Car Sales System Tutorial

A Beginner’s Guide to Building a Python Project

By Phines Macharia

Preface

*Welcome to the **Car Sales System Tutorial**! This book guides beginners through building a Python-based car sales system. Each chapter offers clear explanations, analogies, and tips to ensure success. Whether you're new to coding or brushing up, you'll create a secure, functional program. Let's dive in!*

Who This Book is For

This book targets absolute beginners with little programming experience. If you know basic Python (e.g., variables), great! If not, we explain everything step-by-step to make learning fun and accessible.

What You'll Learn

- How to store data in a **database**.
- Creating **classes** to organize code.
- Building a terminal interface with menus.
- Adding security with logins.
- Testing and optimizing your program.

Table of Contents

Chapter 1: Your First Step into the Car Sales Adventure

***Chapter Overview**: Welcome to your programming journey! This chapter introduces you to the car sales system you'll build using Python. We'll explain what the system does, why it's exciting, and what you need to get started. Whether you've never written code or are brushing up, this chapter will spark your enthusiasm for creating a real-world project!*

1.1 Welcome to Your Programming Journey!

Imagine owning a car dealership where you manage cars, customers, and sales—all through a computer program you build yourself! In this book, you'll create a **car sales system** using **Python**, a beginner-friendly programming language. This system will act like a digital assistant for a car lot, helping you store car details, track customers, and even suggest cars to buyers. The best part? You don't need any coding experience—we'll guide you step-by-step, explaining every piece as if you're learning for the first time.

Programming is like giving instructions to a super-smart robot (your computer) to do tasks for you. For example, you'll tell your program to save a car's details (like 'Toyota Camry, 2020, $20,000') or show a list of cars to a customer. By the end of this book, you'll have a working program you can show off, and you'll feel like a coding superhero!

Why This Matters: Building this project teaches you skills used in real jobs, like organizing data, creating secure logins, and making user-friendly interfaces. Plus, it's incredibly fun to see your code come to life as a working system!

1.2 Why Python?

Python is one of the easiest programming languages to learn, making it perfect for beginners. It's like writing instructions in plain English, so your computer understands exactly what you want. Python is also super popular—companies like Google, Netflix, and even car companies use it to build apps, analyze data, and more.

Here's why Python is great for this project:

- **Easy to Read**: Python code looks clean and simple, like a recipe.
- **Powerful Tools**: Python has libraries (pre-built code) to handle databases, APIs, and interfaces.
- **Beginner-Friendly**: You'll write short, clear code that does big things.
- **Fun!**: Python makes it easy to focus on building cool features without getting stuck on complex rules.

For Beginners: Think of Python as a friendly guide who helps you talk to your computer. You say, 'Save this car,' and Python makes it happen!

1.3 What You'll Build: Your Digital Car Dealership

Your car sales system will be like a digital version of a car dealership, complete with a virtual lot, customer records, and a cashier's desk. You'll build a program that runs in your computer's terminal (a text-based window) and lets users interact with menus to buy, sell, or browse cars. Here's a detailed look at what your system will do:

- **Store Car Details**: Save information like make (e.g., Toyota), model (e.g., Camry), year, price, and more in a **database**, like a digital filing cabinet.
- **Manage Cars**: Let admins add, update, or remove cars, like stocking a car lot.
- **Track Customers**: Store customer names and contact info to keep track of buyers.
- **Search Cars**: Allow customers to search for cars by make or model, like browsing an online store.
- **Fetch Real Data**: Use **APIs** (internet services) to get car details or market prices, like checking a car's value online.
- **Smart Suggestions**: Add **AI** to recommend cars based on what customers like, like a salesperson suggesting similar models.
- **Secure Logins**: Protect the system with usernames and passwords, so only authorized users can access it.
- **Easy Menus**: Create a text-based interface with menus, so users can choose options by pressing numbers, like a video game menu.

Analogy: Your system is like a LEGO car dealership. Each chapter adds a new piece—cars, customers, menus—until you have a complete, working model. We'll explain how every piece fits, so you're never lost!

1.4 A Sneak Peek at the Code

Wondering what your project will look like? Let's preview some real code you'll write in later chapters. Don't worry if it looks unfamiliar now—we'll explain every line when you get there. These examples show the exciting things you'll create!

Example 1: Setting Up the Database

You'll create a database to store car details, like a digital notebook. This code sets up a table for cars:

Error rendering code block: Preformatted.__init__() got an unexpected keyword argument 'maxWidth'

What's Happening?

- This code creates a file called **car_sales.db** to store data.
- The **cars** table holds details like VIN (unique car ID), make, and price.
- You'll use this to save and retrieve cars, like adding a Toyota Camry to your lot.

Example 2: Defining a Car

You'll create a **Car** class to represent each car, like a blueprint for a vehicle. Here's how it looks:

Error rendering code block: Preformatted.__init__() got an unexpected keyword argument 'maxWidth'

What's Happening?

- The **Car** class stores details for one car, like a digital car file.

- The **save** method puts the car’s info into the database.
- You’ll use this to add cars like ‘VIN001, Toyota, Camry, 2020’ to your system.

Example 3: Creating a Menu

Your system will have a text-based menu, like a game menu. Here’s a simple welcome screen:

Error rendering code block: Preformatted.__init__() got an unexpected keyword argument 'maxWidth'

What’s Happening?

- This code displays a welcome message in the terminal.
- Users press a key to continue, like starting a game.
- You’ll expand this into menus for viewing cars or adding customers.

For Beginners: These snippets are like puzzle pieces. You’ll learn to build each one, and by the end, they’ll fit together into a complete system!

1.5 What You Need to Start

You don’t need fancy equipment or prior knowledge to begin. Here’s a simple checklist to get ready:

- **Python 3**: A free program you download from **python.org**. It’s like the toolbox for your project.
- **A Computer**: Any computer with Windows, macOS, or Linux will work.
- **A Terminal**: A text window to run your code. Windows has Command Prompt or PowerShell; macOS and Linux have Terminal.
- **Curiosity**: No coding experience needed—just a willingness to learn!

To check if Python is installed, open your terminal and type:

Error rendering code block: Preformatted.__init__() got an unexpected keyword argument 'maxWidth'

If you see a version number (e.g., ‘Python 3.10.0’), you’re ready! If not, visit **python.org**, download Python 3, and follow the installation instructions. It’s as easy as installing a game!

Troubleshooting Tip: If **python3 --version** doesn’t work, try **python --version**. If both fail, download Python from **python.org** and ensure it’s added to your system’s PATH (the installer has an option for this).

1.6 How to Approach This Book

This book is designed for absolute beginners, so we’ll take things slowly and clearly. Here’s how to make the most of it:

- **Follow Along**: Type the code examples as you read. It’s like practicing a recipe by cooking it yourself.
- **Make Mistakes**: Errors are part of learning. If code doesn’t work, we’ll show you how to fix it.
- **Read the Analogies**: We use examples like LEGO or car dealerships to make ideas clear.

- **Use the Troubleshooting Tips**: Each chapter has tips to solve common problems.
- **Have Fun**: Treat this like a game—each chapter unlocks a new feature for your system!

Each chapter builds on the previous one, starting with setting up your tools (Chapter 2) and ending with sharing your project (Chapter 15). By the end, you'll have a complete car sales system and the confidence to code more projects!

For Beginners: Don't worry if terms like 'database' or 'API' sound scary now. We'll explain them like teaching a friend, and you'll be using them like a pro!

1.7 Key Takeaways

- You're building a car sales system with Python, a beginner-friendly language.
- The system includes real-world features like databases, APIs, and secure logins.
- You need Python 3, a computer, and a terminal—no prior coding experience required.
- This book guides you step-by-step with analogies and troubleshooting tips.
- You'll see real code examples, like saving cars and creating menus, that bring your system to life.

Chapter 2: Setting Up Your Coding Workspace

Chapter Overview*: This chapter helps you set up your project folder, virtual environment, and tools. It's like preparing a workbench for your car sales system.*

2.1 Creating Your Project Folder

Your project needs a dedicated folder. This keeps everything organized.

Open your terminal and type:

Error rendering code block: Preformatted.__init__() got an unexpected keyword argument 'maxWidth'

What's Happening?

- **mkdir car_sales_system**: Creates a folder.
- **cd car_sales_system**: Enters the folder.

Why This Matters: A clean folder is like a tidy desk for coding.

Troubleshooting Tip: Type **ls** (or **dir** on Windows) to confirm the folder exists. Check spelling if it's missing.

2.2 Setting Up a Virtual Environment

A **virtual environment** isolates your project's tools, avoiding conflicts.

Error rendering code block: Preformatted.__init__() got an unexpected keyword argument 'maxWidth'

What's Happening?

- **python3 -m venv venv**: Creates a virtual environment.
- **source venv/bin/activate**: Activates it (shows '(venv)').

For Beginners: It's like a private toolbox for your project's tools.

Troubleshooting Tip: If activation fails, ensure Python 3 is installed (**python3 --version**).

2.3 Installing Libraries

Libraries are pre-built tools for your project. We need a few.

Error rendering code block: Preformatted.__init__() got an unexpected keyword argument 'maxWidth'

What's Happening?

- **pip**: Installs libraries.
- **reportlab**: Creates PDFs.
- **requests**: Fetches internet data.
- **bcrypt**: Secures passwords.
- **pytest**: Tests code.

Why This Matters: Libraries save time by providing ready-made features.

Troubleshooting Tip: Ensure '(venv)' is active. Reactivate with **source venv/bin/activate** if needed.

2.4 Testing Your Setup

Let's confirm everything works with a simple test.

Error rendering code block: Preformatted.__init__() got an unexpected keyword argument 'maxWidth'

Save as **test_setup.py** and run:

Error rendering code block: Preformatted.__init__() got an unexpected keyword argument 'maxWidth'

What's Happening?

- **print**: Displays text.
- Seeing 'Setup is working!' means you're ready.

Troubleshooting Tip: If nothing runs, check your folder (**pwd**) and virtual environment.

Key Takeaways

- Organize with a project folder.
- Use a virtual environment for tools.
- Install libraries with pip.
- Test your setup.

Chapter 3: Building Your Data Storage (Database)

***Chapter Overview**: This chapter teaches you to create a **database** using **SQLite** to store cars, customers, and sales data. You'll learn what databases are and how to set them up.*

3.1 What's a Database?

A **database** is a digital filing cabinet for organized data. **SQLite** is a simple database stored in a file, perfect for this project.

For Beginners: Think of a database as a super-organized notebook that never loses data.

Why This Matters: Databases ensure your system remembers everything reliably.

3.2 Designing the Database

Databases use **tables** (like spreadsheets) with **columns** (headings) and **rows** (entries). We'll create tables for cars, customers, sales, users, and favorites.

Error rendering code block: Preformatted.__init__() got an unexpected keyword argument 'maxWidth'

What's Happening?

- 1. **import sqlite3**: Imports the database tool.
- 2. **conn = sqlite3.connect(...)**: Creates a database file.
- 3. **c.execute(...)**: Defines tables and columns.
- 4. **PRIMARY KEY**: Ensures unique entries.
- 5. **FOREIGN KEY**: Links tables (e.g., sales to cars).
- 6. **conn.commit()**: Saves changes.

For Beginners: Tables are like spreadsheets. Each car is a row, with columns for details like price.

3.3 Starting the Database

Create a **main.py** file to initialize the database.

Error rendering code block: Preformatted.__init__() got an unexpected keyword argument 'maxWidth'

Run it:

Error rendering code block: Preformatted.__init__() got an unexpected keyword argument 'maxWidth'

Troubleshooting Tip: If errors occur, check **database.py** for typos or ensure you’re in the project folder (**pwd**).

3.4 Testing the Database

Verify the tables were created.

Error rendering code block: Preformatted.__init__() got an unexpected keyword argument 'maxWidth'

Save as **test_database.py** and run:

Error rendering code block: Preformatted.__init__() got an unexpected keyword argument 'maxWidth'

Troubleshooting Tip: If tables are missing, re-run **python3 main.py**.

Key Takeaways

- Databases organize data in tables.
- **SQLite** is simple and file-based.
- Test your database setup.

Chapter 4: Creating Your First Class (The Car)

***Chapter Overview**: This chapter introduces **classes** by building a **Car** class to represent cars. You'll learn how classes organize code.*

4.1 What is a Class?

A **class** is a blueprint for objects. The **Car** class defines car attributes like VIN or price. Each car you create is an **object**.

For Beginners: A class is like a cookie cutter. Each cookie (object) has the same shape but different flavors (details).

Why This Matters: Classes keep code organized, reusing the same blueprint for all cars.

4.2 Building the Car Class

Step 1: Start the Class

Error rendering code block: Preformatted.__init__() got an unexpected keyword argument 'maxWidth'

What's Happening?

- **class Car:** Defines the blueprint.
- **pass**: Placeholder for code.

Step 2: Add Attributes

Error rendering code block: Preformatted.__init__() got an unexpected keyword argument 'maxWidth'

What's Happening?

- 1. **def __init__(...):** Initializes car attributes.
- 2. **self**: Refers to the car object.
- 3. **self.vin = vin**: Stores the VIN.

For Beginners: **__init__** is like filling out a car's registration form. **self** is the car holding the details.

Step 3: Connect to the Database

Add methods to save and load cars in **models.py**.

Error rendering code block: Preformatted.__init__() got an unexpected keyword argument 'maxWidth'

What's Happening?

- 1. **save(self):** Saves the car to the database.
- 2. **load_all()**: Retrieves all cars.
- 3. **@staticmethod**: No **self** needed for **load_all**.

Troubleshooting Tip: If saving fails, ensure the **cars** table exists (**python3 main.py**).

4.3 Testing the Car Class

Test saving and loading a car.

Error rendering code block: Preformatted.__init__() got an unexpected keyword argument 'maxWidth'

Save as **test_car.py** and run:

Error rendering code block: Preformatted.__init__() got an unexpected keyword argument 'maxWidth'

Troubleshooting Tip: If tests fail, check **models.py** and **database.py**.

Key Takeaways

- **Classes** are blueprints for objects.
- The **Car** class manages car data.
- Test classes to ensure they work.

Chapter 5: More Classes (Customer and User)

***Chapter Overview**: This chapter builds **Customer** and **User** classes for buyers and logins. You’ll learn to create classes for different purposes.*

5.1 The Customer Class

The **Customer** class stores buyer details like name and contact info.

Error rendering code block: Preformatted.__init__() got an unexpected keyword argument 'maxWidth'

What’s Happening?

- 1. **__init__**: Sets customer details.
- 2. **save**: Adds or updates a customer.
- 3. **load_all**: Retrieves all customers.

For Beginners: The **Customer** class is like a contact card for buyers, stored digitally.

5.2 The User Class

The **User** class handles secure logins with usernames and passwords.

Error rendering code block: Preformatted.__init__() got an unexpected keyword argument 'maxWidth'

What’s Happening?

- 1. **bcrypt**: Hashes passwords for security.
- 2. **save**: Saves a user with a hashed password.

For Beginners: The **User** class is like a secure key card. **bcrypt** locks the password safely.

5.3 Testing the Classes

Test both classes to ensure they save correctly.

Error rendering code block: Preformatted.__init__() got an unexpected keyword argument 'maxWidth'

Save as **test_classes.py** and run:

Error rendering code block: Preformatted.__init__() got an unexpected keyword argument 'maxWidth'

Troubleshooting Tip: Ensure **bcrypt** is installed (**pip install bcrypt**) and the database is initialized.

Key Takeaways

- **Customer** class tracks buyers.
- **User** class secures logins.
- Test classes for reliability.

Chapter 6: Creating a Simple Terminal Interface

***Chapter Overview**: This chapter uses the **curses** library to build a text-based interface, starting with a welcome screen.*

6.1 What is a Terminal Interface?

A **terminal interface** is a text-based menu system where users press keys to choose options, like a simple video game.

For Beginners: It's like a restaurant menu in your terminal—press '1' to pick an option!

Why This Matters: A terminal interface makes your system user-friendly without needing a graphical app.

6.2 Building a Welcome Screen

Create **ui.py** to display a welcome message.

Error rendering code block: Preformatted.__init__() got an unexpected keyword argument 'maxWidth'

What's Happening?

- 1. **curses.curs_set(0)**: Hides the cursor.
- 2. **stdscr.addstr(...)**: Displays text.
- 3. **stdscr.getch()**: Waits for a key press.
- 4. **curses.wrapper**: Sets up the terminal.

Troubleshooting Tip: If **curses** fails, install **ncurses** (**sudo apt-get install libncurses5-dev**) on Linux.

6.3 Testing the Interface

Error rendering code block: Preformatted.__init__() got an unexpected keyword argument 'maxWidth'

What Should Happen: See a welcome message and press any key to exit.

Troubleshooting Tip: Ensure **ui.py** has no typos and you're in the virtual environment.

Key Takeaways

- Terminal interfaces are interactive.

- **curses** controls the terminal.
- Start with a welcome screen.

Chapter 7: Building Advanced Menus

***Chapter Overview**: This chapter adds interactive menus for admins and customers, making the system easier to navigate.*

7.1 Why Add Menus?

Menus let users pick options (e.g., 'View Cars') without typing complex commands, improving usability.

For Beginners: Menus are like a game controller—press a number to act!

Why This Matters: Good menus make your system intuitive for users.

7.2 Creating Menus

Update **ui.py** to add admin and customer menus.

Error rendering code block: Preformatted.__init__() got an unexpected keyword argument 'maxWidth'

What's Happening?

- 1. **admin_menu**: Shows admin options.
- 2. **customer_menu**: Shows customer options.
- 3. **view_all_cars**: Lists all cars.
- 4. **while True**: Keeps menus active until logout.

Troubleshooting Tip: If menus don't work, check **ui.py** imports and **Car** class.

7.3 Testing Menus

Error rendering code block: Preformatted.__init__() got an unexpected keyword argument 'maxWidth'

What Should Happen: Run the program, see the welcome screen, and test menu options (e.g., '1' to view cars).

Troubleshooting Tip: Add cars via a test script if none appear.

Key Takeaways

- Menus improve usability.
- Separate menus for admins and customers.
- Test menus thoroughly.

Chapter 8: Adding Search Functionality

Chapter Overview: This chapter adds a search feature so customers can find cars by make, model, or other criteria.

8.1 Why Search is Important

Search lets customers find specific cars, like typing ‘Toyota’ to see only Toyota models.

For Beginners: It’s like searching a library for books by one author—fast and focused.

Why This Matters: Search makes your system more user-friendly and efficient.

8.2 Creating the Search Function

Create **search.py** to handle car searches.

Error rendering code block: Preformatted.__init__() got an unexpected keyword argument 'maxWidth'

What’s Happening?

- 1. **search_cars**: Builds a dynamic query.
- 2. **params**: Safely passes search criteria.
- 3. **return**: Converts rows to **Car** objects.

For Beginners: This is like asking a clerk to find all Toyota cars in the lot.

8.3 Adding Search to the UI

Update **ui.py** to include a search option.

Error rendering code block: Preformatted.__init__() got an unexpected keyword argument 'maxWidth'

What’s Happening?

- 1. **search_cars_ui**: Prompts for a make and shows results.
- 2. Integrated into the customer menu.

Troubleshooting Tip: If search fails, check **search.py** and ensure cars exist.

8.4 Testing Search

Error rendering code block: Preformatted.__init__() got an unexpected keyword argument 'maxWidth'

What Should Happen: In the customer menu, press ‘2,’ search for ‘Toyota,’ and see Toyota cars.

Troubleshooting Tip: Add cars via the admin menu if none appear.

Key Takeaways

- Search enhances usability.
- Safe queries prevent errors.
- Test search functionality.

Chapter 9: Managing Customers

***Chapter Overview**: This chapter adds features for admins to view and add customers, enhancing system management.*

9.1 Why Manage Customers?

Admins need to track customers to manage sales, like keeping a guest list for your shop. This ensures smooth transactions.

For Beginners: It's a digital notebook for customer details, easy to update and search.

Why This Matters: Customer management is key for sales tracking and relationships.

9.2 Adding Customer Management

Update **ui.py** to include customer management options.

Error rendering code block: Preformatted.__init__() got an unexpected keyword argument 'maxWidth'

What's Happening?

- 1. **manage_customers_ui**: Creates a customer menu.
- 2. **view_customers**: Lists all customers.
- 3. **add_customer_ui**: Adds a new customer.
- 4. Integrated into admin menu.

Troubleshooting Tip: If customers don't save, ensure the **customers** table exists (**python3 main.py**).

9.3 Testing Customer Management

Error rendering code block: Preformatted.__init__() got an unexpected keyword argument 'maxWidth'

What Should Happen: In the admin menu, press '3,' then '2' to add a customer, and '1' to see the list.

Troubleshooting Tip: Check **ui.py** and **models.py** for errors.

Key Takeaways

- Customer management tracks buyers.
- Menus simplify admin tasks.

- Test to ensure functionality.

Chapter 10: Connecting to the Internet (APIs)

***Chapter Overview**: This chapter introduces **APIs** to fetch car details, enhancing your system with external data.*

10.1 What’s an API?

An **API** (Application Programming Interface) is like a messenger fetching data from another system, such as car specs online.

For Beginners: It’s like asking a librarian to grab car info for you!

Why This Matters: APIs make your system dynamic and informative.

10.2 Creating an API Function

Create **api.py** with a placeholder API function.

Error rendering code block: Preformatted.__init__() got an unexpected keyword argument 'maxWidth'

What’s Happening?

- 1. **requests**: Fetches internet data.
- 2. **get_car_details**: Returns fake data for testing.

Troubleshooting Tip: Install **requests** (**pip install requests**).

10.3 Adding API to the UI

Update **ui.py** to show API data when adding cars.

Error rendering code block: Preformatted.__init__() got an unexpected keyword argument 'maxWidth'

What’s Happening?

- 1. **get_car_details**: Fetches API data.
- 2. **add_new_car_ui**: Saves car and shows details.

Troubleshooting Tip: If API data fails, check **api.py** imports.

10.4 Testing the API

Error rendering code block: Preformatted.__init__() got an unexpected keyword argument 'maxWidth'

What Should Happen: Add a car (admin menu, ‘2’) and see fake API data (e.g., ‘Engine: V6’).

Troubleshooting Tip: Ensure **Car** class and database are set up.

Key Takeaways

- APIs add external data.
- Placeholders help test.
- Test API integration.

Chapter 11: More API Fun (Market Value)

***Chapter Overview**: This chapter adds market value data to show fair car prices, using another API function to enhance transparency.*

11.1 Why Market Value Matters

Market value shows if a car's price is competitive, like checking similar cars online. It builds trust with customers by ensuring fair pricing.

For Beginners: It's like looking up a car's worth on a website to ensure you're not overpaying.

Why This Matters: Transparency through market value helps customers make informed decisions.

11.2 Updating api.py

Add a placeholder function to **api.py** to fetch market values.

Error rendering code block: Preformatted.__init__() got an unexpected keyword argument 'maxWidth'

What's Happening?

- **get_market_value**: Returns a fake market value ($20,000) for testing.

Troubleshooting Tip: Ensure **api.py** is in your project folder and imported correctly.

11.3 Showing Market Value in the UI

Update **view_all_cars** in **ui.py** to display market value next to each car's price.

Error rendering code block: Preformatted.__init__() got an unexpected keyword argument 'maxWidth'

What's Happening?

- 1. **from api import get_market_value**: Imports the market value function.
- 2. **market_value = get_market_value(...)**: Fetches the value for each car.
- 3. Displays price and market value together for clarity.

Troubleshooting Tip: If market values don't appear, check **ui.py** imports and ensure cars exist in the database.

11.4 Testing Market Value

Test the updated **view_all_cars** function.

Error rendering code block: Preformatted.__init__() got an unexpected keyword argument 'maxWidth'

What Should Happen: In the admin or customer menu, press ‘1’ to view cars and see prices with market values (e.g., ‘Market: $20000’).

Troubleshooting Tip: Add test cars via the admin menu if none appear.

Key Takeaways

- Market value enhances pricing transparency.
- APIs integrate external data seamlessly.
- Test UI changes to ensure functionality.

Chapter 12: Adding Smart Features (AI)

***Chapter Overview**: This chapter introduces a basic AI feature to recommend cars based on user searches, making the system smarter.*

12.1 What's AI in This Context?

Here, **AI** means simple logic to suggest cars, like recommending similar models based on a customer's search. It mimics intelligent behavior.

For Beginners: It's like a shop assistant saying, 'You like Toyota? Check out these other Toyotas!'

Why This Matters: Recommendations improve user experience by suggesting relevant cars.

12.2 Adding Recommendations

Update **search.py** to include a recommendation function.

Error rendering code block: Preformatted.__init__() got an unexpected keyword argument 'maxWidth'

What's Happening?

- 1. **recommend_cars**: Finds available cars of the same make.
- 2. Returns a list of **Car** objects.

12.3 Showing Recommendations

Update **search_cars_ui** in **ui.py** to show recommendations.

Error rendering code block: Preformatted.__init__() got an unexpected keyword argument 'maxWidth'

What's Happening?

- 1. Imports **recommend_cars**.
- 2. Shows recommendations if a make is searched.
- 3. Displays recommendations below search results.

Troubleshooting Tip: If recommendations don't appear, ensure cars with the searched make exist.

12.4 Testing Recommendations

Test the search with recommendations.

Error rendering code block: Preformatted.__init__() got an unexpected keyword argument 'maxWidth'

What Should Happen: In the customer menu, press ‘2,’ search for ‘Toyota,’ and see search results plus recommended Toyota cars.

Troubleshooting Tip: Add Toyota cars via the admin menu if none appear.

Key Takeaways

- Simple AI improves user experience.
- Recommendations use database queries.
- Test features for accuracy.

Chapter 13: Locking Down Your System (Security)

***Chapter Overview**: This chapter adds a secure login system to protect your system, using **bcrypt** for password hashing.*

13.1 Why Security Matters

A login system ensures only authorized users access the system, protecting sensitive data like customer info or car prices.

For Beginners: It’s like a key card that only lets the right people into a secure building.

Why This Matters: Security prevents unauthorized access and builds trust.

13.2 Creating the Login System

Create **auth.py** to handle logins.

Error rendering code block: Preformatted.__init__() got an unexpected keyword argument 'maxWidth'

What’s Happening?

- 1. **create_user**: Saves a user with a hashed password.
- 2. **login**: Verifies username and password, returns user ID and role.
- 3. **bcrypt**: Secures passwords.

13.3 Adding Default Users

Update **init_db** in **database.py** to add default users for testing.

Error rendering code block: Preformatted.__init__() got an unexpected keyword argument 'maxWidth'

What’s Happening?

- Adds an admin and a customer user.

13.4 Requiring Login

Update **main_ui** in **ui.py** to require login.

Error rendering code block: Preformatted.__init__() got an unexpected keyword argument 'maxWidth'

What’s Happening?

- 1. Prompts for username and password.
- 2. Directs to admin or customer menu based on role.

Troubleshooting Tip: If login fails, ensure the **users** table exists and default users are added.

13.5 Testing Login

Test with default credentials.

Error rendering code block: Preformatted.__init__() got an unexpected keyword argument 'maxWidth'

What Should Happen: Log in with ‘admin’/‘adminpass’ for admin menu or ‘customer1’/‘custpass’ for customer menu.

Troubleshooting Tip: Re-run **init_db** if users are missing.

Key Takeaways

- Secure logins protect the system.
- **bcrypt** ensures safe passwords.
- Test logins with default users.

Chapter 14: Making Your System Fast and Reliable

***Chapter Overview**: This chapter optimizes the system with database indexes and adds **pytest** tests for reliability.*

14.1 Speeding Up Searches

Adding an **index** to the database makes searches faster, like adding a table of contents to a book.

For Beginners: An index is like a shortcut that helps the database find data quickly.

Why This Matters: Faster searches improve user experience.

14.2 Adding an Index

Update **init_db** in **database.py** to add an index on the **make** column.

Error rendering code block: Preformatted.__init__() got an unexpected keyword argument 'maxWidth'

What's Happening?

- **CREATE INDEX**: Speeds up searches by **make**.

14.3 Writing Tests

Create **test_models.py** to test the **Car** and **Customer** classes.

Error rendering code block: Preformatted.__init__() got an unexpected keyword argument 'maxWidth'

What's Happening?

- 1. **pytest.fixture**: Sets up the database.
- 2. Tests saving and loading cars and customers.

14.4 Running Tests

Error rendering code block: Preformatted.__init__() got an unexpected keyword argument 'maxWidth'

What Should Happen: Tests pass, confirming classes work correctly.

Troubleshooting Tip: Install **pytest** (**pip install pytest**) if not found.

14.5 Testing the System

Error rendering code block: Preformatted.__init__() got an unexpected keyword argument 'maxWidth'

What Should Happen: Log in, test menus, searches, and recommendations to ensure everything works smoothly.

Troubleshooting Tip: Check logs in **pdf_generation.log** for errors.

Key Takeaways

- Indexes improve database performance.
- **pytest** ensures reliability.
- Test the entire system.

Chapter 15: Sharing Your Awesome System

***Chapter Overview**: This chapter shows how to share your car sales system with others and explores ideas for future enhancements, celebrating your achievement.*

15.1 Why Share Your Project?

Sharing your system allows others to use or learn from it, like passing on a well-crafted recipe. It's a way to showcase your skills and contribute to the community.

For Beginners: Think of it as giving someone a LEGO kit with clear instructions to build the same model you did.

Why This Matters: Sharing spreads knowledge and opens opportunities for collaboration or feedback.

15.2 Deployment Steps

To share your system, include a **README.md** file with setup instructions. This file guides others to run your project.

Error rendering code block: Preformatted.__init__() got an unexpected keyword argument 'maxWidth'

What's Happening?

- The **README.md** provides step-by-step instructions for setup and usage.
- Includes default login credentials for immediate access.

Troubleshooting Tip: Ensure recipients have Python 3 installed and follow the **README** exactly. Check the virtual environment is activated.

15.3 Ideas for Future Enhancements

Your system is complete, but there's room to grow! Here are some ideas to take it further:

- Build a **web app** using Flask or Django for online access.
- Develop a **mobile app** for customers on the go.
- Integrate advanced **AI** for price predictions or customer preferences.
- Connect to a real car data **API** for live market values.

For Beginners: These ideas are like adding new features to your LEGO model, making it even cooler!

15.4 Congratulations on Your Achievement!

You’ve built a fully functional car sales system from scratch! You’ve mastered databases, classes, APIs, AI, and security. Take a moment to celebrate your hard work!

Why This Matters: These skills are used in real-world programming jobs, and you’ve proven you can do it!

Key Takeaways

- Share your project with clear, concise instructions.
- Consider future enhancements to keep growing.
- Be proud of your programming journey!