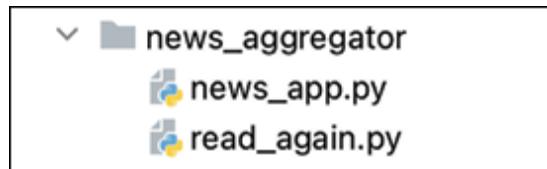


Figure 11.1: Project structure for news aggregator application

Let us start with the news home page for the application. The code is as follows:

```

import Streamlit as st
import requests
from read_again import *
  
```

```
API_KEY = 'xx'

BASE_URL = 'https://newsapi.org/v2/top-headlines'

@st.cache_data
def load_news(category, language):
    return get_news(category, language)
def get_news(category, language='en'):
    url = f"{BASE_URL}?category={category}&language={language}&apiKey={API_KEY}"
    try:
        response = requests.get(url)
        data = response.json()

        if data.get('status') == 'ok':
            articles = [article for article in
data['articles'] if article['title'] != '[Removed]']
            return articles
        else:
            st.error(f"API returned an error: {data.get('message', 'No error message provided')}")
```

```
    return []

def toggle_favorite(title):
    if title in st.session_state.favorites:
        st.session_state.favorites.remove(title)
    else:
        st.session_state.favorites.append(title)

def news_show():
    st.title("Daily News Digest")
    st.write('----')
    st.sidebar.title('Language')
    languages = {'English': 'en', 'Spanish': 'es',
    'French': 'fr', 'German': 'de'}
    selected_language =
    st.sidebar.selectbox('Select language',
    options=list(languages.keys()), index=0)
    st.sidebar.title('News Categories')
    categories = ['general', 'business',
    'entertainment', 'health', 'science', 'sports',
    'technology']
    selected_category =
    st.sidebar.selectbox('Select a category',
    categories)

    news_items = load_news(selected_category,
    languages[selected_language])
```

```
for article in news_items:

    col1, col2 = st.columns((4,1))

    with col1 :

        if article['title'] != '[Removed]':
            st.write(f"### {article['title']}")

            if article['description']:
                st.write(article['description'])

                if article['urlToImage']:
                    st.image(article['urlToImage'],
                             use_column_width=True)

                    with st.expander('More Details'):

                        st.write(f"**Author:**\n{article.get('author', 'N/A')}")

                        st.write(f"**Source:**\n{article['source']['name']}")

                        st.write(f"**Published At:**\n{article['publishedAt']}")

                if 'video' in article:
                    st.video(article['video'])

            st.markdown(f"[Read more]\n({article['url']}))", unsafe_allow_html=True)

    with col2 :
```

```

        if st.button('Favorite',
key=f"fav_{article['title']}"):

            toggle_favorite(article)

st.write("---")

if __name__ == "__main__":
    tab1, tab2= st.tabs(["News", "Read again"])
    with tab1:
        news_show()
    with tab2:
        read_again()

```

The following is the news API home page; this website provides the API endpoint, which can be used to get news updates:

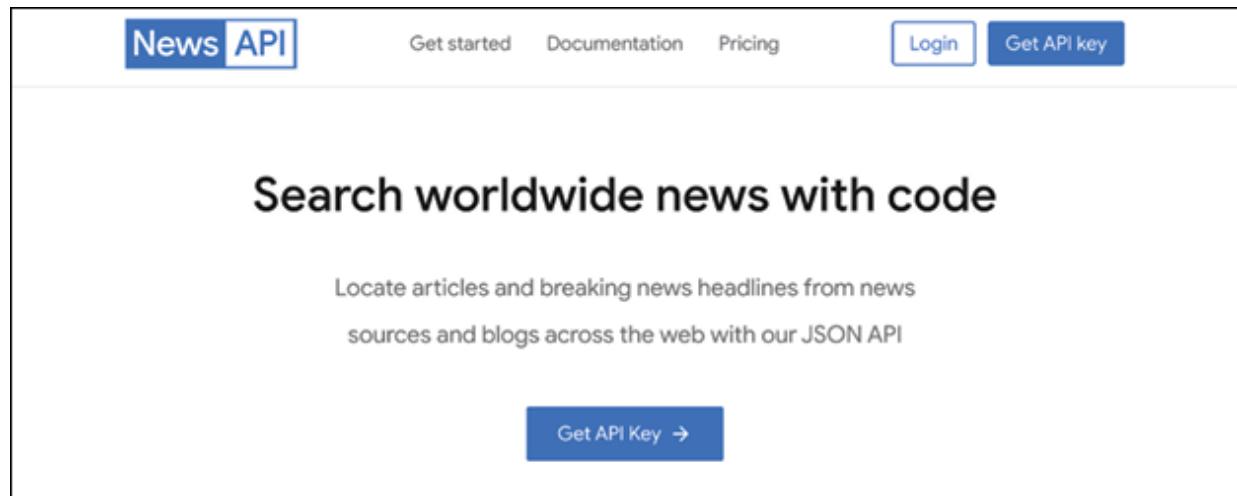


Figure 11.2: News API home page to get the API key

The home page of the news application contains a sidebar with an option to select the news's language and category to

be displayed. The news is requested from newsapi.org; the developer needs to request the API key from the website. As this page is the application's landing page, the tabs for interaction are added to this page, as well as respective functions to access the pages. To make the application robust, we have added error handling and various edge cases to trap exceptions based on the response returned from the API. The favorite button was added to choose the respective article as a favorite.

Next, we will build the Read Again page using the following code:

```
import Streamlit as st

# Function to remove the articles from favorites
def remove_fromFavorites(article):
    if article in st.session_state.favorites:
        st.session_state.favorites.remove(article)
        st.rerun()

# Function for the read again page
def readAgain():
    if 'favorites' not in st.session_state:
        st.session_state.favorites = []
        st.info('No favorite articles added.')
    return

    if not st.session_state.favorites:
```

```
        st.info('No favorite articles added.')
        return

    else:
        for article in st.session_state.favorites:
            col1, col2 = st.columns((4, 1))
            with col1:
                if article.get('title') and
article['title'] != '[Removed]':
                    st.write(f"###{article['title']}")

                    if article.get('description'):

                        st.write(article['description'])

                        if article.get('urlToImage'):

                            st.image(article['urlToImage'],
use_column_width=True)

                            with st.expander('More
Details'):

                                st.write(f"**Author:** {article.get('author', 'N/A')}")

                                st.write(f"**Source:** {article['source'].get('name', 'N/A')}")

                                st.write(f"**Published
At:** {article.get('publishedAt')}")
```

```

        if 'video' in article:

    st.video(article['video'])

    with col2:

        if st.button('Read',
key=f"read{article['title']}"):

            remove_fromFavorites(article)

    st.write('----')

```

This page lists the articles that have been added to your favorite list for reading again. Once the article is read and the Read button is clicked, it is removed from the favorite list. Once the article is removed from the favorite list, we call the Streamlit rerun function to stop any further interaction on the page and load the page again to get the refreshed list of articles.

Next, let us see the news aggregator application in action:

The screenshot shows the Streamlit application interface. On the left, there's a sidebar with 'Language' set to English and a 'News Categories' dropdown showing 'general' selected. The main content area has a header 'Daily News Digest'. Below it, a news item is displayed with the title 'Conception captain gets 4 years in prison - KTLA Los Angeles'. The text of the article is partially visible, mentioning the captain of a Southern California dive ship sentenced to four years in prison for killing 34 people. At the bottom of the news item is a small 'Favorite' button. To the right of the news item is a large, dark image of the sunken ship 'Conception'.

Figure 11.3: The news home page with the drop-down in the sidebar to select the language and category

Currently, no articles are added to the favorite list, as shown in the following figure:

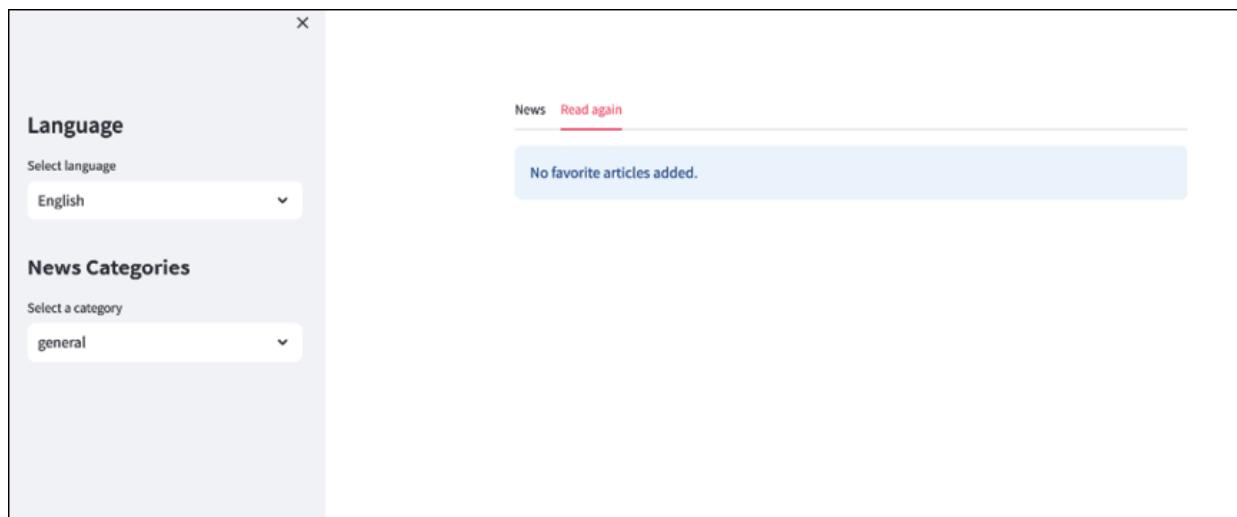


Figure 11.4: Currently, no articles are added as favorites, and hence, the Read Again page has no list

Let us add some articles to your favorite list for later reading. The step has been highlighted in the following figure:

Language

Select language

English

News Categories

Select a category

sports

Daily News Digest

Adrian Newey says he has "no plan" for next F1 challenge - Motorsport.com

Adrian Newey insists he currently has "no plan" to join one of Red Bull's Formula 1 rivals, admitting he needs "a bit of a break".



More Details

Figure 11.5: For selected category sports, the first article is added as a favorite
Select the technology category in the sidebar and add the article as a favorite:

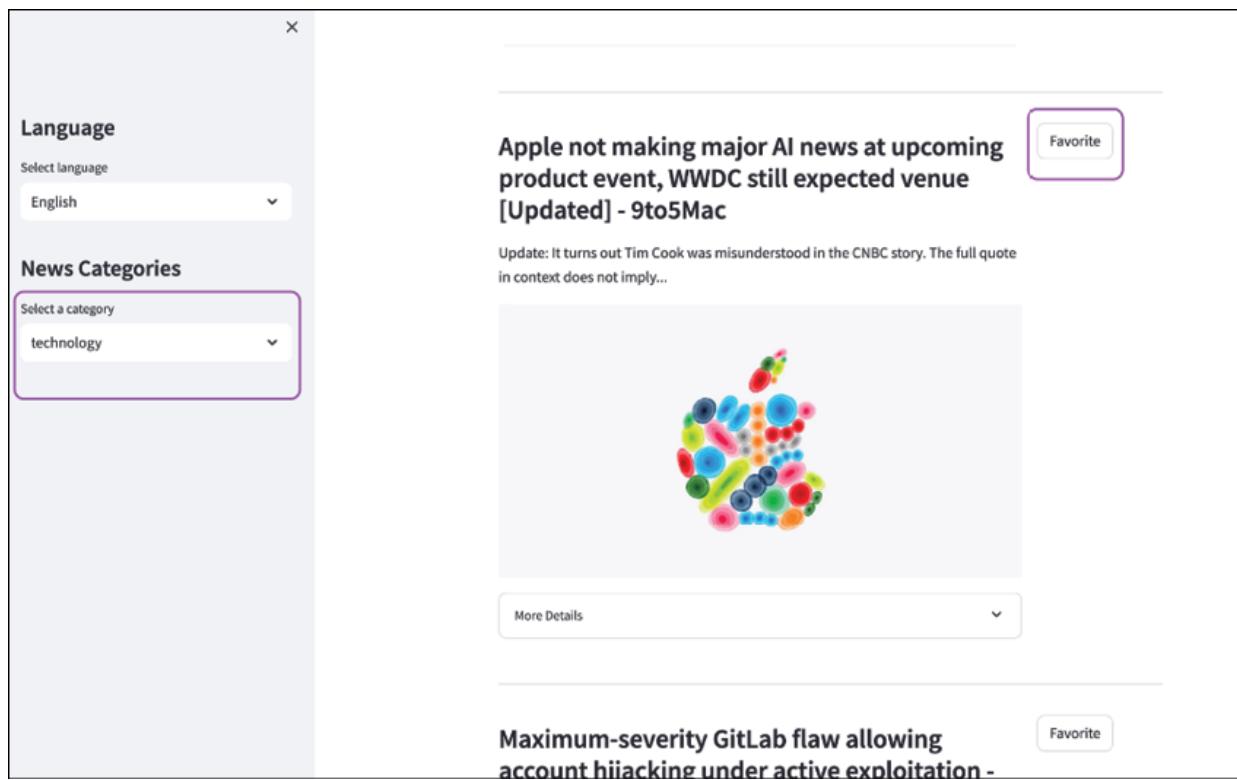


Figure 11.6: For selected category technology, the second article is added as a favorite

Once the articles are added to the favorite list, they will be displayed on the Read Again page and can be referenced later, as highlighted in the following figure:

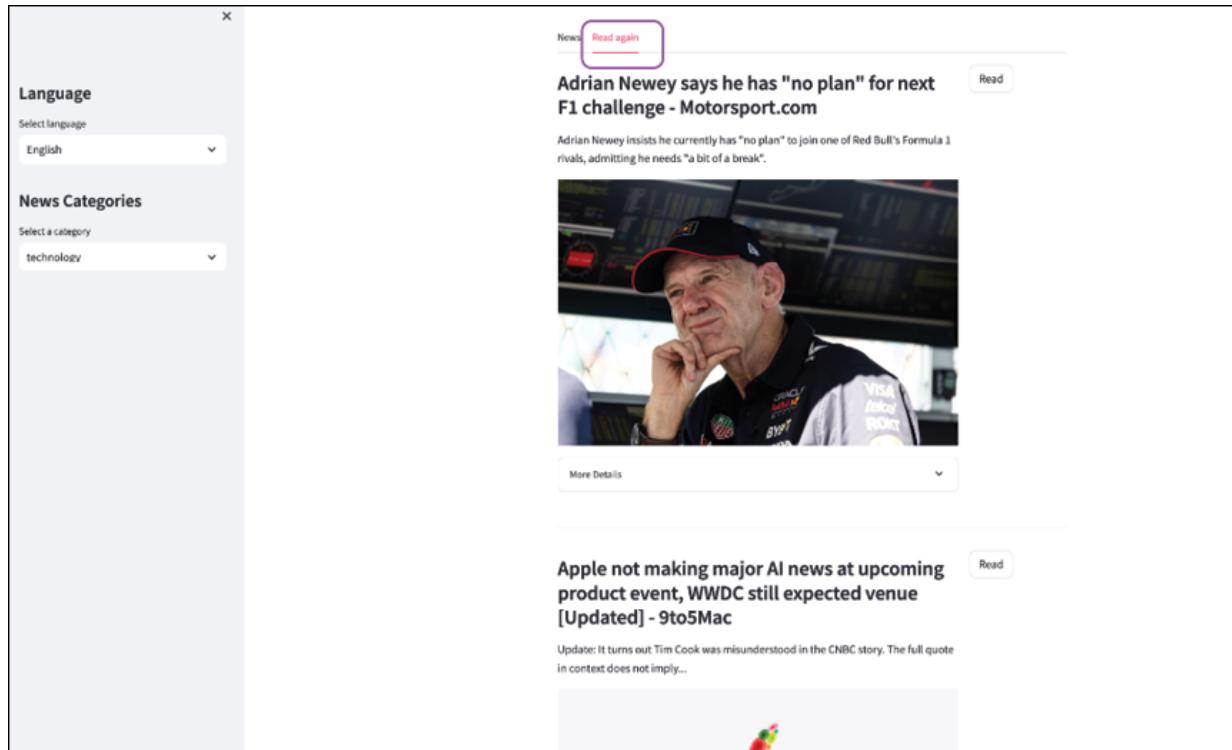


Figure 11.7: Both articles are added to the Read Again page.

Once the articles are read, the user can click on the read button to remove the articles from the favorite list.

When the **Read** button is clicked, the page reloads and refreshes immediately:

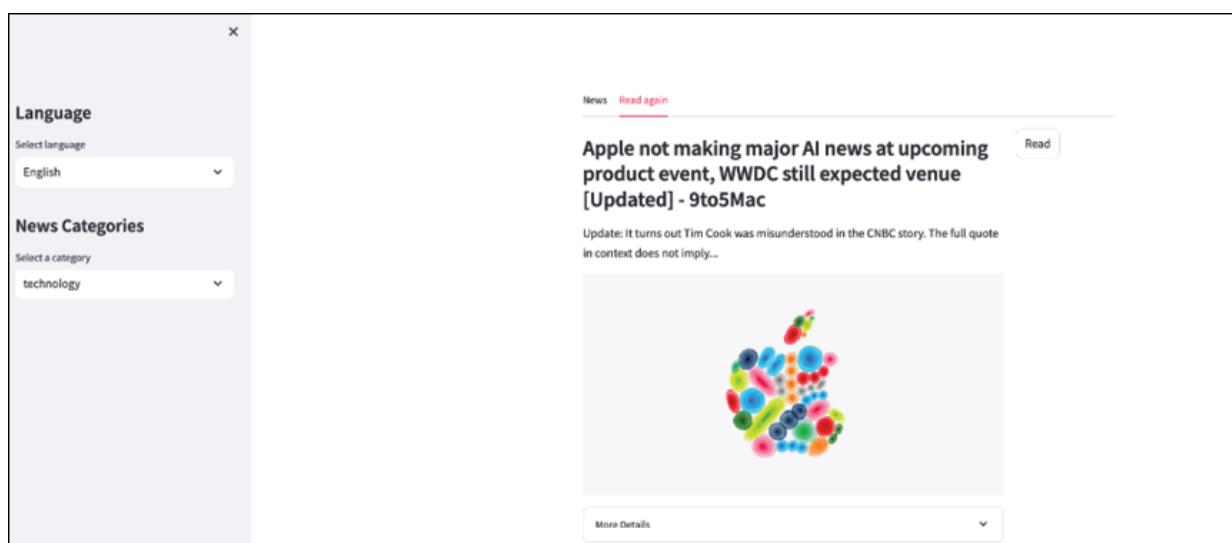


Figure 11.8: The read article is removed from the list.

If the user wants to get details on the article or read the original source, they can click on the **More Details** expander and extract more details from the API response. The details also contain the original news publisher link with the entire article:

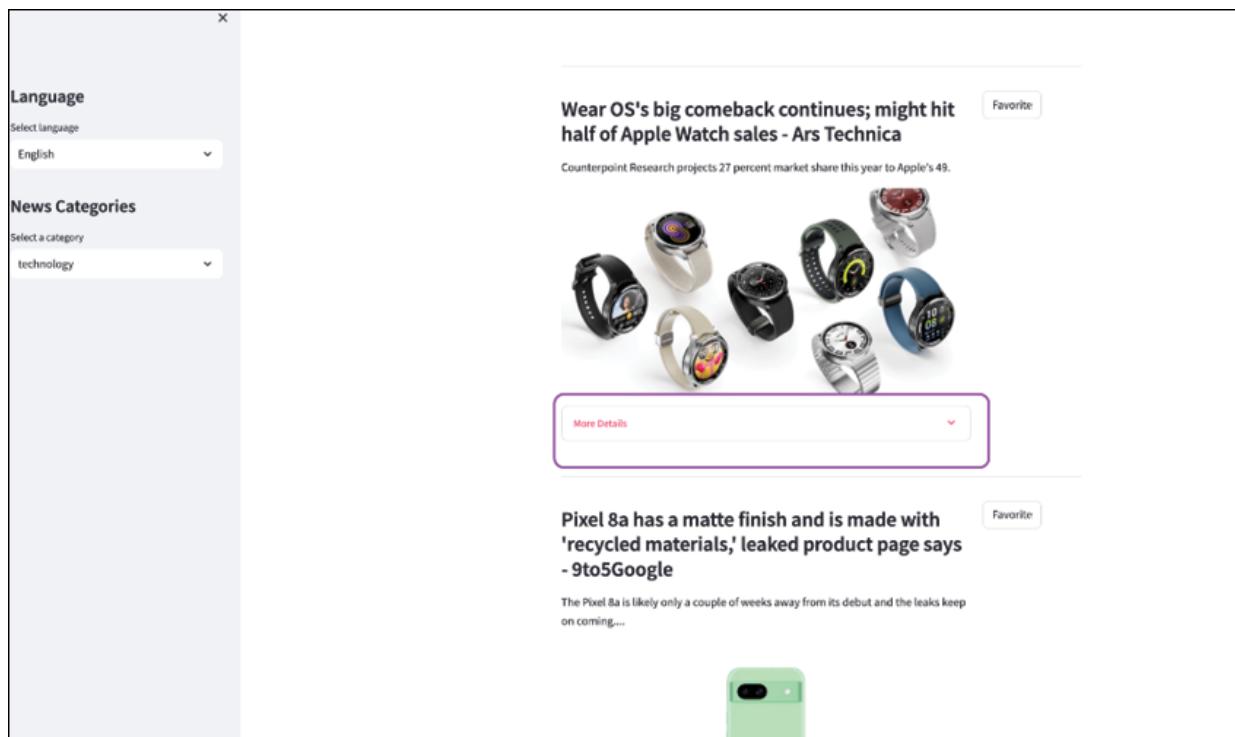


Figure 11.9: More details expander for each article

Following is the expander in action, which contains the details or metadata for the article:

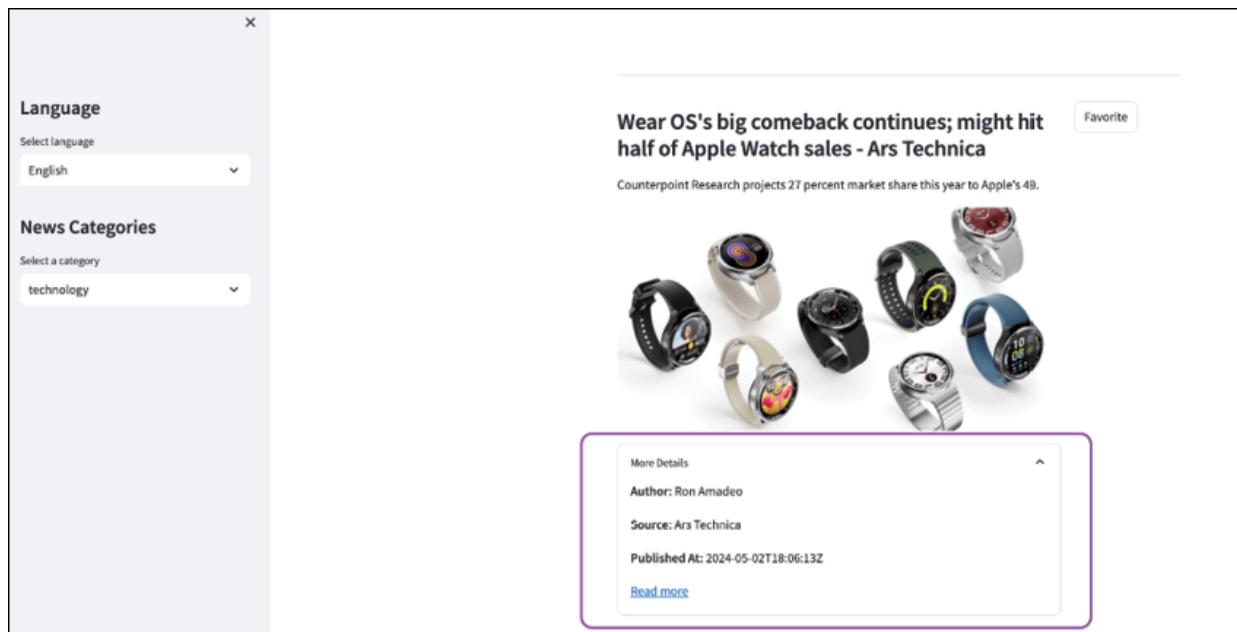


Figure 11.10: Contains more details about the article like source, author, and published date

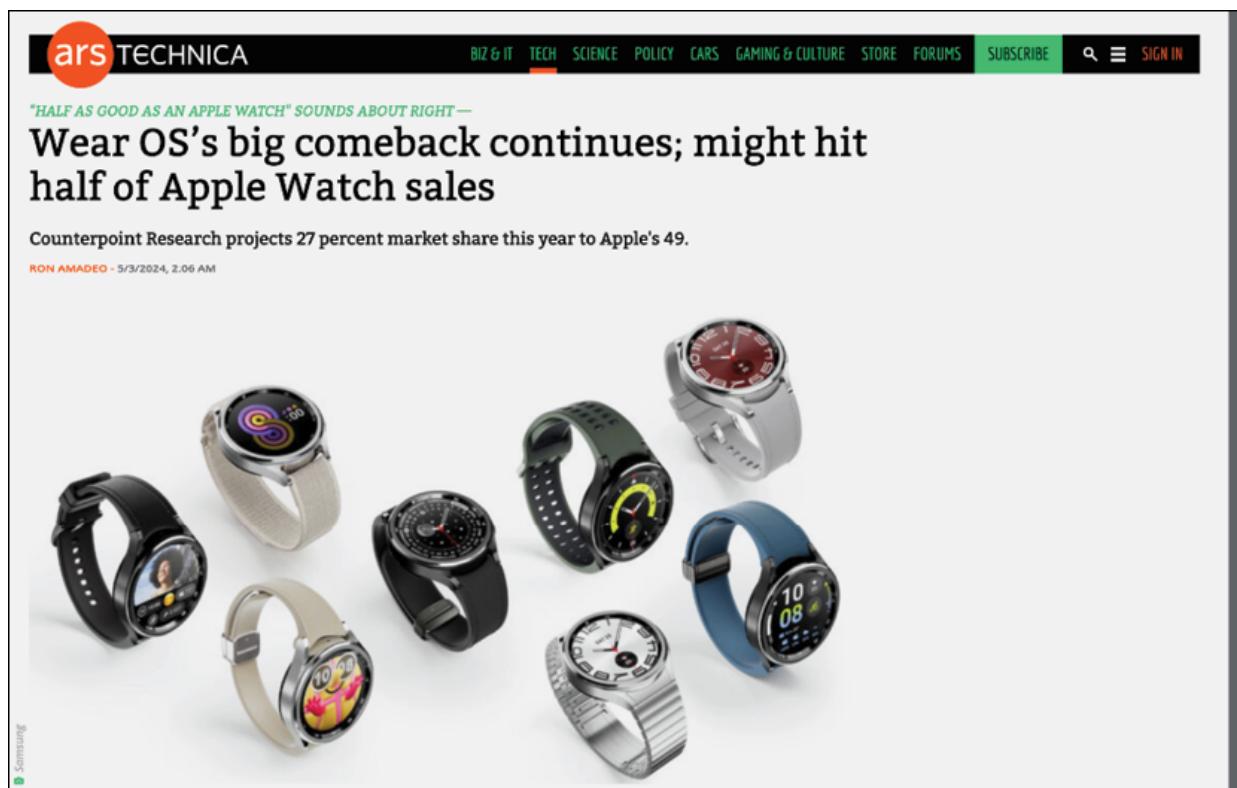


Figure 11.11: Read more in the details section directs the user to the original content of the news

Conclusion

In this chapter, we discussed implementing advanced Streamlit projects, leveraging the framework's capabilities to create sophisticated and interactive applications. Streamlit's simplicity and effectiveness lie in its ability to abstract complex functionalities into user-friendly components, allowing developers to focus on delivering value rather than managing boilerplate code. By incorporating advanced techniques such as session tracking and caching, we demonstrated how Streamlit can handle stateful interactions and optimize data processing tasks, which is essential for developing responsive and efficient applications.

As we conclude the hands-on chapters discussing and building easy to advanced projects, mastering Streamlit involves understanding its components and requires a strategic approach to application design and user interaction. The projects we built testify to the platform's flexibility and power, encouraging developers to think beyond conventional web development paradigms.

In the next chapter, we will shift our focus from individual projects to the bigger picture: building and enhancing a professional portfolio. This will help you pull everything together, ensuring that your work not only functions well but also shines in front of potential employers and collaborators. We will also discuss the channels and communities to follow to stay up to date with the latest market trends and share and contribute your knowledge.

Join our book's Discord space

Join the book's Discord Workspace for Latest updates, Offers, Tech happenings around the world, New Release and Sessions with the Authors:

[https://discord.\(bpbonline.com](https://discord(bpbonline.com)



CHAPTER 12

Build and Enhance Your Portfolio

Introduction

In this chapter, we will focus on building and enhancing your professional profile. A well-crafted portfolio is not just a collection of your work; it is a powerful tool that communicates your skills, creativity, and professional potential to the world. We will also explore key resources and the importance of engaging with the Streamlit community. Being part of a community provides you with ongoing support, new ideas, and opportunities for collaboration, ensuring that you stay at the forefront of the field. Sharing and collaborating in community projects is the best way to learn and expand your skills.

Structure

The chapter discusses the following topics:

- Streamlit journey and next steps
- Key resources and how to engage
- Crafting your professional portfolio

Objectives

In this chapter, we will transition from developing applications to showcasing mastery and creativity through a professional portfolio. After enhancing your skills in Streamlit, it is essential to communicate expertise and professional projects to the world. A well-crafted portfolio displays the technical capabilities and sets the stage for professional opportunities. In this chapter, we will summarize what we have learned so far, understand the potential of these skills in the market, and how to effectively present work across various professional platforms.

Streamlit journey and next steps

In the preceding chapters, we looked at Streamlit comprehensively, from its basic functionalities to its more sophisticated capabilities. Throughout the journey, we built projects that progressively built upon our understanding and skills.

As we progressed, we integrated complex elements such as APIs, interactive widgets, and session states, transforming static data scripts into dynamic and interactive web applications. Each project was designed not only to challenge the understanding but also to enhance the ability to solve real-world problems using Streamlit.

This foundational knowledge equips you to advance further in your development career. The next crucial step is continuously engaging with the evolving landscape of technology and community. In this section, let us look at how to remain at the forefront of innovation by leveraging resources and participating actively within the Streamlit community.

Importance of community engagement

Streamlit has a vibrant community of developers and enthusiasts who share a passion for data science and user-friendly applications. Engaging with this community can provide invaluable resources, from troubleshooting assistance to innovative ideas on using Streamlit in ways you might have yet to consider.

Key resources and how to engage

Let us check the most interesting communities and resources for Streamlit.

Official forums and discussion boards

Streamlit's official community forum, app gallery, and components gallery offer advice, guidance, and application library by Streamlit developers throughout the globe.

Streamlit community forum

The Streamlit Community Forum is an invaluable resource for developers of all skill levels working with Streamlit. It is not just a platform for troubleshooting, it is a vibrant hub where ideas, strategies, and innovations converge. Whether you are a beginner needing guidance on the basics or an advanced user looking to discuss complex implementation issues, the forum offers a direct line to a wealth of knowledge and experience from fellow Streamlit enthusiasts and experts.

This forum also provides information about getting started, the latest news, deployment, and local or virtual events in your area.

Streamlit community: <https://discuss.streamlit.io/>.

The key features of the Streamlit community forum are as follows:

- **Expert solutions:** The forum is frequented by experienced Streamlit developers and the creators of Streamlit themselves. This gives you access to authoritative advice and solutions that are both practical and optimized. If you are encountering a bug or need advice on best practices, the forum is the place to get reliable answers swiftly.
- **Collaborative problem-solving:** Engage in discussions that help the entire community and the person asking the question. Solutions and creative ideas are often generated through collaborative effort, providing a broader perspective and fostering a deeper understanding of Streamlit's capabilities.
- **Showcase and feedback:** The forum is an excellent place to showcase your projects and get peer feedback. This peer review can be invaluable for improving your projects, learning new techniques, and gaining confidence in your abilities.
- **Stay updated:** Streamlit is continuously evolving, with new features and updates regularly rolled out by the development team. The community forum is often the first place where these updates are discussed in depth. This lets you stay updated on the latest advancements and learn how to integrate them into your projects effectively.
- **Build your network:** By participating actively in discussions, you build your reputation as a knowledgeable professional and expand your network within the tech community. These connections can lead to collaborative opportunities, mentorship, and job offers.

The following is the Streamlit community's official home page:

The screenshot shows the Streamlit official forum home page. On the left, there's a sidebar with navigation links: Topics (1), My Posts, More, Categories (Using Streamlit, Community Cloud, Custom Components, Show the Community, Deployment), All categories, Tags, IIRMS, streamlit-cloud (1), All tags, Messages (Inbox, DMs). The main area has tabs at the top: cat: Questions about deploying apps on Streamlit (Community Cloud) [red], Latest, New (27), Unread (3), Top, + New Topic. Below are several boards:

- Using Streamlit**: 64 / week, 1 unread, 15 new. Topics: 🔒 Using Streamlit: how to post a question in the Streamlit forum (0, Jul 2023), Usage: streamlit run [OPTIONS] TARGET [ARGS] (0, 4m ago).
- Community Cloud**: 15 / week, 6 new. Topics: Usage: streamlit run [OPTIONS] TARGET [ARGS] (0, 4m ago).
- Custom Components**: 7 / week, 2 unread, 2 new. Topics: St.page_link additional attributes (2, 14m ago).
- Show the Community**: 11 / week, 2 new. Topics: Local image file not shown in components.html (0, 42m ago).
- Deployment**: 10 / week. Topics: 🔒 Elmento - AI chat to PDF and Image Files (0, 2h ago).
- Events**: 1 / week. Topics: Profanity in App URL - "parser" (0, 3h ago).

Figure 12.1: Streamlit official forum home page

Some interesting boards in the Streamlit community forum:

The screenshot shows the Streamlit Jobs board. On the left, there's a sidebar with navigation links: Topics (1), My Posts, More, Categories, Tags, Messages (Inbox, DMs). The main area has tabs at the top: 📈 Jobs (selected), tags, Latest, Top, + New Topic, 📈. Below are job postings:

- About the 📈 Jobs category**: Job postings related to Streamlit and the data science space. Posted by Streamlit Advocate at Hugging Face (21 replies, 3.2k views, 10d ago).
- I need one expert to do my project 'sentiment analysis of restaurant reviews using machine learning NLP'**: I have the model but I want to turn it to a web app using streamlit. I will pay the fees, if anyone can help please respond debugging. Posted by Streamlit Tutor (4 replies, 222 views, 15d ago).
- Wanted - streamlit/python dev to build dashboard for startup**. Posted by Streamlit Tutor (11 replies, 1.1k views, Feb 29).
- I need two buy 1-2 hours of someone's time to help me finish a project**. Posted by Streamlit Tutor (5 replies, 1.6k views, Jan 18).

Figure 12.2: Jobs board, where you can find interesting job postings around the globe

Streamlit official announcement board is a perfect place for keeping updated with the latest features launched:

The screenshot shows a community board titled "Official Announcements". The top navigation bar includes "Latest" (highlighted in red), "New (1)", and "Top". On the left, a sidebar lists "Topics" (1), "My Posts", "More", "Categories", "Tags", "Messages" (selected), "Inbox", and "DMs". The main content area displays seven posts:

- About the Announcements category**: Announcements from the Streamlit team about releases and other major events. Replies: 0, Views: 2.4k, Date: Jan 2023.
- Version 1.34.0**: release-notes. Replies: 1, Views: 164, Date: 1d.
- [Share your feedback] What processes are a pain to build?**: feedback. Replies: 14, Views: 532, Date: 11d.
- AppTest: test & build interactive Python data apps faster**. Replies: 12, Views: 1.2k, Date: 16d.
- Community Cloud switching from pip to uv**: streamlit-cloud. Replies: 0, Views: 162, Date: 18d.
- [Share your feedback] Help us improve Community Cloud**: feedback. Replies: 6, Views: 173, Date: 19d.
- Version 1.33.0**: release-notes. Replies: 12, Views: 1.8k, Date: 25d.
- Version 1.32.0**: release-notes. Replies: 22, Views: 2.4k, Date: Mar 27.

Figure 12.3: Streamlit Official Announcement Board, for the latest features in the new version launches

Streamlit community board shows the community and is used by worldwide contributors to share their new components. This is an interesting page to get to know what people around the world are working on:

The screenshot shows a community board titled "Show the Community!". The top navigation bar includes "Latest" (highlighted in red), "New (2)", and "Top". On the left, a sidebar lists "Topics" (1), "My Posts", "More", "Categories", "Tags", "Messages" (selected), "Inbox", and "DMs". The main content area displays eight posts:

- About the Show the Community! category**: Apps, tutorials, code snippets, blog posts, videos, and other Streamlit-related projects. Replies: 1, Views: 2.5k, Date: Jun 2021.
- Elmento - AI chat to PDF and Image Files**: streamlit-cloud, pandas, windows. Replies: 0, Views: 10, Date: 2h.
- Enterprise-grade production app 100% in Streamlit for you to try**. Replies: 0, Views: 16, Date: 7h.
- New Component: Streamlit-Ldap-Authenticator**: session-state, authentication, cookies. Replies: 32, Views: 1.3k, Date: 1d.
- Make apps faster by moving heavy computation to a separate process**. Replies: 0, Views: 95, Date: 1d.
- [Feedback requested!] Proposal - Native authentication support for Streamlit**. Replies: 5, Views: 335, Date: 2d.
- New Component: auth0_component, a simple way to authenticate a user**: authentication. Replies: 33, Views: 5.9k, Date: 3d.
- New Component: Arabic Support components**. Replies: 0, Views: 28, Date: 3d.

Figure 12.4: Streamlit Show the community board

The Streamlit event board talks about upcoming conferences, meetups, and events. You can join local events and be part of the local developer community:

The screenshot shows a web-based community board interface. On the left, there's a sidebar with navigation links: Topics (blue dot), My Posts, More, Categories, Tags, Messages, and DMS. The main area has tabs at the top: Events (with a dropdown arrow), tags (with a dropdown arrow), Latest (highlighted in red), and Top. There's also a '+ New Topic' button and a notification icon. Below the tabs, there's a 'Topic' section with a post by 'Hosting an event or speaking about Streamlit?' which is a 'livestream' and 'events'. This is followed by several other posts: '[May 13] Livestream: How Microsoft is supporting AI innovation in the federal government' (ai, livestream, ml), '[PyCon DE & PyData Berlin 2024]' (discussion, events), '[Pycon US 2024]' (discussion, events), '[April 23] Livestream: Supercharge your apps with real-time data' (livestream), and '[April 17] Streamlit Meetup: San Francisco' (events, meetup). Each post includes a user profile picture, the number of replies, views, and the date it was posted.

Figure 12.5: Streamlit events board to find the meetup events around your area

The random board is for the random posts and some fun banter:

The screenshot shows a dark-themed web interface for a community forum. On the left, there's a sidebar with navigation links: Topics (1), My Posts, More, Categories, Tags, Messages, Inbox, and DMs. At the top, there are buttons for Random, tags, Latest (which is highlighted in red), and Top. To the right of these are buttons for + New Topic and a search icon. The main area displays a list of topics:

Topic	Replies	Views	Activity
>About the Random category Anything you'd like to share with the Streamlit community	0	1.2k	Jan 2023
How can you be sure of the number of users? streamlit-cloud discussion	5	165	3d
Looking for ideas/feedback to make enhancements discussion	0	15	3d
Mobile app framework for Streamlit discussion	4	277	3d
Redundant display of a confirmation form debugging	1	53	14d
Does anyone have experience with Google's Gemini Pro Vision? discussion limits computer-vision	0	121	15d
It really was THAT easy! discussion	1	80	17d
How to optimize an app for "offline" use? session-state discussion cache	0	77	19d

Figure 12.6: Streamlit random board to share anything with the community

Streamlit Official App Gallery

The Streamlit Official App Gallery is a curated collection of applications developed by the Streamlit community worldwide. It serves as a dynamic showcase of the innovative ways developers are utilizing Streamlit to turn data into engaging, interactive web applications. Whether you are new to Streamlit or an experienced developer, the App Gallery is a valuable resource for inspiration and practical learning.

The highlights of the Streamlit Official App Gallery are as follows:

- **Diverse range of applications:** The App Gallery features a wide array of applications across different domains, such as finance, health care, machine learning, and more. This diversity not only demonstrates the versatility of Streamlit but also provides insights into how various data-driven challenges can be addressed with creative solutions.

- **Open-source projects:** One of the key advantages of the App Gallery is that most of the projects are open-source. This means you can easily access the code, study it, modify it, and even integrate parts of it into your projects. Open-source projects encourage a hands-on learning approach and foster a spirit of collaboration within the community.
- **Quality documentation:** Each project in the App Gallery comes with thorough documentation, which includes details about the project's purpose, the technology stack, setup instructions, and usage guidelines. Good documentation is crucial as it allows you to understand the project quickly and deeply, making it easier to learn from the examples provided.
- **Learning and experimentation:** By exploring various projects in the App Gallery, you can see firsthand the range of functionalities Streamlit offers and how different features can be implemented. These projects can be excellent learning tools, especially when dissecting how particular visualizations or interactions are coded.
- **Inspiration for your projects:** Seeing what others have built with Streamlit can spark ideas for your projects. The App Gallery can act as a brainstorming tool, where browsing through different applications might inspire a solution to a problem you are working on or give you ideas for completely new projects.

The points to be noted to utilize the Streamlit App Gallery effectively are as follows:

- **Explore regularly:** Make it a habit to check the App Gallery regularly for new submissions. This will keep you updated on the latest trends and innovations within the Streamlit community.

- **Interact with creators:** Many developers provide contact information or links to their GitHub profiles. Do not hesitate to reach out with questions, feedback, or requests for collaboration. Engaging with creators can enhance your learning and expand your professional network.
- **Contribute your projects:** If you have developed a Streamlit app that benefits others, consider submitting it to the App Gallery. Contributing to the gallery can increase your visibility in the community and provide you with valuable feedback that can help improve your work.
- **Apply learnings to your work:** Apply techniques and ideas you learn from the App Gallery to your Streamlit projects. Applying these concepts will reinforce your learning and improve your development skills.

App gallery is a collection of Streamlit applications:

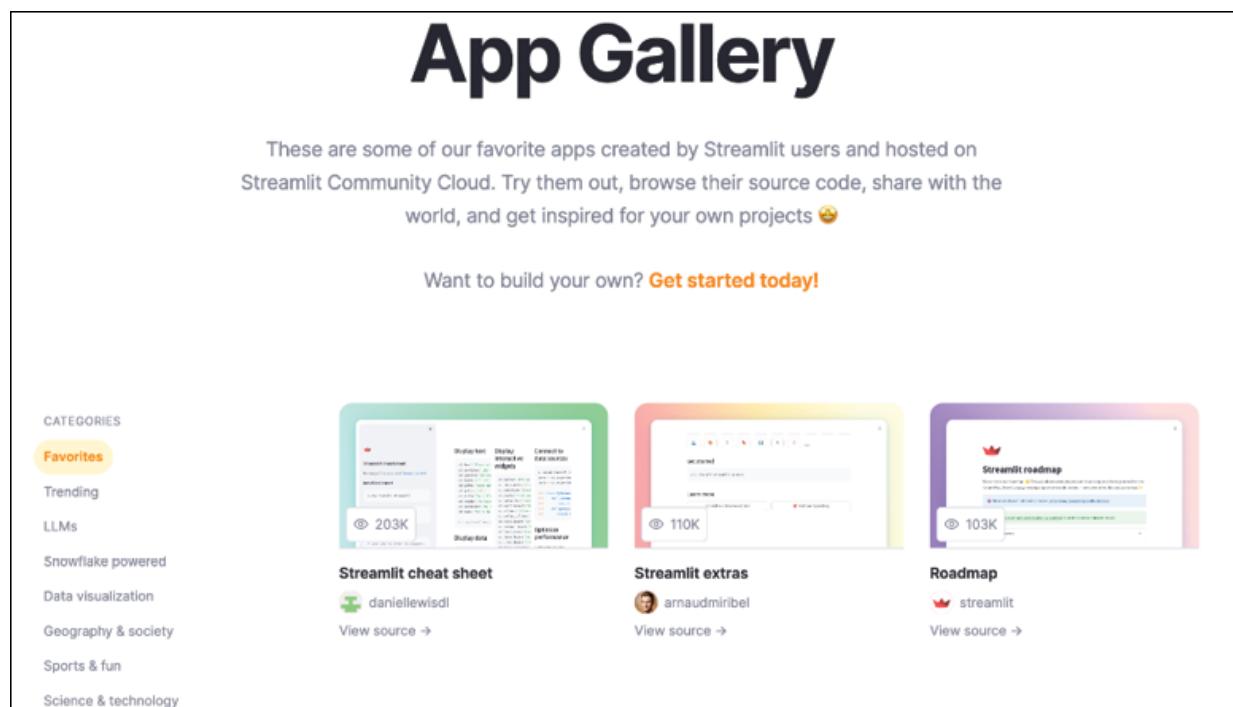


Figure 12.7: Streamlit App Gallery home page

Some interesting Streamlit apps from the gallery. These applications can be directly forked and can be used in your projects:

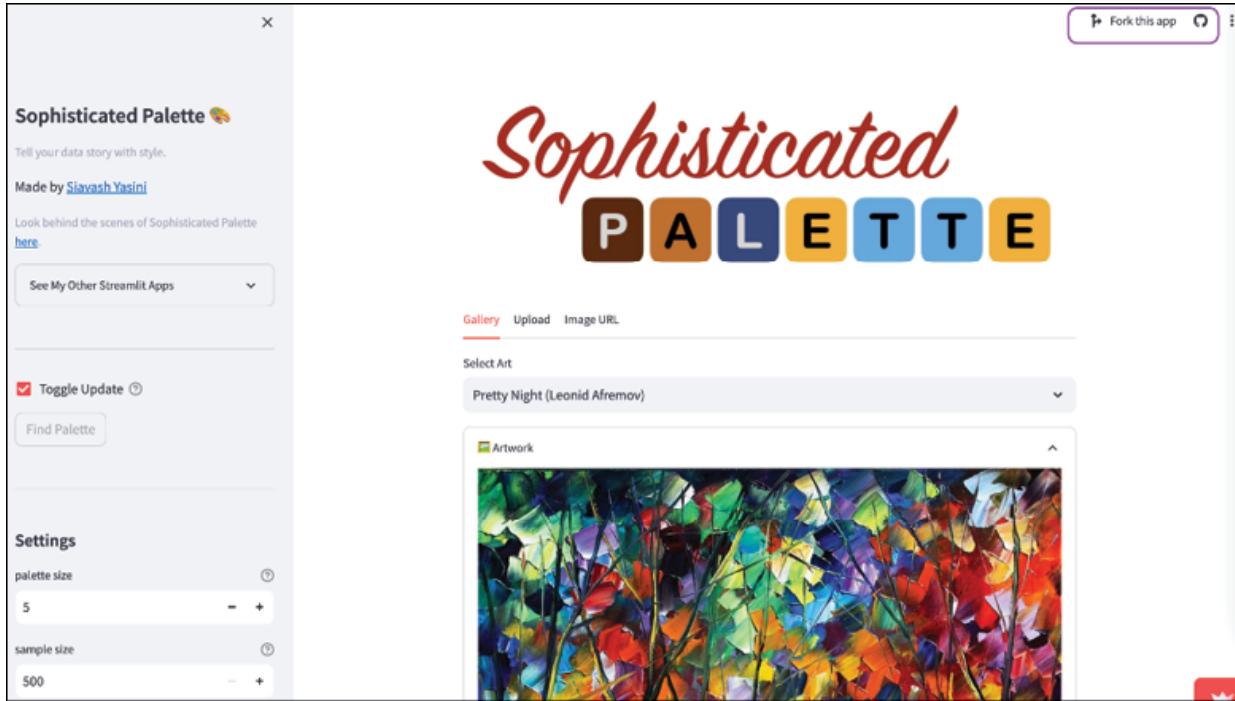


Figure 12.8: *sophisticated-palette.Streamlit.app* by Siavash Yasini

This application lets you upload a picture and provide the color palette from the image. The great thing about community applications is that most can be forked, and you can build on top of them or create an enhanced version:

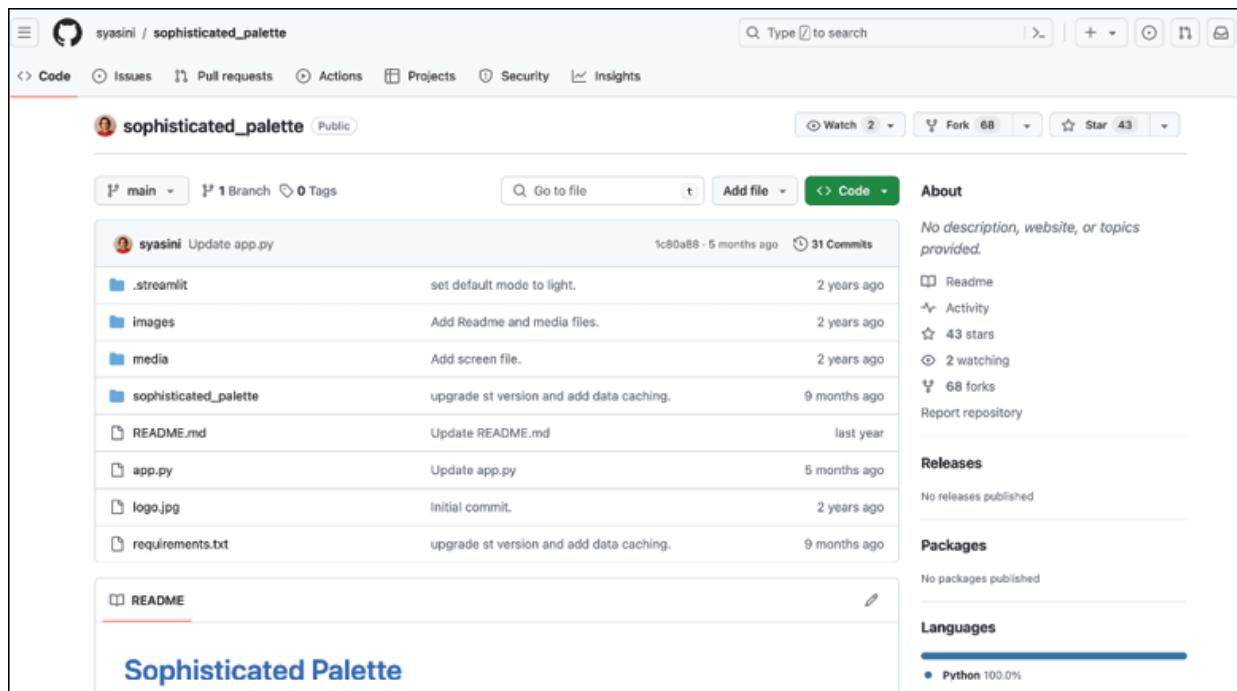


Figure 12.9: GitHub code for *sophisticated_palette*

You can view the code and understand the workings of the application in depth:

intro start game

| Intro



Explore the depths of an ancient dungeon in the first streamlit-based dungeon crawler game! Navigate through dangerous traps, defeat fearsome monsters and uncover the secrets of the DuNgeOn. With intuitive controls and beautiful graphics, this game will keep you entertained for hours. Experience the thrill of adventure as you progress through levels and uncover powerful treasures. Join the adventure today and become the hero of the dungeon!

▶ 0:22 / 0:26 ⏪ ⏴ ⋮

The Dungeon: a streamlit dungeon crawler game

| Game start

To start the game go to the "start game" tab. Please be sure to switch to dark mode or the custom theme. The Dungeon is meant to be played in the dark!

| Controls

Desktop: please use keyboard arrows | Mobile (Android, Chrome): please use on-screen buttons | iOS: unfortunately, the auto-scrolling feature does not work yet for iOS.

Figure 12.10: *dungeon.Streamlit.app* by Tom John

The following figure shows a maze game developed on Streamlit. It has many interesting features:

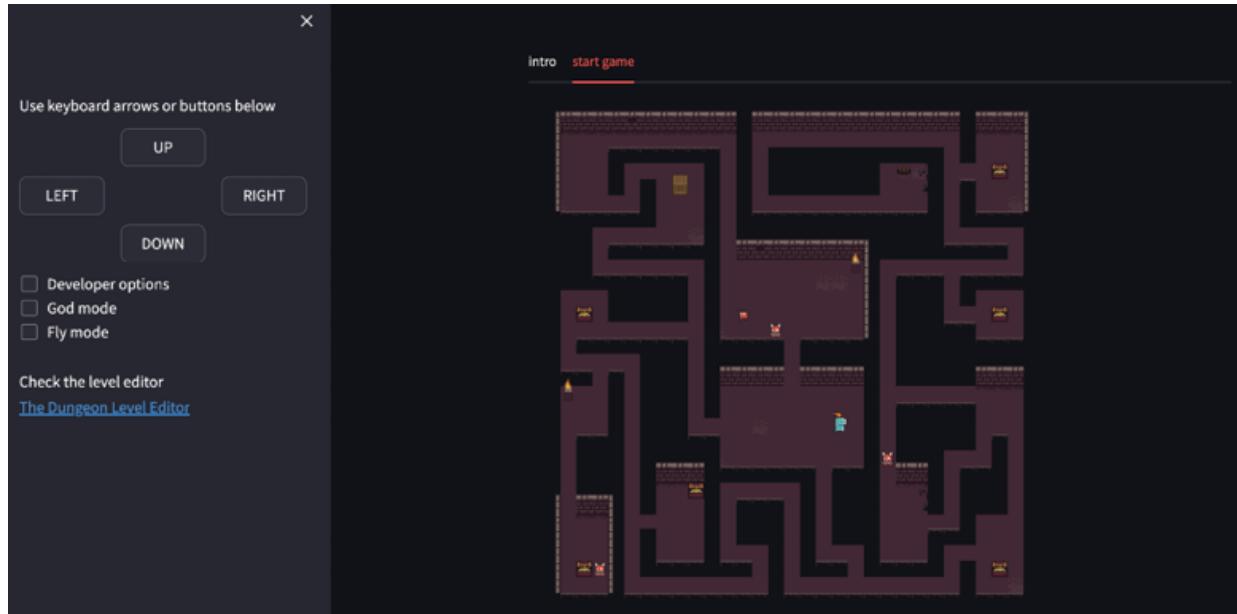


Figure 12.11: Maze game from dungeon app.

The cheat sheet is as follows:

	Display text	Display interactive widgets	Connect to data sources
Streamlit cheat sheet			
Summary of the docs , as of Streamlit v1.25.0.			
Install and import			
\$ pip install streamlit			
# Import convention >>>> import streamlit as st	st.text('Fixed width text') st.markdown('__Markdown__') # see # st.caption('Balloons. Hundreds of st.latex(r''' e^{i\pi} + 1 = 0 ''') st.write('Most objects') # df, err st.write(['st', 'is <', 3]) # see st.title('My title') st.header('My header') st.subheader('My sub') st.code('for i in range(8): foo()') # = optional kwarg unsafe_allow_ht	st.button('Hit me!') st.data_editor('Edit data', data) st.checkbox('Check me out') st.radio('Pick one!', ['nose', 'ear']) st.selectbox('Select!', [1,2,3]) st.multiselect('Multiselect', [1,2,3]) st.slider('Slide me', min_value=0, st.select_slider('Slide to select') st.text_input('Enter some text') st.number_input('Enter a number') st.text_area('Area for textual ent')	st.experimental_connection('pets_d conn = st.experimental_connection(conn = st.experimental_connection(>>> class MyConnection(Experimenta >>> def _connect(self, **kwargs >>> return myconn.connect(* >>> def query(self, query): >>> return self._instance.qu
Add widgets to sidebar			
# Just add it after st.sidebar: >>> a = st.sidebar.radio('Choose			
Magic commands	'_This_ is some __Markdown__' a=3 'dataframe:', data	st.dataframe(my_dataframe) st.table(data.iloc[0:10]) st.json({'foo':'bar','fu':'ba'}) st.metric(label="Temp", value=273	# E.g. Dataframe computation, stor >>> @st.cache_data ... def foo(bar): ... # Do something expensive and ... return data
Command line		st.image('./header.png') st.audio(data)	# Executes foo >>> d1 = foo(ref1) # Does not execute foo # Returns cached item by value, d1 >>> d2 = foo(ref1) # Different arg, so function foo e >>> d3 = foo(ref2)
	Display data	Display media	Optimize performance
			Cache data objects

Figure 12.12: cheat-sheet.Streamlit.app by Daniel Lewis

This is a very handy application with consolidated Streamlit command lists and functions.

The Streamlit Official App Gallery is more than just a collection of projects; it is a vibrant marketplace of ideas and a testament to Streamlit's capabilities. By actively participating in this community resource, you gain knowledge and contribute to the ecosystem, enhancing your skills and helping others.

Streamlit Component Gallery

The Streamlit Component Gallery is an invaluable resource for Streamlit developers looking to enhance the functionality of their applications. Like the App Gallery, the Component Gallery is populated with user-developed components that extend beyond Streamlit's built-in capabilities. These components range from advanced data visualization libraries

to integrations with other web technologies, allowing developers to incorporate sophisticated features into their applications with minimal coding effort.

The benefits of using Streamlit Components are as follows:

- **Enhanced functionality:** The Component Gallery offers many tools that add significant value to your applications. From interactive maps to complex charts, these components allow you to implement functionalities beyond the default Streamlit widgets.
- **Rapid development:** By utilizing pre-built components, developers can save time and reduce the boilerplate code they need to write. This accelerates the development process, allowing you to focus more on the application's logic and user experience.
- **Community support:** Components in the gallery are often supported by a community of developers. This means that issues are frequently discussed and resolved through community contributions, which can enhance their reliability over time.

While the Streamlit Component Gallery is a powerful resource, it comes with risks that developers should be aware of. They are as follows:

- **Cross-platform consistency:** Not all components are built with cross-platform compatibility in mind. Some might work well on one operating system or browser but not on others. This can lead to inconsistent behavior across different environments, potentially affecting the user experience.
- **Maintenance and support:** Components created by the community might not always adhere to the same maintenance and support standards as those developed by the Streamlit team. If they are not regularly

updated, some components might become deprecated or less compatible with newer versions of Streamlit.

- **Security concerns:** Integrating third-party components into your applications risks introducing security vulnerabilities. Before incorporating a component into your project, it is important to review the code and ensure it adheres to security best practices.

The best practices for using Streamlit Components are as follows:

- **Thorough testing:** Before integrating a new component into your application, thoroughly test it in a controlled development environment. Check for both functionality and compatibility across different platforms and browsers. You can utilize App Testing features from Streamlit (details in *Chapter 7, Advanced Streamlit Concepts*).
- **Read the documentation:** Always read the documentation provided with the component. Understanding how it is supposed to work and any dependencies it has can save you from future headaches.
- **Engage with the community:** If you encounter issues, engage with the community. Many developers will appreciate feedback and may provide fixes and updates in response to user problems.
- **Contribute back:** If you make improvements or discover fixes for components, consider contributing back to the community. This helps improve the component for others and strengthens your reputation within the community.

Streamlit Component Gallery provides the list of components submitted by the developers, unlike Streamlit App Gallery,

which hosts the Streamlit applications on Streamlit sharing. Streamlit components are available as Python packages and must be installed in the application environment using the instructions shared by the developers. Most of the component creators provide a demo application and detailed documentation to demonstrate the features and usage of the component.

The Streamlit Component Gallery showcases the list of the Streamlit components:

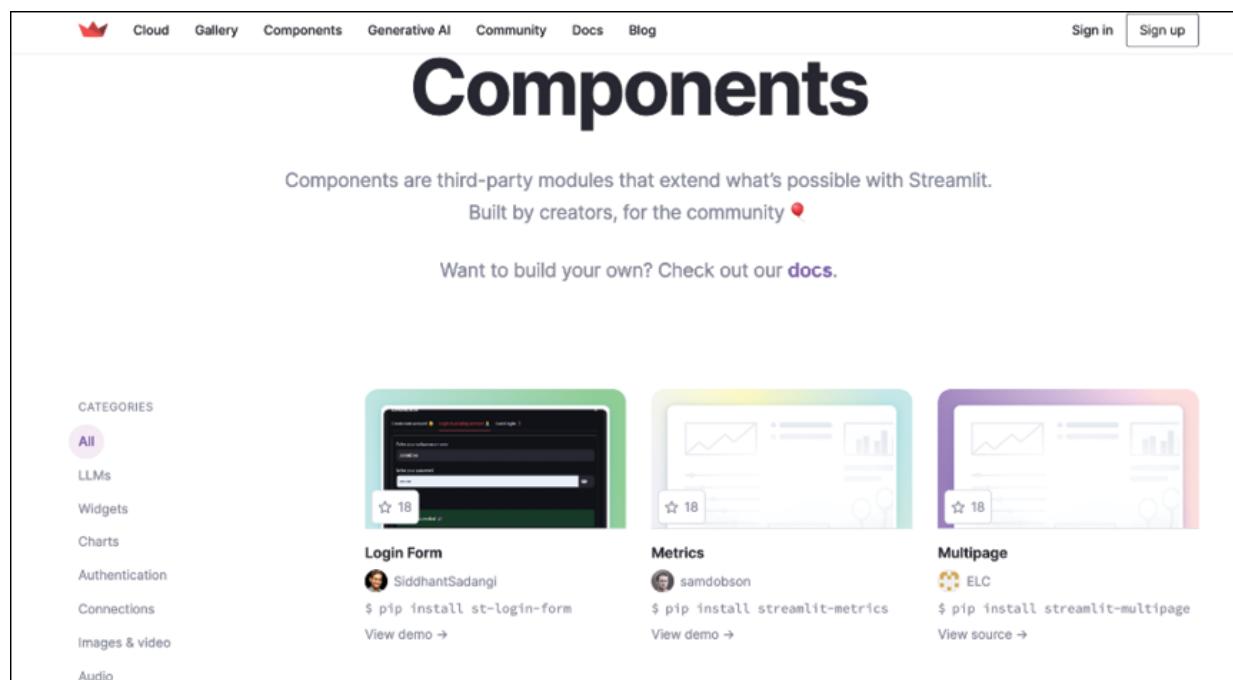


Figure 12.13: Streamlit component gallery

Crafting your professional portfolio

As you progress in your Streamlit journey and broaden your technical skills, creating a professional portfolio becomes a crucial step in showcasing your development and achievements. This section will guide you through the importance of building a comprehensive portfolio and how to use various tools to document and share your work effectively.

The idea behind a comprehensive portfolio

A well-maintained portfolio does more than display your projects; it tells the story of your journey from a novice to an advanced developer. It also serves as a dynamic platform to articulate your evolving ideas and technical prowess. Your portfolio serves the following functions:

- **Reflective of your journey:** Your portfolio should capture the breadth and depth of your experience, including projects that showcase simple beginnings and, later, more complex undertakings that illustrate your growth and ability to learn and adapt. This not only makes your portfolio more relatable but also more inspiring to those who are at earlier stages of their journey.
- **A living document:** Your portfolio is a living document that evolves with you. Regular updates with new projects and insights keep it relevant and reflective of your current skills and focus areas. This ongoing process of adding and refining projects can also help you evaluate your progress and set future goals.

Leveraging tools for effective documentation

Documenting and sharing are the most potent tools for journaling your progress. With numerous free platforms available, it is important to use them to enhance your portfolio and share your ideas with the community. Let us consider how to get started with them:

- **GitHub:** Start with GitHub, not just as a repository for your code but as a portfolio tool. GitHub lets you showcase your projects, complete with README files that explain and contextualize your work. Integrating CI/CD tools, issue tracking, and the ability to fork and

star projects enhances how interactive and professional your portfolio appears to potential employers or collaborators. With GitHub, you can join open-source projects that interest you and start contributing. This will help build your skills as a developer and help you understand the right practices in the market.

- **Blogging platforms:** Use platforms like Medium or Substack to write about your projects, challenges, and insights. This not only helps document the non-code aspects of your projects, such as design thinking and problem-solving approaches, but also improves your visibility and engagement with the tech community. These articles will not only help you grow but also provide guidance and support to other folks around the world.
- **Participation in community events:** Engaging in hackathons, conferences, and meetups and sharing these experiences in your portfolio can significantly boost your professional image. These events are learning opportunities and a chance to network and collaborate, providing fresh content for your portfolio and showing your commitment to staying connected and relevant in your field.

Marketing your skills through your portfolio

Marketing yourself and building your brand is the most underrated skill for tech enthusiasts. The right platform and focus can help you connect with like-minded folks and provide numerous opportunities that might not be feasible with traditional means. Marketing yourself should include the following:

- **Showcase diverse skills:** Ensure your portfolio reflects a range of skills, from data analysis and

machine learning models to web development and user interface design. Highlighting various projects demonstrates your versatility and ability to handle different aspects of software development.

- **Interactive elements:** Whenever possible, include interactive elements or live demos in your portfolio. Streamlit apps can be hosted and linked within your GitHub READMEs or blog posts, providing a direct way for viewers to interact with your projects.
- **Feedback and iterations:** Show how feedback from peers, mentors, or users has shaped your projects. This openness to learning and adapting is highly valued in any professional field. Contact fellow developers for feedback, advice, or mentoring opportunities. Cold calls might seem daunting at first, and you might not be successful in your first attempt, given that everybody has their time commitments, but this will help you learn and work on your soft skills, communication, and presentation skills.

Conclusion

In this chapter, we explored the valuable resources provided by the Streamlit community and broader networking and marketing tools that are crucial for professional development in technology. From the Streamlit Component and App Galleries, which showcase the creative possibilities within the Streamlit ecosystem, to the platforms for documenting and sharing your work, we discussed how each resource can enhance your skills and visibility as a developer.

Remember that the technology landscape is ever-evolving as you engage with these resources and tools. Staying active in communities, continuously updating your portfolio, and adapting to new tools and technologies will ensure your growth remains ongoing. The path of learning is limitless,

and each step you take builds your expertise, professional network, and opportunities. These elements are fundamental to not just surviving but thriving in the fast-paced and ever-changing tech industry.

In the next chapter, we will explore how to harness the power of AI tools to enhance your Streamlit applications further, build fast, and deliver more. These tools can add an extra layer of interactivity and intelligence to your projects, taking them to new heights and providing even more value to your users.

CHAPTER 13

Enhancing Streamlit Development with AI Tools

Introduction

In this chapter, we will explore practical ways to incorporate AI into the various phases of the development of Streamlit apps, from brainstorming the design, exploring the helper utilities, and building test coverage by leveraging various AI tools to create more interactive and engaging user experiences.

Structure

The chapter discusses the following topics:

- Introduction to AI-assisted development tools
- Exploring categories of AI tools
- Hands-on: Using ChatGPT in Streamlit development

Objectives

In previous chapters, we have explored the depths of Streamlit, from foundational concepts to advanced techniques, complemented by hands-on project exercises. Now, we focus on leveraging various AI tools that can significantly enhance the speed, quality, and creativity of your development processes.

In this chapter, we will discuss a spectrum of AI technologies, from code completion tools and AI-powered coding assistants to prompt engineering platforms and beyond. These tools are designed to automate mundane tasks, suggest innovative coding solutions, optimize performance, and even foresee coding errors before they occur. Whether you are refining your code with real-time suggestions, generating complex code blocks from simple prompts, or ensuring your code's robustness against potential bugs, AI tools offer a compelling advantage.

We will explore how these AI capabilities can seamlessly integrate into your Streamlit projects, transforming how you develop, test, and deploy applications. Moreover, we will address the ethical implications of using AI in development, ensuring you are equipped with the knowledge to use these tools responsibly and effectively.

Introduction to AI-assisted development tools

In recent years, **artificial intelligence (AI)** has seen unprecedented growth, driven by advances in machine learning, data availability, and computational power. This surge has revolutionized many sectors, with software development being a prime beneficiary. AI's integration into development tools has transformed traditional coding methodologies, bringing in a new era of efficiency and innovation.

AI's role in software development has expanded beyond simple automation. Today, AI tools can assist in complex decision-making processes, predict development outcomes, and even engage in creative aspects of coding. Their ability to significantly reduce development time, automate mundane tasks, and tackle complex problems that were once the sole preserve of human intellect has accelerated the adoption of AI.

Shift from manual coding to AI-assisted development

So far, the traditional approach to software development has been effort-intensive, often requiring extensive coding sessions to build, debug, and optimize applications. However, the landscape is changing. AI-assisted tools can enable developers to move from manual coding practices to more strategic roles. These tools offer

suggestions, write code, and even refactor existing codebases, which allows developers to focus on higher-level design and strategy rather than the intricacies of syntax and boilerplate code.

Advantages and guardrails of using AI-assisted tools

AI-assisted tools in software development can be categorized based on their functionalities, including code completion, bug detection, automated refactoring, and more. These tools leverage machine learning algorithms and large datasets to learn from patterns and offer predictive coding assistance. Some of the key advantages of using these tools include:

- **Increased efficiency:** AI tools can automate repetitive tasks, helping developers focus on more complex problems.
- **Enhanced creativity:** By handling routine aspects of coding, AI allows developers to explore more innovative solutions and experiment with new technologies.
- **Improved code quality:** With AI's ability to analyze vast amounts of code, these tools can suggest optimizations and identify potential errors before they become problematic.

Although AI tools are constantly refined, and the benefits are substantial, it is crucial to establish guardrails when using AI in development:

- **Accuracy verification:** Regularly check the accuracy of the AI's code suggestions to ensure they align with project requirements.
- **Bias mitigation:** Be aware of potential biases in AI tools, which may stem from the data on which they were trained.
- **Security measures:** Implement security protocols to protect the AI tools from injecting vulnerable code into the system.

As AI continues to evolve, its integration into development tools represents a significant leap forward for software engineers. By understanding and effectively leveraging these AI-assisted tools, developers can enhance their productivity and contribute to building more robust, innovative applications.

Exploring categories of AI tools

Integrating AI into software development tools has created a diverse ecosystem of solutions, each tailored to address specific challenges and enhance different aspects of the development process. When this chapter was written, the following are the main categories of AI tools that are revolutionizing how developers build, test, and deploy applications.

Code completion tools

These tools help developers by automatically completing lines of code or suggesting possible continuations. They use machine learning models trained on large code datasets to predict and offer completions based on context.

Some popular tools in this category are as follows:

- **GitHub copilot**: Uses OpenAI Codex to suggest code snippets and functions based on user input.
- **TabNine**: Utilizes deep learning to provide code completions in various programming languages.

AI-powered coding assistants

The AI-powered coding assistants are more sophisticated than simple code completers, offering solutions to coding problems, explanations, and even refactoring suggestions. They can interact dynamically with the user, understanding more complex queries and generating more contextually relevant code.

Some popular tools in this category are as follows:

- **Kite**: Offers code completions powered by deep learning but also provides documentation and examples in line with the code editor.
- **Codota**: Analyzes the codebase to provide intelligent code completions and insights to improve code quality.

Prompt engineering platforms

These platforms are based on prompt engineering, where the user crafts specific prompts to generate useful outputs from AI. This is

particularly relevant with language models that process natural language prompts to generate text, code, or other outputs.

Some popular tools in this category are as follows:

- **ChatGPT**: This can generate coding solutions, explanations, and even entire scripts based on prompts in natural language.

When writing this content, other noteworthy and leading generative AI platforms similar to ChatGPT are Google Gemini and Meta's LLaMa. Although these platforms are built with different specializations, design philosophies, and technical foundations, the scope, strengths, and architecture of each platform differ, and they can be used for similar purposes, such as NLP tasks and prompt engineering.

- **OpenAI Codex**: Specifically designed to understand and generate code based on natural language prompts.

Bug detection and code review tools

AI in this category focuses on improving code quality by detecting bugs, vulnerabilities, or anti-patterns before the code goes into production.

Some popular tools in this category are as follows:

- **DeepCode**: Analyzes code and suggests improvements based on learned patterns from a code dataset.
- **SonarQube**: Provides continuous code quality checks and can integrate AI to enhance its detection algorithms.

Developer bots

These bots integrate into the development environment or workflow to automate repetitive tasks, set up environments, manage dependencies, etcetera.

Some popular tools in this category are as follows:

- **GitBot**: Automates GitHub workflows and can handle tasks like PR merges, issue labeling, and more.

Comparison of tool category and Streamlit usage

The following table includes popular tools in each category, their pricing, and how they can be practically integrated into a Streamlit project, including sample prompts and usage scenarios.

This should give a high-level understanding of how these tools can be leveraged to enhance their development process and make their projects more efficient and robust.

Many of these tools offer rebates on student plans and different pricing for enterprise editions. Check the individual official sites for the latest pricing details and offers. Some of the tools are as follows:

Tool category	Tool name	Cost	Streamlit usage pattern
Code completion	Copilot	\$10/month for individual developer	Suppose you are building a data visualization in Streamlit. You can start typing a comment like: <code># plot data using matplotlib</code> And Copilot might suggest a complete snippet that prepares and displays the plot within your Streamlit app.
Code assistant	Kite	Free Basic Pro Plan \$19.90/month	When working on a Streamlit app that interacts with databases, Kite can suggest optimized SQL queries or the best practices for integrating SQLAlchemy based on the partial code you write.
Prompt engineering platform	ChatGPT	Free Basic Plus Plan \$20/month	Use ChatGPT to generate ideas or code snippets. For example, you can prompt ChatGPT with Generate a Streamlit script that takes user input and queries an SQL database, and receive a code snippet as a response.
Code review and bug detection	DeepCode	Free for Opensource Repos Private Repo \$22/month	Set up DeepCode to scan your Streamlit project repository regularly. It can identify potential vulnerabilities, such as improper input sanitization, and suggest fixes.
Developer Bots	GitBot	Free	Integrate GitBot to manage your Streamlit project repository on GitHub.

Tool category	Tool name	Cost	Streamlit usage pattern
			It can automatically handle pull requests, issue tagging, and merges based on predefined rules.

Table 13.1: Streamlit tools

Hands-on: Using ChatGPT in Streamlit development

When using any AI tool for your development projects, it is important to understand how it works at a high level, the technology behind it, and the data privacy terms and conditions. This will help you better understand whether the tool can be used for the specific use case.

For this hands-on exercise, we will explore ChatGPT, which will help us understand how this prompt engineering works and delivers high-level results.

The latest release of ChatGPT was 4.0 when this chapter was written; hence, we will use this model to prototype a user feedback analysis tool.

ChatGPT overview

ChatGPT is an advanced language model developed by OpenAI. It is based on the **Generative Pre-trained Transformer (GPT)** architecture, designed to generate human-like text by predicting the next word in a sequence given all the previous words within some text. This technology is a form of deep learning, a subset of machine learning in artificial intelligence, where large neural networks are trained on vast amounts of data.

The versatility of GPT-4 makes it suitable for a wide range of applications, including but not limited to customer support, educational tools, content creation, coding assistance, and more.

ChatGPT data privacy, intellectual property, ethical usage recommendations

Using tools like ChatGPT involves careful consideration of data privacy, **intellectual property (IP)**, and ethical usage. Here is an overview of these aspects to be considered when using them in

your projects. For more details and references on the ethical usage of AI and the future of AI, refer to Appendix B.

Data privacy with ChatGPT

ChatGPT and similar AI models by OpenAI are trained on diverse data sets that include a mixture of licensed data, data created by human trainers, and publicly available data. When interacting with ChatGPT, the following should be kept in mind:

- **User data:** OpenAI can use the inputs you provide to ChatGPT to improve model performance. Users must understand that sensitive information should not be input into ChatGPT, as data privacy cannot be guaranteed. It is important to review the code regulation and compliance guidelines when working on enterprise-level projects.
- **Data handling:** OpenAI documentation states that it has implemented guidelines and practices to protect the data and comply with GDPR. However, users should avoid sharing personal or confidential information through such platforms.

Intellectual property guardrails with ChatGPT

ChatGPT states that *The output from ChatGPT is based on patterns and information it has learned during training and does not directly recall from specific sources, which helps mitigate direct IP infringement.*

With the above in mind, although ChatGPT design ensures that the generated output is based on the learning pattern of the GPT's AI model, the user should be mindful of how they utilize the output and data generated concerning originality and IP violation.

Users should consider the following:

- **Generated content:** The content generated by ChatGPT can be considered a derivative work. While OpenAI allows users to own the rights to the model's outputs, the user is responsible for ensuring that the content does not infringe on others' IP.
- **Reuse and publication:** Depending on the nature of the output, users should exercise caution when reusing or

publishing content generated by ChatGPT, especially in commercial contexts.

Other important terms and conditions with ChatGPT

Once you sign up for your ChatGPT account, the following are some important points to note from the terms of the agreement:

- Prohibitions against using the service for illegal activities.
- Limitations on the use of automated querying tools.
- Restrictions on sharing accounts or API keys.

In the following exercise, we will use ChatGPT to help build a prototype for a Streamlit application that collects and analyzes user feedback using natural language processing. Responses and content generated here are generated with the assistance of ChatGPT by OpenAI. The steps are as follows:

1. Brainstorm and conceptualization:

- **Prompt:** I want to develop a user feedback analysis tool using Streamlit and Python. Help me conceptualize the features of this application.
- The output generated is shown in the following figure:

The screenshot shows a ChatGPT 4 interface. At the top, there are icons for a document, a user profile, and a refresh symbol, followed by the text "ChatGPT 4". Below this is a section titled "Core Features of the User Feedback Analysis Tool" which is numbered 1 through 6. Each number is preceded by a small black dot and a bulleted list of features. At the bottom of the interface, there is a message input field containing "Message ChatGPT" and a note that says "ChatGPT can make mistakes. Check important info."

Core Features of the User Feedback Analysis Tool

1. **Feedback Collection Interface**
 - Text input fields for submitting feedback.
 - Optional user authentication for tracking repeat feedback.
2. **Data Storage**
 - Feedback data stored in a structured database.
 - Easy retrieval features for analysis and reporting.
3. **Feedback Analysis**
 - Sentiment analysis to categorize feedback as positive, negative, or neutral.
 - Keyword extraction to identify main themes.
 - Trend detection to monitor changes and patterns over time.
4. **Reporting and Visualization**
 - Interactive dashboards displaying sentiment trends, keyword clouds, and more.
 - Customizable reports based on specific criteria like date ranges or keywords.
5. **Alerts and Notifications**
 - Real-time alerts for critical feedback.
 - Periodic summary reports via email.
6. **User Feedback Loop**
 - Mechanism for admins to respond to feedback directly.
 - Visibility for users on how their feedback leads to changes.

Message ChatGPT

ChatGPT can make mistakes. Check important info.

Figure 13.1: ChatGPT response to features of the tool

2. Generate code:

- **Prompt:** Can you help generate code for the above features? The application front end is in Streamlit, adding functions for feedback collection, data storage in SQLite, feedback analysis, and visualization.
- The output generated is shown in the following figure:

ChatGPT 4

Can you help to generate code for the above features. The application front end is in streamlit, add function for feedback collection, datastorage in sqlite, feedback analysis and visualization

Certainly! Let's start by creating a basic structure for your Streamlit application that includes functions for feedback collection, storing feedback in SQLite, performing basic feedback analysis, and visualizing the results. We'll develop this step-by-step.

1. Set Up Your Project

First, ensure you have Python and Streamlit installed. You will also need `pandas` for data handling and `sqlite3` for database operations.

```
bash
pip install streamlit pandas sqlite3
```

2. Create the Streamlit App

Create a new Python file, `feedback_app.py`, and start by importing the necessary libraries and setting up a basic Streamlit interface.

```
python
import streamlit as st
import pandas as pd
import sqlite3
from textblob import TextBlob # For sentiment analysis

# Establish a connection to the SQLite database
conn = sqlite3.connect('feedback.db')
c = conn.cursor()

# Create the feedback table if it doesn't already exist
def create_table():
    pass
```

Message ChatGPT



ChatGPT can make mistakes. Check important info.

Figure 13.2: ChatGPT output for the code generation, setup

The following figures showcase the entire output for the Streamlit application as generated by GPT:

The screenshot shows the ChatGPT 4 interface with the following content:

```
# Create the feedback table if it doesn't already exist
def create_table():
    c.execute('''
        CREATE TABLE IF NOT EXISTS feedback_table(
            feedback_text TEXT,
            sentiment TEXT,
            keywords TEXT
        )
    ''')

create_table()

def add_feedback(feedback_text, sentiment, keywords):
    c.execute('INSERT INTO feedback_table (feedback_text, sentiment, keywords) VALUES (%s, %s, %s)', (feedback_text, sentiment, keywords))
    conn.commit()

def main():
    st.title("User Feedback Analysis Tool")

    # Feedback Collection Interface
    feedback = st.text_area("Enter your feedback:")

    if st.button("Submit Feedback"):
        # Basic Feedback Analysis (Sentiment Analysis)
        sentiment = TextBlob(feedback).sentiment.polarity
        sentiment_text = 'Positive' if sentiment > 0 else 'Negative' if sentiment -< 0 else 'Neutral'

        # Just a placeholder for keywords extraction logic
        keywords = "keyword1, keyword2" # This should be replaced with actual keywords

        # Store the feedback in the database
        add_feedback(feedback, sentiment_text, keywords)

        st.success("Feedback submitted successfully!")

    # Data visualization
```

At the bottom of the code block, there is a note: "# Data visualization".

Below the code block, there is a message input field with the placeholder "Message ChatGPT" and a "Send" button with a downward arrow icon.

At the bottom of the interface, there is a footer note: "ChatGPT can make mistakes. Check important info."

Figure 13.3: ChatGPT output for the code generation, data ingestion, and main stub

The screenshot shows the ChatGPT 4 interface with the following content:

```

# Data visualization
if st.button("Show Feedback Analysis"):
    st.subheader("Feedback Sentiment Analysis")
    feedback_df = pd.read_sql_query("SELECT * FROM feedback_table", conn)
    st.write(feedback_df)

    # Show sentiment count
    sentiment_count = feedback_df['sentiment'].value_counts().rename_axis('sentiment').reset_index()
    st.bar_chart(sentiment_count.set_index('sentiments'))

if __name__ == '__main__':
    main()

```

Explanation:

- **Database Setup:** We start by connecting to an SQLite database and creating a table for storing feedback if it doesn't already exist.
- **Feedback Collection:** Users can enter their feedback through a text area in the Streamlit app.
- **Sentiment Analysis:** We're using `TextBlob` for simple sentiment analysis. You might want to replace this with a more robust NLP model depending on your needs.
- **Storing Feedback:** Feedback along with its sentiment analysis result and keywords are stored in the database.
- **Data Visualization:** Feedback and a basic sentiment count bar chart are displayed.

Next Steps:

- **Improve Sentiment Analysis:** Integrate a more complex NLP model for better sentiment analysis.
- **Keyword Extraction:** Implement actual keyword extraction using NLP libraries.
- **Enhance Data Visualization:** Use more sophisticated charts and perhaps time-series analysis if the data includes timestamps.

↓

This script provides a functional foundation for your user feedback analysis tool. You can expand

Message ChatGPT G ↑

ChatGPT can make mistakes. Check important info.

Figure 13.4: ChatGPT output for the code generation, Visualization, explanation, and next steps

3. Refine and Interact with ChatGPT:

- **Prompt:** Could you help to add a UI and visualization function to separate the logic from the main?
- The output generated is shown in the following figures:

Could you help to add a UI and visualization function, separate the logic from the main.

Certainly! Let's refactor the code to separate the user interface (UI) and visualization functions from the main logic, which will make the code cleaner and more modular. This approach enhances readability and maintainability.

Refactored Streamlit Application Code

We will break down the application into separate functions for handling the database, UI, and visualizations.

```
python
import streamlit as st
import pandas as pd
import sqlite3
from textblob import TextBlob # For sentiment analysis

# Database Connection
conn = sqlite3.connect('feedback.db', check_same_thread=False)
c = conn.cursor()

def create_table():
    c.execute('''
        CREATE TABLE IF NOT EXISTS feedback_table(
            feedback_text TEXT,
            sentiment TEXT,
            keywords TEXT
        )
    ''')

def add_feedback(feedback_text, sentiment, keywords):
    c.execute('INSERT INTO feedback_table (feedback_text, sentiment, keywords) VALUES(?, ?, ?)', (feedback_text, sentiment, keywords))

st.title('Feedback App')
st.subheader('Add Feedback')
feedback_text = st.text_input('Feedback Text')
sentiment = st.selectbox('Sentiment', ['Positive', 'Negative'])
keywords = st.text_input('Keywords')

if st.button('Add Feedback'):
    add_feedback(feedback_text, sentiment, keywords)
    st.success('Feedback added successfully!')
```

Message ChatGPT

ChatGPT can make mistakes. Check important info.

Figure 13.5: ChatGPT output for the code refactoring.

ChatGPT 4

```
def get_all_feedback():
    return pd.read_sql_query("SELECT * FROM feedback_table", conn)

def setup_ui():
    st.title("User Feedback Analysis Tool")
    feedback = st.text_area("Enter your feedback:")
    return feedback

def analyze_feedback(feedback):
    sentiment = TextBlob(feedback).sentiment.polarity
    sentiment_text = 'Positive' if sentiment > 0 else 'Negative' if sentiment < 0 else 'Neutral'
    keywords = "Keyword1, Keyword2" # Placeholder for keywords extraction
    return sentiment_text, keywords

def display_feedback_analysis():
    feedback_df = get_all_feedback()
    if not feedback_df.empty:
        st.write(feedback_df)
        sentiment_count = feedback_df['sentiment'].value_counts().rename_axis('sentiment').reset_index()
        st.bar_chart(sentiment_count.set_index('sentiments'))

def main():
    create_table()
    feedback = setup_ui()

    if st.button("Submit Feedback"):
        sentiment, keywords = analyze_feedback(feedback)
        add_feedback(feedback, sentiment, keywords)
        st.success("Feedback submitted successfully!")

    if st.button("Show Feedback Analysis"):
        st.subheader("Feedback Sentiment Analysis")
        display_feedback_analysis()

if __name__ == '__main__':
    main()
```



Message ChatGPT



ChatGPT can make mistakes. Check important info.

Figure 13.6: ChatGPT output for the code refactoring, encapsulating the UI functions

```

st.success("Feedback submitted successfully!")

if st.button("Show Feedback Analysis"):
    st.subheader("Feedback Sentiment Analysis")
    display_feedback_analysis()

if __name__ == '__main__':
    main()

```

Code Breakdown:

1. **Database Functions:** `create_table` and `add_feedback` manage the database interactions. The connection is opened with `check_same_thread=False` to avoid threading errors with Streamlit.
2. **UI Setup:** The `setup_ui` function initializes the Streamlit UI components. It returns the feedback text entered by the user, which can then be processed.
3. **Feedback Analysis:** The `analyze_feedback` function performs the sentiment analysis and keyword extraction. This modular approach allows for easy updates or changes to the analysis logic.
4. **Visualization:** The `display_feedback_analysis` function retrieves all feedback entries from the database and visualizes them using Streamlit's built-in functions. It shows both raw data and a bar chart of sentiment counts.
5. **Main Logic:** The `main` function orchestrates the application flow, tying together UI setup, feedback submission, and data visualization.

This refactoring separates concerns into distinct functions, simplifying the overall structure and enhancing the ease of modifications or expansions to the application.



Message ChatGPT



ChatGPT can make mistakes. Check important info.

Figure 13.7: ChatGPT output for the code refactoring, explanation, and description

4. Explore options for deploying and documenting the application:
 - **Prompt:** Could you provide an option for deploying this application and documentation?
 - The output generated is shown in the following figures:

ChatGPT 4

Could you provide option for deploying this application and documentation

Deploying your Streamlit application can be efficiently done using several platforms that support Python applications. One of the easiest and most popular ways to deploy Streamlit apps is through Streamlit Sharing, but you can also use Heroku, AWS Elastic Beanstalk, or even a Docker container on any cloud platform. Here's a step-by-step guide on deploying your app using Streamlit Sharing and creating documentation for your project.

Deploying with Streamlit Sharing

Streamlit Sharing is built specifically for Streamlit apps and offers a straightforward deployment process with generous free resource limits, making it an excellent choice for small to medium-sized projects.

Steps to Deploy on Streamlit Sharing:

1. Prepare Your Application:
 - Ensure your app is in a Git repository, with all necessary files, including any data files it needs.
 - Make sure to include a `requirements.txt` file in the root directory with all the Python libraries your app depends on listed. For your app, it would look something like this:

```
Copy code
streamlit
pandas
sqlite3
textblob
```

2. Create a Streamlit Sharing Account:

Message ChatGPT

ChatGPT can make mistakes. Check important info.

Figure 13.8: ChatGPT output for the code deployment, with Streamlit sharing

The screenshot shows a ChatGPT 4 interface with the following content:

ChatGPT 4

2. Create a Streamlit Sharing Account:

- Sign up or log in to [Streamlit Sharing](#).
- Connect your GitHub account to Streamlit Sharing.

3. Deploy:

- Once logged in, click on 'New app', then choose your GitHub repository and branch.
- Select the main file of your Streamlit app, usually `app.py` or whatever your file is named.
- Click "Deploy" and Streamlit will start the process. It will install the dependencies and launch your app. This can take a few minutes.

4. Manage Your App:

- After deployment, you can manage your app's settings, such as privacy, access controls, and secrets (for API keys or database passwords).

Documentation

Good documentation is crucial for both users and future developers. It should clearly explain how to install, run, and deploy the application, as well as provide an overview of its functionality.

Suggested Documentation Structure:

- 1. Overview:**
 - Describe what the app does and its main features.
- 2. Installation:**
 - Instructions on setting up a local development environment.
 - This should include steps like cloning the repository, installing dependencies using `requirements.txt`, and any environment setup needed.

Message ChatGPT

ChatGPT can make mistakes. Check important info.

Figure 13.9: ChatGPT output for the code documentation

The screenshot shows the ChatGPT 4 interface. At the top, there are icons for a file, a refresh, and a profile, followed by the text "ChatGPT 4". Below this is a list of bullet points:

- Instructions on setting up a local development environment.
- This should include steps like cloning the repository, installing dependencies using `requirements.txt`, and any environment setup needed.

3. Usage:

- How to run the app locally using Streamlit's run command:
arduino
streamlit run app.py

Copy code

- Explanation of how to interact with the app once it's running.

4. Deployment:

- Detailed steps on how to deploy the app, similar to the deployment guide provided above.
- Include any additional configurations needed for different platforms if necessary.

5. Contributing:

- Guidelines on how others can contribute to the app, including coding standards, submitting pull requests, or reporting issues.

6. License:

- Information on the software license under which your app is released.

This structure will help ensure that anyone who comes across your project can understand its purpose, how to use it, and how to deploy it. Tools like `MkDocs` or `Sphinx` can be used to create professional-looking documentation from markdown or reStructuredText files. You can also host your documentation easily on platforms like Read the Docs or GitHub Pages.

Message ChatGPT

ChatGPT can make mistakes. Check important info.

Figure 13.10: ChatGPT output for the code deployment and running

Thus, working with ChatGPT can help you build a quick prototype with your ideas. This application code can then be refined, refactored, and fine-tuned based on the requirements and complexity. The prompt engineering platform can help generate the boilerplate code, saving time and effort.

Conclusion

In this chapter, we explored various categories of AI tools for faster and more efficient development. These tools introduce new levels of accuracy and creativity, but it is also essential to consider the ethical implications of using these tools as part of your development project. As the acceptance and maturity around these AI tools expand as we move forward, we hope these AI enhancements can be applied in Streamlit projects across your entire development workflow. This chapter just scratched the surface of possibilities and set the foundation, and users are encouraged to explore beyond the basics and discover the full potential of AI.

Join our book's Discord space

Join the book's Discord Workspace for Latest updates, Offers, Tech happenings around the world, New Release and Sessions with the Authors:

<https://discord.bpbonline.com>



APPENDIX A

Streamlit Cheat Sheet

Objectives

Let us summarize the Streamlit commands we learned. This sheet can be used as a quick reference for your Streamlit projects.

Streamlit commands

The following are the most commonly used Streamlit commands:

- **Setup and configuration:**

- Set page-wide configurations like the title and layout:

```
st.set_page_config(page_title="Page Title",  
                   layout='centered',  
                   initial_sidebar_state='auto')
```

- **Layout and structure:**

- Add any widget to the sidebar:

```
st.sidebar.<widget>()
```

- Create a list of columns to layout widgets horizontally:

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

- Create a container to hold a block of widgets:

```
st.container()
```

- Create an expandable section that users can open and close:

```
st.expander("Label", expanded=False)
```

- **Widgets:**

- Create a button that users can click with `Click Me` text:

```
st.button("Click Me")
```

- Create a checkbox that users can toggle:

```
st.checkbox("Select if applicable")
```

- Create a set of radio buttons for selecting options:

```
st.radio("Choose one option",  
        ['Option1', 'Option2', 'Option3'])
```

- Create a dropdown menu for selecting options:

```
st.selectbox("Label", options)
```

- Create a dropdown menu for selecting multiple options:

```
st.multiselect("Label", options)
```

- Create a slider for selecting a range of values:

```
st.slider("Label", min_value, max_value,  
         value)
```

- Create a text input box:

- ```
st.text_input("Enter your text Input")
```
- Create a number input box:  

```
st.number_input("Label", min_value,
max_value, value)
```
- Create a large text input area:  

```
st.text_area("Enter your large text input")
```
- Create an input for selecting a date:  

```
st.date_input("Select the Date input")
```
- Create an input for selecting a time:  

```
st.time_input("Select Date Time input")
```
- Create an input for uploading files:  

```
st.file_uploader("Label", type=['csv'])
```
- Create an input for selecting a color:  

```
st.color_picker("Label", '#000000')
```
- **Display data:**
  - Write data or text to the app:  

```
st.write("Hello, Streamlit!")
```
  - Display a dataframe as an interactive table:  

```
st.dataframe(data)
```
  - Display a static table of data:  

```
st.table(data)
```
  - Display JSON content:  

```
st.json(data)
```
  - Display a metric with an optional change indicator:

```
st.metric(label, value, delta)
```

- **Media and static content:**

- Display an image:

```
st.image(image='path/to/your/image.jpg')
```

- Embed an audio player:

```
st.audio(data='path/to/your/audiofile.mp3')
```

```
st.audio(data='https://example.com/your-audiofile.mp3')
```

- Embed a video player:

```
st.video('path/to/your/videofile.mp4')
```

```
st.video('https://example.com/your-videofile.mp4')
```

- Create a download button for downloading data from the app:

```
st.download_button(label, data, file_name, mime)
```

- **Control flow:**

- Create a placeholder to dynamically update content:

```
st.empty()
```

- **Styling and themes:**

- Display text using Markdown:

```
st.markdown("Text in **bold**")
```

- Display a caption under widgets or media:

```
st.caption("Text")
```

- MMDisplay mathematical expressions formatted as LaTeX:

```
st.latex(r'Formula')
```

- Display code with syntax highlighting:

```
st.code("Code")
```

- **Caching and performance:**

- Decorate a function to cache its data loaded:

```
@st.cache_data(ttl=xx, show_spinner = True,
max_entries = xxx, hash_function =
Custom.Logic)
```

- Similar to `st.cache` but for caching a resource that should be available globally:

```
@st.cache_resource
```

- **Debugging and error handling:**

- Display an error message:

```
st.error("Error message")
```

- Display a warning message:

```
st.warning("Warning message")
```

- Display an informational message:

```
st.info("Info message")
```

- Display a success message:

```
st.success("Success message")
```

- **Advanced features:**

- Manage session state to persist user inputs across interactions:

```
st.session_state
```

- Rerun the script from the top:

- ```
st.rerun()
```
- Stop execution of the script:

```
st.stop()
```

Join our book's Discord space

Join the book's Discord Workspace for Latest updates, Offers, Tech happenings around the world, New Release and Sessions with the Authors:

[https://discord.\(bpbonline.com](https://discord(bpbonline.com)



APPENDIX B

Additional Resources and References

Introduction

The following are the curated resources, including GitHub repos, documentation, books, and research papers to enhance your understanding of Streamlit, user experience design, the future of AI, and ethical considerations in AI. This section is a gateway to continue your journey beyond this book, empowering you to design and build cutting-edge and ethical great applications.

Streamlit projects

The following are the additional resources and references:

- Streamlit official documentation:
<https://docs.streamlit.io/>
- Streamlit official app gallery:
<https://streamlit.io/gallery>
- Streamlit official component gallery:

<https://streamlit.io/components>

- Streamlit official community boards:
<https://discuss.streamlit.io/>
- Streamlit official GitHub repo:
<https://github.com/streamlit>
- Streamlit and Snowflake official guide:
<https://docs.snowflake.com/en/developer-guide/streamlit/about-streamlit>
- *Jrieke* GitHub repo for Streamlit projects:
<https://github.com/jrieke/best-of-streamlit>
- *MarcSkovMadsen* GitHub repo for Streamlit projects:
<https://github.com/MarcSkovMadsen/awesome-streamlit>

User Experience design

The following are some books and references to expand understanding of User experience design:

- This is a great place to start if you want to understand the basic principles of UX design.
[https://lawsофux.com/](https://laws офux.com/)
- First published in Jan 2000, this book is considered a classic book on web usability.
Don't Make Me Think, Revisited: A Common Sense Approach to Web Usability by Steve Krug
- This book provides a structured approach for building your projects and using UX principles.

A Project Guide to UX Design: For User Experience Designers in the Field or in the Making by *Russ Unger* and *Carolyn Chandler*

- The *Nielsen Norman Group* is one of the leading authorities on UX design. Their website offers articles, videos, and reports on best practices, usability testing, and design principles. This is good for beginners as well as advanced users.

<https://www.nngroup.com/>

Future of AI and ethical usage

The following are the additional resources and references for the usage of AI tools:

- Ethics of Artificial Intelligence and Robotics by *Vincent C. Müller* (Stanford Encyclopedia of Philosophy)

<https://plato.stanford.edu/entries/ethics-ai/>

- The Ethics of Artificial Intelligence by *Nick Bostrom* and *Eliezer Yudkowsky* (Cambridge Handbook of Artificial Intelligence)

<https://nickbostrom.com/ethics/artificial-intelligence.pdf>

- Ethics Guidelines for Trustworthy AI by the European Commission's High-Level Expert Group on Artificial Intelligence

<https://digital-strategy.ec.europa.eu/en/library/ethics-guidelines-trustworthy-ai>

- Envisioning the future of the AI research ecosystem by *Sethuraman Panchanathan*

<https://academic.oup.com/pnasnexus/article/3/2/pgae021/7610118?login=false>

- Survey Paper Thousands of AI Authors on the Future of AI

<https://arxiv.labs.arxiv.org/html/2401.02843>

- Interesting AI research papers from 2024

<https://www.analyticsinsight.net/artificial-intelligence/10-breakthrough-ai-research-papers-to-read-in-2024>

- Interesting AI research papers from 2023

<https://magazine.sebastianraschka.com/p/10-ai-research-papers-2023>

Join our book's Discord space

Join the book's Discord Workspace for Latest updates, Offers, Tech happenings around the world, New Release and Sessions with the Authors:

<https://discord.bpbonline.com>



APPENDIX C

Docker 101: Beginner's Guide to Containers

Objectives

If you have ever heard about *containers* and wondered what they are, this guide can explain them. It is designed for those who are new to the world of containers.

You will learn what containers are, why they are useful, and how to use them with Docker. By the end, you will clearly understand the essentials and be on your way to exploring the world of containers.

Introduction to containers

Containers are like little boxes where you can pack everything an application needs to run. This means that the application will always have what it needs wherever you move this box. It is a way to ensure that your software works the same way regardless of where it is running.

Let us try to understand this using an analogy.

Understanding containers with traveling stage show analogy

The stage show analogy effectively simplifies the concept of containers for beginners. It helps users grasp the idea of consistent environments across different systems.

Imagine you are the director of a popular stage show that travels from city to city. Every city has its theaters, but they are all different in size, design, and equipment. To ensure your show remains consistent in every city, you decide to bring along not just the actors and the costumes but also the stage setup, props, lighting, and sound system. Everything is packed up in a special truck so that wherever you go, you simply unpack, and the show looks and sounds the same, regardless of the differences in local theaters.

Containers are like that special truck for the traveling show. Instead of stage props and lighting, they carry the application, related software, libraries, and dependencies. No matter where you *run* your software, be it your personal computer, a friend's laptop, or a production server halfway across the world, the *show* (software) plays out just as expected because it uses the familiar setup from the container.

It is quite common to hear, *But it works on my machine!* when software behaves differently in different environments. Containers essentially say, No worries, I have brought my own stage! ensuring the performance is consistent every time.

In a nutshell, containers provide a consistent environment for software to run, much like our traveling stage show, which always has a familiar setup to deliver its performance perfectly, city after city.

Docker: The stage crew for your traveling show

Now that you have your magical truck that packs everything needed for the show to run smoothly, how do you ensure that the unpacking and setup in every city are done efficiently?

Enter the stage crew, a group of experts who know exactly how to set up the props, where to position the lights, and how to arrange the sound system. They ensure everything is done right so that when the curtain rises, the show begins flawlessly:

Docker is like that stage crew for your software.

When you want to pack up your software with all its components into a container, Docker helps you create the container (or pack the truck). When you want to run the software, Docker helps you deploy the container efficiently (*set up the stage*). What about when the show is over and it is time to move to a new city or update the act? Docker helps you manage, update, or remove containers as needed.

However, it is not just about setup and takedown. The stage crew also ensures the backstage operations run smoothly, like controlling the lighting or managing quick costume changes. Similarly, Docker gives you tools to monitor your software, handle networking, manage storage, and ensure security.

In essence, while containers are the magic trucks that ensure your software's performance remains consistent, Docker is the stage crew that makes it all happen seamlessly behind the curtains.

Docker basics

Before we dive into how to run docker images, it is important to understand *two* essential terms:

- **DockerFile:** This is the set of instructions for creating the docker image.

- **Docker image:** Think of this as a blueprint or a template. It is a snapshot of a system with everything needed to run a particular application.
- **Docker container:** This is a running instance of an image. If an image is like a class in programming, a container is like an object, an instance of that class.

With reference to the above analogy for setting up the stage for the traveling show, the following figure demonstrates the role of DockerFile, Docker image, and Docker container.

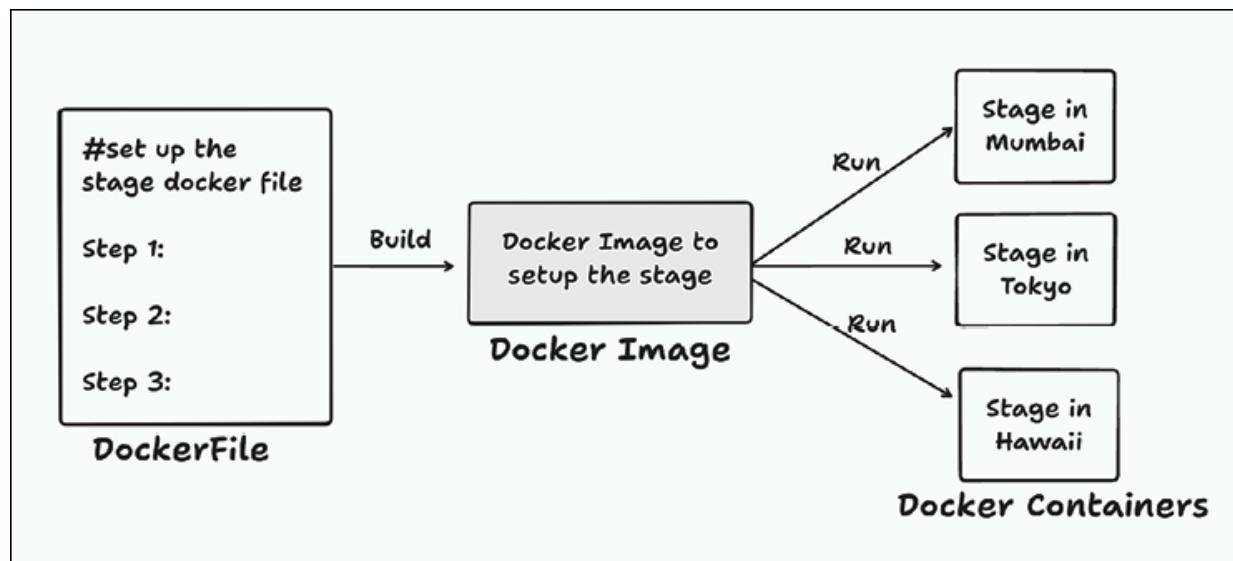


Figure C.1: Understanding Docker components.

Run docker image

Running a Docker container is a straightforward process, but understanding what happens behind the scenes can make it even more exciting. Let us look at the running of a container:

The prerequisites are as follows:

- You need to install the docker in your system:
 - Docker Desktop
 - [\(https://docs.docker.com/desktop/\)](https://docs.docker.com/desktop/)

Getting started

Let us start by running a container to understand the concept of containerization better:

1. Before running a container, you need an image to base it on. This image is like a blueprint containing everything the container needs to run. For our first container, let us use the `hello-world` image. It is a simple image created by Docker to test if your installation works correctly.
2. First, ensure you have the image on your local machine. If not, Docker can fetch (or pull) it for you from Docker Hub, a cloud-based registry service that houses many public images.

The following command tells Docker to get the hello-world image. This step is optional, and if you do not explicitly pull an image, Docker will do it automatically in the next step when you try to run the container.

```
docker pull hello-world
```

3. Now that you have the image, you can start a container based on it. Run the following command:

```
docker run hello-world
```

- a. When you issue this command, several things happen:
 - Docker checks if the hello-world image is available locally.
 - If not found, Docker will pull it from Docker Hub.
 - Docker creates a new container from that image.
- b. Once the image container is created
 - Docker loads the image's filesystem, a lightweight, stand-alone, executable package

that includes everything needed to run the software.

- Docker creates an isolated environment for the container. This environment contains everything the software needs to run (code, runtime, system tools, system libraries, and settings).
- The container starts and runs its designated process.
- Once the container is up and running, it runs a command that sends a message to your console, ensuring Docker works correctly.

Cleaning up

After running containers, especially when experimenting, cleaning up unused containers and images is good practice to save space. This helps free up the disk space and avoid the clutter.

- To see a list of all containers:

```
docker ps -a
```

- To remove a container:

```
docker rm [container_id_or_name]
```

- To remove an image

```
docker rmi hello-world
```

Building custom images

Now, after getting an idea of Docker's workings and applications, let's dive into the most common use case. We will build a custom image for Streamlit deployment.

To build a custom Docker image, you will rely on a Dockerfile. A Dockerfile is a script that contains instructions for telling Docker how to build the image.

To understand the structure of Dockerfile, let us use the Dockerfile from [Chapter 2, Getting Started with Streamlit](#).

The following is the code snippet with an explanation for each line:

```
#This line indicates that the base image for this Dockerfile is python:3.9-slim. The Docker container will be based on a slim version of Python 3.9. The slim version is a lightweight variant that omits certain packages and functionalities not required for running Python.
```

```
FROM python:3.9-slim
```

```
#This line tells Docker to expose port 8501 on the container. It's a hint to those using the image that the containerized application will be listening on port 8501. In this case, it's the port that the Streamlit app will use.
```

```
EXPOSE 8501
```

```
#Before we move on with installing our Python packages, we're ensuring that pip (Python's package installer) is upgraded to its latest version
```

```
RUN pip install --upgrade pip
```

```
#This sets the working directory in the container to /app. All subsequent commands in the Dockerfile will be run in this directory. If the directory doesn't exist, Docker will create it.
```

```
WORKDIR /app
```

```
#This line copies the requirements.txt file from  
the current directory (on your machine or the build  
context) to the current location inside the  
container, which is /app (as defined by the  
WORKDIR).
```

```
COPY requirements.txt .
```

```
#This installs the Python packages listed in  
requirements.txt. The --no-cache-dir flag ensures  
that the Docker image doesn't store any cache from  
the pip installation, making the image smaller.
```

```
RUN pip install --no-cache-dir -r requirements.txt
```

```
#Just like we copied the requirements.txt file,  
we're now copying the data_app.py script into the  
/app directory inside the container.
```

```
COPY data_app.py .
```

```
#This line defines the default command to run when  
the container is started. In this case, it will run  
the data_app.py script using Streamlit on port  
8501. The flags --server.port and --  
browser.serverAddress are specific configurations  
for Streamlit, telling it which port to listen on  
and which address to bind to (in this case, any  
available address).
```

```
CMD streamlit run data_app.py --server.port=8501 --  
browser.serverAddress="0.0.0.0"
```

Another important addition to the docker workflow could be the .dockerignore file, which could be used in Docker projects to specify which files and directories should be ignored when building a Docker image.

Its purpose is to improve the efficiency and security of the Docker image build process by excluding unnecessary or

sensitive files from being included in the image.

Sample `.dockerignore` file.

```
# Ignore all .git files  
.git  
  
# Ignore all node_modules folders  
node_modules  
  
# Ignore environment files  
.env  
  
# Ignore logs  
*.log
```

In summary, the actions performed by this Dockerfile are as follows:

- Uses a slim Python 3.9 image.
- Exposes port 8501 for Streamlit.
- Sets `/app` as the working directory.
- Copies and installs the necessary Python dependencies.
- Copies the Streamlit app script (`data_app.py`).
- Runs the Streamlit app when the container is started.

Once your dockerfile is ready, you can build it. Use the following code:

```
docker build -t image_name:latest .
```

The functions performed are as follows:

- `-t` allows you to tag the image with a name (optionally a version).
- The `.at` the end tells Docker to use the current directory for the build context (where the Dockerfile is located).

Docker reads the Dockerfile, executes the commands in order, and then returns a new image.

The next step is to verify the built image:

```
docker images
```

Your new image, `my-custom-image`, should be on the list.

Now that we have the image ready, we can create containers from it just like any other image:

```
docker run image_name:latest
```

Building custom Docker images provides you with the flexibility to capture your applications and their environments precisely. Over time, as you become more familiar with Docker and its intricacies, you will discover even more advanced ways to optimize and refine your Docker images.

The basic commands are as follows:

- **Pulling an image:** To download (or pull) an image from Docker Hub, a cloud-based registry service

```
docker pull [image_name]
```

- **Listing image:** To see a list of images on your system:

```
docker images
```

- **Running a container:** To start a container from an image:

```
docker run [image_name]
```

- **Listing running containers:** To see containers currently running on your system:

```
docker ps
```

- To see all containers, including those not currently running:

```
docker ps -a
```

- **Stopping a container:** To stop a running container:

```
docker stop [container_id_or_name]
```

Note: You can find the container ID or name using the docker ps command.

- **Removing a container:** If you want to delete a container from your system:

```
docker rm [container_id_or_name]
```

- **Removing an image:** To remove an image from your system:

```
docker rmi [image_name]
```

These commands will help you get started with Docker. For more advanced manuals and guides, you can refer to the official docker website

(<https://docs.docker.com/reference/>).

Join our book's Discord space

Join the book's Discord Workspace for Latest updates, Offers, Tech happenings around the world, New Release and Sessions with the Authors:

[https://discord\(bpbonline\).com](https://discord(bpbonline).com)



Index

A

advanced time series forecasting application project 212
implementation guide 212, 213
AI-assisted development tools 240
additional resources and references 262, 263
advantages 240, 241
guardrails 241
manual code, shifting from 240
AI-powered coding assistants 242
Codota 242
Kite 242
AI tools categories 241
AI-powered coding assistants 242
bug detection and code review tools 242
code completion tools 241
comparison 243, 244
developer bots 243
prompt engineering platforms 242
app testing 151-153
best practices 153
area charts 78
artificial intelligence (AI) 240

B

bar charts 78
Basic Stock Price Viewer project 186
implementation guide 187
BMI calculator project 188
implementation guide 188
budget planner project 197
implementation guide 197
bug detection and code review tools 242
DeepCode 242
SonarQube 242
button widgets 36, 37

C

callbacks [142-144](#)
ChatGPT [242](#)
 data privacy [244, 245](#)
 for Streamlit development [244-255](#)
 intellectual property guardrails [245](#)
 overview [244](#)
 terms and conditions [245](#)
CI/CD pipeline [160](#)
Cloud Run
 metrics and alert information, obtaining from [180, 181](#)
code completion tools [241](#)
 GitHub copilot [241](#)
 TabNine [241](#)
Codota [242](#)
community engagement [226](#)
 key resources [226](#)
 official forums and discussion boards [226](#)
 Streamlit Community Forum [227](#)
 Streamlit Component Gallery [235](#)
 Streamlit Official App Gallery [230](#)
containers [265](#)
 cleaning up [269](#)
 with traveling stage show analogy [266](#)
continuous deployment
 setting up, for data application [176-179](#)
currency converter project [189](#)
 implementation guide [189](#)
 project structure [193, 194](#)
custom classes
 strategies, for defining [153, 154](#)
custom components [67](#)
customizable news aggregator project [212](#)
 implementation guide [212](#)

D

database connections [149, 150](#)
 best practices [151](#)
data presentation
 challenges [2, 3](#)
data visualization
 importance in data analysis [78](#)
data widgets [37-42](#)
date and time widgets [27-29](#)
DeepCode [242](#)
design patterns pitfalls [153](#)

developer bots 243
 GitBot 243
display and aesthetics widgets 91
 chat widgets 97-99
 display message widgets 94, 95
 display progress and status widgets 92-94
 media widgets 95, 96
 metric widgets 96, 97
display widgets 24, 26
Distributed Denial of Service (DDoS) attacks 159
Docker 8, 265, 266
 basics 267
Docker container 267
DockerFile 18, 267
Docker image 267
 custom image, building 269-272
 running 267, 268

G

Generative Pre-trained Transformer (GPT) architecture 244
GitBot 243
GitHub 237
GitHub copilot 241
Google Cloud Platform (GCP) 163
Google Cloud Run 163

I

Image Gallery project 187
 implementation guide 187
Interactive Geographic Information System (GIS) project 213
 implementation guide 213
Interactive Quiz project 189
 implementation guide 190

J

job application tracker project 196
 implementation guide 196, 197

K

Kite 242

L

Large Language Models (LLMs) 97
line charts 78

M

machine learning [118, 119](#)
machine learning environment
 essential Python libraries [119](#)
machine learning trained model integration [121, 122](#)
ML models, preparing [120](#)
 setting up [119](#)
 simple machine learning model [120, 121](#)
machine learning model deployment project [211](#)
 implementation guide [211](#)
map charts [79](#)
Medium [237](#)
model hyperparameters [122](#)
 interactive controls, implementing [122-124](#)
 performance improvement tips [124](#)
movie recommendation system project [188](#)
 implementation guide [188](#)
multi-page data analysis dashboard project [196](#)
 implementation guide [196](#)
multi-page event planner [200](#)
 implementation guide [200](#)
multi-page portfolio website
 implementing [201-207](#)
multi-page portfolio website project [198](#)
 implementation guide [198](#)
multi-page product catalog [199](#)
 implementation guide [199](#)
multi-page recipe organizer [200](#)
 implementation guide [200](#)
multiple-choice widgets [35](#)
mutating charts
 best practices [149](#)
 building [148, 149](#)

N

news aggregator application
 sample project, implementing [213-223](#)
numerical input widgets [27](#)

O

OpenAI Codex [242](#)

P

personal fitness tracker project [197](#)

implementation guide 197
portfolio app
 project structure 201
product recommender app
 building 16
 deploying 164-176
 enhancements 68-74
 ML integration 125-137
 project structure 16, 17, 18
 visual elements, integrating 100-114
 widgets, adding 43
 widgets, selecting 43-47
professional portfolio
 comprehensive portfolio 237
 crafting 236
 skills, marketing through 238
 tools, for documentation 237
projects 185
prompt engineering platforms 242
 ChatGPT 242
 OpenAI Codex 242

R

random quote generator project 190
 implementation guide 190
real-time data analytics dashboard project 210
 implementation guide 210

S

scatter charts 78
secret management 145, 146
 best practices 147
secure financial portfolio manager project 210
 implementation guide 211
simple expense tracker 189
 implementation guide 189
single-choice widgets 32-34
SonarQube 242
sports team management application 199
 implementation guide 199
state management 62-66
Streamlit 1
 additional resources and references 261, 262
 advanced topics 142
 core concepts 11

installation options 8
journey 226
machine learning 118
purpose 3-5
running locally, using virtual environment 8
running, using Docker 8
running, using Streamlit community cloud 9-11
setting up 8
UI design 50
visualization options 78
widgets 23

Streamlit advanced projects 210
advanced time series forecasting application 212
customizable news aggregator 212
Interactive Geographic Information System (GIS) 213
machine learning model deployment 211
real-time data analytics dashboard 210
secure financial portfolio manager 210

Streamlit application
debugging tips 163
deployment options 181
hosting 160
hosting considerations 158, 159
hosting, with GCP Cloud Run 163
hosting, with Streamlit Share 160-163
running 18-20

Streamlit app model 12
Streamlit commands 257-260
Streamlit Community Forum 227-230
Streamlit Component Gallery 235
benefits 235
best practices 235, 236

Streamlit core widgets 24
button widgets 36, 37
data widgets 37-42
date and time widgets 27-29
display widgets 24-26
multiple-choice widgets 35
numerical input widgets 27
other input widgets 30, 31
single-choice widgets 32-34
text input widgets 26

Streamlit easy projects 186
Basic Stock Price Viewer 186, 187
BMI calculator 188
currency converter 189

Image Gallery project 187
Interactive Quiz 189, 190
movie recommendation system 188
random quote generator 190
simple expense tracker 189
to-do list application 187
weather dashboard 186
Streamlit hello app
exploring 12-15
Streamlit intermediate projects
budget planner with visualizations 197
job application tracker 196
multi-page data analysis dashboard project 196
multi-page event planner 200
multi-page portfolio website 198
multi-page product catalog 199
multi-page recipe organizer 200
personal fitness tracker 197
sports team management application 199
task management application 198
Streamlit Magic 42, 43
Streamlit Official App Gallery 230-234
highlights 231
Substack 237

T

TabNine 241
task management application project 198
implementation guide 198
text input widgets 26
timezones
best practices 147
working with 147
to-do list application 187
implementation guide 187

U

UI design, Streamlit 50
control flow widgets 55-57
HTML and CSS, rendering 60-62
layouts and containers 51-54
theming capabilities 58-60
utility widgets 54, 55
User experience design 262

V

visual elements

- integrating, in product recommender app [100-114](#)
- visualization options, Streamlit [78](#)
 - chart types and use cases [83, 84](#)
 - core chart elements [78-82](#)
 - supported charting libraries [84-91](#)

W

weather dashboard project [186](#)

- implementation guide [186](#)
- project structure [191, 192](#)
- sample project, implementing on [190](#)
- widgets [23](#)
 - adding, to product recommender app [43](#)
 - overview [24](#)
 - selecting, for product recommender app [43-47](#)