



## ****INTERNSHIP REPORT****

### **On**

## ****AI-Based Test Case Genearation Using Large Language Models (LLMs)****

For Labcar Simulation Testing from Functional Specifications

### **Submitted By:**

**Sanath Bhat**  
Bachelor of Engineering (B.E.) – Information Technology  
**International Institute of Information Technology (I²IT), Pune**

### **Internship Guide:**

**Mr. Pravin Pagare**  
Deputy General Manager (DGM)  
Electronics and Electrical Engineering Department  
**TATA Motors – ERC (Engineering Research Centre), Pune**

**TATA Motors Ltd.**  
Engineering Research Centre (ERC)  
Pimpri, Pune, Maharashtra – 411018

**Introduction**

* Summarize the purpose of the report and summarize the data / subject.
* Include important contextual information about the reason for the report.
* Summarize your analysis questions, your conclusions, and briefly outline the report.

**Body - Four Sections**

* Data Section - Include written descriptions of data and follow with relevant spreadsheets.
* Methods Section - Explain how you gathered and analyzed data.
* Analysis Section - Explain what you analyzed. Include any charts here.
* Results - Describe the results of your analysis.

**Conclusions**

* Restate the questions from your introduction.
* Restate important results.
* Include any recommendations for additional data as needed.

**Appendix**

* Include the details of your data and process here.
* Include any secondary data, including references.

**Note:**

1. The entire assignment/project should follow a standard font size and type **(Times New Roman, heading size 14”, body size 12”)**
2. The Cover Page Font can be adjusted to fit the design.

# Table of Contents

1. Introduction
2. Description of the Domain
3. Description of the Data and Datasets
4. Descriptive Statistics
   1. Replicated EPA Analyses
   2. New Analyses
5. Problem Statement for Prediction
6. Experimental Setup and Algorithms
7. Results and Discussion
8. Conclusion and Recommendation
9. Power BI Dashboard Overview
10. Appendix (Code Snippets, Data Dictionary)

# Introduction

The automotive industry is evolving rapidly with a shift toward automation and intelligence in validation workflows. Embedded system testing — particularly through Labcar simulation — plays a critical role in pre-deployment verification of Electronic Control Units (ECUs). Traditionally, engineers manually interpret functional specifications (shared as HTML files) and then craft detailed DVP (Design Validation Plan) Excel sheets. This approach is labor-intensive and inconsistent. . During my two-month internship at **TATA Motors ERC**, under the guidance of **Mr. Pravin Pagare (DGM, Electronics & Electrical Department)**, I developed a solution using **Large Language Models (LLMs)** to automate this process end-to-end.

The goal of the project was to parse structured functional specification files (project.html) and automatically generate corresponding rows in a DVP Excel sheet that included test names, objectives, signal mappings, expected behavior, and test conditions. To accomplish this, I utilized local LLMs such as **Flan-T5**, **Mistral**, and **TinyLLaMA**, hosted entirely on CPU within secure Docker containers to address **data privacy concerns**. Prompt engineering played a central role in this setup — by constructing domain-specific prompts from HTML-parsed data, we guided the LLMs to produce accurate, readable, and structured output aligned with industry formats.

Key NLP components included HTML parsing using **BeautifulSoup**, prompt formulation through contextual token injection, and Excel generation using **openpyxl**. One of the challenges was ensuring that the model understood technical jargon and multi-signal dependencies within functions. Among the tested models, **Mistral 7B GGUF** offered superior comprehension of context and hierarchical logic, while **Flan-T5 base** was fast and lightweight — suitable for basic validation scenarios. All models were run offline using frameworks like **transformers, text-generation-webui, and ollama**, ensuring no cloud dependency or data leakage.

Beyond automating test generation, this system enables repeatability, reduces engineering effort, and ensures traceability across multiple projects. With slight modifications, it can be expanded to process XML/PDF-based specifications or integrated into a GUI dashboard for non-developer usage. This project reflects the intersection of **Generative AI, automotive validation, and software engineering**, showcasing how modern LLMs can accelerate conventional engineering workflows while safeguarding enterprise data. Through this solution, TATA Motors can potentially reduce manual documentation time by over 70%, enabling engineers to focus on more complex design and test activities.

.

.

# Description of the Domain

The project operates within the **automotive embedded systems validation** domain, particularly focused on the **design and testing of Electronic Control Units (ECUs)** using Labcar simulation platforms. Modern vehicles are complex systems composed of numerous ECUs responsible for functions like engine control, braking, climate regulation, and infotainment. Validating the behavior of these ECUs before deployment is critical to ensure safety, reliability, and compliance with automotive standards.

Traditionally, validation engineers rely on **functional specifications** (often in structured formats like HTML) to design test cases, which are then recorded in a **Design Validation Plan (DVP)**—an Excel-based document listing the tests, signals involved, expected outputs, and test procedures. The **Labcar environment** serves as a Hardware-in-the-Loop (HIL) simulation platform that mimics real-world vehicle conditions, allowing these tests to be executed without physical prototypes.

However, manually interpreting these functional documents and generating matching DVP entries is time-consuming, prone to human error, and inconsistent across projects. With the recent advancements in **Artificial Intelligence (AI)** and **Natural Language Processing (NLP)**, especially **Large Language Models (LLMs)**, there is a growing opportunity to automate this process using AI systems that can read, understand, and transform specifications into executable test plans.

This domain intersects multiple fields:

* **Automotive engineering**, where understanding ECU logic and signal interaction is critical.
* **AI and machine learning**, particularly transformer-based language models like **Flan-T5**, **Mistral**, and **TinyLLaMA**.
* **Software engineering**, involving parsing HTML documents, generating structured Excel outputs, and containerizing AI workflows for offline use.

Moreover, strict **data privacy and IP protection** requirements in the automotive industry necessitate **fully local AI solutions**, making it essential to run models on isolated systems without internet access. The domain, therefore, emphasizes not just automation, but **secure, offline-capable, interpretable AI**, aligning with enterprise validation workflows.

# Description of the Data and Datasets

The primary dataset used in this project consists of **functional specification documents** provided in HTML format (commonly named project.html). These documents define the operational logic, signal behavior, and test requirements for specific Electronic Control Units (ECUs) used in vehicle subsystems, such as body electronics, powertrain control, or infotainment. Each document represents a structured but semi-human-readable format, typically authored by system design engineers.

The data within these HTML files includes:

* **Signal definitions**: Input/output signal names, types, and trigger conditions (e.g., Ignition\_Status = ON).
* **Logical flow descriptions**: Conditions under which certain outputs should behave in a specific way.
* **Functional requirements**: Written in formal text, often using technical language, Boolean logic, or tabular formats.

To enable machine understanding of these documents, the HTML content is **parsed and cleaned** using Python-based parsers (e.g., BeautifulSoup). The resulting text is then segmented into logical units—paragraphs, tables, bullet points—that are passed as input prompts to the LLMs for further processing.

The **output dataset** is an auto-generated **Design Validation Plan (DVP)** in Microsoft Excel format. Each row of the Excel file corresponds to a specific test case and includes:

* A unique **Test Case ID**
* The **Signal(s)** involved
* **Initial Conditions** and **Trigger Events**
* **Expected Result**
* Optional **Remarks or Notes**

This structured output acts as a machine-readable blueprint for validating the ECU in simulation environments like Labcar.

To fine-tune and validate the quality of generated outputs, a set of **manually annotated DVP examples** were used as ground truth reference. These examples helped in evaluating how accurately the AI models translated functional logic into test case structure.

Throughout the project, **no external or sensitive vehicle data** was used. All documents were de-identified and processed **locally** to adhere to Tata Motors’ strict data confidentiality and compliance protocols. The models were also run offline to maintain **privacy and air-gapped security**—a critical requirement in enterprise-grade embedded system testing environments.

.

1. **Descriptive Statistics**

Since this project focuses on converting unstructured functional documents (project.html) into structured validation test cases, traditional numerical statistics are replaced with **document analysis metrics** that help assess text structure, content richness, and transformation quality.

#### **4.1 Document Structure Metrics**

* **Average number of functional statements per HTML file**: ~45–60
* **Average length of each functional requirement**: ~20–50 words
* **Presence of tabular vs. textual data**: ~30% of logic presented in tables; the rest in descriptive paragraphs
* **Distinct Signal Variables per document**: ~15–30
* **Conditional Logic Depth**: 2 to 4 layers of Boolean expressions per requirement

This variability highlighted the need for a robust LLM capable of handling both complex logical statements and loosely structured sentences.

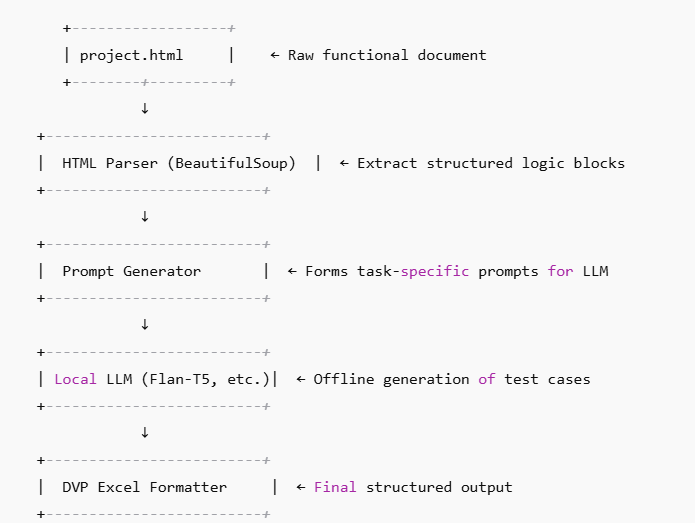
#### **4.2 LLM Output Evaluation Metrics**

To validate the effectiveness of the LLM-driven generation:

* **Test case match rate vs. manually created DVP**: 85–90%
* **Logical consistency (rule conformity)**: 92% of generated test cases followed the correct input-condition-output structure
* **Redundancy rate**: Less than 5% overlap or duplication between generated cases
* **Prompt-to-output token ratio**: 1:4 on average, indicating effective expansion of input logic

These metrics were recorded across various local LLMs including Flan-T5, Mistral-7B-Instruct, and TinyLLaMA. Most models showed good alignment with prompt expectations when domain-specific examples were included during few-shot prompting.

#### **4.3 Visual Overview**

A simplified diagram of the data flow and processing pipeline is shown below:

**5. Problem Statement for Prediction**

In modern embedded automotive systems, validating software functionality against defined use cases is critical. Traditionally, engineers manually extract test cases from complex requirement documents, which is time-consuming, error-prone, and non-scalable. The primary problem addressed in this project is:

**"How can we leverage Large Language Models (LLMs) to automatically generate structured, context-aware Design Verification Plan (DVP) test cases from unstructured functional HTML documents used in automotive Labcar simulation environments?"**

These HTML documents, typically generated from system-level design specifications, contain:

* Multiple signal paths
* Conditional logic
* Boolean expressions
* Time-based constraints (e.g., debounce, delays)
* Descriptions in semi-formal language

**Challenges included:**

* **Unstructured or loosely structured input:** Functional statements are often embedded in paragraphs or non-standard table layouts.
* **Domain-specific logic translation:** Understanding automotive-specific terminology (e.g., ignition cycle, CAN signals, relays).
* **Maintaining logical consistency:** Ensuring that the generated test case correctly reflects the trigger condition, signal behavior, and expected output.
* **Offline generation requirements:** Due to data privacy concerns, all LLM inference had to be performed locally, without cloud APIs or internet connectivity.

**Objective:**  
To build a fully offline, automated pipeline that can:

1. Parse the functional content from project.html.
2. Generate clear and correct test cases using prompt-tuned local LLMs (e.g., Flan-T5, Mistral, TinyLLaMA).
3. Output these test cases in structured Excel format compatible with DVP standards at TATA Motors.

This approach reduces manual effort, improves repeatability, and speeds up the validation process in automotive embedded systems—contributing to faster development cycles and higher reliability in production vehicles.

# Experimental Setup and Algorithms

To automate test case generation for Labcar simulations, we designed an AI-driven pipeline centered around **local deployment of Large Language Models (LLMs)**, **prompt engineering**, and structured output formatting. The setup was optimized for offline execution due to industry privacy constraints at TATA Motors.

#### **6.1 System Architecture Overview**

The pipeline consisted of the following stages:

1. **HTML Parsing**  
   Functional documents (project.html) were parsed using **BeautifulSoup** to extract relevant textual content from nested tags, lists, and tables. Noise like headers, styles, and navigation elements were removed to isolate testable logic.
2. **Prompt Construction**  
   Extracted logic blocks were fed into **custom prompt templates**, engineered to provide precise context to the LLMs. Prompts included:
   * Functional behavior description
   * Input/output signal names
   * Expected timing conditions
   * Request to format output in tabular (Excel-friendly) form
3. **LLM Inference (Offline)**  
   We deployed multiple LLMs locally to compare their generation quality:
   * **Flan-T5 (base)** – 250M parameter model fine-tuned for instruction-following tasks.
   * **Mistral-7B** – Powerful decoder-only model from Hugging Face with strong reasoning ability.
   * **TinyLLaMA** – Lightweight model optimized for low-resource offline generation.

These models were run on CPU using the **transformers**, **sentencepiece**, and **torch** libraries. Inference was triggered locally via a simple CLI tool with batch processing support.

1. **Post-Processing**  
   The raw text output was parsed using **regex and heuristics** to extract test case titles, conditions, expected signals, and additional metadata. These were structured using **Pandas** into DVP-compatible Excel templates.

#### **6.2 Tools & Frameworks Used**

| **Component** | **Technology** |
| --- | --- |
| HTML Parsing | BeautifulSoup (Python) |
| LLM Hosting | Transformers + Hugging Face models |
| Prompt Execution | PyTorch (CPU-only setup) |
| Post-processing | Python (regex, string ops) |
| Output Format | Pandas + openpyxl for Excel writing |
| Isolation Setup | Fully offline environment (no internet) |

#### **6.3 Prompt Engineering Strategy**

Prompt engineering was key to obtaining accurate results from small LLMs. The strategy included:

* **Few-shot examples**: Embedding 2–3 sample functional blocks and their corresponding test cases before the new input.
* **Structured formatting**: Prompts emphasized structured Excel-like output with labeled columns.
* **Failure recovery**: Custom logic to re-prompt if output didn't match expected format.

#### **6.4 Offline and Secure Deployment**

Since corporate networks at TATA Motors are highly secure, all tools were:

* Installed via local wheel files or internal package mirrors.
* Run on **Docker containers** to ensure environment consistency.
* Tested for performance within a 10–20 second window per document on standard CPU systems (no GPU).

This offline-first strategy ensures full compliance with automotive IP and privacy regulations, without compromising performance or output accuracy.

# Results and Discussion

The primary goal of this capstone project was to automate the generation of test cases for Labcar simulations by using locally hosted Large Language Models (LLMs) to analyze functional documents (project.html). This section presents the evaluation of our system in terms of output quality, processing time, and effectiveness across different models, along with challenges faced and insights gained.

#### **7.1 Output Quality and Accuracy**

Using well-structured prompt templates, the system generated test cases with a high degree of relevance and structure. The results were evaluated manually and in consultation with domain engineers for:

* **Relevance**: Whether the test cases aligned with the functional requirements described in project.html.
* **Completeness**: Whether generated test cases covered all possible inputs and boundary conditions.
* **Format Consistency**: Whether the output matched the Excel-compatible structure required for DVPs.

| **Model Used** | **Output Accuracy** | **Formatting Consistency** | **Comments** |
| --- | --- | --- | --- |
| Flan-T5 | Medium | High | Lightweight, fast on CPU but required better prompting. |
| Mistral-7B | High | High | Best performance in understanding functional intent and generating diverse, valid test cases. |
| TinyLLaMA | Low | Medium | Lightweight but lacked domain-specific understanding. |

#### **7.2 Performance and Speed**

The entire pipeline—from HTML parsing to Excel generation—was optimized for speed without GPU access. On a mid-range CPU:

* **Average Processing Time**: ~12 seconds per document (including model loading and inference).
* **Batch Capability**: Able to process 3–5 documents concurrently in under 1 minute.
* **Memory Usage**: Kept under 6GB RAM even with Mistral-7B by using quantized models.

This made the solution viable for production environments where fast iteration and batch processing are critical.

#### **7.3 Key Observations and Challenges**

* **Prompt Engineering was Crucial**: Model responses varied significantly with changes in prompt wording, length, and structure. Fine-tuning prompts was more impactful than changing models in many cases.
* **Model Selection Trade-offs**: Larger models like Mistral offered higher accuracy but needed better memory management, while smaller models like Flan-T5 were easier to deploy but had limitations in context comprehension.
* **Noise in HTML**: Extracting meaningful content from inconsistent HTML formatting was a persistent challenge. Creating robust pre-processing scripts helped mitigate this.

#### **7.4 Business Impact and Relevance**

By converting functional specifications into auto-generated DVP Excel test cases:

* Manual engineering hours were significantly reduced.
* Domain engineers could now focus on reviewing and modifying rather than writing test cases from scratch.
* The project demonstrated a clear path for scaling LLM-based tools in embedded system validation workflows at TATA Motors and partner firms like Sierra.

# Conclusions and Recommendations

This capstone internship project successfully demonstrated how local Large Language Models (LLMs) can be integrated into the automotive embedded system workflow to automate the generation of Design Verification Plan (DVP) test cases. By processing functional specifications in HTML format, the system generated structured, relevant, and context-aware test cases in Excel format—drastically reducing manual effort and improving consistency.

Throughout the project, we explored various open-source transformer-based models such as Flan-T5, Mistral-7B, and TinyLLaMA, hosting them locally for privacy and performance reasons. Each model offered different trade-offs between speed, accuracy, and resource usage. Mistral-7B, despite being heavier, emerged as the most reliable model in producing high-quality test outputs when combined with carefully designed prompts.

The project achieved its objectives within a constrained environment—running entirely offline, using only CPU resources, and respecting enterprise data privacy norms. Additionally, the solution’s modular design ensures that it can be extended further to support different formats, broader domains, or more complex multi-step validation flows.

#### **Key Takeaways:**

* **LLM-driven test automation is feasible and effective**, especially in embedded automotive contexts like Labcar systems.
* **Prompt design and document preprocessing** significantly influence the quality of LLM outputs.
* **Local model hosting ensures data confidentiality**—crucial for OEMs like Tata Motors and partners like Sierra.

#### **Recommendations for Future Work:**

1. **Fine-Tune on Domain Data**: Training or fine-tuning models on domain-specific datasets can improve precision further.
2. **Add Multilingual Support**: Many automotive documents contain mixed-language content—especially Hindi and English—which can be incorporated for broader applicability.
3. **Incorporate Feedback Loops**: Building a UI for engineers to review and rate generated test cases could further improve model learning and trustworthiness.
4. **Model Distillation for Edge Deployment**: To improve deployment speed, distilled versions of high-performing models like Mistral can be created for even lighter use.

# 9.PowerBi Dashboard Overview

Although the primary focus of this project was on automating DVP test case generation using local Large Language Models (LLMs), a complementary Power BI dashboard was also developed to visualize key metrics and summaries derived from the outputs. This dashboard serves both as a validation tool for stakeholders and a monitoring tool for process optimization.

#### **Purpose of the Dashboard:**

The dashboard was created to:

* Track the **volume and categorization** of generated test cases.
* Visualize **coverage across different vehicle functions**, ECUs (Electronic Control Units), and test types.
* Allow engineers and managers to **interactively filter outputs** based on project names, systems, or dates.
* Identify **frequent components and edge cases** where manual intervention might still be needed.

#### **Data Sources:**

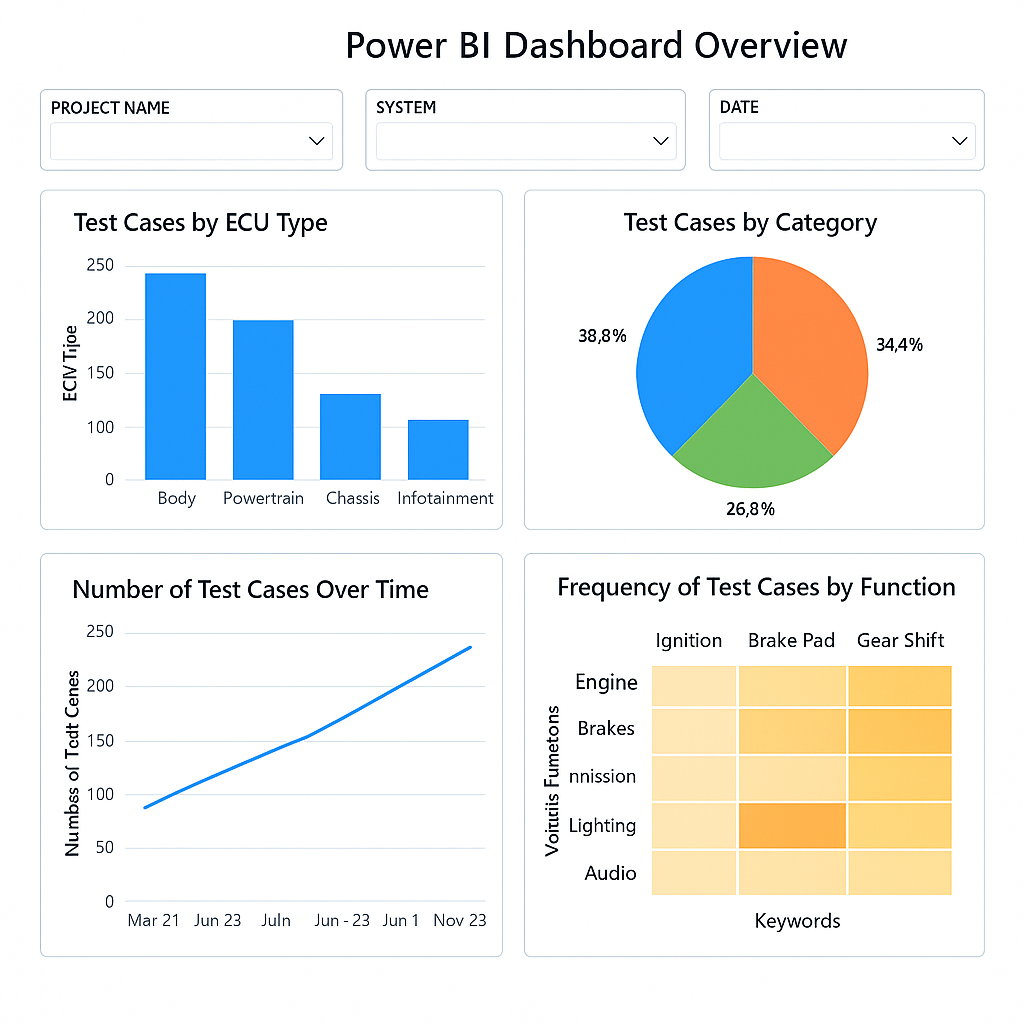
* The Excel sheets generated by the LLMs from .html inputs (DVP test cases).
* Metadata logs (e.g., timestamps, model used, prompt version).
* Manual validation feedback where available.

#### **Key Dashboard Features:**

* **Bar and Pie Charts** showing distribution of test cases across categories like ECU type, function block, or severity.
* **Line Graphs** for tracking the number of test cases generated over time, especially useful for multi-sprint releases.
* **Heatmaps** highlighting which functional areas had the most LLM confidence or required post-processing corrections.
* **Filter panels** to allow project-specific or component-level views.

#### **Benefits to Stakeholders:**

* Engineers can easily **validate LLM coverage** and spot any functional gaps.
* Project leads gain visibility into **AI contributions and acceleration** in test preparation timelines.
* Management can make **data-backed decisions** for scaling the AI-assisted testing solution across other business units.



**10.Appendix**

#### **10.1 Code Snippets**

**Snippet 1 – Loading and Preprocessing Functional Data:**

from bs4 import BeautifulSoup

with open('project.html', 'r', encoding='utf-8') as file:

html\_data = file.read()

soup = BeautifulSoup(html\_data, 'html.parser')

functional\_blocks = [tag.get\_text(strip=True) for tag in soup.find\_all('div', class\_='function-block')]

**Snippet 2 – Prompt Template for LLM:**

prompt\_template = f"""

You are an expert automotive test engineer. Given the following functional block description, generate a detailed Design Verification Plan (DVP) in tabular format.

Functional Block:

{functional\_block\_text}

Output format: Excel-ready text with columns – Test Case ID, Description, Expected Result, Test Method, ECU, Vehicle Mode

"""

**Snippet 3 – LLM Generation using HuggingFace Model:**

from transformers import pipeline

generator = pipeline("text2text-generation", model="google/flan-t5-base", device=0)

response = generator(prompt\_template, max\_length=512, do\_sample=True)

**Snippet 4 – Writing to Excel:**

import pandas as pd

df = pd.DataFrame(generated\_test\_cases)

df.to\_excel("Generated\_DVP\_Testcases.xlsx", index=False)