Product Title Scraper: User Guide

Version 1.0

Welcome to your new Product Title Scraper! This guide will walk you through everything you need to know, from the initial technical setup to strategic use, all in simple, easy-to-understand language.

Note: This automation is designed to run on your local computer and requires a one-time setup to connect to your Google Account and install the necessary software.

Sheet Link: Page Optimizing Myntra

Table of Contents

- Your Daily Workflow
- How to Monitor Progress
- Understanding the Output

Chapter 4: Strategies for Better Data Collection & Customization

- Strategic Use Cases for Scraped Data
- How to Customize the Scraper
    - Adjusting Scrape Depth and Performance
    - Expanding to Scrape Other Websites (Advanced)

Chapter 5: How the Magic Happens (A Simple Look at the Script)

- What are Python and Selenium?
- The Journey of a Scrape: The Script's Logic
- Important Settings to Know

Appendix: Troubleshooting & Frequently Asked Questions (FAQ)

- Common Problems and Solutions

Chapter 1: Introduction to the Tool

What is this Automation Tool?

Imagine you need to collect the names and brands of all products from 20 different category pages on an e-commerce website. Normally, you would have to visit each page, scroll down, and manually copy and paste every single product title into a spreadsheet. This is incredibly time-consuming, boring, and prone to errors.

This tool solves that problem.

It is a smart script that automatically visits a list of URLs you provide, scrapes all the product titles and brands from each page (even navigating through multiple pages like "Page 1, 2, 3..."), and neatly organizes the data for you in a Google Sheet. It acts as your personal data collection assistant.

Who is this Tool For?

This tool is designed for anyone who needs to gather product listing data from websites without manual effort. You do not need to be a programmer to use it, but you will need to follow the one-time setup instructions carefully.

It's perfect for:

- **SEO Analysts:** To analyze competitor product titles and identify keywords.
- **E-commerce Managers:** To monitor competitor product assortments and brand presence.
- **Market Researchers:** To gather data on product trends and market saturation.
- **Data Analysts:** To collect raw data for reports and business intelligence dashboards.
- **Small Business Owners:** To keep an eye on the market and find product opportunities.

Key Features at a Glance

- **Fully Automated Scraping:** Just provide a list of URLs and run the script. The tool does the rest.
- **Multi-Page Navigation:** Automatically clicks the "Next Page" button to scrape data from all available pages (up to a set limit).
- **Centralized Data Output:** All scraped data is saved directly into a designated Google Sheet, ready for you to analyze.

- **Error Resilient:** If a single URL fails, the script will retry it multiple times before skipping it, ensuring the entire process doesn't stop.
- **Performance Optimized:** Automatically restarts the browser periodically to prevent slowdowns during large scraping jobs.
- **Crash Alerts:** If the script encounters a critical failure, it can automatically send an email notification to you.
- **Complete Logging:** Every action—starting, processing a URL, success, or failure—is recorded in a local log file so you can see exactly what happened.

How It Works: A Simple Overview

The process is straightforward. Think of it as giving a set of instructions to a very fast robot.

1. **You Provide the URLs:** You add a list of website category pages you want to scrape into the 'Header scraper' sheet.
2. **You Configure the Settings:** In a simple configuration file, you tell the script your Google Sheet name and your email for alerts.
3. **The Automation Kicks In:** You run the script from your computer's terminal.
4. **It Opens a Browser:** The script launches a hidden web browser in the background.
5. **It Visits Each URL:** For every URL on your list, the script navigates to the page.
6. **It Scrapes and Navigates:** It reads the product brands and names, then clicks "Next Page" and repeats the process until all pages are done.
7. **It Records the Data:** It sends all the collected data back to the 'Scraped Products' sheet, organizing it with the source URL.
8. **It Logs Its Work:** It keeps a detailed record of its progress in a `product_titles.log` file on your computer.

This cycle repeats for every URL, saving you hours or even days of manual work.

Chapter 2: Getting Started: The Complete Setup Process

This chapter will guide you through the one-time setup of your automation tool. Follow these steps carefully to ensure everything works perfectly. We have provided sample CSV files of the Google Sheets to help you set it up correctly.

Part A: Google Sheet & API Setup

Step 1: Making Your Own Copy of the Google Sheet

First, you will use our master Google Sheet template. You cannot edit this master template directly. You must make your own personal copy.

1. Open this link to the Google Sheet: Product Title Scraper Sheet
2. In the menu, click **File > Make a copy**.
3. Give your copy a new name (e.g., "My Company's Product Scraper") and save it to your own Google Drive.
4. All your work from now on will be done in **your copy** of the sheet.

Step 2: Understanding Your Workspace: The 2 Key Sheets

At the bottom of your Google Sheet, you will see two tabs. Each has a specific purpose.

- **Header scraper:** This is your **input** sheet. You will paste the list of URLs you want the script to scrape into Column A of this sheet.
- **Scraped Products:** This is your **output** sheet. The script will automatically clear this sheet and fill it with the data it collects. You don't need to edit this sheet.

Step 3: Configuring Google Cloud Platform (GCP) for API Access

The script needs permission to talk to your Google Sheet. We do this securely through a Google Cloud "Service Account."

1. **Go to Google Cloud Console:** Open the Google Cloud Console. If you're new, you may need to agree to the terms of service.
2. **Create a New Project:**
    - Click the project dropdown at the top of the page and click **"New Project"**.
    - Give it a name like "Data Scraper Project" and click **"Create"**.

3. **Enable APIs:**
    - Make sure your new project is selected.
    - In the search bar, type **"Google Drive API"** and press Enter. Click on it and then click the **"Enable"** button.
    - Go back to the search bar, type **"Google Sheets API"**, and enable it as well.
4. **Create a Service Account:**
    - In the search bar, type **"Service Accounts"** and go to that page.
    - Click **"+ Create Service Account"**.
    - Give it a name (e.g., "sheets-scraper-bot") and a description. Click **"Create and Continue"**.
    - For the "Role", select **"Basic" > "Editor"**. This gives it permission to edit your sheets. Click **"Continue"**, then **"Done"**.
5. **Generate an API Key (JSON file):**
    - Find the service account you just created in the list. Click the three dots under "Actions" and select **"Manage keys"**.
    - Click **"Add Key" > "Create new key"**.
    - Choose **JSON** as the key type and click **"Create"**.
    - A `.json` file will be downloaded to your computer. **This file is your password! Keep it safe.** Rename it to `gcp_credentials.json`.

Step 4: Sharing Your Sheet with the Service Account

The script now has a key, but it needs access to your specific Google Sheet.

1. Open the `gcp_credentials.json` file you downloaded. Find the line that says `"client_email"`. It will look something like `sheets-scraper-bot@data-scraper-project.iam.gserviceaccount.com`. Copy this entire email address.
2. Go to **your copy** of the Google Sheet.
3. Click the **"Share"** button in the top right corner.
4. Paste the service account's email address into the sharing box, give it **Editor** permissions, and click **"Send"**.

Part B: Local Computer Setup

Step 5: Installing Python

This script is written in Python. If you don't have it, you'll need to install it.

1. Go to the official Python website: python.org/downloads/
2. Download the latest version (e.g., Python 3.11).
3. Run the installer. **Important:** On the first screen of the installer, make sure to check the box that says **"Add Python to PATH"**. This is crucial. Then, click "Install Now".

Step 6: Setting Up the Project Folder & Virtual Environment (venv)

It's best practice to keep project files organized and isolated.

1. **Create a Project Folder:** On your computer, create a new folder where you will store all the project files (e.g., `C:\Users\YourName\Documents\Product_Title_Scraper`).
2. **Copy Project Files:** Place all the files provided with the project (`product_titles.py`, `config.py`, `requirements.txt`) into this new folder.
3. **Open Command Prompt/Terminal:**
    - **Windows:** Open the Start Menu, type `cmd`, and open "Command Prompt".
    - **Mac:** Open "Terminal".
4. **Navigate to Your Folder:** In the terminal, type `cd` followed by the path to your folder.
    - Example: `cd C:\Users\YourName\Documents\Product_Title_Scraper`
5. **Create a Virtual Environment:** This creates an isolated space for this project's libraries. Run this command:
   `python -m venv venv`
6. **Activate the Environment:** You must do this every time you want to run the script.
    - **Windows:** `venv\Scripts\activate`
    - **Mac/Linux:** `source venv/bin/activate`
    - You'll know it's active because you'll see `(venv)` at the beginning of your command prompt line.

Step 7: Installing Required Python Libraries

With the virtual environment active, run the following command to install all the necessary packages automatically:
`pip install -r requirements.txt`

Part C: Final Configuration & Activation

Step 8: Configuring the

This file holds all the key settings for the script.

1. Move the `gcp_credentials.json` file you downloaded in Part A into your main project folder.
2. Open the `config.py` file with a text editor (like Notepad or VS Code).
3. Update the following settings:
    - `PROJECT_ROOT`: Change this to the full path of your project folder. Use the format shown in the file.
    - `SHEET_NAME`: Enter the exact name of **your copy** of the Google Sheet (e.g., "My Company's Product Scraper").
    - **Email Notification Settings (Optional but Recommended):**
        - `ENABLE_EMAIL_NOTIFICATIONS`: Set to `True` if you want crash alerts.
        - `SENDER_EMAIL`: Your Gmail address that will send the alert.
        - `SENDER_PASSWORD`: **Important:** Do not use your regular Gmail password. You need to generate an "App Password". Follow Google's instructions here.
        - `RECIPIENT_EMAIL`: The email address(es) that should receive the alerts.

Step 9: Preparing Your Input URLs

1. Open your Google Sheet.
2. Go to the **'Header scraper'** tab.
3. Paste the list of Myntra (or other supported website) category URLs you want to scrape into Column A, starting from cell A2.

Step 10: Activating the Automation (Running the Script)

You are now ready to run the scraper!

1. Open your Command Prompt or Terminal.
2. Navigate to your project folder (`cd C:\Path\To\Your\Folder`).
3. Activate the virtual environment (`venv\Scripts\activate`).
4. Run the script with this command:
   `python product_titles.py`

The script will now start. You will see log messages appearing in your terminal, showing its progress.

Congratulations! Your Product Title Scraper is now fully set up and running.

Chapter 3: Using the Automation Day-to-Day

Now that the setup is complete, using the tool is simple.

Your Daily Workflow

1. **Add New URLs:** Open your Google Sheet and go to the **'Header scraper'** tab. Clear any old URLs and paste the new list of URLs you want to process.
2. **Run the Script:** Open your terminal, navigate to the project folder, activate the virtual environment, and run `python product_titles.py`.
3. **Wait for Completion:** Let the script run. It will process each URL one by one. This can take a while for a long list of URLs.
4. **Check Your Results:** Once the script finishes, open your Google Sheet and go to the **'Scraped Products'** tab. It will be filled with the newly scraped data.

How to Monitor Progress

There are two ways to see what the script is doing:

1. **The Terminal:** The command prompt window where you ran the script will show live updates, telling you which URL it's currently processing, how many pages it's scraping, and when it successfully writes data to the sheet.
2. **The Log File:** A file named `product_titles.log` is created in your project folder. This is a permanent record of everything the script has done, including any errors it encountered. If something goes wrong, this is the first place to look.

Understanding the Output

The **'Scraped Products'** sheet will contain three columns:

- **Source URL:** The original URL from which the data was scraped. This helps you trace the data back to its origin.
- **Html 1:** The product brand (e.g., "Wonderchef").
- **Html2:** The product name or title (e.g., "Regenta Coffee Maker 800 W").

Chapter 4: Strategies for Better Data Collection & Customization

This tool is powerful, but using it strategically will give you the best results.

Strategic Use Cases for Scraped Data

- **Competitor Analysis:** Scrape all of a competitor's main category pages. You can then analyze which brands they carry most, how they name their products, and identify gaps in their product offerings.
- **SEO Keyword Optimization:** Analyze the most common words and phrases used in product titles for a specific category. This can give you ideas for optimizing your own product listings or ad campaigns.
- **Market Trend Spotting:** Regularly scrape the "New Arrivals" or "Bestsellers" sections of multiple websites to track which products are trending over time.
- **Building a Product Database:** Use the scraper as the first step in building a larger database. The collected data can be enriched with prices, descriptions, and other information later.

How to Customize the Scraper

Adjusting Scrape Depth and Performance

You can fine-tune the script's behavior by editing the `product_titles.py` file.

- `MAX_PAGES_TO_SCRAPE = 120`: This variable at the top of the file controls the maximum number of pages the script will scrape for a single URL. Lower this number if you only need a sample, or increase it for very large categories.
- `RESTART_DRIVER_AFTER_N_URLS = 25`: This setting tells the script to close and reopen the hidden browser after processing 25 URLs. This helps keep the process fast and stable. You can adjust this number if you wish.
- `MAX_RETRIES_PER_URL = 3`: If a URL fails (e.g., due to a network error), the script will try again 3 times. You can increase this for less reliable connections.

Expanding to Scrape Other Websites (Advanced)

The script is pre-configured for Myntra, but it can be adapted for other websites. This requires identifying the correct CSS selectors for the new site.

1. Open `product_titles.py` and find the `WEBSITE_CONFIGS` dictionary.

2. You would need to add a new entry (e.g., for "Amazon").
3. You would then need to find the correct CSS selectors for:
    - `data_selectors`: The selectors for the brand and product title elements.
    - `product_container`: The selector for the main box that contains a single product's information.
    - `total_pages_info`: The selector for the text that says "Page 1 of X".
    - `next_page_button`: The selector for the "Next" button.
4.

This is an advanced modification and requires some knowledge of web development tools (like Chrome's "Inspect" feature).

Chapter 5: How the Magic Happens (A Simple Look at the Script)

You don't need to understand code to use this tool, but it can be helpful to know what's happening behind the scenes.

What are Python and Selenium?

- **Python:** A popular programming language used for everything from web development to data science. Our script is written in Python.
- **Selenium:** A powerful tool that allows code to control a web browser. When our script needs to visit a URL, scroll, or click a button, it uses Selenium to do so.

The Journey of a Scrape: The Script's Logic

When you run the script, it follows a very logical, step-by-step process:

1. **Wake Up & Connect:** The script starts and uses your `gcp_credentials.json` file to connect to the Google Sheets API.
2. **Read the To-Do List:** It reads all the URLs from your 'Header scraper' sheet.
3. **Prepare the Workspace:** It completely clears the 'Scraped Products' sheet to make way for new data.
4. **Launch the Browser:** It opens a "headless" (invisible) Chrome browser instance.
5. **Pick a URL:** It takes the first URL from its to-do list.
6. **Scrape the Page:** It visits the URL, waits for the products to load, and scrapes all the brand and title information.
7. **Look for More Pages:** It checks if there is a "Next" button. If so, it clicks it and repeats Step 6 until it has scraped all pages (or reached the `MAX_PAGES_TO_SCRAPE` limit).
8. **Record the Results:** Once finished with a URL, it sends all the data it found to the 'Scraped Products' sheet.
9. **Check for Maintenance:** It checks if it's time to restart the browser (based on `RESTART_DRIVER_AFTER_N_URLS`).
10. **Handle Errors:** If any step for a URL fails, it logs the error, restarts the browser, and retries the *same URL* again. After `MAX_RETRIES_PER_URL` failures, it gives up on that URL and moves to the next.
11. **Repeat:** The script moves to the next URL in the list and starts the process all over again.
12. **Send Crash Alert:** If the entire script crashes due to a critical error (like losing internet connection or a bug), it will trigger the email notification before shutting down.

Important Settings to Know

These key settings, located in `config.py` and `product_titles.py`, control the entire operation:

- **(config.py):** Tells the script which Google Sheet to work with.
- **(product_titles.py):** Prevents infinite loops and limits data collection.
- **(product_titles.py):** The "map" that tells the script where to find data on a specific website.

Appendix: Troubleshooting & Frequently Asked Questions (FAQ)

Q: I ran the script, but nothing happened in my Google Sheet.
A: The first place to look is the `product_titles.log` file in your project folder. It will tell you why. Common reasons are:

- **Authorization Error:** The most common issue. Double-check that you shared your Google Sheet with the correct `client_email` from the `.json` file and gave it "Editor" permissions.
- **Error:** The `SHEET_NAME` in your `config.py` file does not exactly match the name of your Google Sheet.
- **Website Structure Changed:** Websites change their code all the time. If Myntra updates its website, the CSS selectors in `WEBSITE_CONFIGS` might be outdated and need to be updated. The log file will likely show "Timeout" errors.

Q: The script crashes immediately with a
A: This means the script cannot find your API key file. Make sure:

1. You have renamed the downloaded file to exactly `gcp_credentials.json`.
2. The file is located in the same folder as the `product_titles.py` script.

Q: I'm not receiving error emails.
A: This is almost always an issue with the sender's email credentials in `config.py`.

- Make sure `ENABLE_EMAIL_NOTIFICATIONS` is set to `True`.
- For `SENDER_PASSWORD`, you **must** use a Google "App Password," not your regular account password. Follow the official guide to create one.
- Check for typos in `SENDER_EMAIL` and `RECIPIENT_EMAIL`.

Q: How do I stop the script while it's running?
A: Go to the terminal window where the script is running and press **Ctrl + C** on your keyboard.

Q: Can I scrape a website other than Myntra?
A: Yes, but it requires an advanced change. You need to add a new configuration to the `WEBSITE_CONFIGS` dictionary in `product_titles.py` with the correct CSS selectors for the new site. See Chapter 4 for more details.