Machine Learning Operations Project:

Hotel Churn Prediction

Group K

José Ramirez Fernandes - 20220641

Pedro Ferreira - 20220589

Ricardo Montenegro Dona - 20221359

Adriana Monteiro - 20220604

Quintino Fernandes - 20220634

Dataset Selection

For the purposes of this project, and taking into consideration the various tools we were required to utilize, we concluded that constructing a predictive supervised learning solution would be preferable.

After considering various possible topics, we decided to use a hotel reservation dataset, which we had access to due to a separate course in our Master's, to develop a predictive model which predicts customer churn, meaning reservation cancellations. This is a very common issue in the real world, as hotels struggle to balance between the costs of client compensation for overbooking and the costs of lost demand. This type of solution can help hotels such as the one from which the dataset originated not only optimize their operations and inventory management but also target customers which are likely to churn and attempt to retain them.

This dataset is composed of around 79000 records of hotel bookings, of which around 42% correspond to cancellations. It also comprises a total of 31 features, mixed between numerical and categorical. Due to this high number of features, naming and describing each and every one of them here would be impractical, but this description can be found in our initial data exploration notebook.

The primary business goal of this project is to arrive at a solution that, once implemented in a real-world scenario, would have the ability to help the hotel drastically reduce their cancellation ratio by properly managing churning clients. Furthermore, by aiding the hotel in the management of its logistics and operations, the solution should result in a substantial cost reduction and subsequent profit increase.

In terms of machine learning goals, our objective is to create a solution that, with the necessary adjustments, could be put into production and be scalable and sustainable. By implementing procedures such as unit tests and data drift tests and working within the Kedro framework, we want the quality of our solution and its performance to be maintainable over time and easily adjustable whenever necessary. Finally, in terms of performance, the F1 Score will be our primary technical success criteria, and our goal is to achieve a score of at least 0.8.

Project Planning

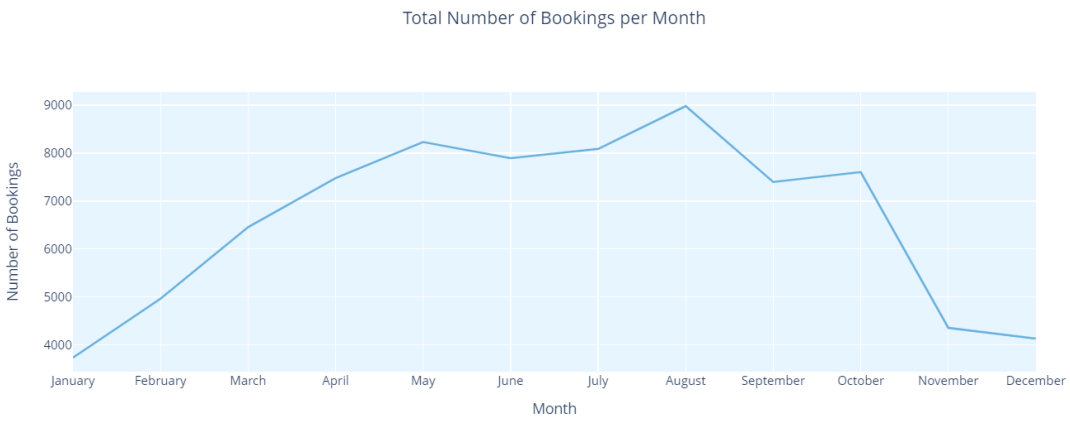
In order to address the challenges presented to us, we decided to adopt Agile methodology practices into our project development. Our workload was divided into multiple smaller tasks which constituted our sprints. The total time allocated to this project was 14 days, and a total of 7 sprints were laid out. The following table constitutes a brief overview of all these sprints and the tasks associated to each one of them:

Sprints	Tasks	Duration
Sprint 1	Data Exploration	2 Days
Sprint 2	Creation of the Data Preprocessing Pipeline	3 Days
Sprint 3	Creation of the Feature Selection Pipeline	2 Days
Sprint 4	Creation of the Hyperparameter Search Pipeline	2 Days
Sprint 5	Creation of the Data Split, Model Train and Model Predict Pipelines	3 Days
Sprint 6	Creation of the Data Unit Tests Pipeline	1 Day
Sprint 7	Creation of the Data Drift Pipeline	1 Day

Sprint 1 had a 3-day duration, and it corresponded to the data exploration phase. The dataset was explored in order to find not only the data's patterns and characteristics, but also its inconsistencies and most relevant issues. Following that, the 5-day-long **Sprint 2** was started in order to tackle one of our most time-consuming tasks – data preparation. To get the data in a state that is appropriate for modeling, we first needed to clean it, create the appropriate features and encode it. This process culminated in our data preprocessing pipeline. **Sprint 3** was where the feature selection pipeline creation took place. In order to properly optimize any machine learning model, one must use the appropriate features, and therefore we had to trim through the large number of features present in our dataset after encoding. This sprint lasted a total of 2 days. Then followed the creation of the hyperparameter search pipeline in the 2-day-long **Sprint 4**. In this sprint, the tool to find the optimal parameters for the model is created, so that the model utilized in the training and prediction stages can be optimized. After that, the 5-day-long **Sprint 5** took place, where the data split, model train and model predict pipelines were created. In this sprint, the data was split, the model was built and trained on the training dataset and a pipeline was also created to use the trained model to predict on new data, in this case the test data from the previously mentioned split. Additionally, an MLFlow component was built into the training pipeline to allow for proper model versioning and management. This sprint lasted for 3 days. Afterwards, **Sprint 6** took place. The data unit tests pipeline was built, in order to set up a number of tests to assure the quality of the data and its transformation is up to standard. Finally, the final step in our development process, **Sprint 7**, corresponded to the construction of the data drift pipeline. This tool is an extremely important part of the final solution and it will be responsible for the detection of significant changes in future data that might lead to model performance drops and require re-adjustments.

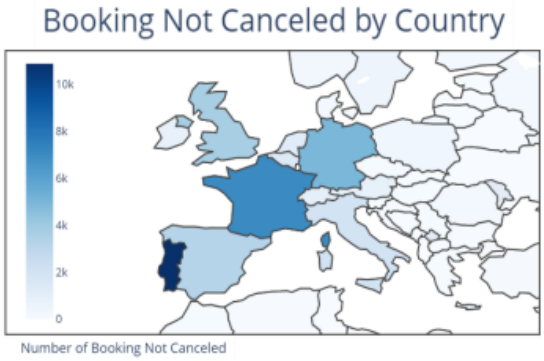
Results and Conclusions

From our data exploration, several initial conclusions were drawn. First and foremost, in this particular dataset, the target variable is relatively balanced, with nearly 42% of observations corresponding to cancellations. Additionally, the distribution of bookings per month of the year shows that demand peaks in May, July and August, indicating a strong likelihood of the presence of seasonality in demand.



The majority of bookings originate from Portugal, which makes sense considering it is the place where the hotel is located, and most of these bookings are from first-time guests. Additionally,

room type “A” is by far the most common one, and most of these bookings come from Online Travel Agents.



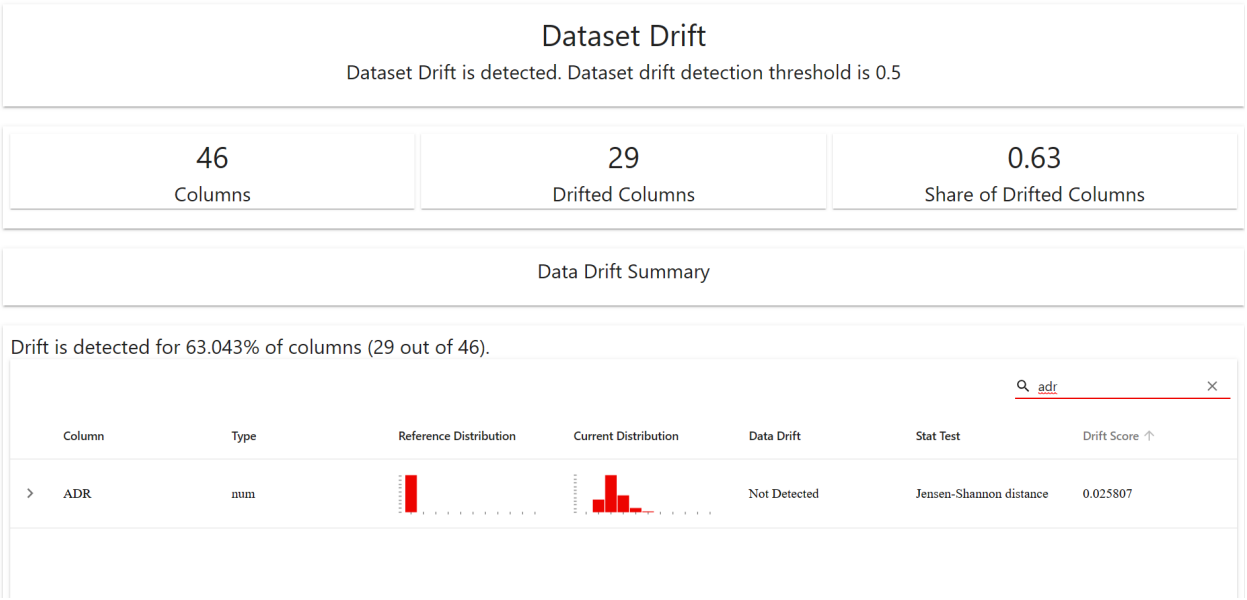
After setting up the cleaning, feature engineering and encoding stages, we proceeded to feature selection. This step is of special importance, not only for optimizing model performance, but also to figure out which variables are most relevant to the model and how these variables can be further researched to learn more about business success drivers. After applying the RFE method, our results for feature importance were the following (ordered from most to least important):

Feature	Importance
Season_Winter	0.084798281
PreviousCancellations	0.083888294
CustomerType_Transient	0.073091217
Semester_2	0.059089836
IsRepeatedGuest	0.053153113
ArrivalDateMonth_3	0.052074994
DepositType_NoDeposit	0.051032524
Agent_42	0.042703892
Agent_128	0.041234476
Quarter_2	0.03664191

This important “Top 10” ranking could give us some relevant insights into some characteristics of this hotel’s business. First of all, the importance of the winter season in the predictive model could indicate some type of relevance of the seasonality in predicting cancellations, since the winter is the season with the least demand. Additionally, the “NoDeposit” deposit type feature being relevant could indicate that customers that do not make a deposit are more likely to cancel, which would be a logical conclusion. With this information in mind, some type of hotel regulation change regarding deposit requirements could be taken into consideration in order to mitigate the high incidence of cancellations. Finally, the “IsRepeatedGuest” feature relevance could indicate some type of correlation between recurring customers and cancellations. By looking at the initial data exploration, it would seem that repeated customers are proportionally less likely to cancel and, therefore, it might make sense to target retention efforts primarily at first-time clients. Other angles of exploration into these and other features could prove useful for future analysis, but at first glance, these are some of the most relevant trends we noticed.

After properly building, versioning and optimizing the model parameters, we arrived at a final model solution. Our final model is a Random Forest Classifier implementation, and it boasted an F1 Score of 0.83 and an Accuracy of 0.86, which is a very satisfactory result and falls within the threshold of our technical success criteria.

We tested data drift by getting a subset of data and changing it in a way where we would suddenly get half of the data as English visitors to our hotel, where 75% of them would cancel their reservation and spend 50€ more in their ADR. We used the Jenson-Shannon test to check if the distributions of the train set variables were kept into this drifted set. Our tests ended up figuring out data drift in many variables, not only the ones we altered. This is normal, as we got a truncated subset of the original test data to do this, so other columns' values also are a bit different in terms of distribution, which would probably not happen if we were doing random uniform sampling. A sample of the obtained dashboard using altered data is in the figure below.



We actually did multivariate data drift using nannyML, which is included in the datadrift pipeline but its results were not analyzed (in spite of being exported into a csv) due to time constraints.

Finally, it is relevant to mention that a multitude of Data Unit Tests were developed and deployed. These tests targeted several aspects of our data and the consistency of the transformations applied to it. This includes checks for data inconsistencies, null values, data types and target distribution. These procedures serve not only to ensure that our pipelines are currently solid, but also to allow for future monitoring of data quality.

Overall, we believe the final version of our solution is solid and, although various limitations, constraints and possible next steps are delineated in the section below, we are satisfied with the performance of our final model, which fulfilled our technical goals and is likely to be of high business value to the hotel should it ever be properly implemented in a real-world setting.

Advantages, Limitations and Next Steps

The frameworks we used for this project's development are clearly advantageous for this kind of task and although we believe our solution to be solid, thorough and well-implemented, we

recognize it still only constitutes a proof-of-concept and, therefore, there are some limitations associated to it.

First of all, Kedro was really helpful for the whole projects' harmony and organization. By using Kedro and its template disposal, we managed to not only have very organized code and pipelines, clearly stated as to what they all do starting by their naming, but also work in a much smoother way by separating all those datasets within their respective folders. It also allowed us to go back and forth when running and testing pipelines, isolating them in an easy to debug way. It also allowed for quick installation of all the project's requirements and for easy access to MLflow UI.

Speaking of MLflow, it was really advantageous for model tracking. Because our project is more dedicated to the project's integration than to performance itself, we ended up only using MLflow for tracking the best models when we did hyperparameter search. Because of it, we got five models with good performance (we ended up having to delete the other 45 models we saved as a result of the random search because we did not need them and they were taking way too much space) and chose the one with the best testing score. MLflow comes in clutch not only in tracking the models, but by saving their accessible artifacts, which comes in handy, as this way we do not have to train the models all over again when we want to use said model. This is also an advantage of the Kedro organization, because it also allows for saving pickle files with the models for later use. It also allowed us to register the model for production.

MLflow would also be very helpful if we used spark, as it is supported by it. If we wanted to use more data or to parallelize our jobs to make our runtimes more efficient, we could use (py)spark and make use of Databricks and its distributed file system, which would also allow our local machines to be less of a bottleneck. This means packages like pandas could be substituted by spark's RDDs or pySpark's version of pandas. However, with the data we have, the project is feasible, we would really just need spark if the amount of data escalated a lot.

NannyML, Great Expectations and Evidently were helpful packages when dealing with data drift and unit testing. Their problems related only to installations and implementation errors, specifically great expectations which did not run in some computers.

Overall, there is a lot where we could improve our work. There could be more in-depth analysis of data drift, using more tests like PSI or Kolmogorov-Smirnov, more unit tests, more model testing and so on. We could also do all the Kedro configuration from scratch to get a more personalized environment building the pipelines, as we have used a template to add and customize to our project's needs.

To deploy the model, we could use container frameworks such as Docker. This would allow us to share the models without having to replicate the development phase and spend precious time in getting it to work. By bundling the code into containers it allows for standardized operations to manipulate them and run the project smoothly. It would also help ensure the model's reliability throughout its lifetime.

Packages and Versions

black~=22.0

flake8>=3.7.9, <5.0

ipython>=7.31.1, <8.0; python_version < '3.8'

isort~=5.0

jupyter~=1.0

jupyterlab_server>=2.11.1, <2.16.0

jupyterlab~=3.0, <3.6.0

kedro~=0.18.8

kedro-telemetry~=0.2.0

nbstripout~=0.4

pytest-cov~=3.0

pytest-mock>=1.7.1, <2.0

pytest~=7.2

evidently>=0.3.3

great_expectations>=0.17.1

matplotlib>=3.5.2

mlflow>=2.3.0

nannyml>=0.9.0

numpy>=1.21.5

pandas>=1.4.4

scikit_learn>=1.0.2

setuptools>=63.4.1

shap>=0.41.0