A Statistical Analysis of Riipen's Request Expiry Feature

Diving into how the request expiry feature affected user behaviour

Habiba Zaghloul - 1005566807 , Mark Vartolaș - 1005292354 , Iris Shtutman - 1005198778 , Karl Hendrik Nurmeots - 1005356730 - STA130 - TUT203 - Group ID:203-1

What is Riipen?

- Riipen is a Vancouver-based company that connects students, educators and employers.
- Employers can apply to work with professors and vice versa by sending “requests”.
- People were frustrated that their requests were not getting answered; they were often left with no response.
- Riipen implemented a request expiry date on November 21st, 2018.

Objectives

- How did the expiry feature affect response times?
- Did this allow for more requests/connections between users?
- Does the 14 day expiry limit make sense?

Hypothesis 1 - Volume of Requests

- The request expiry feature would allow Riipen to connect institutions and experiential learning employers faster.
- Unanswered requests wedge requestees into an unfavorable situation.
- A request acceptance deadline forces request recipients to act more promptly.

Test Question - Request Buffer Time:

The difference in time between any one user's consecutive requests, is influenced by the request expiry feature.

Formal Hypotheses: H_0: $\overline{b} - \overline{a} = 0$ (H_A: $\overline{b} - \overline{a} \neq 0$)

where $\overline{b}$ is the mean request buffer time for requests before the expiry feature implementation, and $\overline{a}$ is the same for requests after.

Data Wrangling - Overview

- Our primary source of data was the requests dataframe provided by Riipen. It includes data on requests made between April 12, 2018 and March 18, 2019.

Variables added, using **mutate()**:

Using the **if_else()** f'n, we added:

- **pre_expiry**, evaluating to: TRUE, iff the request was made before Nov. 21, 2019, and FALSE, iff the request was made after Nov. 21, 2019.
- We converted dates to **dmy** objects, using the **dmy()** functions and then subtracted the appropriate dates.

Methods - Testing Request Buffer Time

Data Wrangling - Setting up the Test

- Sorted the wrangled data frame in ascending order of *Actor.Id*, and then by *Date.Created*
- Used a **for loop** so that: if two adjacent obs were created by the same Actor.Id, we appended the difference in time between observations to a new variable (time_before). This gave us request buffer time.

Data Wrangling - Unusable Observtions

Used **filter()** to remove:

- requests sent on *November 21st, 2018*; there is no indication of the time of day that the feature was added at, and
- requests sent on *August 30th, 2018*; it was indicated by Riipen Product Owner, Emily Masching, that the Actor.Id variable went missing due to some unknown error for these observations.

Methods - Testing Request Buffer Time

Test Statistic: the difference in the means of request buffer times before the expiry feature was implemented, and after.

- We conducted a simulation under the null hypothesis that: request buffer time was not affected by addition of expiry date.
- We shuffled, using **sample()**, the labels of pre_expiry. Then, we calculated the difference of means for the new groups. This was repeated 10000 times, assuming H_0, using a **for loop**.
- Comparing the extremity of the test statistic to the distribution provided a p-value.

Hypothesis 2 - Request Response Time

- We expected response time to be significantly lower for requests made after the expiry time implementation.
- We expected a similar behaviour for subsets of the dataset, however we were curious to find out if there are any types of requests that were not affected, or if different subsets were affected noticeably more or less than others.

Formal Hypotheses: H_0**:** $\overline{b} - \overline{a} = 0$ (H_A: $\overline{b} - \overline{a} \neq 0$)

where $\overline{b}$ is the mean response time for requests before the expiry feature implementation, and $\overline{a}$ is the mean response time for requests after.

More Data Wrangling - Testing Response Time

A New Variable

- **response_time**, the number of days it took for a request to recieve a response (accepted/rejected/cancelled/expired). Because the issue was that people who made requests got frustrated from getting no feedback at all, we considered statuses of cancelled or expired as responses; they still gave feedback to the requester.

More Exclusions

- We used **filter()** to filter out requests, where state == "pending". These requests had not recieved a response yet.

Methods - Testing Response Time

Randomized simulation test

- We calculated true difference in mean response times for requests before and after expiry time implementation to get our **test statistic**.
- We conducted a simulation under the null hypothesis that: request buffer time was not affected by addition of expiry date.
- Our method of simulation was analogous to the test conducted in testing request buffer time.

Methods - Testing Response Time

Linear Regression Models

- The model uses the formula $Y_i = \beta_0 + \beta_1 x_i$, where x_i is equal to 1 if pre_expiry is TRUE, and false otherwise, and β_1 is the difference in average response time between requests with and without the expiry feature. β_1 is the intercept parameter. Y_i, gives us the mean response time for any arbitrary request made.
- We analyzed the value of β_1; this told us how much larger the mean response time for requests made before expiry time implementation was on average.
- **lm()** was used to create the model; **summary()** was used to find the optimized β_1 value.

Methods - Does The 14 Day Expiry Time Make Sense?

Classification Tree Creation

- We created variables **response_in_14**, which evaluates to “Yes” if a request recieved a response within 14 days and, false otherwise, and **response_type**, which is “Accepted/Rejected” if the request recieved a response of accepted or rejected, and false otherwise.
- We used 80% of the pre_expiry data for training, and 20% for testing.

Visualizations

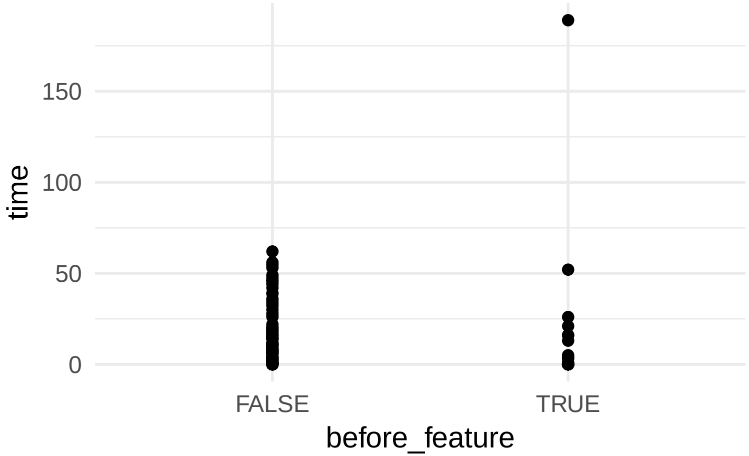
- Create a classification tree using **rpart()**, which predicts response_type based on response_in_14.
- Create a ROC curve for the tree using the **ROCR** package.

Results - Request Buffer Time

- After conducting the afforementioned hypothesis test, assuming our null hypothesis, that request buffer time remained stagnant after implementation of the expiry feature, we found a p-value of 0.043.
- The results of fitting a respective linear regression model, below, exhibit a decrease of approximately 6.79 days in request buffer time, after the expiry feature addition.

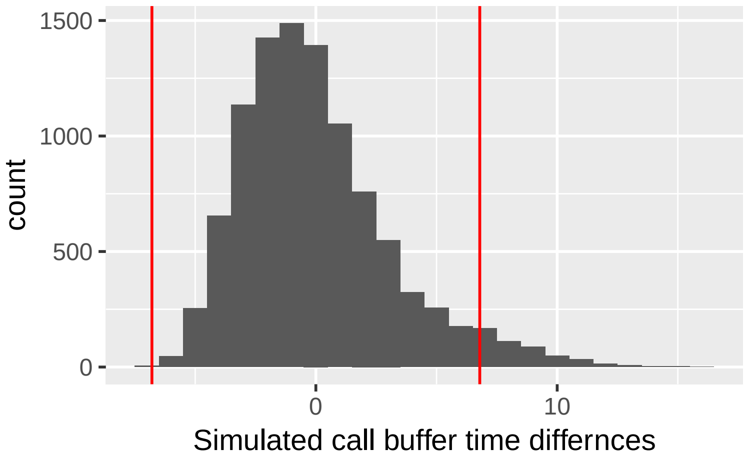
E	stimate S	td. Error	t value	Pr(>|t|)
(Intercept)	6.022654	0.8803973	6.840836	0.0000000
before_featureTRUE	6.792161	3.1057499	2.186963	0.0294384

Results - Request Buffer Time



- A plot of the model clearly conveys the afforementioned observation.

Results - Request Buffer Time



The figure above shows that simulations under the null hypothesis are rarely at least as extreme as our test statistic, of 6.79 days.

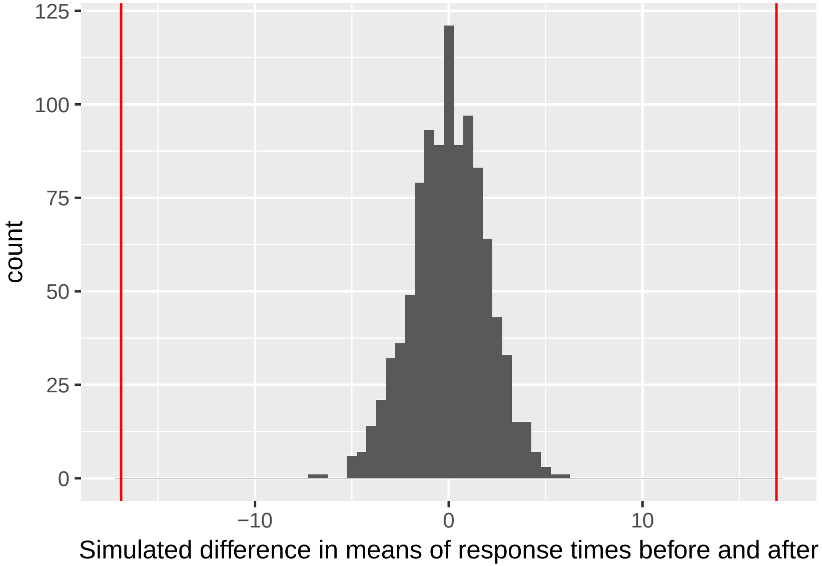
Results - Request Response Time

By doing the test for the entire dataset, we get a p-value of 0 assuming the null hypothesis of there being no difference in the means of response times for before and after expiry time implementation is true.

	Estimate	Std. Error	t value	Pr(>|t|)
(Intercept)	8.274725	1.467482	5.638724	0
pre_expiryTRUE	16.911230	1.851400	9.134293	0

The linear regression model shows us that the mean for response times was reduced by nearly 17 days after the implementation of expiry times.

Results - Request Response Time

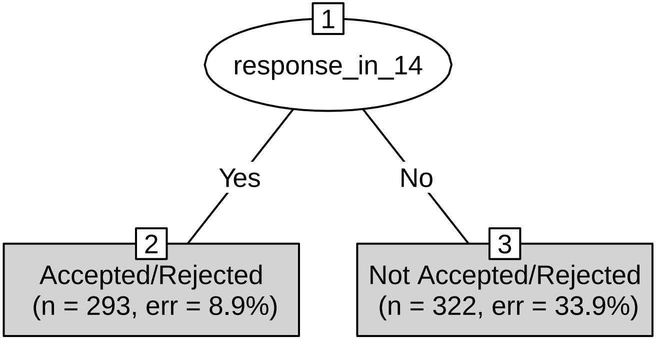


Results - Request Response Time

	p-value	Decrease in mean response time
Requests by educators	0	19.38942
Requests by employers	0	15.63062

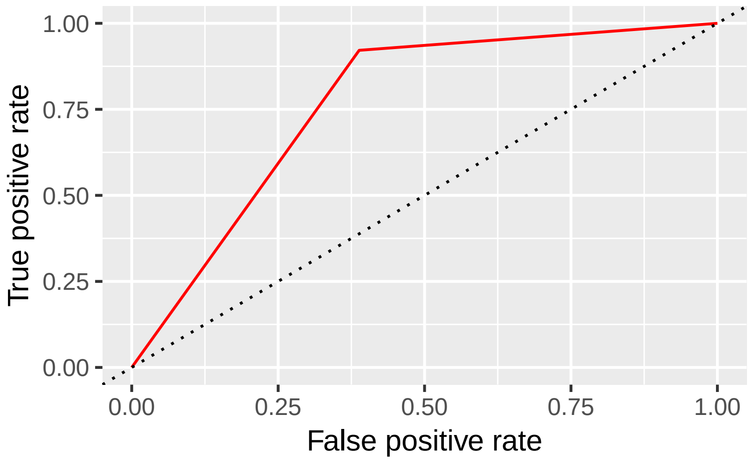
By separately filtering out requests made by educators and employers, we can redo the test and see how the response times for these requests changed. As shown in the table above, the implementation of expiry times affected response times for requests by educators more.

Results - Classification Tree



By using only a single predictor, response_in_14, we managed to create a very efficient classification tree. In fact, adding additional predictors did not change the structure of the tree.

Results - Classification Tree ROC Curve



Because the curve for the tree is close to the top-left corner of the graph, we can conclude that this tree is fairly accurate.

Conclusions

- We found moderate evidence against our null hypothesis, that request buffer time remained constant after implementation of the request expiry feature.
- This implies that we have significant evidence that the request buffer time remained constant after Riipen's implementation of the request expiry feature.
- With more requests being sent, Riipen further fulfills their place as a marketplace conncecting institutions with the workplace.

Conclusions

- Using randomization tests and linear regression we established that expiry times reduced average response times both for all requests, and for specific subsets of requests.
- For the given data, mean response time decreased from 25.18 days down to 8.29 days. Our linear regression model expects the mean response time for future requests to stay at around 8.27.
- Choosing to implement a 14 day expiry time is a logical decision, as shown by our classification tree - this is derived from the fact that the tree predicted requests that took longer than 14 days to recieve any sort of feedback to not be accepted nor rejected.

Limitations

- Users' activity differs with time of year (probably more active at beginning of school year), we do not have a full year's worth of data for requests made before and after expiry time implementation. (The analysis was observational, and we cannot conlcude causation in our tests.)
- For certain subgroups, there were very few observations (less than 15), which means the analysis is less meaningful and random chance plays a bigger role. Therefore the data for these specific subgroups is not be representative of the subgroup itself and the values we calculated do not tell us anything significant.
- The time of day the expiry date feature was launched was not specified so data required on that day could fall either with data before the feature was launched($<$ 2018-11-21), or after($>$2018-11-21).