Monthly Report

Oisín Hodgins

21/09/2020

1. Overview

The aim of this project is to generate a report for Econiq's customers, which will summarise the data collected by said customer and present it in a presentation style. This report should contain general overview statistics, such as the total number of calls, the average duration etc. as well as some more in-depth statistics to see if we can predict what factors most drive success in the business.

At current there are six major sections in the report;

a general overview of the month, an analysis of agent performance, an analysis of guided conversations and possible improvements going forward.

2. Architecture

2.1 Front-end

A homepage has been built in vue.js using node.js.

The user will upload a JSON file here that contains the raw data collected throughout the month, which will be used to generate a report.

This upload is done through a POST request.

2.2 Back-end

A http based RESTful API, designed using swaggerhub.io and operating on a java spring boot server, handles a POST request from the homepage, which uploads the JSON file.

A GET request from the homepage will execute an r markdown file, which generates the report.

The finished report is then stored in a public folder.

The address of this public folder is sent to the homepage.

2.3 Using R

2.3.1 R the Language, and R Studio

R is the coding language used to generate this monthly report, and R Studio is the interface I used during the development process. The r base language handles data in an untidy manner, so I chose to use the *tidyverse* package group throughout the design process. Graphing and plotting were done using the *ggplot2* package, statistics using both base R and the *aod* package, while the *formattable* package was used to display tables in the presentation.

2.3.2 R Markdown

R Markdown was used to generate the HTML presentation and allows the implementation of several other languages such as python and C++ if you are so inclined.

2.3.3 R Packages

I chose to create my own package to store the various functions I wrote, instead of writing all of the code inside the r markdown file (HTML Monthly Report.Rmd). This process allows us to simply call “library(<our package name>)” and we can access all the predefined functions in our presentation. Going forward I would highly recommend adding any new code to functions inside this package, the method of doing this is outlined briefly here, and in more detail here and here. Note, we do need to distribute this package or make it accessible to other R users in any way, we are simply organising code.

3. Statistics

3.1 Overview

The initial section of the report displays a general overview of the month, reporting simple call statistics such as the average duration of all calls, and the number of:

-Calls made

-Calls made with each outcome (Sale, Referral, Guided Conversation)

-New customers

-Customers served

Some bar charts are used here to visualise the outcome and duration variables.

3.1.1 R Functions (overview)

We use the read_csv_monthly_report() and find_total() functions in this section, as well as some ggplot objects. We make calls to several agent specific functions too; these are relevant in section 3.2.

3.2 Analysis of Agent Performance

We begin this analysis in the various *process_agent_data()* functions, where the relevant data is extracted from our core data set. We have some options in regards to outliers here, by default we only analyse agents who have a minimum of 10 calls, and at least 1 call of each outcome. These can be changed in the function call for *process_agent_data_calls()*, where the minimum calls value can be changed and the at least one call in each outcome requirement can be set to 'FALSE'.

3.2.1 Ranking Agents

Our goal here is to rank the agents numerically from best to worst, with the intent of identifying struggling agents who may need coaching or rewarding high performing agents. This particular problem belongs to the field of multiple-criteria decision analysis, and the method I have chosen to solve this problem with is goal programming.

First, we must consider our metrics, and we will divide them into three categories, based of our perception of their effect on agent performance.

The following metrics are seen to increase performance as they increase (directly proportional).

-The **total** number of **calls** made

-The proportion of **sales, referrals** and **guided conversations** that make up those calls

-The **rate of success** an agent has with **guided conversations**

-The average number of **red** clicks an agent has in their calls

The following metrics are seen to decrease performance as they increase (inversely proportional).

-The **AHT** (average handling time) of an agent's calls that belong to the *other* category

The following metrics give the best score when they are close to the mean.

-The **AHT** of an agent's calls not in the *other* category

-The average number of **blue**, **green** and **purple** clicks in a call (note: these are ranked against the average clicks in a successful call)

Now that are metrics are categorised, we can set about combining them. We use feature scaling here to standardise and normalise our data. First, we use z-score standardisation to transform our data, now an agent's score in each metric is shown as the number of standard deviations their value lies from the mean, where all positive values indicate an increase in performance. Next, we use min-max normalisation to transform the values onto a [0,1] scale where 0 is the worst value and 1 is the best value, and finally we average all metrics to find the agent performance score. This all takes place in the *agent_performance_analysis()* function, and is finally displayed in a HTML table through the *display_agent_performance_table()* function.

(Note: This is not an area I have studied before, and I am not wholly convinced this is a perfect solution. There may be a need to weight certain values, for example we could say an agent's sale percentage is twice as important as their referral percentage, thus we would multiply their sale percentage value by 2. There may also not be a need to perform min-max normalisation here at all. This stack exchange answer led me to multiple-criteria decision analysis and goal programming).

3.2.2 Visualising Performance

In this section we display a series of graphs to visualise agent performance across a number of categories, highlighting the number of calls made, proportion of calls belonging to each outcome, AHT and overall performance.

3.2.3 R Functions (Analysis of Agent Performance)

The following functions are used to create the *agent_data* table, which stores the relevant data for this section: *process_agent_data_calls(), process_agent_data_colours(), process_agent_data_duration().* The ranking of agents is implemented in the *agent_performance_analysis()* function, which makes calls to *max_standardise(), min_standardise()*, *mean_standardise* and *normalize().* The table itself is displayed in *display_agent_performance_table().*

3.3 Analysis of Guided Conversations

3.3.1 Logistic Regression

Question: What makes a guided conversation successful?

Here we are trying to predict a binary response variable, success (which is [0,1]) using a number of predictors such as the percentage of overall clicks made (Blue, Green, Red, Purple) and the duration of the call.

We use multi-variate logistic regression to complete this task. An explanation can be found here.

The Wald test is used to rank the various predictors in terms of their significance, and it's results are stored in the *guided_logit_stat_signif* table. This table reports each predictor's p-value, and whether or not the predictor is statistically significant (by convention we say a predictor is significant if its p-value is less than the significance level of 0.05).

We must also evaluate if our model is a good fit for the data, when compared to the null model. Here the null model is a model where we have no information about the predictors. If our model fits better than the null model we know at least some of our predictors are significant, however if our model does not fit better than the null model we must conclude our chosen predictors are not significant.

The method I have chosen to use here is a likelihood ratio test, which results in a p-value. This is performed in the *logit_reg_guided()* function, where the result of the test is stated. Here the p-value tells us with what

confidence we can say, “Our model fits significantly better than the null model”, with a p-value less than 0.05 we can be 95% confident with our statement, while a p-value less than 0.01 means we can be 99% confident.

3.3.2 R Functions (Logistic Regression)

The *process_guided_data()* and *eval_guided()* functions creates the relevant data set; *guided_data.* Then the *logit_reg_guided()* function performs the regression, and creates the *guided_logit_stat_signif* table.

3.3.2 Independence between agent choice and success

Question: Does the agent having a guided conversation, significantly affect its outcome?

Because agent choice (or agent username) is a categorical variable, it is not suitable to use the Wald-test from the previous section. Instead a Chi-Square test is used to investigate for independence between agent choice and the outcome of a given conversation.

Of note here is that a Chi-Square independence test is only accurate if there is a frequency of at least 5 in each cell. This means we need each agent to have at least five successful and unsuccessful guided conversations, if we want an accurate test. This is currently addressed in the r function *agent_guided_success_Chi_Square_test()*, where we check each agents logged calls and report any agents who have performed too few calls, then run the test with these agents regardless. A potential change here is to remove any offending agents from the test, and report however may agents are in fact being tested.

On to the test itself, first we state our hypotheses then the p-value is reported, along with a decision to reject or fail to reject the null hypothesis. (note: this is the unusual syntax conventionally used in hypothesis testing)

3.3.3 R Functions (Independence between agent choice and success)

The *agent_guided_Chi_Square_test()* function performs all the actions in this section.

4. Improvements going forward

4.1 Inter-Month comparisons

One possible improvement to consider, would be comparing two different months. I did not have the time to get around to this, however I can give an idea of how to implement it:

- Store the relevant data, by saving the agent_data, guided_data, agent_kpi_analysis tibbles in a file.
- Create an R script to compute the difference between two data sets, and store this in a file.
- Run the current "HTML Monthly Report.Rmd" file, but pass the new file to it. Some minor alterations would need to be made here, as the ".Rmd" file would need to read data that is formatted differently, and pass it to the functions and plots already defined.

4.2 Agent Performance Analysis Section

Should some metrics be weighted here? This could be applied prior to standardisation and normalization or after, also perhaps there is no need to normalize the data here.

4.3 Sale/Referral Specific Analysis

What makes a given call a sale, or a referral?

Can we rank what metrics best predict success?

If this is the case, logistic regression may be the best solution yet again. One method could be to analyse only sales and referrals, where a sale outcome is considered a success and a referral is a failure. One could also analyse referrals only, to see what percentage of them resulted in a sale, and to see if this behaviour can be predicted. I am unsure if answering these questions would provide any useful insight, and I believe the best method to use would be logistic regression. This book is what I used to determine my methods, and I would recommend it if you decide to go forward in this direction.

4.4 Unused Variables

There are a few variables in the original data set (Econiq demo data) that were trivial, such as branch, district and market, and I chose not to include them. Perhaps these will be important metrics to analyse, I would suggest using bar charts to display the number of calls (and calls broken down by outcome) in each of these metrics. Perhaps it would be useful to break down average agent performance in each of these branches? If I were to implement this, I would first append new columns to the core_data set, then I would create grouped bar charts (more info in useful notes section 5).

5. Useful Notes and Resources

5.1 R Notes

With HTML presentation in rmarkdown:

-Use message() for printing text and cat() for printing text with variables in (like sprintf() in C).

` -Plots in ggplot2 require their own slide for formatting reasons, also creating them in functions and saving them as objects, then later calling them to display has caused bugs for me.

-Dont use "%" in variable names if possible, all tidyverse functions will require to surround the variable in `backticks` and knitr will throw an invalid UTF-8 error.

- One line of white space is needed on either side of a slide title

Some R package stuff:

- When testing the package, use devtools::document() to update the functions, then load_all to get it into the global environment {NOTE: working directory must contain the package}

- When finished with the package, or you want to use library(<your package>), you must set the working directory to the one which contains your package, and then devtools::install("<package>")

5.2 Useful Resources

SwaggerHub, where one can design an api using their helpful interface, and then export it elsewhere.

The rJava package for R, which allows some interface between Java and R. I never used it in the end, however it could be useful in running r on the server.

A good book for statistics in r (this is also linked in section 4.3)

A good book for using r markdown, and also a nice cheat sheet which summarises it.

A good book for using r in general, as well as using the tidyverse package.

The vignette for Roxygen2, which is used to document any functions inside a user created package.

The vignette for formattable, which is used to display tables.

A guide on using colours with the ggplot2 package.

The R graph gallery, which shows off many types of plots and graphs, along with a guide to implementing them.

6. Installation Guide

6.1 Installing R

R Studio can be installed here, and the language itself can be installed here. You must first install R the language before you can use R studio.

6.1.1 R Packages

Packages are essential to using R, and can be installed easily by calling “install.packages(“<pkg name>”)” in the R console(R or RStudio).

The packages I used are: “tidyverse” (which includes dplyr), “aod”, “roxygen2”, “formattable”, “ggplot2” and “devtools”.

After a package is installed you must call “library(<pkg name>)” to use it, and this must be done in each new r studio session and every time r is restarted.

6.2 Other software

Here is a list of other software I used, and links to their downloads:

- Node.js
- Vue.js
- Visual Studio Code
- IntelliJ
- Spring Boot
- SwaggerHub