**Crew Absenteeism Prediction**

|  |
| --- |
| **Written by:** Mihir  **Monitored by:** Rajesh |

Abstract

*Implemented a predictive system that forecasts crew absenteeism on a day-of-operation basis. The system leverages a comprehensive database which includes crew profiles, historical absenteeism records and roster activity information. Given a published roster period, the system applies machine learning techniques to predict absenteeism probabilities for all crew members scheduled to be on duty for a given day-of-operation. The system contains features like experiment tracking, model monitoring and hyperparameter tuning which can be easily managed and visualised using the MLflow Dashboard. Additionally, the system incorporates LIME (Local Interpretable Model-agnostic Explanations) to provide interpretability and transparency in its explanations. The reports produced by the model allows clients to understand the factors that influence absenteeism, enabling them to leverage these predictors effectively.*

Motivation

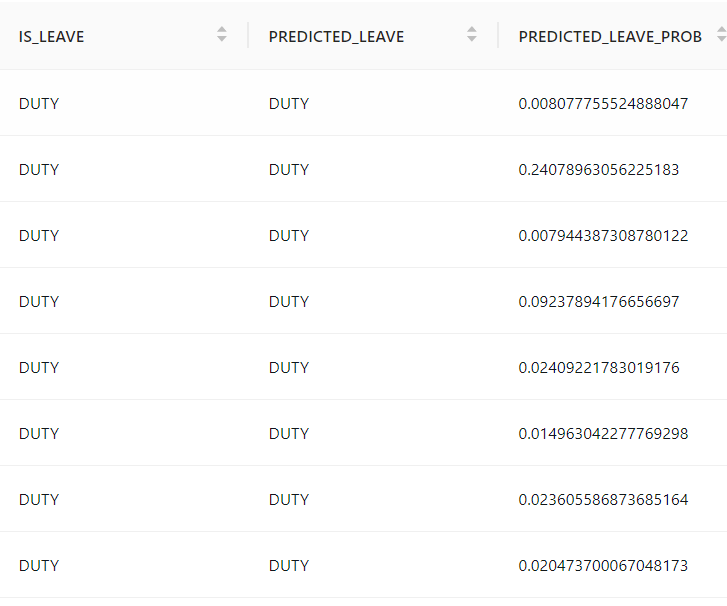
Unplanned absenteeism often necessitates replacements and overtimes and can sometimes lead to disruptions in flight schedules. Ensuring optimal resource allocation is crucial, which involves not only maintaining the right number of standby crew members, but also ensuring they are correctly qualified and meet regulatory requirements.

Instructions to run the model

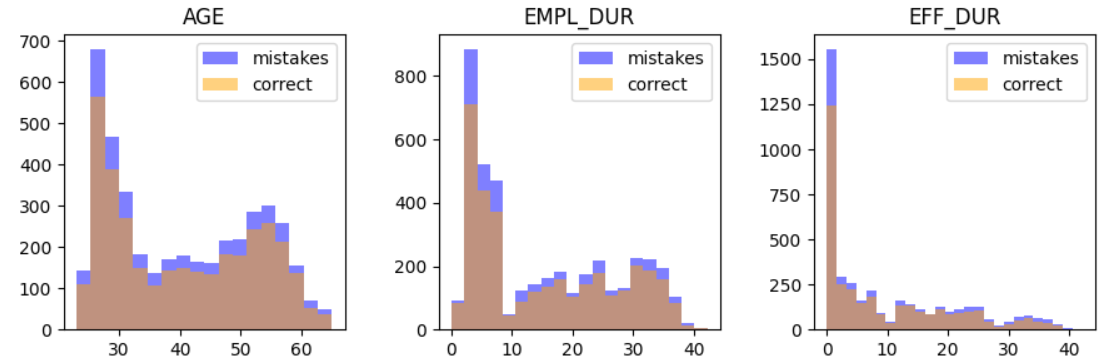
1. Start the server by running python -m uvicorn main:app –reload .
2. Initiate a POST API call to [*http://localhost:8000/train\_model*](http://localhost:8000/train_model) with body: {start\_date, end\_date, run\_name, sampler\_tag, model\_tag} .
3. Initiate the mlflow UI server using python -m mlflow ui . Using the dashboard, compare and explore the visualisations
4. Initiate a POST API call to http://localhost:8000/explain\_prediction with body: {end\_date, sampler\_tag, model\_tag, target\_crew\_id} to create LIME explainability report for a given crew member

System Workflow and Deliverables

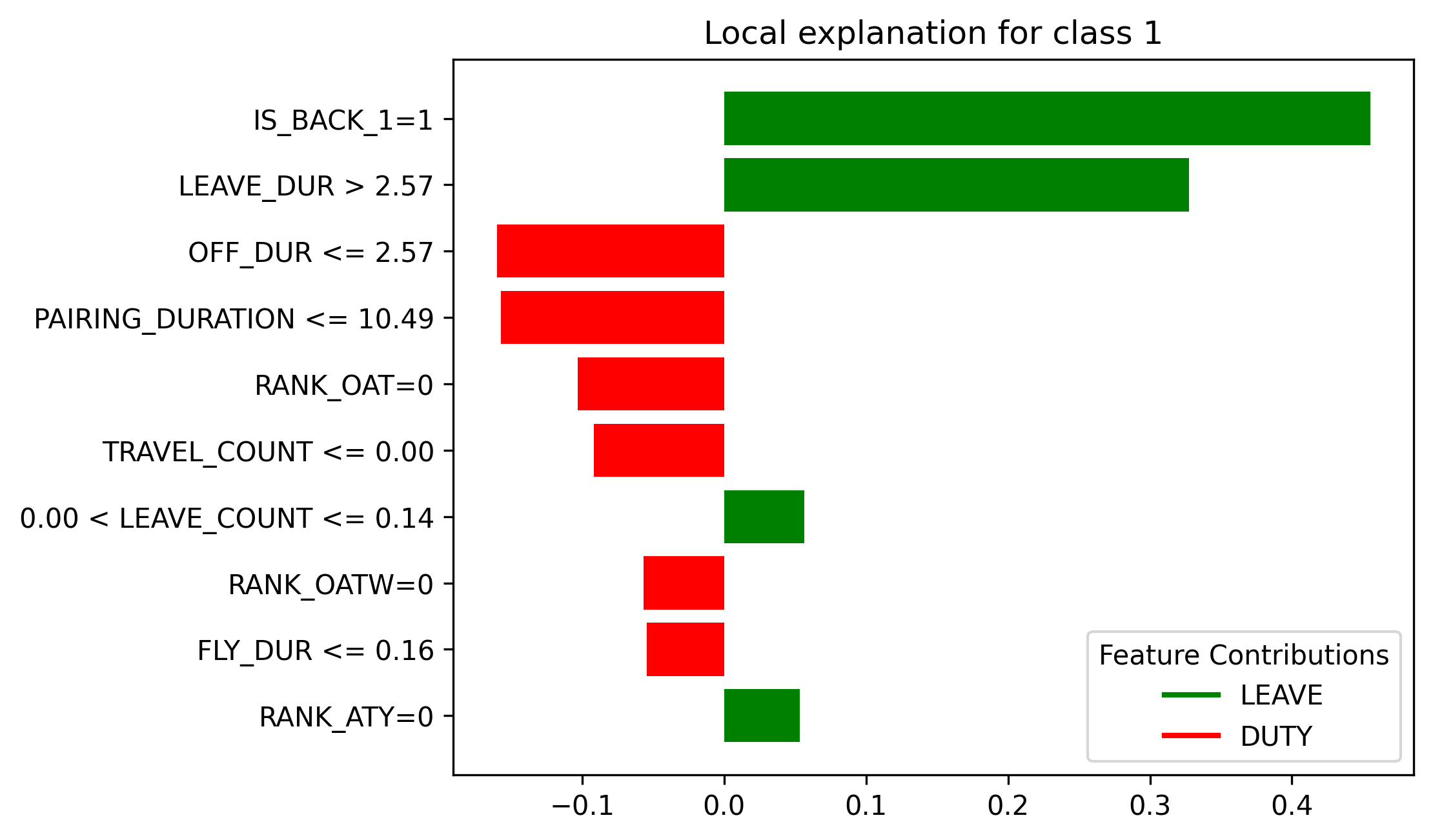
1. Client specifies the roster period by defining a START\_DATE and END\_DATE, which determines the data that the system can access.
2. The system then presents a sheet with the list of all crew members that are scheduled to be on duty for END\_DATE along with it’s predictions (‘DUTY’, ‘LEAVE’), and the probability of prediction.



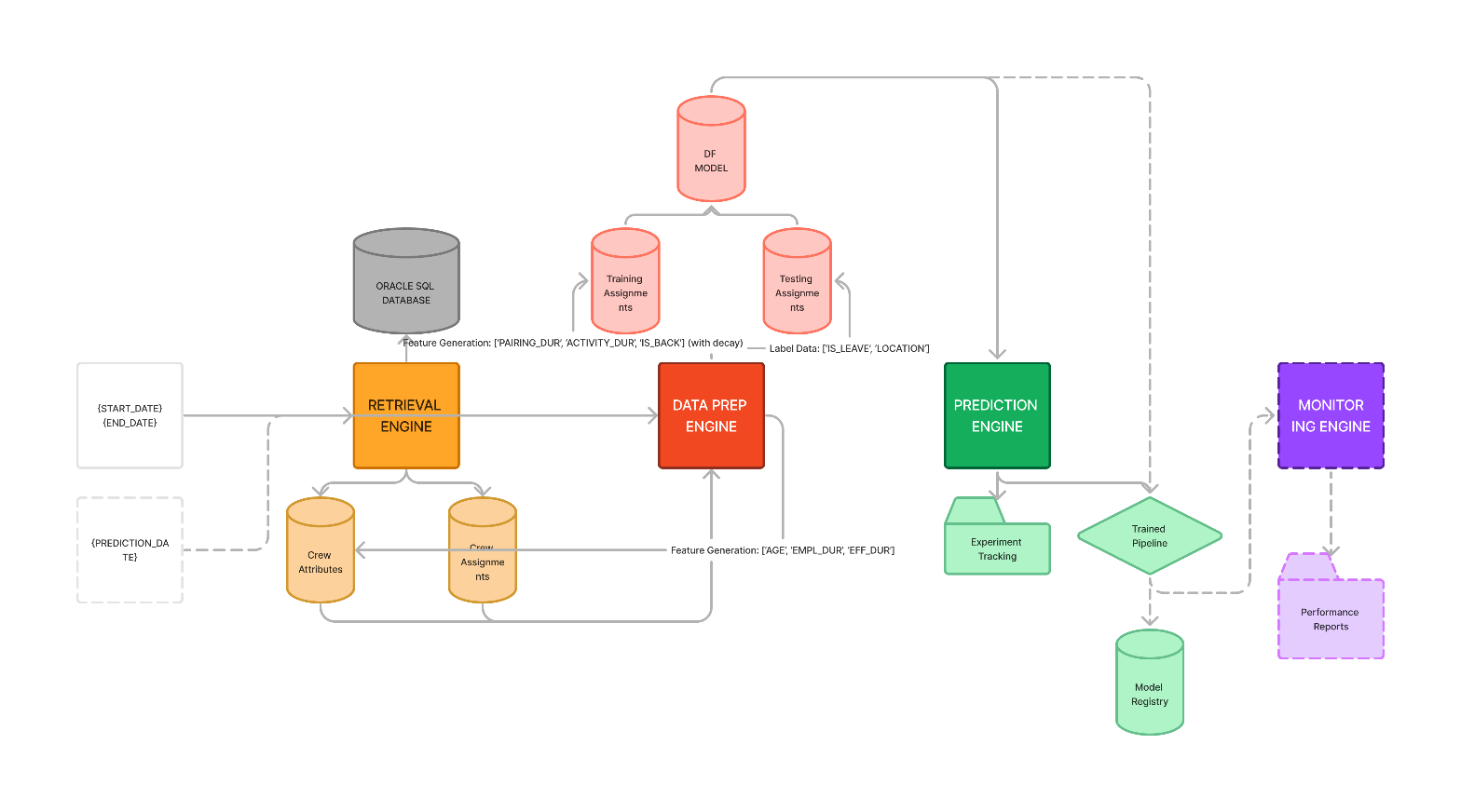
1. The system then retrieves actual absenteeism data to evaluate it’s predictions and creates a diagnostic report



1. The dashboard can be used to evaluate performance of the model across different runs
2. The system has an additional feature, where given a CREW\_ID, it will produce an explanation for it’s prediction, providing a graph that highlights key features that influenced it’s decision. This allows one to understand why the model made a mistake



Architecture



Retrieval Engine

Given the roster training date range, returns

* Crew profile data
* Training Crew Assignment data
* New Crew Schedule data

|  |  |  |
| --- | --- | --- |
| Attribute | Type | Description |
| con | oracledb.Connection | The connection object to the oracle database |
| cur | oracledb.Cursor | The cursor object used to execute SQL queries |
| training\_start\_date | str | The start date of the roster training period |
| training\_end\_date | str | The end date of the training period |

|  |  |  |
| --- | --- | --- |
| Method | Params | Description |
| get\_crew\_attrs |  | Method to retrieve crew profiles. If this data exists in the cache, just read it. Else, retrieve it from the database and save it in the cache |
| get\_training\_crew\_assmts |  | Retrieve crew assignment data in the given date window from oracle database or cache. Save in cache if necessary. |
| get\_new\_crew\_assmts | (curr\_date) | Retrieve crew data that are scheduled to have duty on curr\_date from oracle database or cache. Save in cache if necessary |

Future Improvements:

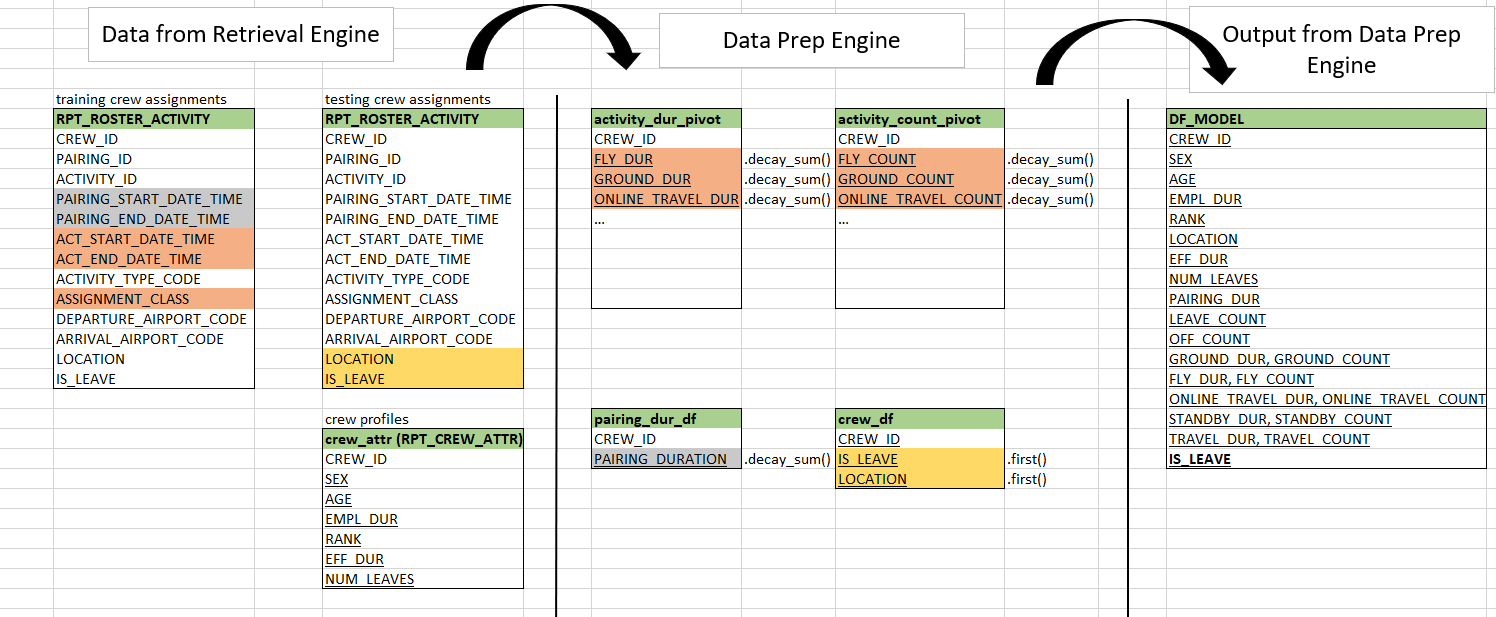
* Modify methods to configure whether client wants to store in cache (due to data piracy)
* Possibly add encryption to cache
* Add multi-threaded parallelism functionality as retrieving large amounts of data from the oracle database takes lot of time

Data Prep Engine

Given the data from the retrieval engine:

* Cleans the dataframes
* Splits them for testing and training
* Extracts features like pairing duration, activity duration, activity count, location
* Labels each crew ID as DUTY/LEAVE

The mind map is given below:



|  |  |  |
| --- | --- | --- |
| Method | Params | Description |
| prep\_training\_data | (crew\_attrs, crew\_assmts) | Prepares training data by cleaning and transforming the provided crew attributes and assignments data. |
| split\_data |  | Splits the crew assignments data into training assignments and testing assignments based on the training end date.   * Training Assignments: Crew Assignments with ACT\_START\_DATE < END\_DATE * Testing Assignments:Crew Assignments with ACT\_START\_DATE >= END\_DATE |
| extract\_features |  | Extracts and computes features from the training assignments data to prepare it for modeling.   * Crew profile contains [SEX, BASE\_CODE, RANK] * (Step 1) Extracts [AGE, EMPL\_DUR, EFF\_DUR] using BIRTH\_DT, EMPL\_DT, EFF\_DT in crew\_attrs * (Step 2) Extracting [PAIRING\_DURATION, ACTIVITY\_DURATION, ACTIVITY\_COUNT] (decay) using decay sum with duration and time\_since * (Step 3) Extracting [LOCATION, IS\_BACK]:   + LOCATION is location of first activity of crew\_ID in testing\_assignments   + IS\_BACK = 1 if LOCATION == BASE\_CODE * Normalizes features |
| label\_data |  | Labels each crew member as (‘DUTY’, ‘LEAVE’) to train the model |
| prep\_new\_data |  | Updates datasets with the most recent information |

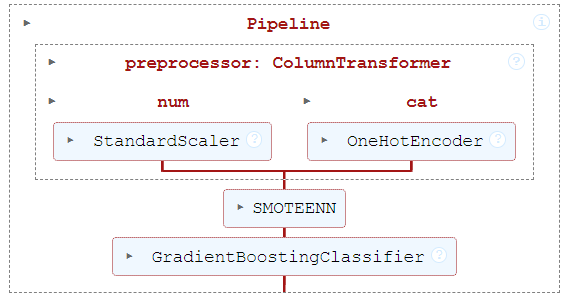
Future Improvements:

* A limitation with the data prep engine is that the resulting dataframe only contains crew data that have data in the training roster period. Consequently, crew members scheduled for duty on a specific day but lacking records in the training period are excluded from predictions. This is because necessary features cannot be extracted for this crew member. To address this issue, the system could be enhanced to enable predictions for such crew members based solely on their crew profile information, ensuring that all scheduled crew members are considered, regardless of their activity during the training period.
* Implement parameter tuning for the decay parameter

Prediction Engine

* Contains a pipeline registry and a run\_pipeline script.
* Each pipeline contains a preprocessor, a sampler and a classifier
* Sampler options: [smotenc, smotetomek, smotenn]
* Classifier options: [GradientBoostingClassifier, XGBoostClassifier]

An example pipeline configured with smotenn sampler and gradientboosting classifier looks like this:



These pipelines can be configured by the client using sampler\_tag and model\_tag

|  |  |  |
| --- | --- | --- |
| Method | Params | Description |
| create\_pipeline | (numeric\_features, categoric\_features, sampler\_tag, model\_tag) | Creates a pipeline according to given configuration |
| train\_pipeline | (pipeline, X\_train, y\_train) | Fits the pipeline with training data |
| evaluate\_pipeline | (trained\_pipeline, X\_test, y\_test) | Evaluates the pipeline and displays the performance metrics |
| save\_pipeline | (model\_name) | Saves the pipeline into a pickle file |

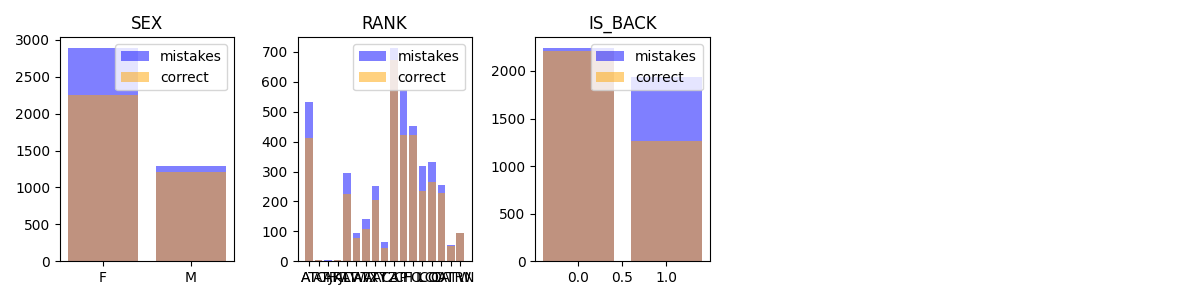
Future Improvements:

* Add a hyperparameter tuning phase. The reason I did not implement this is because each run takes a lot of time. Would require GPU access to properly tune hyperparameters in reasonable amount of time

Monitoring Engine

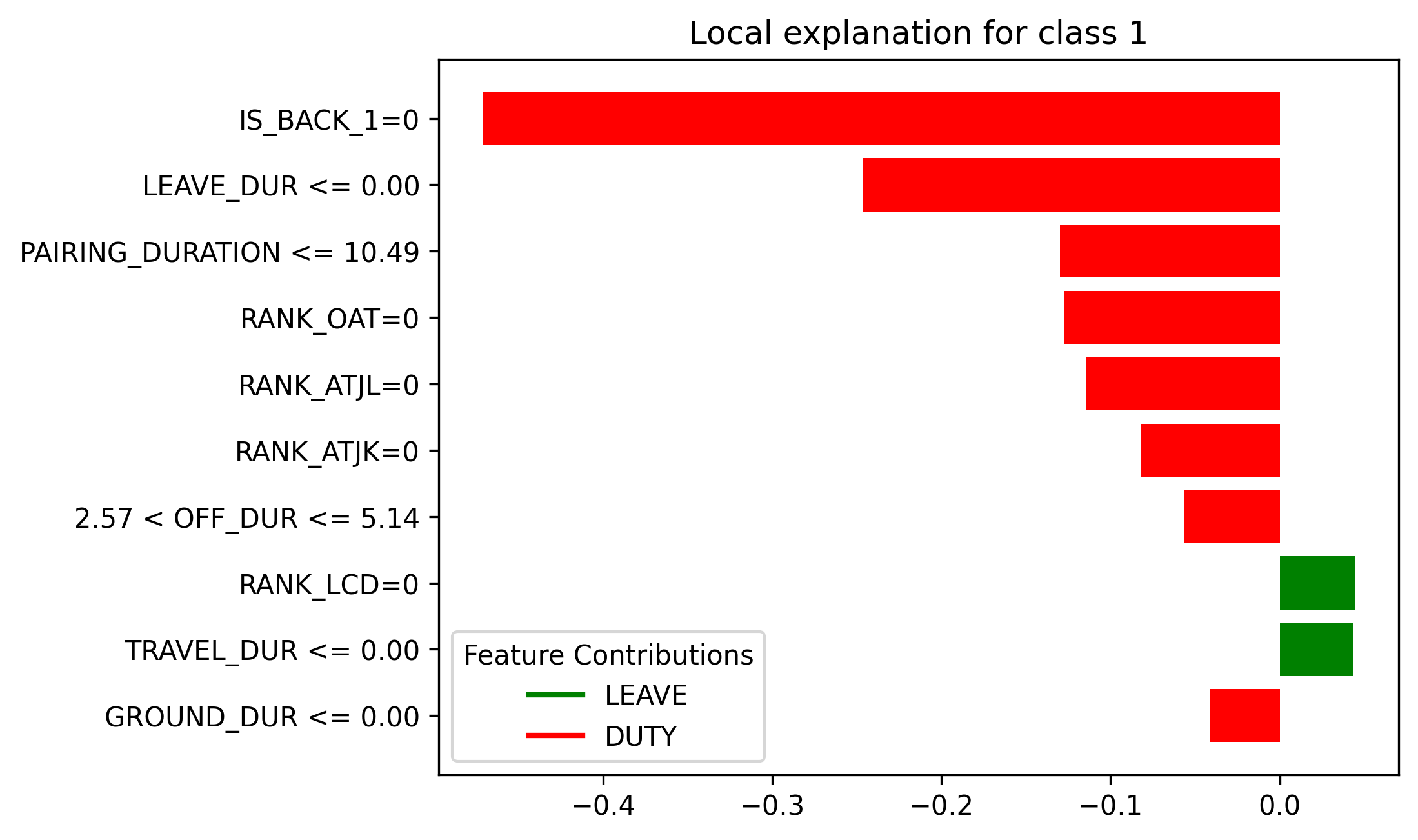
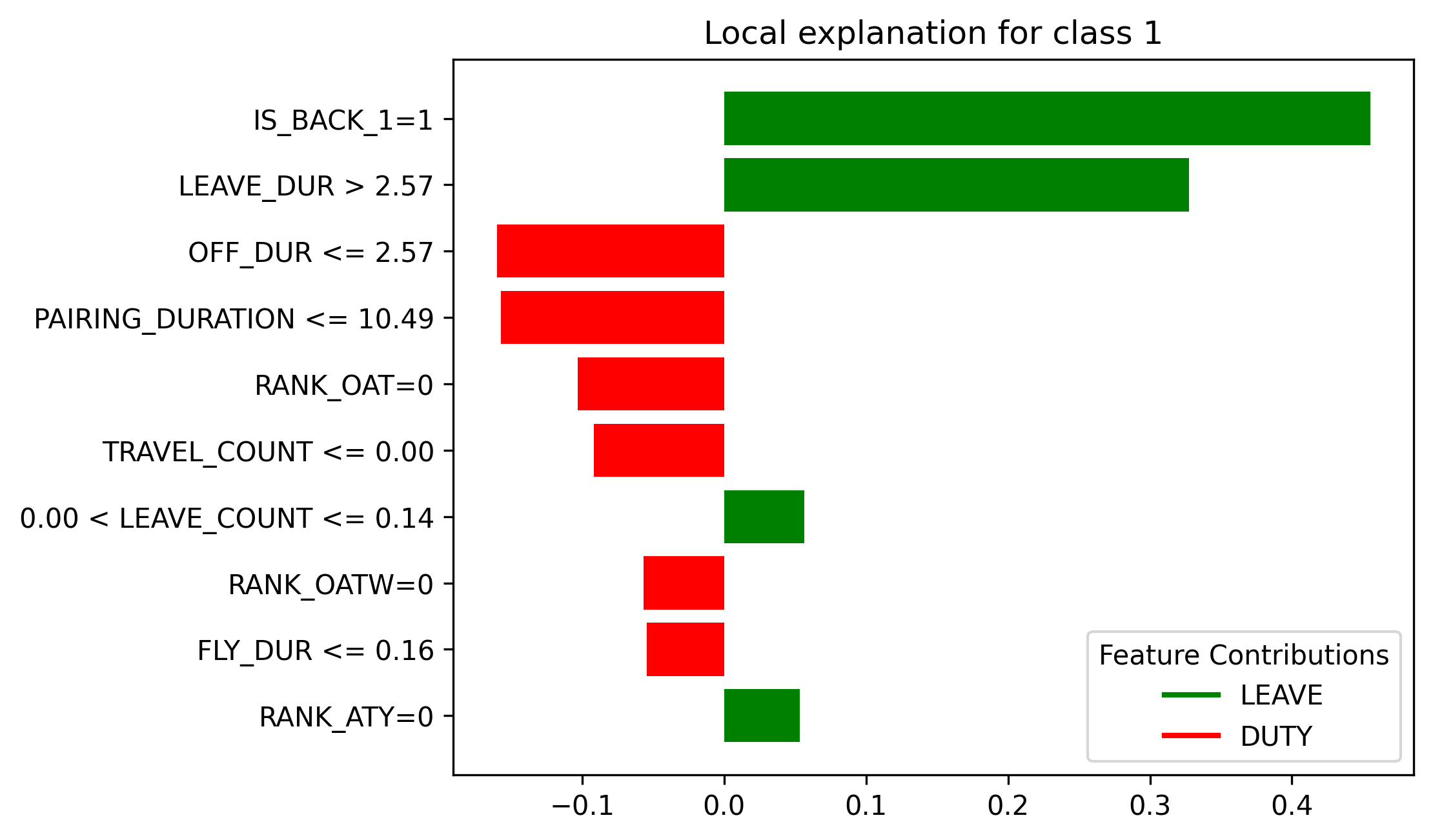
Helps to monitor and visualize the performance of the prediction model. It generates a report containing histograms of correct vs incorrect predictions for each feature. Additionally it explains its predictions for a given crew member

For example:



* One insight that can be drawn from this report is that the histogram for IS\_BACK reveals that most misclassifications happen for crew members that are back in homebase (IS\_BACK=1). This insight aligns with the intuitive understanding that crew members who are away from homebase are less likely to take unplanned leave compared to those who are at home. Thus, the monitoring engine can help towards targeted analysis to identify patterns of absenteeism.

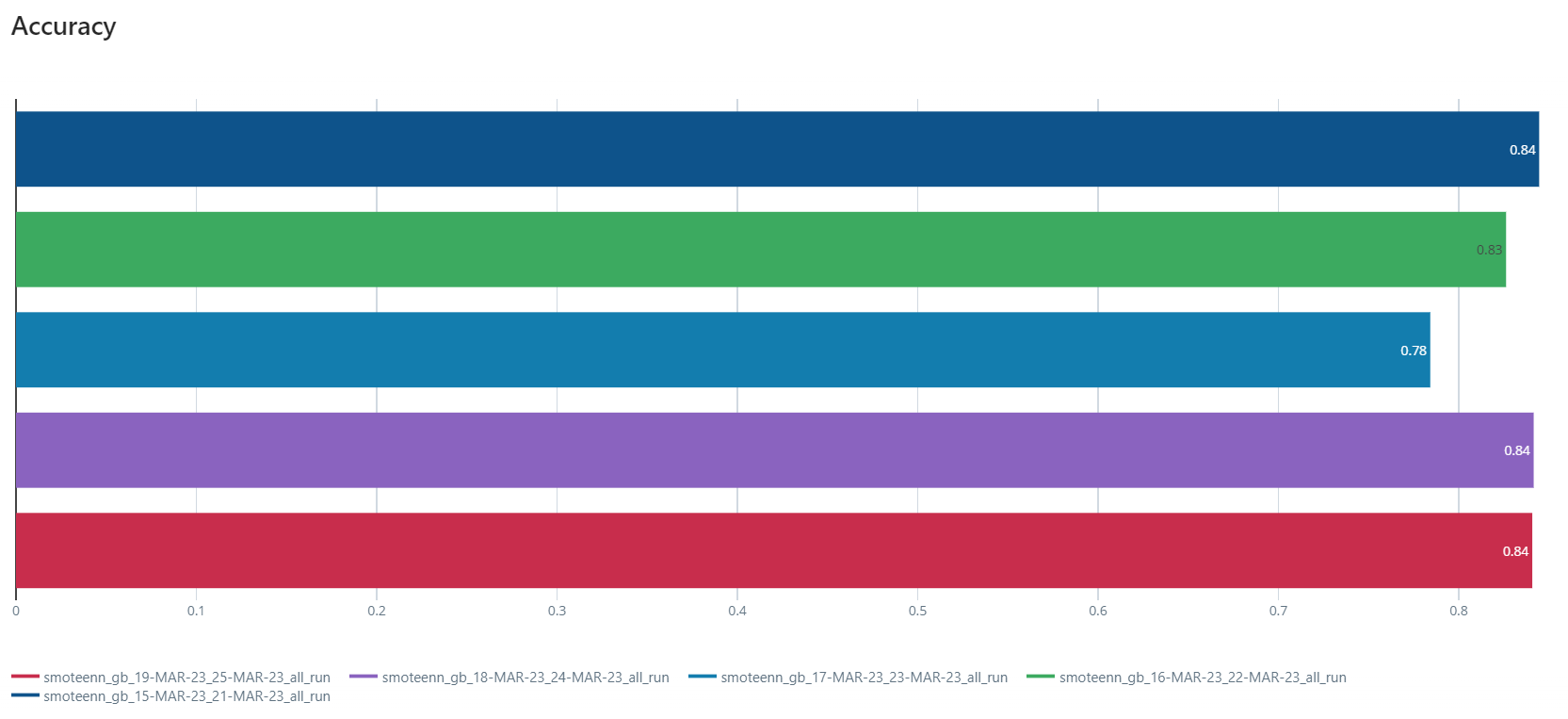
Prediction Explanation Report:

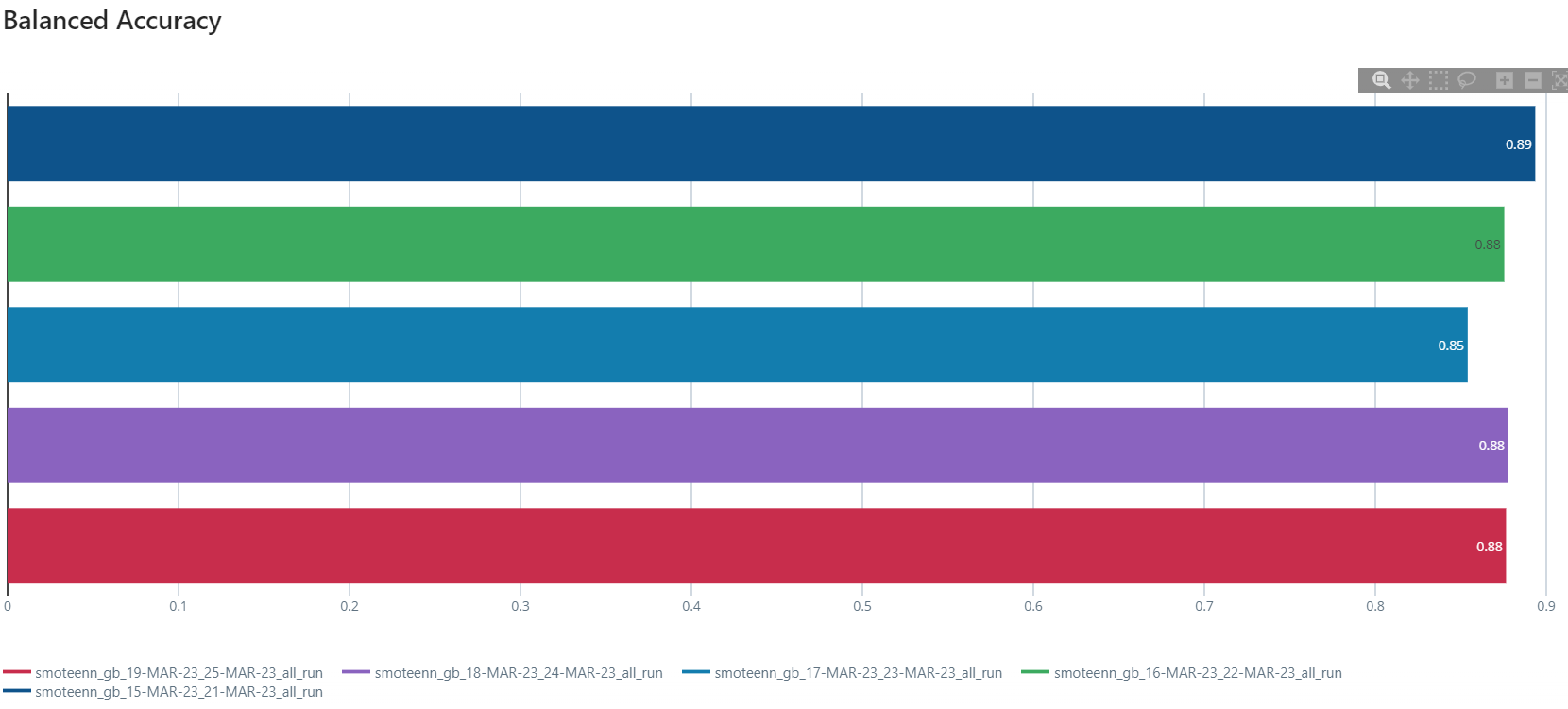
 

Consider the above 2 reports generated for 2 different crew members. Fig 1 is for a crew member who was predicted as ‘DUTY’. The highest factor that reinforced this prediction was that the crew member was not back. Fig2 is for a crew member that was predicted as ‘LEAVE’. One of the highest factors that reinfocred this was a high LEAVE\_DUR. These are the sorts of insights that can be drawn from the monitoring engine.

Evaluation

After running the model for 5 consecutive days, the following was the performance:

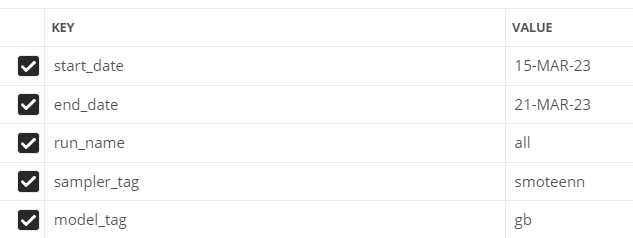




* Takes an average time of 7.9 minutes to retrieve 1 week worth of training data for the model
* Takes an average time of 48.8s to make predictions

Case Study

Ran the model with the following parameters:



This means that the model is trained from 15-MAR-23 to 21-MAR-23 and the predictions are made on 22-MAR-23

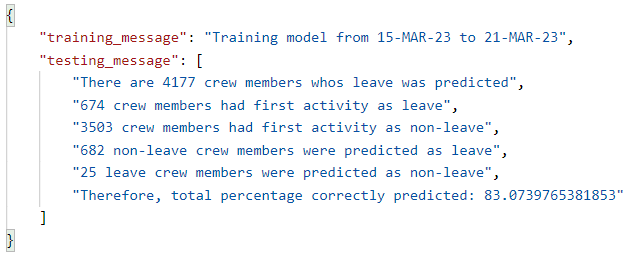
logs.txt ->

Number of crew members with activities on 22-MAR-23: 6342

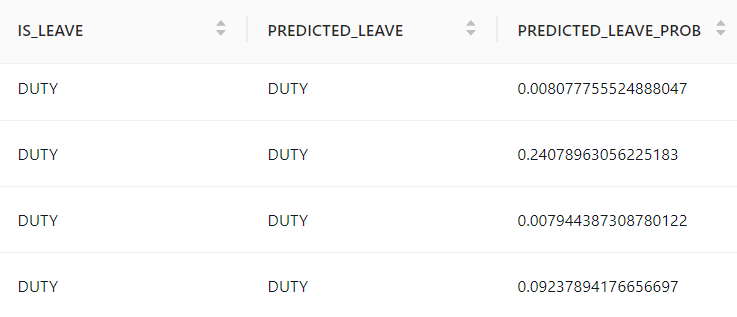
Number of crew members that are supposed to be on duty: 4194 (Removal of OFF and planned leaves)

Number of crew members predicted on: 4177 (17 crew members were not found in training data)

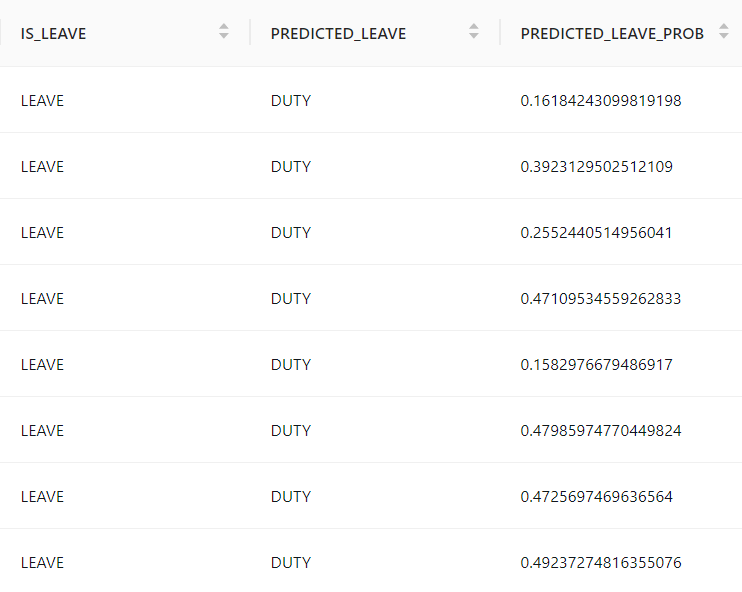
Message sent to API after training and testing is complete ->



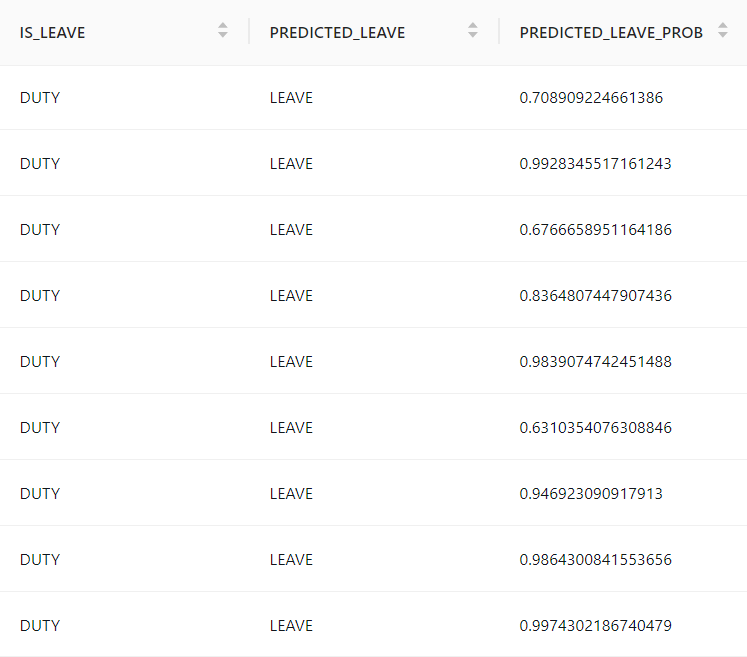
Predictions ->



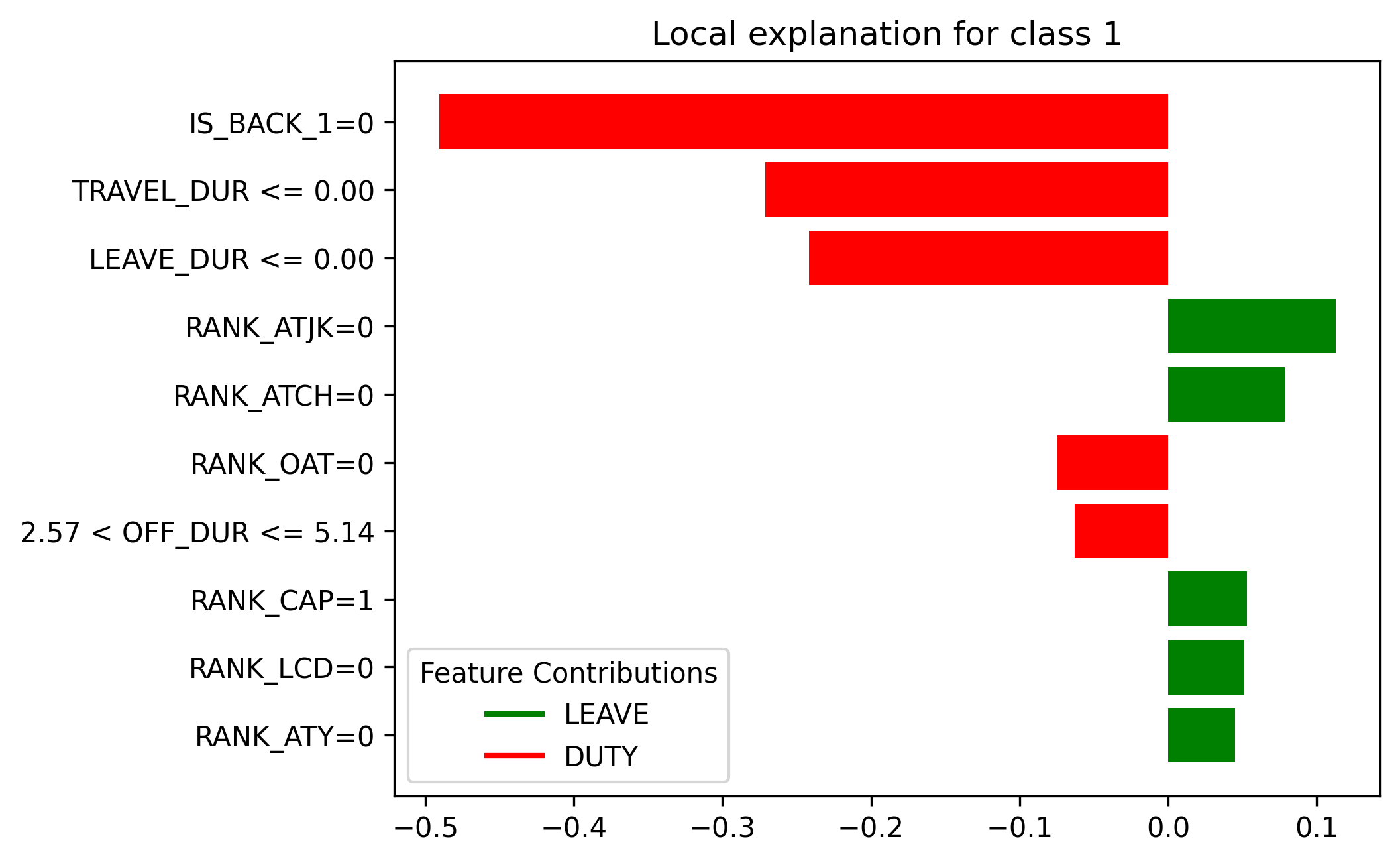
Looking at the false negatives: “leave crew members predicted as non-leave”



Looking at the false positives: “non-leave crew members that were predicted as leave”



Now let us examine why the first FN was predicted as such:



Looks like the model predicted this crew member as duty mainly because

* the crew member is not back in crewbase
* has not travelled in the past week
* has not taken any leaves in the past week

Areas of Improvement

There are several limitations to the model like:

* The model is incapable of accounting for time based factors and temporal events like seasonal flu season, global events that can influence leave patterns.
* The model also does not account for any external factors like weather
* The model does not account for fatigue caused to crew members due to timezone differences and duties during the Work-on-Call (WOCL) cycle.

Concepts

Amazon RDS for Oracle

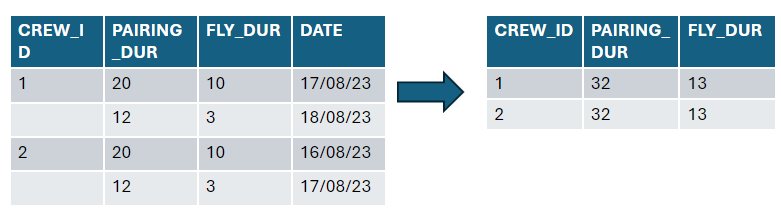
* Used to manage relational databases that are connected to the oracle database

Decay aggregation

* I used a decay sum to generate features like PAIRING\_DURATION, {ACTIVITY}\_DURATION, {ACTIVITY}\_COUNT. This means that the roster activity data, where each row is an activity was used to calculate the above features for every crew member. The naïve approach would be to just sum up the duration of each pairing for a given crew member to give PAIRING\_DURATION.

**Shortcoming of Naïve approach:**

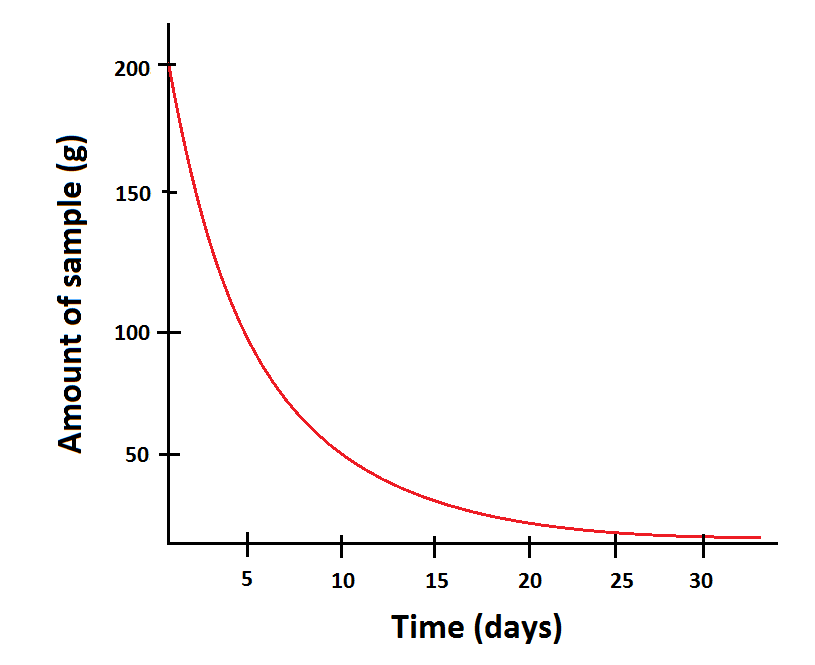
* Consider the naïve .sum() appraoch to calculate the above features. Below is a counter example to showcase the shortcoming of this approach:



Consider the case in which 2 crew members have identical pairing and fly times. However, CREW2 did this sequence of activities few days before CREW1. A naive sum aggregation would produce the same PAIRING\_DURATION and FLY\_DURATION for both implying that the model would get identical information for both. The model does not relay the time component of completeion of these activities which could pose an issue

**Solution: Time decay sum**

* A time-adjustment factor has to be added to to create a weighted sum when calculating activity and pairing durations. Weights can be based on how much time has passed since that activity. This is the underlying principle of the alpha decay



Undersampling and Oversampling methods

* The data fed to the model consists of each crew member scheduled for duty and a target variable indicating whether they took unplanned leave or came for duty. This data is clearly unbalanced with more than 75% of the crew members coming in for duty. This leaves the ‘LEAVE’ class to be a minority. Many classification models underperform in classifiying the minority class in an imbalanced dataset. To address this, we use undersampling and oversampling approaches.
* Undersampling involves reducing the number of overrepresented samples and oversampling involves increasing the number of underrepresented samples.

**SMOTE (oversampling):**

* The idea behind the naïve oversampling method is to duplicate some random examples from the minority class. This does not add any new information from the data. Therefore we use SMOTE (Synthetic Minority Oversampling Tecnique) to synthesize new examples from the minority class. The process behind this is as follows:
  1. Choose random data from the minority class.
  2. Calculate the Euclidean distance between the random data and its k nearest neighbors.
  3. Multiply the difference with a random number between 0 and 1, then add the result to the minority class as a synthetic sample.
  4. Repeat the procedure until the desired proportion of minority class is met.

**SMOTETomek (SMOTE with Tomek Links):**

* Apply SMOTE to generate synthetic samples of minority class
* Remove samples from majority class using Tomek Links. A Tomek Link is a pair of instances (one from each class) that are nearest neighbors to each other. These links are considered to be noise or ambiguous points that do not help in defining the decision boundary clearly.

**SMOTEENN (SMOTE with Edited Nearest Neighbors Classification):**

* Apply SMOTE to generate synthetic samples of minority class
* Remove samples from majority class using ENN, which is a cleaning technique that removes instances that are misclassified by their nearest neighbors.

LIME

LIME, which stands for Local Interpretable Model-agnostic Explanations, is a technique designed to make machine learning models more interpretable.

How it works ->

* 1. Choose an instance for explanation
  2. Generate perturbed data by modifying the above instance to generate a variety of new, similar instances.
  3. Use the model to make predictrions on these samples. This provides insights into how changes in the input features effects the model’s output
  4. Assign weights to the perturbed samples based on their proximity to the original instance.
  5. Train a simple, interpretable model (such as a linear regression or decision tree) using the perturbed data and their corresponding predictions from the complex model. This simple model is designed to approximate the behavior of the complex model in the local region around the instance.
  6. Analyze the simple model to understand the influence of different features on the prediction for the specific instance.

Timeline

|  |  |
| --- | --- |
| Week | Updates |
| Week 1 | * Onboarding and Environment setup * Environment: CiscoVPN, OracleSQL, AWS VPN, VSCode, Jira |
| Week 2 | * Understand and formulate the problem statement: “Identify what KPIs are required for predicting crew absenteeism” * Understand the data (RPT\_ROSTER\_ACTIVITY) * Learnt airline business concepts like pairing, duty, activity, deadhead * Ran a basic model using crew profile and 1 day of data. Model had very low accuracy. Indicates more training data is required, more features need to be extracted like holidays, airport crowd, crew fatigue etc. |
| Week 3 | * Created a pipeline to connect and retrieve data from SQL to python * Performed a preliminary analysis on crew leave pattern |
| Week 4 | * Performed outlier detection and removed crew members that have taken leave for a long duration of time.      * Ran the same basic model using 2-day data with different models * Applied undersampling and oversampling methods (SMOTETomek, SMOTEENC, SMOTEENN) |
| Week 5 | * Performed time series analysis using data from 2023. Observed yearly, monthly and weekly trends * Explored predictions options like: classification, regression, time-series forecasting * Supervised Classification: Around 60% balanced accuracy * Supervised Regression: 0.137 (without outlier removal), 0.53 (with outlier removal) * Time Series Forecasting: Used ARIMA model ->      * Observed weekly correlation      * Prediction was too uniform. Unable to capture the complexities and anomalies |
| Week 6 | * Completed presentation to Nagesha, Rajesh * Zoomed in on supervised classification task * Formulated the prediction task to the business problem (version 1) * Extracted features like Pairing Duration, Activity Duration, Activity count using pivot tables (version 2) |
| Week 7 | * Added a decay parameter to mimic the time component (version 3) * Added crewbase, location, and number of leaves * Bug fixes |
| Week 8 | * Moved it from simple jupyter notebook implementation to a comprehensive ML system with modular code * Added a monitoring engine * Exposed APIs using FASTAPI |
| Week 9 | * Learnt about data-centric models and MLOps * Added Experiment Tracking and Model Registry using MLFlow * Performed experiments on feature importance, weekly performance trend etc |
| Week 10 | * Added LIME model for introducing explainability * Started analysis on crew roster assignment variation * Problem Statement: “To detect abnormal patterns and preferential treatment among captain fly duties” * Used KNN and DBScan to find outliers in fly patterns * Found that outliers do exist in fly patterns when looking at the fly duration vs fly count graph. |
| Week 11 | * Implemented a community detection on a bipartite graph of countries and captains that fly international to find outliers in travel network. * Found new hires that fly certain routes that are only given to old hires * Integrated crew documentation data and found that this behaviour can be explained by the presence of US documentation like Visa, Passport and travel permits. |
| Week 12 |  |
|  |  |