Complete Project Explained: Hotel Booking Dataset

The main goal of this document is to explain how my entire Hotel Booking dataset project functions. I completed two distinct but complementary projects using the same dataset (see the 'Projects Completed' section).

The 'Workflow – Simulation to Production Activities' section describes how these two projects work together and how a tech consulting company could offer its B2B services to clients.

Finally, the 'Automations & Technical Professionalism' section provides a deeper look into the technical aspects of the project. It explains how I reduced operational tasks, improved execution speed, and minimized errors. Specifically, the project was designed so that even users with no prior experience in automation or coding can easily generate updated KPI dashboards by only knowing the basics of a dashboard platform like Looker Studio.

Projects Completed:

Project No1:

Leveraging Machine Learning for Cancellation Prediction: Report to Stakeholders

Project No2:

Static but Fully Updated KPIs Dashboard in Looker Studio

Workflow – Simulation to Production Activities

1) I download the dataset from www.kaggle.com/hotel-booking. (raw data)
2) I load the raw data into a Jupyter notebook, reading it into a DataFrame using the pandas library. I do this in order to later upload it to my local PostgreSQL database (DB), simulating

my company's data storage protocol, which is described in the assumptions section of ***Doc_KPIs_Looker.pdf***. For this, I assume that my company has API key access to the real-time updated dataset, which fetches the updated data to its local PostgreSQL DB.

➔ By doing this, I simulate a situation where the data are updated and stored in my company's local PostgreSQL DB, where I have permission to extract them. At this stage, the data is still raw.

3) I extract the raw but updated data from the company's PostgreSQL DB to my local machine.

➔ This is where Project No1 begins.

4) I run ***preprocessing.py*** to transform the extracted data into a cleaned format.

5) I load the cleaned data into my company's PostgreSQL DB so that stakeholders can access it. The cleaned datasets are ***logreg_rf_data*** and ***dashboard_data***.

6) I extract the cleaned ***logreg_rf_data*** from my company's PostgreSQL DB to use it for Machine Learning (ML) purposes. I apply logistic regression for the baseline model and random forest for the advanced ML model. This is for exploratory ML purposes to identify the most important features for booking cancellations, but it is not a full MLOps setup.

7) The exploratory ML analysis from the previous step, combined with domain research, provides valuable insights about the most important features. I then prepare a report for the stakeholders to explain them the potential causes of booking cancellations.

➔ The company charges for the report above. This is where Project No1 is completed.

8) I extract the cleaned ***dashboard_data*** from my company's PostgreSQL DB.

➔ This is where Project No2 starts.

9) I run ***dashboard_dataframe.py*** to generate the KPI DataFrames (***hotel_market_segments.csv*** and ***hotel_kpis.csv***) based on the insights from the previous exploratory ML analysis, along with domain knowledge. I store these two DataFrames locally on my machine, as they only contain static pre-calculated fields that will be used in dashboards. There is no need to upload them to the company's PostgreSQL DB.

10) I upload the ***hotel_market_segments.csv*** and ***hotel_kpis.csv*** datasets to Looker Studio (LS), using the first for vertical bar charts and the second for creating scorecards. At this

point, the LS KPI dashboard structure is fully prepared. I share the KPI dashboard with stakeholders and provide updates at agreed intervals (e.g., monthly), so stakeholders can monitor how their optimization strategies evolve over time.

➔ The company charges for each dashboard shared, typically on a monthly basis. This is where Project No2 is completed.

Automations & Technical Professionalism

Automate Processes by Running only one File

The data pipeline works properly by running only one file, ***run_all.py***. When running this file, the following steps are executed:

1) The raw data are extracted from the company's PostgreSQL DB.
2) The ***preprocessing.py*** script is executed, generating the cleaned datasets ***logreg_rf_data*** and ***dashboard_data***.
3) These cleaned datasets are loaded back into the company's PostgreSQL DB.
4) The ***dashboard_dataframe.py*** script is executed, extracting the ***dashboard_data*** from the company's PostgreSQL DB and generating two tabular KPI datasets: ***hotel_market_segments.csv*** and ***hotel_kpis.csv***.

Automate the One-File Running

Additionally, the run_all.py file is set to run automatically every time my working PC boots, before I arrive at the office.

This is done by creating a shortcut on desktop and typing in the location field: "path to the project's environment" "path to the file directory". (This is the file that must run every time the PC boots). Between the two paths there is a space. After that, I moved the shortcut inside the Startup folder in Windows.

Final step: I set the PC to boot automatically on working days at a specific hour, usually 20–30 minutes before the work starting time, through BIOS settings.

In this way, a decent level of automation can be achieved. Even if it is not the most professional solution, it can totally be used for automating the workflow I described above.

All in all, with the steps described above, a fresh set of cleaned and preprocessed datasets is ready every morning without any need of human actions and without wasting time or creating errors during the working hours.

Monitor run_all.py Running

To ensure all processes are executed successfully, a log file named ***run_all.txt*** accompanies the ***run_all.py*** script. This log file is generated to track the execution of the entire workflow and helps monitor the success of each stage. It stores timestamps marking the successful execution of key steps, such as:

- Preprocessing (***preprocessed.py***) - Ensures that the raw data is cleaned and transformed as expected.
- Dashboard Data Preparation (***dashboard_dataframe.py***) - Verifies that the data is correctly processed into the final KPI datasets.

The log file provides a simple and efficient way to monitor the execution of ***run_all.py*** and serves as a record of the entire pipeline's success, enabling easy identification of any issues should they arise. By reviewing run_all.txt, I can confirm that each script has been run without errors and that the entire data pipeline has completed as expected.

No Calculations in Looker Studio

Since my case study is about a static but fully updated KPI dashboard, the calculated fields needed in LS can be prepared outside of it using Python. In this way, the dashboard update times are minimized because LS only needs to reconnect to the updated data sources after each new data upload. Also, the dashboard structure stays stable thanks to proper data source management.

All in all, even if the process is not 100% automated, the only human action needed after the first dashboard creation is to upload the fresh data into LS and quickly manage the reconnections,

making sure that the data refreshes correctly. After this short task, the static but fully updated KPI dashboard is ready to be shared with stakeholders.

Finally, the person who works with LS does not need to know low-level dashboard coding. They only need to follow simple, repeated steps in LS data management.

Instructions for data sources management in LS: Resource→ Manage added data sources→ Choose the dataset of interest→ Edit→ Edit connection→ Choose the same dataset that was chosen before→ Delete upload→ Add files→ Upload the fresh dataset→ Reconnect→ Repeat the process for the remaining datasets.

Database Credentials Security with .env File

To manage sensitive information securely, I use an .env file to store database credentials such as the username, password and other configuration settings. The .env file is excluded from version control (listed in .gitignore) to prevent accidental exposure of sensitive data. The Python package python-dotenv is used to load these variables at runtime in PyCharm, allowing the scripts to access them without displaying their literal values.

Future Suggestions (Full Professional Project)

To further enhance the automation and professionalism of the project, a next step could be to fully automate the data upload and connection process in Looker Studio using APIs or other automation tools to avoid any manual work. For the ML part, the exploratory analysis can be improved into a more production-ready version by applying some basic MLOps practices. Since performance tracking is already handled through my own custom function, the only thing left here is to automate the model retraining. As a first step, it would probably be a good idea to include the training in a file similar to run_all.py, so that both Project No1 and Project No2 can be completed by just running one file. Later, replacing the current PC-level automations with more professional solutions would be a strong upgrade. Also, adding alert systems like email notifications when a process fails could help improve the overall reliability and make the whole solution closer to real-world company standards.