# Abstract

As part of my Co-op work term, I had the privilege to work on two projects. My first project was with RBC’s Enterprise Security & Access Management (ESAM) Operations Team, in which I developed a system that could recommend an analyst most suitable to fulfill an employee request. A request is raised as a ticket, through various channels, which eventually gets queued in ticket management systems *(Service Manager 9 & Service Now).* These tickets are resolved by a skilled system analyst. Every day, a manager spends hours to identify system analysts best suited to fulfill several hundred’s of ticket request.

As part of this project, I created a software system which could ingest data from multiple data sources, pre-process & combine it to train multiple machine learning models. This model would consider the type of ticket, its title, the department that raised it, skills required to resolve it, as well as, human factors, such as, an analyst’s vacation schedule, the teams they work for, number of active tickets in their queue, to recommend four analysts most suitable to resolve any given ticket. Evaluation results showed that the system has an accuracy of 85%. A web application was developed to provide a user-friendly interface to upload required data, train the model/estimator & get recommendations. The system has been deployed in the RBC’s private cloud.

# 

# Report Specification

The report is intended for anyone who would like to understand my work. No prior assumption is made, regarding readers understanding of RBC’s internal systems. This is an attempt to explain relevant information, without diving into lower level technical details.

Its recommended for the reader to have some familiarity with Web technologies such as Angular JS, Node JS, Python, Flask, REST Web Service, as well as, Machine Learning *(Supervised Learning, Recommendation Systems)* & Natural Language Processing. Anyone who has prior experience working for a large organization would be able to better understand the problem & appreciate the solution.

The document would also help anyone in RBC’s ESAM Operations Team to enhance my work. Its recommended for developers to review this document along with the source code. The source code can be found at RBC’s private Github, under the names [Automated\_Ticket\_Management\_2](https://rbcgithub.fg.rbc.com/XQ50/Automated_Ticket_Management-2) under XQ50 organization group.

# Table of Contents

[Abstract 1](#_Toc524958280)

[Report Specification 2](#_Toc524958281)

[Table of Contents 3](#_Toc524958282)

[Glossary 3](#_Toc524958283)

[T-Bot 4](#_Toc524958284)

[Introduction 4](#_Toc524958285)

[Background 5](#_Toc524958286)

[System Design 5](#_Toc524958287)

[Pre-Process 6](#_Toc524958288)

[Model / Estimator 8](#_Toc524958289)

[Post-processor 8](#_Toc524958290)

[Evaluation 10](#_Toc524958291)

[Package 10](#_Toc524958292)

[Web Application 11](#_Toc524958293)

[Deployment 12](#_Toc524958294)

[User Guide 13](#_Toc524958295)

[Limitation 13](#_Toc524958296)

[Future Work 14](#_Toc524958297)

[References 14](#_Toc524958298)

# Glossary

|  |  |
| --- | --- |
| API | Application Programming Interface. An interface used to get dynamic data from the database to a web page |
| Client | ESAM Operations Team |
| ESAM | Enterprise Security & Access Management |
| Flask | A microservice server which supports Python. |
| RBC | Royal Bank of Canada |
| REST | **RE**presentational **S**tate **T**ransfer. An architectural style to create Web Services using HTTP protocol. |
| Service Now | A ticket management system, which would replace SM9 in November 2018 |
| SM9 | Service Manager 9. A ticket management system, which would be replaced by Service Now. |
| T-Bot | Ticket Recommendation System created for RBC’s ESAM Operations Team. |
|  |  |
|  |  |

# T-Bot

## Introduction

At RBC, a ticket is created to service an employee request to get privileges. Once a request is approved, it is sent to ticket management systems *(SM9 & Service Now)* for fulfillment. A manager has to analyze the ticket & choose a System Analyst, most adept to fulfill a request.

The choice made by the manager depends on several factors. These include human factors such as the team that is supposed to fulfill the request, team members, vacation schedule, the current workload of the analyst. They also include information about the ticket to be fulfilled, such as the type of ticket, its title, ticket description, the complexity of the ticket, the department that raised the ticket, to name a few. We did not know the skills required to fulfill a ticket.

To automate the decision-making process, I created T-Bot, a ticket recommendation bot which suggests analyst most suitable to fulfill the ticket, based on various parameters. This is expected to reduce the time spent to assign tickets from several hours to a few minutes. T-Bot can be accessed within RBC using the following web address <https://tbot-nonreversed-koko.apps.pcf.devfg.rbc.com/>

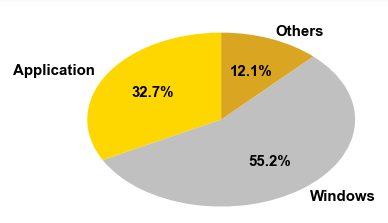
We’ll now try to understand RBC’s Ticket Management System & follow it with the system design.

## Background

At RBC, a ticket request is made to enable employees access systems to gain special privileges on an as-needed basis. Tickets could be requested to – add user to a specific group, provide Github Access to an organization (XQ50), enable VPN access, create a distribution list, are some of the 400 different types of tickets that are raised by employees.

When an employee wants special access, they have to raise a ticket in RBC’s MyMarketplace, a portal to create ticket request. When a ticket is created it begins its life-cycle. Once created, the ticket is sent to the employee’s manager for approval. Once approved, it is sent to the ESAM Operations Team for fulfillment.

A ticket request is categorized into bins. A bin is simply a categorization of a ticket, which depends on the kind of request. For this project, we have 2 bins: Windows *(Tickets related to Windows Operating System)* & Application Services *(Tickets related to various applications in RBC)*. Together they constitute approximately 88 % of the tickets, as shown below.



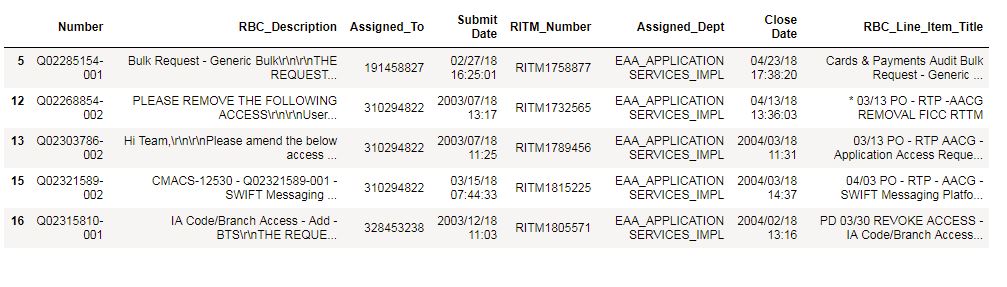
*Figure 1: Ticket Distribution.*

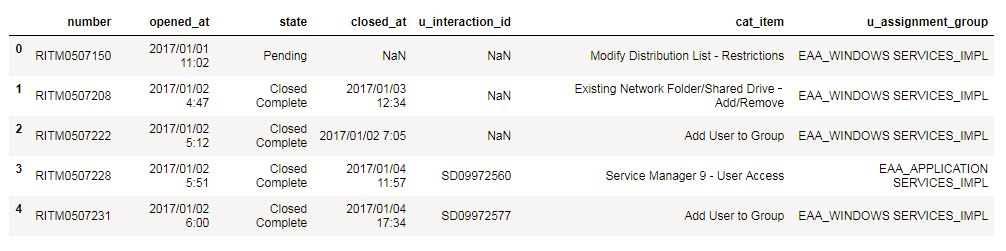
## System Design

We would now look at the steps followed to create T-Bot.

### Pre-Process

There are 2 primary ticket management systems which aggregate tickets created from several systems. Tickets received by ESAM Operations are raised against various bins. However, my work focussed on 2 bins (Windows & Application Services). So, I ensured the tickets we would analyze are either in Windows or Application bin. Below is a screenshot of tickets as seen in Service Manager (SM) 9 & Service Now.

*Figure 2: SM9 tickets.*



*Figure 3: Service Now tickets.*

ESAM Operations Team has automated fulfillment of some ticket types. These tickets are auto-resolved. Such tickets are not assigned to an analyst. Hence, these were not considered for recommendation.

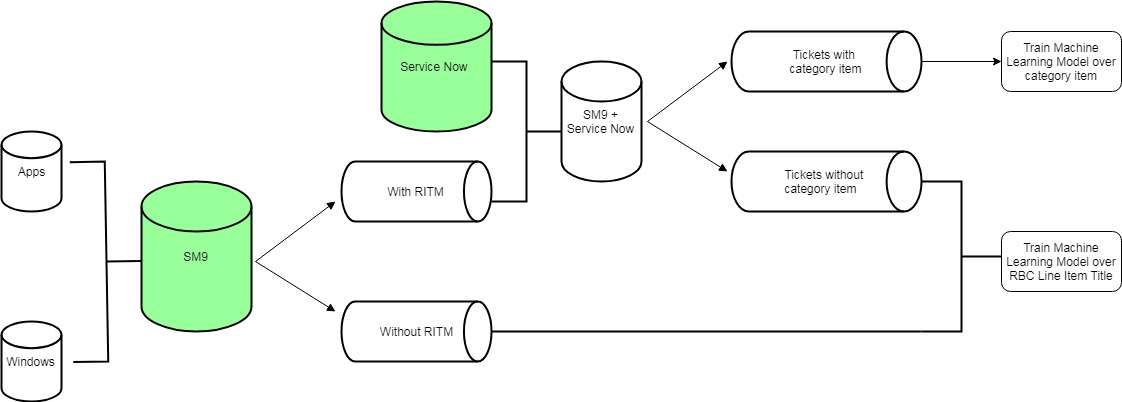
We had tickets for the past 20 months. These were assigned to a total of 105 analysts who had worked on these tickets. Our focus was to assign tickets to analysts who are part of the current team. Hence, we removed tickets that were assigned to analysts, who no longer work with the team.

The ticket title is vital to decide an analyst who should be assigned a ticket. However, the ticket title required cleaning, as it had other redundant, noisy information, which would not help with a recommendation. To remove these words, the title was passed through a series of regular expressions to remove unwanted words. Below screenshot which shows some ticket titles. As can be seen, tickets have information such as dates (03/13) & codes (PO, PA) that are not useful to make recommendations.



*Figure 4: Tickets with Title & without category item.*

Based on my analysis of the various tickets, a strategy was formed to merge data from these systems. The below diagram & the summary that follows it explains the same.



*Figure 5: Pre-Processor Data Flow.*

RITM number *(a secondary identifier for a ticket)* was the only common column, between tickets present in SM9 & Service Now systems. RITM number is created for all tickets present in Service Now, however the same cannot be said for tickets in SM9. We used RITM to merge tickets so that we have all the information about a ticket.

There were tickets in SM9 that did not have RITM number. This was possible for tickets created through external systems, such as Access Manager & others, which are not integrated with Service Now. Hence, we had to divide tickets in SM9 into 2 parts:

1. Tickets which have RITM number

2. Tickets without RITM number.

Tickets which had RITM number were merged with Service Now. The category item column in Service Now, tells us the ticket type, which was only present for tickets in Service Now. This was a critical feature to classify tickets, as it was better organized & well structured.

However, on merging SM9 & Service Now tickets based on RITM we found some tickets in SM9 were not present in Service Now. As a result, for such tickets, we did not have a category item. These tickets, which were without category item & the SM9 tickets without RITM, were combined together & we decided to use the ticket title to make recommendations for such tickets.

Now, we had all the information needed to make recommendations. Two estimators were created. One to give recommendations for tickets having category item & another to make recommendations based on title.

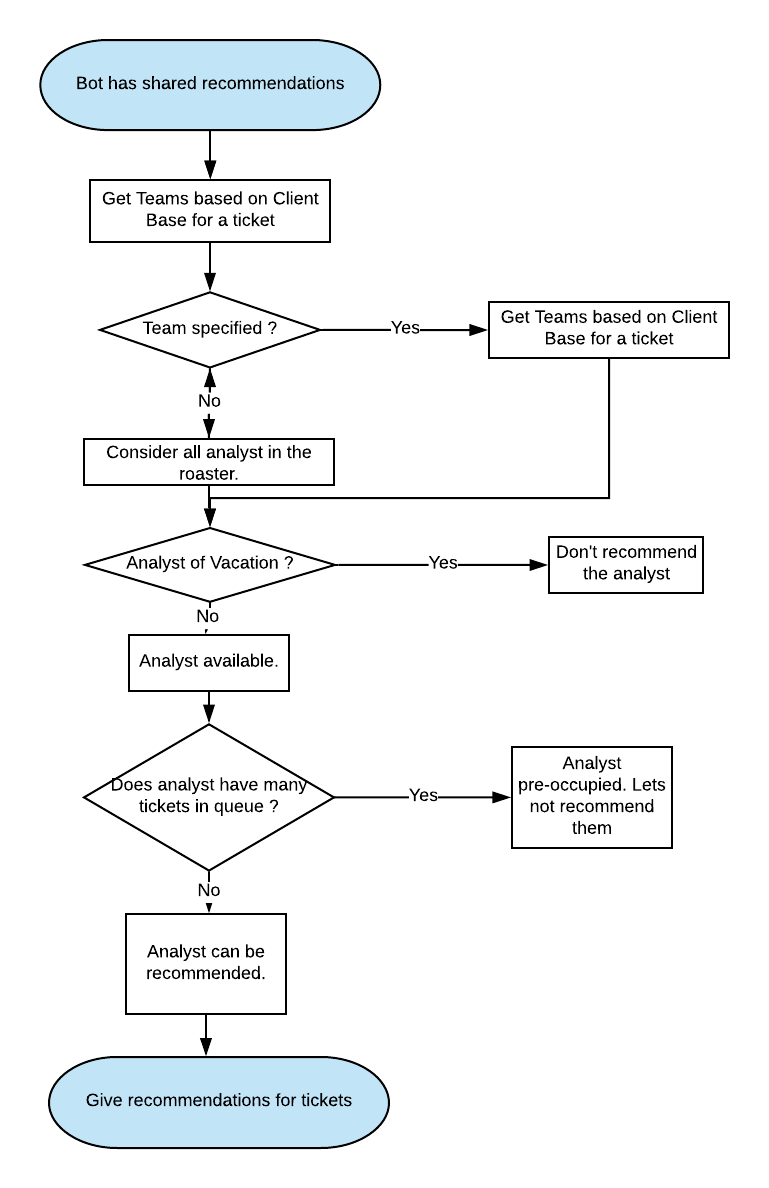
### Model / Estimator

The features *(category item & title)* were processed to remove stop words, the words were stemmed, & bi-grams were derived & count vectorized to generate features for our estimator. The analyst who was assigned the ticket was used as labels. The model was trained over these features using different estimators & hyperparameter values to create a model. We get a prediction probability from these estimators & we map these probabilities to the analyst *(classes/labels)*. These probabilities are then sorted so we can recognize analyst most capable to resolve a ticket.

We found Multinomial Bayes to give best results, over all other classifiers.

### Post-processor

Once the bot has predicted the probability of an analyst to resolve a ticket, we run the post-processor pipeline to refine our results to give useful recommendations. The below flowchart shows the post-processor pipeline, along with a short description of each step. The post-processor is executed, once the estimator has given prediction probability of each analyst capable of resolving the ticket. These probabilities are sorted in descending order & are considered as T-Bot’s recommendations.



*Figure 6: Post Processor Pipeline*

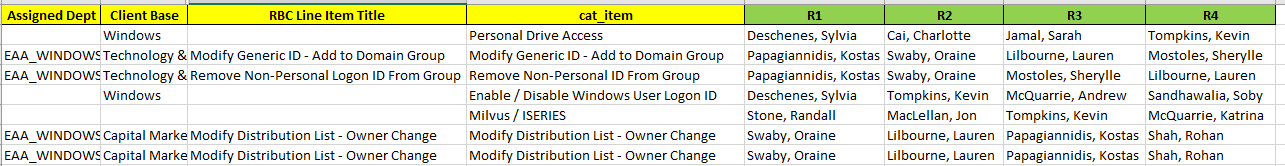
We use the Client Base column in a ticket to recognize teams within ESAM Operations that are expected to work on a ticket.

ESAM Operations has several teams, with analyst assigned to work on a different team, every month. The teams change monthly, to facilitate cross-training among analyst. So, even though the estimator would recommend an analyst who has indeed resolved a ticket many times, this may not be acceptable, provided the analyst is assigned to work for the team meant to resolve such a ticket. As part of this process, we get all analyst that could be considered for recommendation.

The vacation schedule of each employee tells us whether an analyst considered for recommendation is unavailable. This is important, as you don’t want to recommend an analyst who is not working.

The tickets an analyst is working on is provided separately. This file is processed to find the active number of tickets an analyst is working on any given day. This is used to decide whether an analyst should be recommended to resolve tickets. The rationale is if an analyst is already assigned several tickets, then there’s a high chance the analyst would need more time, as compared to an analyst who has fewer tickets assigned & has resolved similar tickets in the past.

The final set of recommendations are then set for each ticket, which is exported as an excel file. Below is a screenshot which shows recommendation (R1, R2, R3, R4) for some tickets.



*Figure 7: Ticket Recommendation*

### Evaluation

To evaluate the model, the ranking on the analyst recommended was compared with the analyst who was assigned the ticket. It was found that about 85 % of the time the top 4 recommendations made by T-Bot matched with the analyst who was assigned the ticket. This gave us confidence, that the recommendations made were useful.

### Package

The code to pre-process, estimate, post-process & evaluate was all part of a Jupyter notebook. These notebooks, although convenient to code, are not readily deployable. Hence, a python package was created to modularize the code across several python modules. A separate directory was created to store data required to train, post-process & make recommendations. The package is hosted as a web application on Flask, which will be described in the section below. The code is deployed on remote cloud instance & can be found at */home/vcap/app*. Following the package structure for Tbot.

Ticket\_Management\_System

- src

- data

- postprocess

- recommendations

- sm9

- train

- test

- test\_archive

- serviceNow

- train

- test

- test\_archive

- templates

upload.html

complete.html

- static

images

* Logo.jpeg

css

* main.css

\_\_init\_\_.py

preprocess.py

postprocess.py

recommender.py

app.py

utility.py

path.py

requirements.txt

- jupyter

- preprocess.ipynb

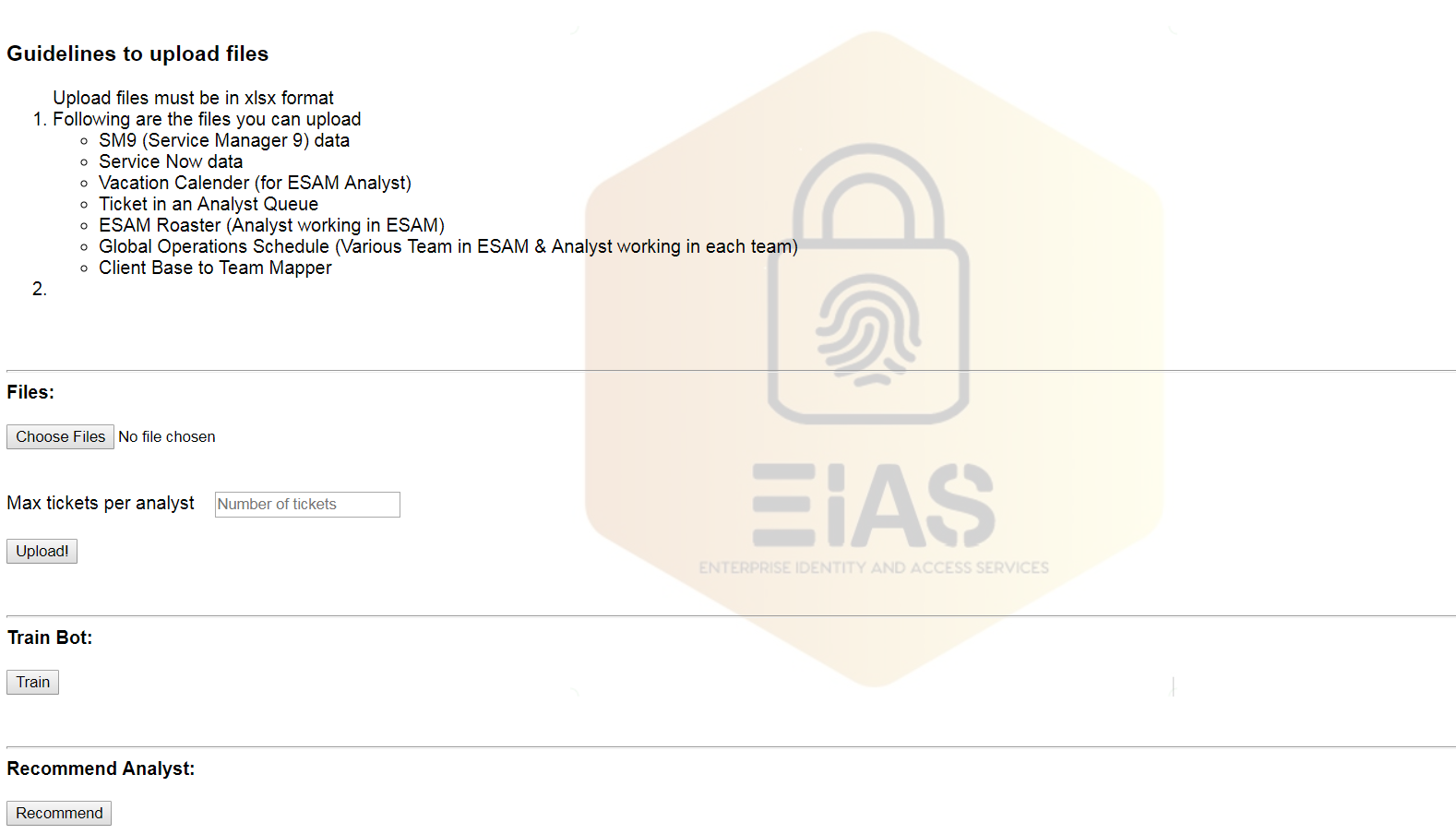
- Ticket\_Recommendation\_System.ipynb

- tms.ipynb

- post-process.ipynb

### Web Application

A web application is hosted on a Flask server, which supports microservices written in Python. This enables our clients to upload files, train & get recommendations through a simplified web interface. Below is a screenshot of the web application in Chrome.



*Figure 8: T-Bot Web Application*

We now have a system which gives useful recommendations for each ticket. However, the system did not have a client-side interface. Using the T-Bot requires technical expertise, which my clients are not familiar with. So, we decided to create an interface that would allow them to upload tickets they seek recommendations for, along with necessary files required by post-processor *(vacation schedule, teams, open tickets per analyst, team list)*.

To ensure the recommendations are meaningful going forward, I created a web application which would help my client to:

* Upload files & configure parameters *(maximum number of tickets to be assigned per analyst)*
* Train the T-Bot
* Get recommendations from T-Bot

We now have a system that has a web interface for clients to interact with. The interface is rudimentary & would be enhanced in future. The main goal of this interface was to provide a means for clients to get recommendations without any technical expertise.

### Deployment

The web interface by itself is not enough unless it is deployed in the cloud, to be readily accessible to clients. Hence, I collaborated with the cloud team, to have the application deployed on the Pivotal Cloud Foundry (PCF), RBC’s private cloud. The application is deployed in the Sandbox environment. It can be accessed using <https://tbot-nonreversed-koko.apps.pcf.devfg.rbc.com/>. After upcoming enhancements, it would eventually be deployed to Production. Further details can be on the applications deployment environment can be found through the [Cloud Manager](https://login.sys.pcf.devfg.rbc.com/login)

## User Guide

Some key points to be considered by clients while using T-Bot, especially before

uploading data for T-Bot.

All data files must be in .xlsx format.

* File names should follow a naming convention, to assist T-Bot to process a file.

Following are the naming conventions that must be followed.

|  |  |
| --- | --- |
| **File Type (All .xlsx)** | **File Names Starting With** |
| SM9 tickets | sm9\_train |
|  | sm9\_predict |
| Service Now | sn\_train |
|  | sn\_predict |
| Open Tickets per Analyst | otpa |
| Vacation Calender | Vacation Calendar |
| Global 2018 Operations Schedule | Global |
| Team List | Team List |
| Map Client Base to Team | Client\_Base\_to\_Team\_Mapper |

* Following are the columns names that should be present in the above-listed files

|  |  |
| --- | --- |
| **File** | **Columns** |
| SM9 tickets | Assigned Dept, Assigned to, Number, RBC Line Item Title, RBCMMPRITM, Rbc Description, Client Base |
| Service Now | cat\_item , number , u\_assignment\_group, state , stage |
| Open Tickets per Analyst | Assigned to, Pending Customer, Client Base |
| Vacation Calender | ALL sheet (Exactly same format) |
| Global 2018 Operations Schedule | MONTHLY SCHEDULE sheet (Exactly same format) |
| Team List | Name, Status, Employee #, Windows & Application (Line of Business) Skillset |
| Map Client Base to Team | Client Base, ESAM Application Teams. Using pipe / vertical bar ( | ) as a separator between teams. |

* T-Bot should be trained every 2 weeks, to ensure it learns from the most recent ticket assignments.

## Limitation

One of the limitations of the system is that it does not Automate Ticket Management. However, the same could not be achieved, as T-Bot would require real-time, on-demand access to SM9 & Service Now database, to recognize tickets which must be assigned, determine tickets assigned per analyst, to find the workload per analyst. Since we could not get real-time access, complete automation was not in scope.

## Future Work

Besides, fully automating ticket assignments, there are other enhancements that could be made to improve T-Bot.

* T-Bot error handling, validation & verification logic could be enhanced to interact with clients, so that they know what T-bot is processing. Also, if an invalid column is specified or required columns are missing, then the client should be clearly notified, along with steps to rectify it.
* An Email service, to provide recommendations via email.
* A skills-based estimator that recognizes the skills required to resolve a ticket & uses this knowledge to assign tickets to the analyst. Used to assign tickets to the new analyst, who don’t have a history of resolving tickets.
* Post-migration from SM9 to Service Now, python modules would have to be updated to ensure T-Bot is aware of the changes in the data source.
* Health Monitoring service to check the status of T-bot, so the clients are notified whenever there’s a downtime. T-bot must be deployed in Production to ensure high availability & fault tolerance.

## References

1. Python <https://www.python.org/>

2. Flask <http://flask.pocoo.org/>

3. SkLearn <http://scikit-learn.org/>

4. Pandas <https://pandas.pydata.org/>