Technical Report: Analysis of Repeat Callers Among Delta Air Lines Customers

Team vAIRiance: Prativa Basnet, [redacted]

Kennesaw State University

DS 7900 Spring 2024

Table of Contents

Problem Overview

In the aviation sector, customer service plays a crucial role in attracting and retaining customers. Customers present a range of issues to airlines, from flight cancellations to baggage challenges. Airlines are responsible for addressing these challenges and preventing them from recurring. Delta Airlines, a prominent company in this industry, is well known for their ability to create excellent customer experiences. However, as operations moved back to normal following the 2020 pandemic, Delta realized they may have a problem with excessive repeat calls from their customers. Delta encounters the issue of customers contacting them multiple times within a short period of time, potentially driven by lack of resolution about the same issue across multiple channels.

Repeated customer calls may indicate inefficiencies in problem resolution within Delta's Reservation sector, leading to potential delays and overall reduced satisfaction. These events can reduce specialist availability, distort the true demand picture, and hinder the efficient handling of inquiries from other customers. Our team thus aims to identify underlying factors in original calls that influence future repeat calls. Rather than assessing the traits of true repeat calls, we feel it is important to identify what happens in the original call that causes the customer to call back to address the issue at its root cause. With the identification of these traits, this analysis can inform decision-making at Delta Airlines and relieve volume pressure at specialist centers, leading to improved operational efficiency and enhanced customer experience. We will be specifically focusing on customers who experience both the IVR and specialist portion of a call to identify factors on both ends that influence future repeats. Likewise, we will also be assessing the potential differences in customer experiences between atypical and "blue sky" days.

Questions to Solve and Key Assumptions

Using call and messaging data accrued from 2023, our goal is to determine patterns related to repeat producing calls on a customer and specialist level, as well as specific customer intents that the Interactive Voice Response (IVR) may have trouble resolving. The IVR is designed to address caller concerns without agent intervention, yet it may not always achieve this goal effectively. If there are many customer intents that the IVR is unable to resolve, specialists will have a much higher call volume, which ultimately will cost Delta more money and reduce overall profit. We aim to identify specific intents that pose challenges for the IVR, shedding light on areas where its performance can be improved. Additionally, we seek to uncover patterns in repeat calls, exploring the demographics and behaviors of customers producing repeat calls compared to those not producing repeat calls. If specific customer types are identified that make repeat calls more frequently, Delta can target their specialist training to these groups and work to find areas of improvement for problem solving.

In our analysis, we operate under several key assumptions. We assume that if a customer calls multiple times in a 12-hour period about the same issue, that customer is making a repeat call. Furthermore, if a customer makes a repeat call, that indicates their issue has not been resolved. When comparing IVR and call intents, we presume the call intent will more accurately reflect the true purpose of a customer's call, as the IVR may misinterpret the problem at times. We also assume that the staffing to call ratio is consistent throughout the year, and that phone numbers who make more than 1,000 calls throughout the year are travel agents. These assumptions are necessary to identify, as they become important throughout our analysis and assist in reducing our scope.

Dataset Creation

Dataset Preparation

Prior to performing our analysis, we undertook essential steps to ensure the data was meticulously cleaned and prepared. This involved identifying anomaly values, removing potential outliers, and taking a representative sample of the data to use for analysis. Delta provided 3 datasets: calls, messages, and intents. Beginning with the calls dataset of 32.8 million calls, we started by dividing the calls into 4 groups based on IVR and specialist time: 1) calls with just IVR time, 2) calls with just specialist time, 3) calls with both IVR and specialist time, and 4) quickly abandoned calls. From this point, only calls from group 3 (IVR and specialist time) were kept, resulting in the removal of 12.8 million calls and a new sample size of approximately 20 million calls. Table 1 below indicates the conditions to assign groups and the number of calls in each group.

Table 1. IVR/Specialist Groups for Full Set of Calls

Group	Conditions	Number of Calls
1. Just IVR Time	IVR Time > 0, Talk Time = 0	12,801,906 (39%)
2. Just Specialist Time	IVR Time = 0, Talk Time > 0	14 (0%)
3. IVR and Specialist Time	IVR Time > 0, Talk Time > 0	20,040,418 (61%)
4. Quickly Abandoned Calls	IVR Time = 0, Talk Time = 0	503 (0%)

With the remaining 20 million calls, the next step was to make sure there were no duplicate call IDs in the dataset, as each observation should represent a unique call. With this assessment, 8 call IDs were found to be duplicated. These duplicate call IDs were removed, resulting in the removal of 16 total calls. The team then worked to identify and remove potential travel agents. As recommended by Delta, phone numbers with more than 1,000 calls throughout

the year were identified as potential travel agents and removed from the dataset. This resulted in the removal of 160,000 additional calls.

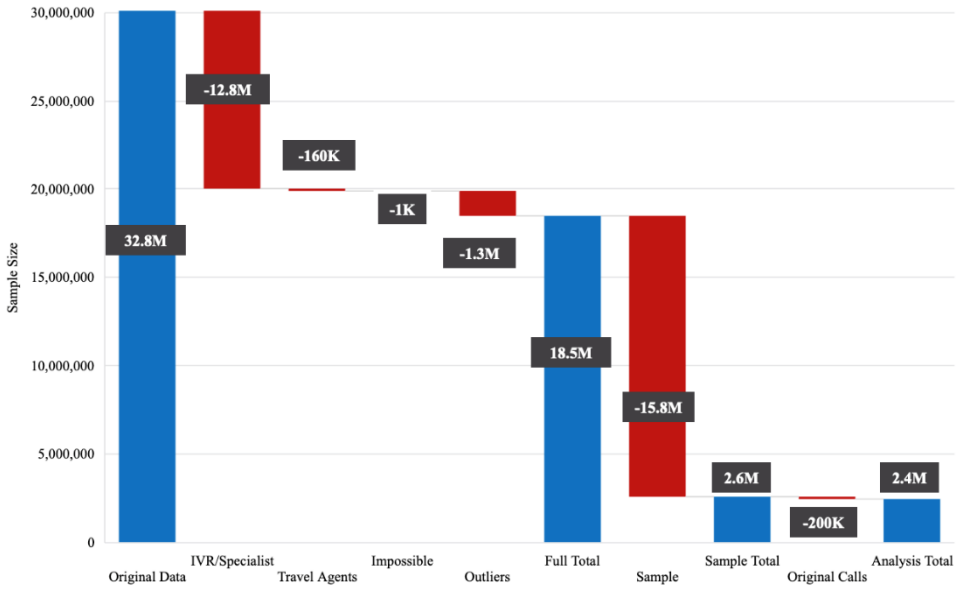
Our team's goal was to focus on the "typical" Delta customer experience; therefore, it was necessary to remove outliers. While assessing the data for outliers, we found several instances of "impossible" values that needed to be removed prior to identifying extreme, but realistic, values. These included customer age greater than 105, negative IVR time, and negative call handle time. Using these criteria, 1,144 calls were identified and removed from the dataset. Outliers were then assessed for the following variables: hold count, longest hold count, talk time, time spent working after the call (ACW time), specialist talk time, quoted wait time, IVR time, call handle time, interactive time, total customer wait time, initial queue wait time, support call queue time, and support call hold time. These variables were selected as they are all related to call time and experience, which could potentially be extremely high on atypical days. Since we wanted to avoid looking at calls that did not represent the typical customer experience, we removed any calls with a z-score greater than 3.5 for any of these variables. This resulted in the removal of 1.3 million calls. See table 2 below for additional information regarding the number of calls identified as outlier values for each variable (please note that multiple variables may identify the same call).

Table 2. Identified Outlier Values for Each Variable

Variable	Number of Calls Identified	Percent of Calls
Hold Count	353,763	1.78%
Longest Hold Count	252,469	1.27%
Talk Time	279,310	1.41%
ACW Time	97,122	0.49%
Specialist Talk Time	319,839	1.61%
Quoted Wait Time	192,132	0.97%
IVR Time	32,539	0.16%
Call Handle Time	296,279	1.49%
Interactive Time	279,310	1.41%
Total Customer Wait Time	185,904	0.94%
Initial Queue Wait Time	193,276	0.97%
Support Call Queue Time	271,576	1.37%
Support Call Hold Time	190,099	0.96%
Sample Size After Outlier Removal: 18,497,198		

Attempting to create the target variable and perform analysis using the full dataset was found to be extremely computationally expensive and unrealistic in the given time frame. Therefore, a representative sample was selected to use for the project's duration. The method decided on for sampling was a random sample of 1,000 phone numbers (roughly 10 percent of all phone numbers). It was important here to sample phone numbers rather than calls, as sampling calls could split up the series of calls from individual customers, therefore preventing appropriate analysis and identification of repeat calls. Following the sample, the final dataset sample size was 2,615,742. Later in the analysis process, following the addition of the “original”, “repeat flag” and “repeat after” variables, the dataset was further subset to include only original calls, resulting in a final analysis sample of size of 2,434,711. Additional information regarding the data preparation process order and the creation of the analysis datasets is outlined in Figure 1 below.

Figure 1. Waterfall Chart of Dataset Preparation Process



Derived Variables

Although many meaningful variables were provided by Delta, several additional variables were created to explore the issue of repeat callers on a deeper level. These variables were centered around our team's specific analytical approach and allowed us to gain more insight into the underlying factors surrounding repeat producing calls. Using both the calls and the messaging dataset, we created the following variables: new intent, message within 12 hours, multiple flag variables related to atypical days, shift, expired, and close to expiration.

The new intent variable was created due to excessive missingness for both IVR intents and call intents. Our analysis, specifically the creation of the target variable, centered around each call having a level 1 intent. We wanted to retain as many calls as possible, which required a combination of IVR and call intents to make a new intent variable for analysis. First, the call and intent datasets were merged, as call intents were provided by a third-party company. From this intent dataset, only call ID, level 1 intent, and level 2 intent were kept. This was done to ensure

each value was unique, as each call segment was represented in the intent dataset, rather than one observation per unique call as in the call dataset. This process did result in the loss of the sentiment score variable, since unique sentiment scores were provided for each call segment. However, we felt this was necessary as several of our calls did not have a level 1 call intent (15.8% of the sample), and therefore would not have any related sentiment scores. For calls with a level 1 call intent, their call intent became their "new intent". This type of intent was favored due to the assumption that intents assigned during the time talking to a specialist are more likely to be accurate. For calls with only an IVR intent, this IVR intent became their "new intent". For calls missing both intents, their "new intent" became either "Missing Base/NM" or "Missing HVC" based on their SkyMiles tier. Following this intent creation, the dataset was transposed so that each row represented one call, with binary variables for each intent variable to indicate all intents for all call segments. This process was crucial for the creation of the "repeat flag" variable, as our definition assessed matching intents between calls for any of the call segments. Additional information regarding the creation of this new intent variable, as well as the format of the transposed dataset can be found in Appendix A (Tables A1 and A2).

Delta also provided us with messaging data, allowing an additional perspective for the assessment of repeat contact and the different methods customers may use to resolve their problem. This dataset included messages from SkyMiles members, with both the time of the message and the related SkyMiles number. Since this was a separate dataset, we merged the calls and messages by the common SkyMiles number variable. We suspected that when repeat calls occur, the customer is also likely to contact Delta through messages to try and resolve their issue. With this information, we decided to identify calls that had a message 12 hours before or 12 hours after the call. Since not all callers made a message or were SkyMiles members (24.7% of

the sample), calls that did not have a related message were identified as not having a message. An example of the creation of this variable can be found in Appendix A (Table A3).

Next, multiple "atypical day" flags were created to assess the potential differences in repeat call trends on these identified days. These flags were created based on days with extreme average times (z-score greater than 2) for the following variables: total customer wait time, quoted wait time, hold time, support queue time, longest hold time, and call length. We felt these variables provided an important overview into extreme call occurrences and allowed us to assess these days in comparison to "blue sky" days. Table 3 below shows the number of days flagged for each variable related to atypical days. Calls on these days were not removed from the dataset, as we found that too many days would be lost due to different days identified by different variables. Likewise, the purpose of these identified days was not to exclude additional calls, but rather provide variables for the assessment of call experiences on "atypical" days.

Table 3. Number of Days and Calls Identified by Different "Atypical Day" Flags

Flag	Number of Days	Number of Calls
Total Customer Wait Time	13	79,625
Quoted Wait Time	15	90,497
Hold Time	6	44,179
Support Queue Time	14	95,342
Longest Hold Time	14	81,069
Call Length	8	44,914

Additional variables "shift" "expired", and "close to expiration" were also created to gain additional insight into the overall Delta call experience and the behaviors/traits of the customer on the call. The "shift" variable was created with 3 categories based on the time of the call, which was represented by 1st Shift (12AM-8AM), 2nd Shift (8AM-4PM), and 3rd shift (4PM-12AM). The "expired" and "close to expiration" variables were based on the given SkyMiles

expiration date and were therefore only present for SkyMiles members. The “expired” variable indicated whether the caller’s SkyMiles membership was expired (“yes” or “no”), and the “close to expiration” variable indicated the absolute value of the time until the caller’s expiration in seconds. These 3 variables gave important insight that extended beyond what was provided from the provided call variables and allowed the team to assess the potential differences related to impending expiration of SkyMiles membership or the time of the call.

Defining the Target Variable

In the context of this project, Delta wanted us to identify the underlying reasons for repeat calls from their customers and offer potential solutions to this problem. Before starting the assessment, it was necessary to identify and create the target variable for repeat calls. Delta allowed us to create our own definition for the repeat call variable, as they wanted to see potentially new ideas for solving this problem. We decided to define repeat flags as multiple calls within a 12-hour period that had the same level 1 intent for any call segments. We felt it was important to require both a time and an intent constraint for a repeat call, as a customer could call multiple times about different issues, rather than calling multiple times because their existing problem was not resolved. This is a generally strict definition of a repeat call; however, this definition allowed us to ensure that only real occurrences repeat calls were analyzed.

For the target variable creation, there were 3 main variables that needed to be created to assist with identifying repeat callers. These variables included 1) Original Calls: Indicating the original call for each phone number in each 12-hour segment, 2) Repeat Flag: Indicating whether the specific call was identified as a repeat or not based on multiple calls with at least 1 matching intent in less than 12 hours, and 3) Repeat After: Indicating whether a repeat call follows the original call. These variables were coded in SAS for phone numbers with 1 or 2 calls, however,

Python was necessary for more in depth programming of phone numbers with more than 2 calls. The code for both sets of target variable creation can be found in Appendix B. In our analysis, the repeat after variable serves as the target, as understanding what happens in the original call to create a repeat call is critical to understanding how to reduce them. Therefore, the data is subset to only include original calls. Examples of these variables' creation both before and after excluding non-original calls can be found in Tables 4 and 5 below.

Table 4. Creation of "Repeat After" Target Variable

Phone Number	Call ID	Call Start	Intent	Original	Repeat After	Repeat Flag
1	A	Jan 1: 2AM	SkyMiles	yes	yes	no
1	B	Jan 1: 6AM	Baggage	yes	no	no
1	C	Jan 1: 10AM	SkyMiles	no	no	yes
2	D	Feb 4: 12PM	Ticket	yes	no	no
2	E	Feb 4: 5PM	Support	yes	yes	no
2	F	Feb 5: 3AM	Ticket	yes	no	no
2	G	Feb 5: 4AM	Support	no	no	yes

Table 5. Example Dataset of Original Calls with "Repeat After" Target Variable

Phone Number	Call ID	Call Start	Intent	Original	Repeat After
1	A	Jan 1: 2AM	SkyMiles	yes	yes
1	B	Jan 1: 6AM	Baggage	yes	no
2	D	Feb 4: 12PM	Ticket	yes	no
2	E	Feb 4: 5PM	Support	yes	yes
2	F	Feb 5: 3AM	Ticket	yes	no

Data Discovery

Univariate Analyses

Prior to analysis, several variables were assessed to determine their overall distribution and degree of missingness. Assessing distribution for variables allowed improved understanding of typical values related to Delta calls and customer experience. Likewise, assessing missingness was crucial prior to building models so that variables could be included or excluded as needed. Variables related to customer demographics, specialist demographics, and overall call experience were assessed including customer age, specialist tenure, call length, total customer wait time, and quoted wait time. Table 6 below outlines the distribution of each of these variables. All numeric variables were also assessed for missingness, and the variables with the most missingness (excluded from complete case models) are outlined in Table 7. Missing categorical variables were recoded to maintain as many as possible, however, some variables were removed due to severe imbalance of categories (Example: "Call Type" variables with 99.7% of calls being inbound). Since the target variable "repeat after" was created, we could now assess the distribution of repeat producing calls, as well as repeat calls in general. Here we found that 6.9% of calls were identified as repeat calls, with 6.5% of original calls identified as producing future repeat calls. This is a critical finding for our analysis, as we found that repeat calls were relatively rare, creating a severely imbalanced dataset.

Table 6. Distribution of Selected Customer, Specialist, and Call Variables

Variable	Minimum	Maximum	Mean	Median	Standard Deviation
Customer Age	0.00	105.00	51.64	52.00	14.74
Specialist Tenure	0.00	48.00	5.62	2.00	8.80
Call Length (Min)	0.02	76.63	10.86	7.40	10.34
Total Wait (Min)	0.00	75.65	10.05	5.12	12.12
Quoted Wait (Min)	0.00	68.55	10.87	6.02	12.09

Table 7. Top Variables with the Most Missingness

Variable	Number Missing	Percent Missing

Million Mile Tier	2,154,555	88.5%
Survey Response	1,995,582	82.0%
Expired/Close Expiration	602,500	24.7%
Customer Age	574,499	23.6%
SkyMiles Tenure	562,772	23.1%

Bivariate Analyses

The next step in data discovery was to explore different relationships between variables, specifically how different variables were related to the "repeat after" variable. Since our analysis's main purpose was to identify underlying characteristics related to calls that produce repeats, it was necessary to perform preliminary assessments from various perspectives. Here, we performed preliminary analyses on average total wait time, average quoted wait time, average call length, SkyMiles tiers, and intents as they relate to calls that produce future repeats.

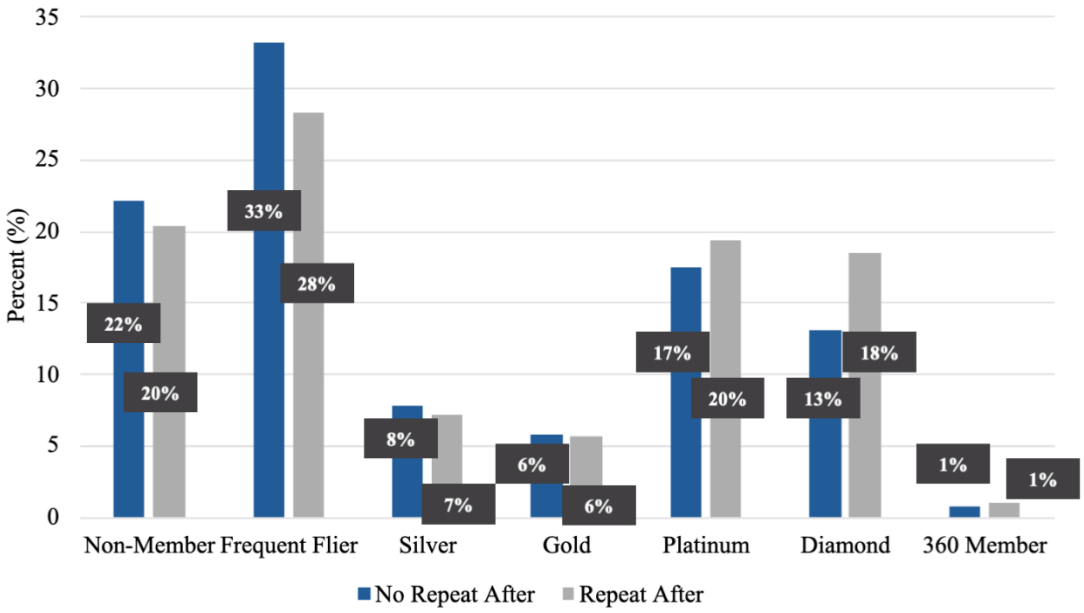
The assessment of total wait time, quoted wait time, and call length were essential to understanding the different experiences of customers who later produce repeat calls, and those who do not produce repeat calls. The average of each of these variables is outlined below in Table 8. Here, we found that calls producing repeats had a lower average wait time, quoted wait time, and average call length. Although not an extremely large difference, this finding was interesting and suggested that those who reached a specialist quicker (possibly those with a higher SkyMiles tier) may be more likely to make a repeat call. This initial finding will be further explored throughout the modeling process.

Table 8. Call Time Assessments for Repeat After vs. No Repeat After

Variable	Repeat After	No Repeat After
Average Total Wait Time	7.8 minutes	10.2 minutes
Average Quoted Wait Time	9.0 minutes	11.0 minutes
Average Call Length	9.6 minutes	10.9 minutes

Delta's SkyMiles tiers are central to their operations and reputation, which offers a unique perspective to examine the problem of repeat calls from different customer groups. Figure 2 below depicts the distribution of “repeat after” and “no repeat after” for different SkyMiles tiers. Here it is found that the majority of repeat producing calls are made by Frequent Fliers (33%), however, Diamond members have a disproportionately high number of repeat producing calls, with repeat producing calls being 5% higher than non-repeat producing calls.

Figure 2. Distribution of “Repeat After” by SkyMiles Tiers



Our team hypothesized that the purpose of the call may be influential in determining whether an original call will produce future repeat calls. To assess the accuracy of this hypothesis, it was first necessary to look at the top intents for calls producing repeats and calls not producing repeats. Ideally, we were hoping to find intents that were common in one group and not the other. Figure 3 identifies the top 5 intents for both repeat-after and no repeat after,

indicating that the most common intents are the same for both groups. Although there are small differences in frequencies, assessments of phi correlations (Table 9) indicate no relationship between these intents and the repeat after variable, as all correlations are very close to 0.

Figure 3. Top 5 Intents for "Repeat After" and "No Repeat After"

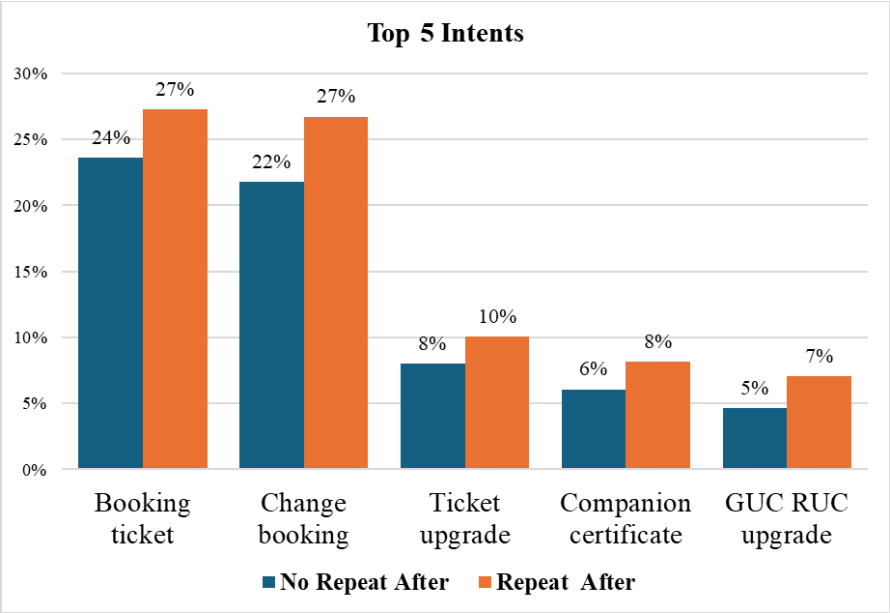


Table 9. Phi Correlations for Top Intents and Repeat After

Intent	Phi Correlation
Book a Ticket	0.02
Change Booking	0.03
Ticket Upgrade	0.01
Companion Certificate	0.02
GUC/RUC Upgrade	0.03

Logistic Regression

Correlations between variables were calculated with correlations greater than 0.7 reported in Table 10. Highly correlated variables were removed before modeling and are highlighted in blue. Continuous variables were discretized into five equal length bins due to the linearity assumption being violated.

Table 10. Pearson Correlations for Continuous Variables

Variable 1	Variable 2	Correlation
tot_wait_flag	qtd_wait_flag	0.78
Lngst_Hld_Cnt	Hld_Cnt	0.73
Hndl_Sec_Ct	Tlk_Tm	0.88
Intr_Sec_Ct	Tlk_Tm	1.00
Support_Call_Cnt	Spclst_Tlk_Tm	0.83
Spclst_Tlk_Tm	Spclst_Cnt	0.83
Support_Call_Cnt	Spclst_Cnt	1.00
Intr_Sec_Ct	Hndl_Sec_Ct	0.88
Tot_Cust_Wait_Time	Initial_Queue_Wait_Time	0.93
Frst_Callback_Ts_ind	Callback_Cnt	0.92
Callback_Cnt	callback_accpt_ts_ind	0.97
callback_accpt_ts_ind	Frst_Callback_Ts_ind	0.98

Logistic regression was used to calculate odds ratios and marginal probabilities to identify potential variables strongly contributing to varying levels of probabilities of repeat calls occurring. Stepwise Akaike Inclusion Criteria was used to identify variables significantly contributing to the model with 36 being identified.

The LOGISTIC procedure in SAS was used to calculate odds ratios and 95% Wald confidence intervals. A large majority of categorical levels were identified with significant odds ratios with the highest six odds ratios reported in Figure 4 and the five lowest reported in Figure 5. The intent with the highest odds was Flight Delayed with an OR estimate of 1.81, while the

intent with the lowest odds was Flight Status with an OR estimate of 0.31. Out of the most frequent intents, Change Booking had the highest OR estimate at 1.59.

Figure 4. Odds Ratios for the Six Highest Increased Odds of a Repeat Call

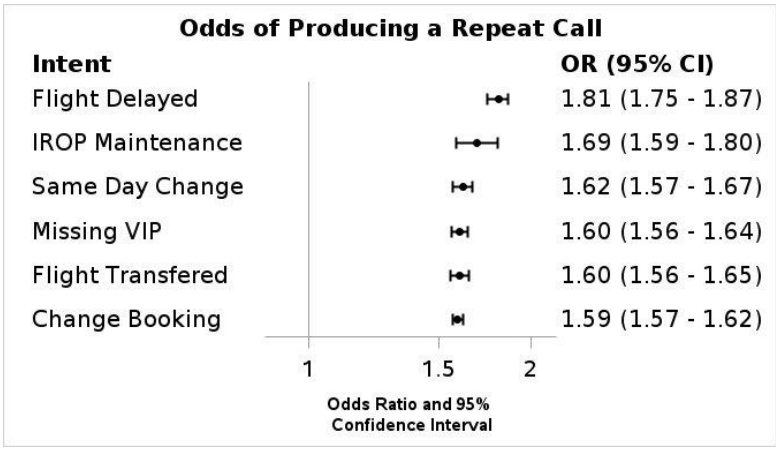
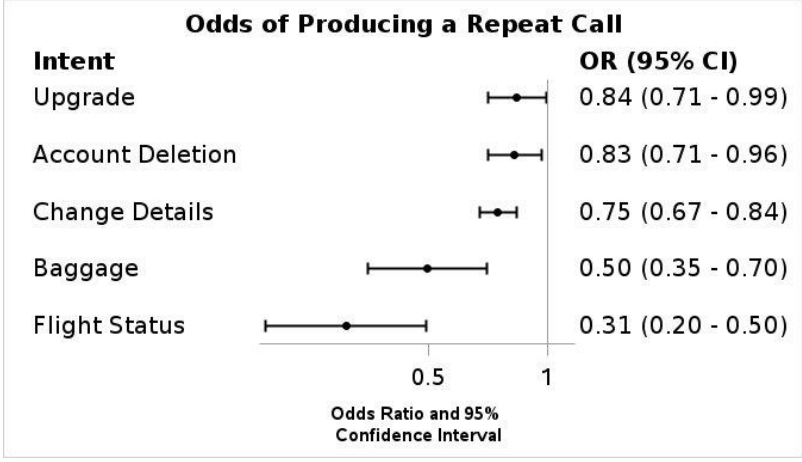


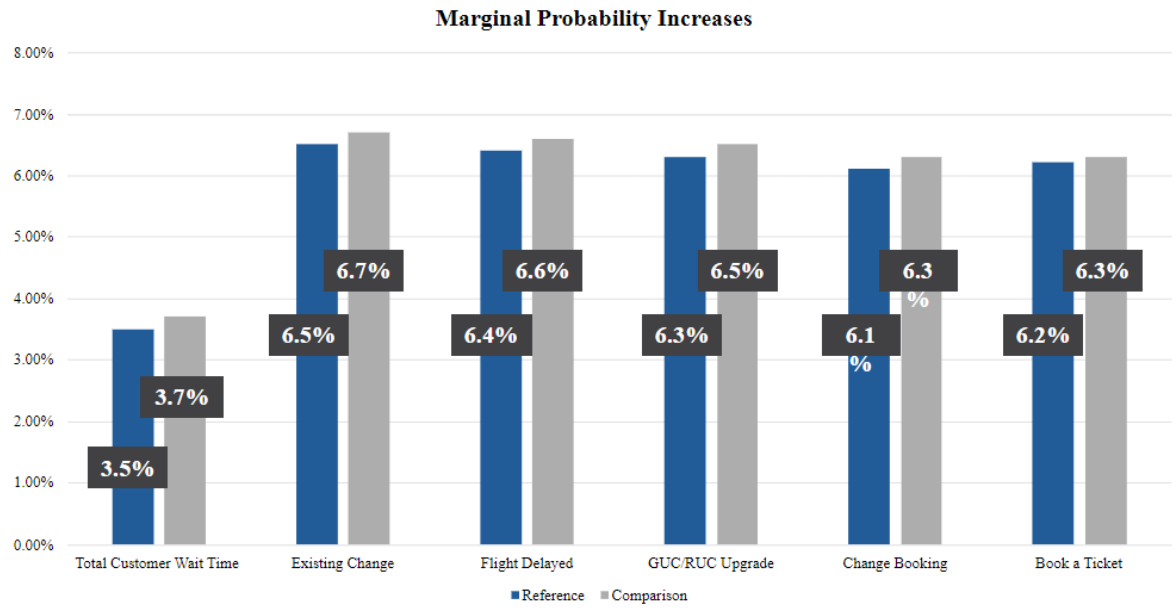
Figure 5. Odds Ratios for the Five Highest Decreased Odds of a Repeat Call



When dealing with small probabilities, marginal probabilities help create a more encompassing picture of how impactful these intents may be. These marginal probabilities were calculated using the QLIM and MEANS procedures with the largest increase in probabilities reported in Figure 6. The two most frequent intents, Change Booking and Book a Ticket, saw

marginal probability increases of 0.2% and 0.1% respectively. Flight Delayed, the intent with the highest odds ratio, saw a marginal increase of 0.2%.

Figure 6. Marginal Probability Increases



XG Boost Modeling

To further investigate the presence of predictive patterns in the data that would allow us to reliably predict whether a current call would be followed by a repeat call, we used an Extreme Gradient Boosting (XGBoost) predictive model from the tree models family. The motivation driving the choice of XGBoost was that XGBoost is currently the dominating predictive model for tabular data both in competitive data science (such as Kaggle competitions) and corporate data science. Another motivation is that XGBoost can natively resolve many issues that are persistent in other approaches such as automatically handling missing values, preventing overfitting, etc. Our main assumption was that if a predictive pattern existed in repeat call

dataset, XGBoost would have detected it even if the pattern was weak, and then we would proceed with further optimization and tuning of the model.

The first model we built made an attempt at predicting repeat calls in the entire dataset based on all available variables. We call it “global” model. This model did not yield acceptable performance even when imbalance of repeat calls was directly addressed in the XGBoost model, as shown in tables below.

Table 11. Confusion Matrix of Global Model

	Predicted calls with no repeat after	Predicted calls with repeats after
Actual calls with no repeat after	664,534	17,200
Actual calls with repeat after	44,808	2,480

Table 12. Performance Metric of Global Model

Accuracy	0.91
Precision	0.13
Recall	0.05
F1-score	0.07
Area under curve	0.60

Based on the low performance of the global model, we hypothesized that the predictive pattern might be weak and might be dominated by noise from other features. The next step of investigation assumed that some intents might be stronger generators of repeat calls, so we decided to subset the data into separate datasets with only one unique intent present. For

example, we would subset only calls that have the intent “luggage”. Creating such intent-specific datasets and then predicting repeat calls within each dataset was expected to improve modeling results.

We had three datasets for three groups of customers – NM, BASE and HVC. The NM and and HVC datasets had 48 intents, while the BASE dataset had 49 intents present. Therefore, we build 145 XGBoost models for all possible combinations of dataset and intent as illustrated below:

Table 13. Dataset Combinations Illustrated

Dataset	**Intent**
HVC	luggage
HVC	upgrade
…	…
…	…
BASE	book a ticket
BASE	refund

Each of these models was also addressed for its own specific imbalance of calls with and without repeat after, therefore our expectation was that at least some models would produce consistent predictions. For instance, it might not be possible to predict a repeat call for customers in BASE group if the call was driven by the “book a ticket” intent but it could have been possible

to reliably predict a repeat call for customer in HVC group if the first call was produced by "upgrade" intent.

However, all 147 XGBoost models did not yield predictive power that would be acceptable at production level. Below is an averaged performance report for all models:

Table 14. Average Performance Report

Average Accuracy	0.73
Average Precision	0.30
Average Recall	0.07
Average F1-score	0.11
Average Area under curve	0.54

These outcomes strongly suggest that there are no consistent patterns and relationships driving repeat calls even when trying to isolate the effect of each specific intent. Therefore, the conclusion would be that repeat calls are indeed driven by random factors which are impossible to control.

Random Forest Modeling

The Random Forest algorithm was performed on all SkyMiles members to assess the predictability of repeat calls. The dataset of SkyMiles members contained 1.9 million observations with 87 variables. The missing values were handled using a complete case function, and categorical variables were transformed into factors using the 'lapply' function. Furthermore,

variables with only one level were removed. The dataset was divided into training (80%) and testing (20%) subsets.

Given the highly imbalanced nature of the dataset, with repeat producing calls comprising only 6% and non-repeat producing calls comprising 94%, the majority class was under sampled in the training set to mitigate this imbalance. Hyperparameter tuning was conducted on the Random Forest to optimize the model's performance. The selected hyperparameters included Mtry(randomly selected predictors), split rule(gini), and Min.node.size (minimum node size). The model was evaluated using kappa as the primary metric, with 5-fold cross-validation.

The performance of the model was based on the testing dataset. Table 10 below shows the confusion matrix. The confusion matrix analysis indicates 13,000 true positives, 190,000 true negatives, 8,000 false negatives, and 122,000 false positives. The true positive and true negative represent correct prediction, while false positive and false negative denote incorrect prediction of the model.

Table 15: Random Forest Model - Confusion Matrix

	Actual: Repeat After	**Actual: No Repeat After**
Predicted: Repeat After	13K / 4% (TP)	122K / 37% (FP)
Predicted: No Repeat After	8K / 2% (FN)	190K / 57% (TN)

The performance result of the model is given in Table 11 below. The model achieved an accuracy of 60.9%. However, due to the significant class imbalance, with no repeat after calls

comprising the majority (94%), accuracy alone may be misleading. Therefore, other metrics such as kappa, precision, and F1 score were evaluated. These statistical performance metrics are relatively low indicating that repeat calls follow a similar pattern to original calls.

Table 16: Performance Metrics of Random Forest Model

Performance Metrics	
Accuracy	0.609
Kappa	0.061
Precision	0.095
F1 Score	0.164
Sensitivity	0.605
AUC	0.651

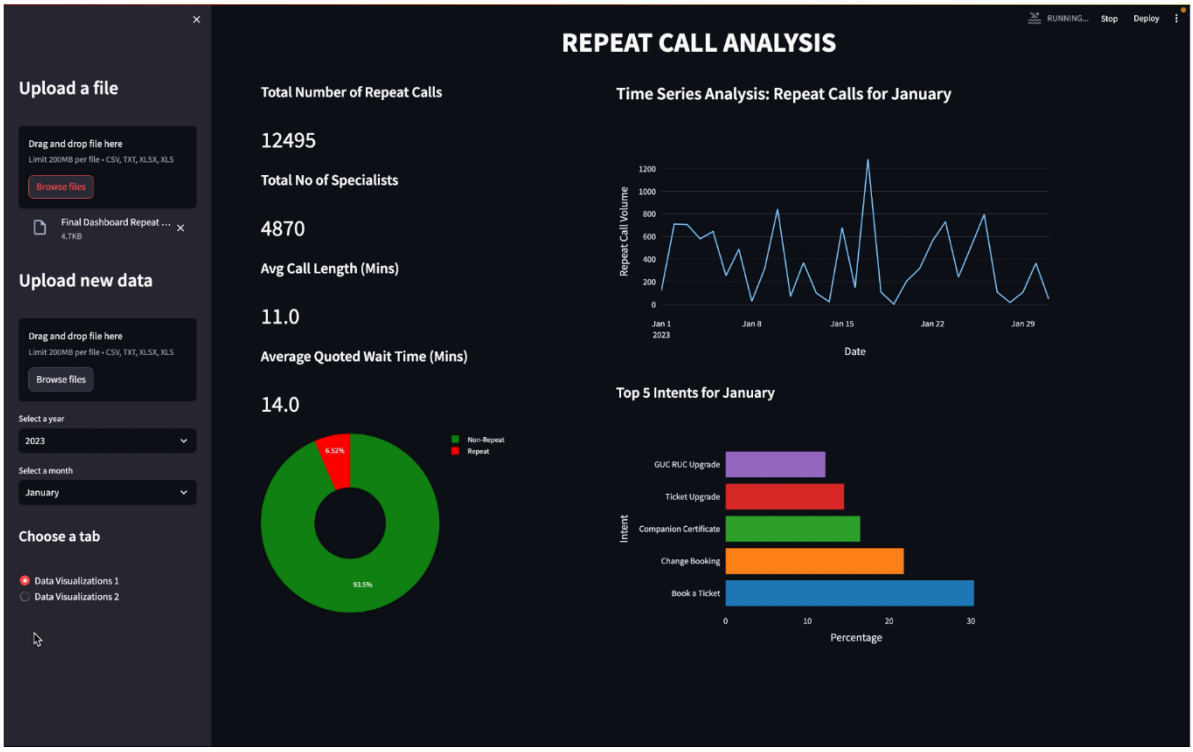
Dashboard Creation

As part of our analysis, we developed a comprehensive dashboard to integrate our key statistical findings into a centralized platform. By incorporating selected variables that were crucial to our analysis, our objective was to provide Delta employees and executives with a tool to monitor patterns related to repeat calls. This approach enables the Reservations Division to proactively identify and address issues concerning customer service, seasonal changes, and early problem detection.

The dashboard offers insights into the behavior of repeat call volume over time, the monthly proportion of repeat calls to non-repeat calls, and the fluctuating number of repeat calls in relation to the average call volume, quoted wait time, our newly created variables, shift and message, and the availability of specialists each month. Additionally, we examined repeat call behavior across demographics such as age, gender, and SkyMiles tier. This demographic analysis

allows users to effectively analyze patterns and trends in repeat calls, providing insights into customer segments more prone to repeat calls. This information can help identify areas for improvement in customer service or product offerings. These statistics were calculated in SAS to create representative synthetic data for the dashboard, and specific code related to these calculations can be found in Appendix B.

Figure 7. Example of Dashboard Statistics



To make the dashboard efficient for users, we implemented several key functionalities. Users can upload a CSV file, which loads the key statistical findings per month, separated into two tabs for easy navigation and organization. This feature enables users to track changes in repeat call behavior on a month-to-month basis. Additionally, the dashboard provides insights into the demographics of repeat callers, allowing for targeted strategies to improve customer

service. Furthermore, it offers real-time monitoring of call volume trends which could enhance operational efficiency and overall customer satisfaction.

Key Findings

Overall, our analysis revealed that calls producing repeats may be unpredictable, as they generally follow the same patterns as calls that do not produce repeats. Likewise, we found that repeat calls may not be as big of a problem as previously suspected, given that only 6.9% of calls are identified as repeat calls. While they may be a nuisance, there is some level of repeat calls that are expected in normal business operations, specifically when dealing with air travel.

Many bivariate relationships were assessed related to repeat producing calls. These include the bivariate relationships initially explored, as well as several analyses performed by month for the dashboard creation. In each example, variables were distributed relatively the same for calls producing repeats and calls not producing repeats and did not represent any impactful differences. To make sure we came to the correct conclusion, marginal probabilities were also assessed. However, even the largest marginal differences were negligible, indicating little variation between these groups of calls. While odds ratios for some variables appeared to be impactful at first glance, these marginal probabilities verified that the actual difference in probability between different variable levels was not practically significant.

Random forest and XG Boost modeling techniques were implored to assess the predictability of repeat calls, and the variables most related to these differences. However, even with several trials of these modeling techniques, the performance was very poor for both models. We found that each trial was producing different variables of importance, indicating that the predictive performance of these models was not reliable, and not appropriate for the current

problem. However, these results gave a new perspective that repeat calls may be a result of a variety of underlying characteristics that cannot be measured and are likely outside of both Delta's and the customers' control.

Our original hypothesis that intents may influence repeat calls was not found to effectively differentiate between repeat and non-repeat producing calls. The most common intents of calls that produce repeats are the same as the most common intents of calls that do not produce repeats. This indicates that the drivers behind calls that produce repeats seem to be the same as non-repeats, with no differences in patterns to distinguish between the two.

Business Recommendations

Based on our key findings from the data presented to us, we derived business recommendations that we believed would aid the Delta team in resolving their issue with repeat calls. We noticed that calls which led to future repeats, as well as calls that did not, were driven by similar intents, with no significant variation in the distribution of these intents between the two types of calls. This uniformity obstructs the creation of targeted strategies for reducing repeat calls alone. To address this, Delta can continue to monitor the behavior of calls.

Additionally, reducing the call volume overall will lead to a reduction in repeat calls. Delta can focus on implementing more self-service options when calls are made to reduce the likelihood of reaching a specialist and the need to call in for future common inquiries. From our analysis, the most common reasons for calls, including those that lead to repeat calls, are related to booking a ticket and changing an existing booking. The IVR system cannot process these requests, requiring callers to speak with a specialist. When trying to book a ticket over the phone, the IVR sends an optional text message or deflects to messaging with a specialist. However,

information provided to the IVR does not transfer over, leading to customer frustration and potential repeat calls.

To test the effectiveness of the IVR system, we called the Delta customer service number looking to make inquiries about flights. However, when offered a deflection, we were redirected to the Travel Planning Center, which we found to be difficult to navigate and not accessible on a phone. From this information, we theorized that customers may call back after a successful deflection to speak with a specialist due to this complicated process, which could potentially be avoided with additional IVR capabilities. Delta should enhance the accessibility and user-friendliness of the Travel Planning Center, making it easier for customers to navigate and find the information they need without the need for additional assistance.

Overall, we believe that Delta should consider integrating a system into the IVR that allows ticket purchases for SkyMiles members with saved cards and options to change bookings for all customers. This would streamline the booking process, reduce the need for customers to speak with a specialist for common requests and overall reduce the repeat call volume. Additionally, Delta can perform a cost analysis on providing an incentive, such as a small SkyMiles bonus, to see if this will encourage people to book through alternative means. This could help reduce call volume and incentivize customers to use self-service options, reducing the likelihood of repeat calls.

Appendix A: Additional Figures

Table A1. Creation of "New Intent" Variable

IVR Intent	Call Intent	New Intent
Present	Present	Call Intent
Absent	Present	Call Intent
Present	Absent	IVR Intent
Absent	Absent	"Missing Base/NM" or "Missing HVC"

Table A2. Example of Multiple Intents Per Call

Call ID	Baggage	SkyMiles	Book a Ticket	Child Ticket
A	0	0	1	1
B	1	1	0	0
C	1	0	1	0

Table A3. Creation of "Message Within 12 Hours" Variable

Phone Number	Event	Event Start	Message Within 12 Hours
1	**Message**	Jan 1: 2AM	
1	Call	Jan 1: 6AM	Yes
1	Call	Jan 2: 10 AM	No
2	Call	Feb 4: 12PM	No
2	**Message**	Feb 5: 2AM	
2	Call	Feb 5: 3AM	Yes
2	**Message**	Feb 5: 4AM	
3	Call	Feb 7: 2AM	No Message

Appendix B: SAS, R, and Python Code

SAS Code: Dataset Creation, Data Discovery, Logistic Regression, and QLim Modeling

```
libname archive "/gpfs/user_home/os_home_dirs/kdunca49/ds7900_spring24_delta/archive";
libname team4 "/gpfs/user_home/os_home_dirs/kdunca49/ds7900_spring24_team4";
/************************ DATASET CREATION ***********************/
/*****************************************************************/
/* create copy of calls dataset */
data calls;
set archive.allcalls;
run;
/* check call_id > 1 */
proc sql;
select call_id, count(call_id) as call_count
from calls
group by call_id
having count(call_id) > 1;
quit;
/* number of calls with tlk_tm > 0 */
proc sql;
select count(call_id)
from calls
where tlk_tm > 0;
quit;
/* 20040434 */
/*************************** NEW VARS ****************************/
/* new variables */
data calls_newvar;
set calls;
day_of_week = compbl(put(datepart(call_strt_gts), downame.));
hour = hour(call_strt_gts);
if 0 <= hour < 8 then shift = 1;
else if 8 <= hour < 16 then shift = 2;
else if 16 <= hour < 24 then shift = 3;
if SM_tier_eff_dt = . then expired = 99;
else if call_strt_gts - SM_tier_eff_dt > 0 then expired = 1;
else if call_strt_gts - SM_tier_eff_dt < 0 then expired = 0;
close_expire = abs(call_strt_gts - sm_tier_eff_dt);
if expired = 99 then expired = .;
drop hour;
```

```
run;
/* make an indicator for those who were in ivr and specialist */
/* ivr_sec_tm > 0 and tlk_tm > 0 */
data calls_newvar2;
set calls_newvar;
if ivr_sec_tm > 0 & tlk_tm > 0 then ivr_spec = 1;
else ivr_spec = 0;
run;
/* make an indicator for those who were only in ivr */
/* ivr_sec_tm > 0 and tlk_tm = 0 */
data calls_newvar2;
set calls_newvar2;
if ivr_sec_tm > 0 & tlk_tm = 0 then just_ivr = 1;
else just_ivr = 0;
run;
/* make an indicator for those who were only in specialist */
/* ivr_sec_tm = 0 and tlk_tm > 0 */
data calls_newvar2;
set calls_newvar2;
if ivr_sec_tm = 0 & tlk_tm > 0 then just_spec = 1;
else just_spec = 0;
run;
/* only 14 people have just specialist*/
/* make an indicator for those who abandoned quickly */
/* ivr_sec_tm = 0 and tlk_tm = 0 */
data calls_newvar2;
set calls_newvar2;
if ivr_sec_tm = 0 & tlk_tm = 0 then abandon = 1;
else abandon = 0;
run;
/* frequency tables for new variables */
proc freq data = calls_newvar2;
tables ivr_spec just_ivr just_spec abandon / nocol nocum norow;
run;
/* only keep people where ivr_spec = 1 */
data team4.calls_ivrspec;
set calls_newvar2;
where ivr_spec = 1;
run;
/* 20040418 */
/************************ DATA CLEANUP ***************************/
/* delete duplicate call ids */
data team4.calls_ivrspec;
set team4.calls_ivrspec;
if call_id = "12265c2a-5d61-47e3-acd5-255d58765437"
or call_id = "2158d7c9-36cb-4183-aad9-ae18015a3da8"
or call_id = "2a1cb013-4232-4f48-87ed-ffee59c3802f"
or call_id = "6316fa3e-4cf4-48fe-bb60-9c68f2aaa559"
or call_id = "91ae599c-2560-460a-97b7-3a3f10d890f2"
or call_id = "94f8708b-454d-4929-a9a2-6fc50aee90b3"
or call_id = "984c83ad-ade0-41d0-863a-07db2dd06754"
or call_id = "a0c7a123-fd84-4e52-9ac7-97b78bdf75fb"
or call_id = "afbced4c-337c-41fc-9790-2b784515736a"
```

```
or call_id = "d30a6d2e-0e6b-4288-8a44-55952c96b3ea"
or call_id = "d7a3a46f-b0a3-4de0-a23d-649885a448ec"
or call_id = "f5334c48-0e84-4827-8bdd-54853ecda5dc"
then delete;
run;
/* 20040401 observations */
/********************* REMOVING TRAVEL AGENTS ***********************/
/* create dataset for removing travel agents */
data calls_agents;
set team4.calls_ivrspec;
run;
/* table with number of phone calls made by each phone number */
proc sql;
create table PhoneNumberCounts as
select cust_phn_nm, count(*) as phone_count
from calls_agents
group by cust_phn_nm;
quit;
/* creating a table with an agent flag variable */
proc sql;
create table agent as
select *,
case when phone_count > 1000 then 1
else 0
end as agent_flag
from PhoneNumberCounts;
quit;
/* check flag count for agent */
proc freq data = agent;
table agent_flag;
run;
/* 31 phone numbers (162562) flagged as agents */
/* creating a table with agent flag and customer phone number */
proc sql;
create table agent1 as
select cust_phn_nm, agent_flag
from agent;
quit;
/* sort agent1 and calls_agents by cust_phn_nm */
proc sort data = agent1;
by cust_phn_nm;
run;
proc sort data = calls_agents;
/* merge agent_flag1 and delta_data_processed_time by cust_phn_nm */
data delta_flags;
merge
calls_agents agent1;
by cust_phn_nm;
run;
/* 20040401 observations */
/* check merge */
proc freq data = delta_flags;
table agent_flag;
```

```
run;
/* 162562/0.81% flagged as agents */
/* delete observations where agent = 1 */
data delta_flags;
set delta_flags;
if agent_flag = 1
then delete;
run;
/* 19877839 calls remaining */
/* check delete */
proc freq data = delta_flags;
table agent_flag;
run;
/* all agents successfully removed from the dataset (N = 19877839) */
/******************** REMOVING IMPOSSIBLE VALUES ********************/
proc means data = delta_flags;
run;
/* customer age > 105 */
proc sql;
select count(cust_age)
from delta_flags
where cust_age > 105;
quit;
/* 1143 */
/* ivr time negative */
proc sql;
select count(ivr_sec_tm)
from delta_flags
where ivr_sec_tm < 0;
quit;
/* 0 */
/* hndl_sec_ct negative */
proc sql;
select count(hndl_sec_ct)
from delta_flags
where hndl_sec_ct < 0;
quit;
/* 1 */
/* remove based on above criteria */
data delta_flags;
set delta_flags;
if cust_age > 105 then delete;
if ivr_sec_tm < 0 then delete;
if hndl_sec_ct < 0 then delete;
run;
/* 19876695 calls remaining */
/************************ REMOVE OUTLIERS ************************/
/* define macro to calculate z scores and flag for z >= 3.5 */
/* frequency tables will also be created for each flag variable */
%macro outlier_flag(var =);
proc stdize data = delta_flags out = delta_flags OPREFIX SPREFIX = ZSCORE;
var &var.;
run;
```

```
data delta_flags;
set delta_flags;
if ZSCORE&var. >= 3.5 then &var._flag = 1;
else &var._flag = 0;
run;
/* proc freq data = delta_outlier;
table &var._flag;
run; */
%mend;
%outlier_flag(var = Hld_Cnt);
%outlier_flag(var = Lngst_Hld_Cnt);
%outlier_flag(var = Tlk_Tm);
%outlier_flag(var = Acw_Tm);
%outlier_flag(var = Spclst_Tlk_Tm);
%outlier_flag(var = Est_Qtd_Sec_Tm);
%outlier_flag(var = IVR_Sec_Tm);
%outlier_flag(var = Hndl_Sec_Ct);
%outlier_flag(var = Intr_Sec_Ct);
%outlier_flag(var = Tot_Cust_Wait_Time);
%outlier_flag(var = Initial_Queue_Wait_Time);
%outlier_flag(var = Support_Call_Queue_Sec_Ct);
%outlier_flag(var = Support_Call_Hld_Sec_Ct);
/* delete calls flagged for any outlier variable flags */
data delta_no_flags;
set delta_flags;
if
Hld_Cnt_flag = 1 or
Lngst_Hld_Cnt_flag = 1 or
Tlk_Tm_flag = 1 or
Acw_Tm_flag = 1 or
Spclst_Tlk_Tm_flag = 1 or
Hndl_Sec_Ct_flag = 1 or
Intr_Sec_Ct_flag = 1 or
Tot_Cust_Wait_Time_flag = 1 or
Initial_Queue_Wait_Time_flag = 1 or
Support_Call_Queue_Sec_Ct_flag = 1 or
Support_Call_Hld_Sec_Ct_flag = 1
then delete;
run;
/* 18497198 */
data team4.no_flags_0301_1;
set delta_no_flags;
run;
/* make temp dataset */
data no_flags;
set team4.no_flags_0301_1;
run;
proc contents data = no_flags;
run;
proc freq data = delta_flags;
tables
Hld_Cnt_flag
Lngst_Hld_Cnt_flag
```

```
Tlk_Tm_flag
Acw_Tm_flag
Spclst_Tlk_Tm_flag
Est_Qtd_Sec_Tm_flag
IVR_Sec_Tm_flag
Hndl_Sec_Ct_flag
Intr_Sec_Ct_flag
Tot_Cust_Wait_Time_flag
Initial_Queue_Wait_Time_flag
Support_Call_Queue_Sec_Ct_flag
Support_Call_Hld_Sec_Ct_flag / nocol norow nocum;
run;
/* remove unnecessary variables */
data no_flags_clean;
set no_flags;
drop
Hld_Cnt_flag
Lngst_Hld_Cnt_flag
Tlk_Tm_flag
Acw_Tm_flag
Spclst_Tlk_Tm_flag
Est_Qtd_Sec_Tm_flag
IVR_Sec_Tm_flag
Hndl_Sec_Ct_flag
Intr_Sec_Ct_flag
Tot_Cust_Wait_Time_flag
Initial_Queue_Wait_Time_flag
Support_Call_Queue_Sec_Ct_flag
Support_Call_Hld_Sec_Ct_flag
ZSCOREHld_Cnt
ZSCORELngst_Hld_Cnt
ZSCORETlk_Tm
ZSCOREAcw_Tm
ZSCORESpclst_Tlk_Tm
ZSCOREEst_Qtd_Sec_Tm
ZSCOREIVR_Sec_Tm
ZSCOREHndl_Sec_Ct
ZSCOREIntr_Sec_Ct
ZSCORETot_Cust_Wait_Time
ZSCOREInitial_Queue_Wait_Time
ZSCORESupport_Call_Queue_Sec_Ct
ZSCORESupport_Call_Hld_Sec_Ct;
run;
/* 18497198 */
/************************* TAKE SRS TO USE ************************/
/* dataset of distinct phone numbers */
proc sql;
create table distinct_phone as
select distinct cust_phn_nm
from no_flags_clean;
quit;
proc surveyselect data = distinct_phone
seed = 117
```

```
method = srs N = 1000000 out = delta_sample;
run;
proc sort data = delta_sample out = sample_sort;
by cust_phn_nm;
run;
proc sort data = no_flags_clean out = clean_sort;
by cust_phn_nm;
run;
data delta_flags;
merge clean_sort(in=a) sample_sort(in=b);
by cust_phn_nm;
if a and b;
run;
/* 2615742 */
/********************* MERGE CALLS AND INTENTS **********************/
/* make copy of intents dataset */
data intents;
set archive.allintnt;
run;
/* rename to call_id */
data intent_rename;
set intents;
rename
CsGpCl_Sg_Id = call_id;
run;
/* adjust format for call_id in intent dataset */
data intent_format;
length call_id $38;
set intent_rename;
run;
/* create table with unique call_id/intent rows from intent */
proc sql;
create table IntentCallId as
select distinct
call_id, lvl_1_int, lvl_2_int
from intent_format;
quit;
/* sort */
proc sort data = delta_flags out = flags_sort;
by call_id;
run;
proc sort data = IntentCallId out = intent_sort;
by call_id;
run;
/* merge */
data merge_test;
merge flags_sort(in=a) intent_sort;
by call_id;
if a;
run;
/* 4640007 */
/* make sure distinct */
proc sql;
```

```
create table distinct_merge_0302 as
select distinct *
from merge_test;
quit;
/* 4640007 */
/* permanent copy */
data team4.long_start;
set merge_test;
run;
/* assess missingness for level 1 intent */
proc freq data = team4.long_start;
tables lvl_1_int / nocol norow nocum missing;
run;
/******************** INTENT FORMAT/NEW INTENT ********************/
/* rename intent categories to avoid duplicates */
data merge_transpose;
set distinct_merge_0302;
if Lng_Int_Nm_Lv2 = "Change in booking" then lng_int_nm_lv2 = "Change in Booking";
if Lng_Int_Nm_Lv2 = "Flight status" then lng_int_nm_lv2 = "Flight Status";
if Lng_Int_Nm_Lv2 = "Medallion Questions" then lng_int_nm_lv2 = "Medallion Question";
if Lng_Int_Nm_Lv2 = "Post-trip Support" then lng_int_nm_lv2 = "Post Trip Support";
if Lng_Int_Nm_Lv2 = "Skymiles Support" then lng_int_nm_lv2 = "SkyMiles Support";
if shrt_int_nm_lv1 = "Pre Trip Support" then shrt_int_nm_lv1 = "Pre-Trip Support";
if shrt_int_nm_lv1 = "Voluntary change" then shrt_int_nm_lv1 = "Voluntary Change";
if shrt_int_nm_lv1 = "Skymiles" then shrt_int_nm_lv1 = "SkyMiles";
if shrt_int_nm_lv1 = "SkyMiles Support" then shrt_int_nm_lv1 = "SkyMiles";
run;
/* update lvl_2_int categories */
data merge_transpose;
set merge_transpose;
if lvl_2_int = "Baggage - Claim" then lvl_2_int = "Baggage";
if lvl_2_int = "Baggage - Damaged" then lvl_2_int = "Baggage";
if lvl_2_int = "Baggage - Lost & Found" then lvl_2_int = "Baggage";
if lvl_2_int = "Baggage - Missing/Delayed" then lvl_2_int = "Baggage";
if lvl_2_int = "Baggage - Status" then lvl_2_int = "Baggage";
if lvl_2_int = "Upgrade Availability" then lvl_2_int = "Upgrade";
if lvl_2_int = "SkyMiles Redemption" then lvl_2_int = "SkyMiles";
if lvl_2_int = "Card Features and Benefits" then lvl_2_int = "Card Features/Benefits";
run;
/* update lng_int_nm_lv2 categories */
data merge_transpose;
set merge_transpose;
if lng_int_nm_lv2 = "Account Support" then lng_int_nm_lv2 = "Support";
if lng_int_nm_lv2 = "Change in Booking" then lng_int_nm_lv2 = "Change Booking";
if lng_int_nm_lv2 = "Class Upgrade" then lng_int_nm_lv2 = "Upgrade";
if lng_int_nm_lv2 = "Credit Card" then lng_int_nm_lv2 = "Card Features/Benefits";
if lng_int_nm_lv2 = "Customer Care" then lng_int_nm_lv2 = "Care - Complaint";
if lng_int_nm_lv2 = "External Support" then lng_int_nm_lv2 = "Support";
if lng_int_nm_lv2 = "In-Flight Meals" then lng_int_nm_lv2 = "Other";
if lng_int_nm_lv2 = "Name Change" then lng_int_nm_lv2 = "Name Correction";
if lng_int_nm_lv2 = "Post Trip Support" then lng_int_nm_lv2 = "Support";
if lng_int_nm_lv2 = "Pre-Trip Support" then lng_int_nm_lv2 = "Support";
if lng_int_nm_lv2 = "Refunds" then lng_int_nm_lv2 = "Refunds Request";
```

```
if lng_int_nm_lv2 = "Shopping & Booking" then lng_int_nm_lv2 = "Other";
if lng_int_nm_lv2 = "Sky Club Question" then lng_int_nm_lv2 = "SkyMiles";
if lng_int_nm_lv2 = "SkyMiles Benefits" then lng_int_nm_lv2 = "SkyMiles";
if lng_int_nm_lv2 = "SkyMiles Enrollment" then lng_int_nm_lv2 = "SkyMiles";
if lng_int_nm_lv2 = "SkyMiles Redemption" then lng_int_nm_lv2 = "SkyMiles";
if lng_int_nm_lv2 = "SkyMiles Support" then lng_int_nm_lv2 = "SkyMiles";
if lng_int_nm_lv2 = "Update Details" then lng_int_nm_lv2 = "Change Details";
if lng_int_nm_lv2 = "eCredit/Voucher" then lng_int_nm_lv2 = "eCredit Redemption";
if lng_int_nm_lv2 = "Check-in issues" then lng_int_nm_lv2 = "Check-in Issues";
run;
/* recode skymiles */
data merge_transpose;
set merge_transpose;
if sm_tier_call = "NM" then sm_tier_callN = 1;
if sm_tier_call = "FF" then sm_tier_callN = 2;
if sm_tier_call = "FO" then sm_tier_callN = 3;
if sm_tier_call = "GM" then sm_tier_callN = 4;
if sm_tier_call = "PM" then sm_tier_callN = 5;
if sm_tier_call = "DM" then sm_tier_callN = 6;
if sm_tier_call = "TS" then sm_tier_callN = 7;
run;
/* create new intent variable */
data merge_transpose;
set merge_transpose;
if lvl_2_int ne " " then new_int = lvl_2_int;
if lvl_2_int = " " then new_int = lng_int_nm_lv2;
if lvl_2_int = " " & lng_int_nm_lv2 = " " & sm_tier_callN >= 4 then new_int = "Missing VIP";
if lvl_2_int = " " & lng_int_nm_lv2 = " " & sm_tier_callN < 4 then new_int = "Missing Low
Tier";
run;
/* adding info from level 1 intent */
data merge_transpose;
set merge_transpose;
if lvl_1_int = "Shopping & Booking" and lvl_2_int = "Upgrade"
then new_int = "Booking Upgrade";
if lvl_1_int = "Ticket Changes" and lvl_2_int = "Upgrade"
then new_int = "Ticket Upgrade";
if lvl_1_int = "Pre-Trip Support" and lvl_2_int = "Baggage"
then new_int = "Pre-Trip Baggage";
if lvl_1_int = "Post-Trip Support" and lvl_2_int = "Baggage"
then new_int = "Post-Trip Baggage";
run;
/* make intent lowercase */
data merge_transpose;
set merge_transpose;
int = lowcase(new_int);
run;
/* make distint */
proc sql;
create table merge_transpose_distinct as
select distinct *
from merge_transpose;
quit;
```

```
/* 4618543 */
/* save permanent copy */
data team4.merge_long_36;
set merge_transpose_distinct;
run;
/****************** TRANSPOSE INTO WIDE FORMAT ********************/
data need_trans (keep = call_id int);
set merge_transpose_distinct;
run;
*creates wide data with each intent as a variable;
data need_trans;
set need_trans;
if int = "baggage" then baggage = 1;
else baggage = 0;
if int = "boarding pass" then boarding_pass= 1;
else boarding_pass = 0;
if int = "booking upgrade" then booking_upgrade = 1;
else booking_upgrade = 0;
if int = "book a ticket" then book_a_ticket = 1;
else book_a_ticket = 0;
if int = "cancellation" then cancellation = 1;
else cancellation = 0;
if int = "card features/benefits" then card_features_benefits = 1;
else card_features_benefits = 0;
if int = "care - complaint" then care_complaint = 1;
else care_complaint = 0;
if int = "change booking" then change_booking = 1;
else change_booking = 0;
if int = "change details" then change_details = 1;
else change_details = 0;
if int = "check-in issues" then checkin_issues = 1;
else checkin_issues = 0;
if int = "child ticket" then child_ticket = 1;
else child_ticket = 0;
if int = "companion certificate" then companion_certificate = 1;
else companion_certificate = 0;
if int = "delete/cancel account" then delete_cancel_account = 1;
else delete_cancel_account = 0;
if int = "existing change" then existing_change = 1;
else existing_change = 0;
if int = "external transfer" then external_transfer = 1;
else external_transfer = 0;
if int = "online/website" then online_website = 1;
else online_website = 0;
if int = "other" then other = 1;
else other = 0;
if int = "post-trip baggage" then post_baggage = 1;
else post_baggage = 0;
if int = "pre-trip baggage" then pre_baggage = 1;
else pre_baggage = 0;
if int = "promo question" then promo_question = 1;
else promo_question = 0;
if int = "refunds request" then refunds_request = 1;
```

```
else refunds_request = 0;
if int = "retro credit" then retro_credit = 1;
else retro_credit = 0;
if int = "ssr" then ssr = 1;
else ssr = 0;
if int = "same day change" then same_day_change = 1;
else same_day_change = 0;
if int = "schedule change" then schedule_change = 1;
else schedule_change = 0;
if int = "seat selection" then seat_selection = 1;
else seat_selection = 0;
if int = "skymiles" then skymiles = 1;
else skymiles = 0;
if int = "specialty fare" then specialty_fare = 1;
else specialty_fare = 0;
if int = "support" then support = 1;
else support = 0;
if int = "ticket upgrade" then ticket_upgrade = 1;
else ticket_upgrade = 0;
if int = "travel requirements" then travel_requirements = 1;
else travel_requirements = 0;
if int = "trip extras" then trip_extras = 1;
else trip_extras = 0;
if int = "upgrade" then upgrade = 1;
else upgrade = 0;
if int = "ecredit redemption" then ecredit_redemption = 1;
else ecredit_redemption = 0;
run;
*collapses rows with the same call_id. no value should be >1;
proc sql;
create table transposed as
select
call_id,
sum(baggage) as baggage,
sum(boarding_pass) as boarding_pass,
sum(booking_upgrade) as booking_upgrade,
sum(book_a_ticket) as book_a_ticket,
sum(cancellation) as cancellation,
sum(card_features_benefits) as card_features_benefits,
sum(care_complaint) as care_complaint,
sum(change_booking) as change_booking,
sum(change_details) as change_details,
sum(checkin_issues) as checkin_issues,
sum(child_ticket) as child_ticket,
sum(companion_certificate) as companion_certificate,
sum(delete_cancel_account) as delete_cancel_account,
sum(ecredit_redemption) as ecredit_redemption,
sum(existing_change) as existing_change,
sum(external_transfer) as external_transfer,
sum(flight_canceled) as flight_transfered,
sum(flight_delayed) as flight_delayed,
sum(flight_status) as flight_status,
sum(fraudulent_activity) as fraudulent_activity,
```

```
sum(guc_ruc_upgrade) as guc_ruc_upgrade,
sum(irop_maintenance) as irop_maintenance,
sum(irop_weather) as irop_weather,
sum(login_issues) as login_issues,
sum(medallion_question) as medallion_question,
sum(medical_fare) as medical_fare,
sum(trip_extras) as trip_extras,
sum(ticket_upgrade) as ticket_upgrade,
sum(upgrade) as upgrade
from need_trans
group by call_id;
quit;
/* 4618543 */
/************* MERGE CALL DETAILS ONTO TRANSPOSE INTENT *************/
/* set new dataset dropping variables not needed */
data call_details
(drop = shrt_int_nm_lv1 lng_int_nm_lv2 lvl_1_int lvl_2_int new_int int);
set merge_transpose_distinct;
run;
proc sql;
create table call_distinct as
select distinct *
from call_details;
quit;
proc sql;
create table merge_wide as
select *
from transposed
natural join call_distinct
order by cust_phn_nm;
quit;
/* create copy of dataset */
data team4.merge_wide_32;
set merge_wide;
run;
/* 2615742 */
/******************* EXCLUDE DAYS FOR WAIT TIME *******************/
/* tot_cust_wait_time, est_qtd_sec_tm, Support_Call_Hld_Sec_Ct, Support_Call_Queue_Sec_Ct
Lngst_Hld_Cnt, hndl_sec_ct */
/* get necessary variables for day/month/year */
data delta_time;
set team4.merge_wide_32;
day_of_year = put(datepart(call_strt_gts), julian.);
day_of_month = put(datepart(Call_strt_GTs), day.);
month_of_year = put(datepart(Call_strt_GTs), MMYY.);
run;
/* getting average hold time by day for each var */
proc sql;
create table day_flags as
select day_of_year, mean(tot_cust_wait_time) as avg_tot_wait,
mean(est_qtd_sec_tm) as avg_qtd_wait, mean(support_call_hld_sec_ct) as avg_hold,
mean(Support_Call_Queue_Sec_Ct) as avg_support_queue, mean(lngst_hld_cnt) as avg_long_hold,
mean(hndl_sec_ct) as avg_length
```

```
from delta_time
group by day_of_year;
quit;
/* calculating z-scores for day flags */
proc stdize data = day_flags out = day_flags OPREFIX SPREFIX =ZSCORE;
var avg_tot_wait avg_qtd_wait avg_hold avg_support_queue avg_long_hold avg_length;
run;
/* creating a table with day_flag flag variable */
proc sql;
create table wait_flag as
select *,
case when ZSCOREavg_tot_wait >= 2 then 1
else 0
end as tot_wait_flag,
case when ZSCOREavg_qtd_wait >= 2 then 1
else 0
end as qtd_wait_flag,
case when ZSCOREavg_hold >= 2 then 1
else 0
end as hold_flag,
case when ZSCOREavg_support_queue >= 2 then 1
else 0
end as support_flag,
case when ZSCOREavg_long_hold >= 2 then 1
else 0
end as long_hold_flag,
case when ZSCOREavg_length >= 2 then 1
else 0
end as length_flag
from day_flags;
quit;
/* checking flag count for days flagged based on avg_wait */
proc freq data = wait_flag;
table tot_wait_flag qtd_wait_flag hold_flag support_flag long_hold_flag length_flag;
run;
/* creating a table with flags and day of year */
proc sql;
create table day_flags1 as
select day_of_year, tot_wait_flag, qtd_wait_flag, hold_flag,
support_flag, long_hold_flag, length_flag
from wait_flag;
quit;
/* sort day_flags1 and delta_time by day_of_year */
proc sort data = day_flags1;
by day_of_year;
run;
proc sort data = delta_time;
by day_of_year;
run;
/* merge wait_flag and delta_no_flags by day_of_year */
/* output as delta_exclusion219 */
data team4.delta_analysis_322;
merge
```

```
day_flags1 delta_time;
by day_of_year;
run;
/* check merge */
proc freq data = team4.delta_analysis_322;
table tot_wait_flag qtd_wait_flag hold_flag support_flag long_hold_flag length_flag;
run;
/* check intents */
proc freq data = team4.delta_analysis_322;
tables baggage boarding_pass booking_upgrade book_a_ticket
cancellation card_features_benefits care_complaint change_booking change_details
checkin_issues
child_ticket companion_certificate delete_cancel_account ecredit_redemption existing_change
external_transfer flight_transfered flight_delayed flight_status fraudulent_activity
guc_ruc_upgrade irop_maintenance irop_weather login_issues medallion_question medical_fare
missed_flight missing_low_tier missing_vip name_correction notification_received
online_website
other post_baggage pre_baggage promo_question refunds_request retro_credit same_day_change
schedule_change seat_selection skymiles specialty_fare ssr support travel_requirements
trip_extras ticket_upgrade upgrade / nocol nocum norow;
run;
/********************** TARGET CODE DATASET ***********************/ data
team4.delta_target_322;
set team4.delta_analysis_322;
keep
call_id cust_phn_nm call_strt_gts baggage boarding_pass booking_upgrade book_a_ticket
cancellation card_features_benefits care_complaint change_booking change_details
checkin_issues
child_ticket companion_certificate delete_cancel_account ecredit_redemption existing_change
external_transfer flight_transfered flight_delayed flight_status fraudulent_activity
guc_ruc_upgrade irop_maintenance irop_weather login_issues medallion_question medical_fare
missed_flight missing_low_tier missing_vip name_correction notification_received
online_website
other post_baggage pre_baggage promo_question refunds_request retro_credit same_day_change
schedule_change seat_selection skymiles specialty_fare ssr support travel_requirements
trip_extras ticket_upgrade upgrade;
run;
/* 2615742 */
/********** FLAG/SEPARATE PHONE NUMBERS OCCURRING ONLY ONCE *********/
data target_occurrence;
set team4.delta_target_322;
run;
proc sql;
create table PhoneCounts as
select cust_phn_nm,
count(*) as Count
from target_occurrence
group by cust_phn_nm;
/* Join the temporary table back to the original dataset and create the indicator variable */
create table calls_indicator as
select A.*,
case when B.Count = 1 then 1
when B.Count = 2 then 2
```

```
else 3
end as call_indicator
from target_occurrence A
left join PhoneCounts B
on A.cust_phn_nm = B.cust_phn_nm;
quit;
data call_single call_double call_multiple;
set calls_indicator;
if call_indicator= 1 then output call_single;
if call_indicator= 2 then output call_double;
if call_indicator= 3 then output call_multiple;
run;
/*************** CREATE CHUNKS TO RUN TARGET CODE *****************/ /* create numeric phone
number variable */
data target_multiple;
set call_multiple;
numeric_phone = input(cust_phn_nm, ?? best32.);
run;
/* 1669413 */
/* sort by phone number */
proc sort data = target_multiple out = target_multi_sort;
by cust_phn_nm;
run;
/* assess value of phone numbers */
proc means data = target_multi_sort min max;
var numeric_phone;
run;
/* range from 239 to 10816526 */
proc means data = target_multi_sort median;
var numeric_phone;
run;
/* median = 5259521 */
/* split into 2 datasets for multiple calls */
data team4.target1_multi322;
set target_multi_sort;
where 239 <= numeric_phone <= 5259521;
run;
data team4.target2_multi322;
set target_multi_sort;
where 5259521 < numeric_phone <= 10816526;
run;
/* 834700 */
/* export multipes as csv */
proc export data = team4.target1_multi322
outfile ='/gpfs/user_home/os_home_dirs/kdunca49/ds7900_spring24_team4/target1_multi322.csv'
dbms = csv replace;
proc export data = team4.target2_multi322
outfile ='/gpfs/user_home/os_home_dirs/kdunca49/ds7900_spring24_team4/target2_multi322.csv'
dbms = csv replace;
run;
/* create permanent copy of single calls with target variables (all 0, 1, 0) */
data team4.calls_single_322;
set call_single;
```

```
repeat_flag = 0;
original = 1;
repeat_after = 0;
run;
/* export single to csv */
proc export data = team4.calls_single_322
outfile ='/gpfs/user_home/os_home_dirs/kdunca49/ds7900_spring24_team4/calls_single_322.csv'
dbms = csv replace;
run;
/* create permanent copy of double calls */
data team4.calls_double_322;
set call_double;
run;
/* export double to csv */
proc export data = team4.calls_double_322
outfile ='/gpfs/user_home/os_home_dirs/kdunca49/ds7900_spring24_team4/calls_double_322.csv'
dbms = csv replace;
run;
/* save full analysis and target datasets as csv */
proc export data = team4.delta_analysis_322
outfile ='/gpfs/user_home/os_home_dirs/kdunca49/ds7900_spring24_team4/delta_analysis_322.csv'
dbms = csv replace;
run;
proc export data = team4.delta_target_322
outfile ='/gpfs/user_home/os_home_dirs/kdunca49/ds7900_spring24_team4/delta_target_322.csv'
dbms = csv replace;
run;
/********** CREATION OF REPEAT VARIABLES FOR DOUBLE CALLS **********/ data call_double;
set team4.calls_double_322;
run;
/* _____________________________________ */
/* splitting the double calls into 2 datasets - calls within 12 hours and calls outside of 12
hours */
/* _____________________________________ */
* Sort the dataset by phone number and call time;
proc sort data=call_double;
by cust_phn_nm call_strt_gts;
run;
*creates two datasets for under 12 hours and over 12 hours;
data calls_over_12hours calls_under_12hours;
set call_double;
by cust_phn_nm;
/* Retain the start time of the previous call for comparison */
retain prev_call_strt_gts;
/* time difference in hours if not first row and phone number same */ if not first.cust_phn_nm
then do;
diff_hours = (call_strt_gts - prev_call_strt_gts) / 3600; /* Convert seconds to hours */
/* Segregate based on time difference */
if diff_hours > 12 then output calls_over_12hours;
else if diff_hours <= 12 then output calls_under_12hours;
end;
/* Update previous call start time */
prev_call_strt_gts = call_strt_gts;
```

```
run;
/* ____________________________________ */
/* merging the flagged calls back onto the doubles dataset */
/* ____________________________________ */
*creates a variable called under_12 and assigns 0 (calls are over 12 hours apart);
data calls_over_12hours (keep = cust_phn_nm under12);
set calls_over_12hours;
under12 = 0;
run;
*creates a variable called under_12 and assigns 1 (calls are under 12 hours apart);
data calls_under_12hours (keep = cust_phn_nm under12);
set calls_under_12hours;
under12 = 1;
run;
*sorting the 3 datasets by phone number for the merge;
proc sort data = call_double;
by cust_phn_nm;
run;
proc sort data = calls_over_12hours;
by cust_phn_nm;
run;
proc sort data = calls_under_12hours;
by cust_phn_nm;
run;
*merging under_12 flag onto the doubles data;
data call_double_time;
merge call_double calls_over_12hours calls_under_12hours;
by cust_phn_nm;
run;
/* ____________________________________ */
/* flagging repeat calls */
/* ____________________________________ */
*sorting by phone number and call start time;
proc sort data = call_double_time;
by cust_phn_nm call_strt_gts;
run;
*calculates repeat calls for under 12 hours;
data processed_dataset;
set call_double_time;
by cust_phn_nm;
array intents[49] baggage boarding_pass booking_upgrade book_a_ticket
cancellation card_features_benefits care_complaint change_booking change_details
checkin_issues child_ticket companion_certificate delete_cancel_account ecredit_redemption
existing_change external_transfer flight_transfered flight_delayed flight_status
fraudulent_activity guc_ruc_upgrade irop_maintenance irop_weather login_issues
medallion_question medical_fare missed_flight missing_low_tier missing_vip name_correction
notification_received online_website other post_baggage pre_baggage promo_question
refunds_request retro_credit same_day_change schedule_change seat_selection skymiles
specialty_fare ssr support travel_requirements trip_extras ticket_upgrade upgrade;
array sums[49] _temporary_; * Use a temporary array to hold sums;
* Initialize sums to 0 for the first record of each phone number;
if first.cust_phn_nm then do;
do i = 1 to 49;
```

```
sums[i] = 0; * Initialize sum for each variable;
end;
end;
* Calculate the running sum for each variable;
do i = 1 to 49;
sums[i] = sums[i] + intents[i]; * Update sum;
end;
retain repeat_flag 0; * Retain repeat_flag across rows;
* Check if any sum is greater than 1 for the last record of the phone number;
if last.cust_phn_nm then do;
repeat_flag = 0; * Reset repeat_flag for each new group;
do i = 1 to 49;
if sums[i] > 1 then do;
repeat_flag = 1; * Set repeat_flag if any sum is > 1;
leave; * Exit the loop early;
end;
end;
output; * Output record;
end;
where under12 = 1;
run;
*merging flag with call_double_time data;
data processed_dataset (keep = call_id repeat_flag);
set processed_dataset;
run;
*sorting flags and double calls dataset by call id for merge;
proc sort data = processed_dataset;
by call_id;
run;
proc sort data = call_double_time;
by call_id;
run;
*merging the flagged call IDs;
data call_double_repeat;
merge call_double_time processed_dataset ;
by call_id;
run;
*finishing repeat flags ;
data call_double_repeat;
set call_double_repeat;
if under12 = 1 and repeat_flag = . then repeat_flag = 0;
if under12 = 0 and repeat_flag = . then repeat_flag = 0;
run;
*sorting for visual check;
proc sort data=call_double_repeat;
by cust_phn_nm call_strt_gts;
run;
/* _____________________________________ */
/* creating the initial call variable */
/* _____________________________________ */
/* Sort the dataset by customer phone number and call start time */
proc sort data=call_double_repeat;
by cust_phn_nm call_strt_gts;
```

```
run;
/* Flag the first call for each phone number */
data flagged_dataset;
set call_double_repeat;
by cust_phn_nm call_strt_gts;
/* Initialize 'original' flag to 0 */
original = 0;
/* If it's the first row for the current phone number, flag it as original */
if first.cust_phn_nm then original = 1;
run;
proc sort data=flagged_dataset;
by cust_phn_nm call_strt_gts;
run;
*setting the second call to original if it is not a repeat;
data flagged_dataset1;
set flagged_dataset;
if original = 0 and repeat_flag = 0 then original = 1;
run;
/* ____________________________________ */
/* creating the repeat_after variable */
/* ____________________________________ */
proc sort data=flagged_dataset1 out=sorted_dataset;
by cust_phn_nm call_strt_gts;
run;
data helper;
set sorted_dataset;
by cust_phn_nm;
if last.cust_phn_nm and repeat_flag=1 then do;
output;
end;
keep cust_phn_nm;
run;
data flagged_dataset2;
merge sorted_dataset(in=a) helper(in=b);
by cust_phn_nm;
if a and first.cust_phn_nm and b then repeat_after=1;
else repeat_after=0;
run;
proc sort data = flagged_dataset2;
by cust_phn_nm call_strt_gts;
run;
proc print data = flagged_dataset2;
where repeat_flag = 1 and original = 1;
run;
/* ____________________________________ */
/* permanent copy of flagged repeats for double calls */
/* ____________________________________ */
data team4.calls_double_326_2;
set flagged_dataset2;
run;
/****************** MERGE TARGETS VARS FROM SINGLE, DOUBLE, MULTI ************************/
/* import multi calls */
```

```
proc import datafile =
"/gpfs/user_home/os_home_dirs/kdunca49/ds7900_spring24_team4/final_labels.csv"
out = team4.calls_multi_326
dbms = csv
replace;
getnames = yes;
run;
/* set call details */
data call_details;
set team4.delta_analysis_322;
run;
/* keep only necessary variables from single calls */
data calls_single;
set team4.calls_single_322;
keep call_id repeat_flag original repeat_after call_indicator;
run;
/* keep only necessary variables from double calls */
data calls_double;
set team4.calls_double_326_2;
keep call_id repeat_flag original repeat_after call_indicator;
run;
/* adjust repeat variables for multiple calls */
data calls_multi;
set team4.calls_multi_326;
if repeat_after = 1 then original = 1;
call_indicator = 3;
run;
data calls_multi;
set calls_multi;
if original = . then original = 0;
if repeat_after = . then repeat_after = 0;
if repeat = . then repeat = 0;
run;
data calls_multi;
set calls_multi;
rename repeat = repeat_flag;
run;
proc freq data = calls_double;
tables repeat_after / nocol nocum norow;
run;
proc freq data = calls_multi;
tables repeat_after / nocol norow nocum;
run;
/* concatenate single, double, and multi calls */
data team4.all_repeat326;
set
calls_single
calls_double
calls_multi;
run;
/* 2615742 */
/* merge call details and repeat info */
proc sort data = call_details out = details_sort;
```

```
by call_id;
run;
proc sort data = team4.all_repeat326 out = repeat_sort;
by call_id;
run;
data repeat_details326;
merge
details_sort(in=a) repeat_sort(in=b);
by call_id;
if a and b;
run;
proc freq data = team4.repeat_details326;
tables original repeat_flag repeat_after / nocol nocum norow;
run;
/* save permanent copy */
data team4.repeat_details326;
set repeat_details326;
run;
/* export as csv */
/* call_indicator needs to be deleted from the model */
proc export data = team4.repeat_details326
outfile ='/gpfs/user_home/os_home_dirs/kdunca49/ds7900_spring24_team4/repeat_details326.csv'
dbms = csv replace;
run;
/****************** IMPLEMENTATION OF MESSAGES ********************/
/* add messaging */
data messages;
set archive.messages;
run;
data calls;
set team4.repeat_details326;
run;
/* keep only necessary variables */
data calls2;
set calls;
keep call_id call_strt_gts cust_sm_nm;
run;
data messages2;
set messages;
keep msgg_cvst_id msgg_strt_tm msgg_cust_sm_nm;
run;
/* change var names */
data calls3;
set calls2;
rename
call_id = ID
call_strt_gts = time
cust_sm_nm = SM;
contact = "call";
run;
data messages3;
set messages2;
rename
```

```
msgg_cvst_id = ID
msgg_strt_tm = time
msgg_cust_sm_nm = SM;
contact = "message";
run;
/* create sm number directories */
proc sql;
create table sm_call as
select distinct SM from calls3;
quit;
proc sql;
create table sm_message as
select distinct SM from messages3;
quit;
by sm;
run;
proc sort data = sm_call;
by sm;
run;
proc sort data = sm_message;
by sm;
run;
data merge_message;
merge
sm_call(in=a) sm_message(in=b);
if a and b;
by sm;
where sm ne .;
run;
data call_message;
set calls3 messages3;
run;
data merge_message;
set merge_message;
if SM = . then delete;
run;
data call_message;
set call_message;
if SM = . then delete;
run;
proc sort data = call_message;
by SM;
run;
proc sort data = merge_message;
by SM;
run;
data call_message2;
merge
call_message(in=a) merge_message(in=b);
by SM;
if a and b;
run;
proc sql;
```

```
select count(distinct sm)
from call_message2;
quit;
proc sort data = call_message2;
by SM time;
run;
proc freq data = call_message2;
tables contact / nocol nocum norow;
run;
/* permanent copy of messages */
data team4.call_message329;
set call_message2;
run;
/* sort by time and sm */
proc sort data = team4.call_message329 out = message_sort;
by sm time;
run;
/* export as csv */
proc export data = team4.call_message329
outfile ='/gpfs/user_home/os_home_dirs/kdunca49/ds7900_spring24_team4/call_message329.csv'
dbms = csv replace;
run;
/* calculate calls/messages within 12 hours */
data result(keep=id time contact sm within_12);
set call_message2;
by sm;
array call_ids[1000] $38. call_times[1000] message_times[1000] n_calls[1000] n_messages[1000];
retain call_times message_times n_calls n_messages;
format within_12 $3.;
if first.sm then do;
/* Reset counters for each account */
n_calls = 0;
n_messages = 0;
end;
if contact_type = 'call' then do;
n_calls + 1;
call_times[n_calls] = time;
call_ids[n_calls] = id;
end;
else if contact_type = 'message' then do;
n_messages + 1;
message_times[n_messages] = time;
end;
if last.account_number then do;
/* Now that we have all calls and messages for an account, check for each call */
do i = 1 to n_calls;
within_12 = 'no';
do j = 1 to n_messages;
if abs(call_times[i] - message_times[j]) <= 12*60*60 then do;
within_12 = 'yes';
leave; /* Exit the loop as soon as a match is found */
end;
end;
```

```
id = call_ids[i];
time = call_times[i];
contact_type = 'call';
account_number = _n_; /* Current BY-group counter */
output;
end;
end;
run;
proc sql;
create table msg_sm as
select sm, count(*) as msg_count
from messages3
group by sm;
quit;
proc sort data = msg_sm;
by descending msg_count;
run;
/* export to csv */
proc export data = team4.repeat_details326
outfile ='/gpfs/user_home/os_home_dirs/kdunca49/ds7900_spring24_team4/repeat_details326.csv'
dbms = csv replace;
run;
/********** MERGE WITH MESSAGES AND CREATE FINAL DATASET ************/
/* import multi calls */
proc import datafile =
"/gpfs/user_home/os_home_dirs/kdunca49/ds7900_spring24_team4/Joseph/call_message_classified12.
csv"
out = team4.call_message_classified
dbms = csv
replace;
getnames = yes;
run;
/* change id to call id */
data call_class;
set team4.call_message_classified;
keep id within_12;
rename
id = call_id;
run;
/* sort by call_id */
proc sort data = team4.repeat_details326 out = details_sort;
by call_id;
run;
proc sort data = call_class out = class_sort;
by call_id;
run;
/* merge */
data team4.repeat_details329;
merge
details_sort class_sort;
by call_id;
run;
/* update missing */
```

```
data team4.repeat_details329;
set team4.repeat_details329;
if within_12 = "" then within_12 = "Missing";
run;
proc freq data = team4.repeat_details329;
tables within_12 / nocol norow nocum;
run;
proc contents data = team4.repeat_details329;
run;
/* only initial calls */
data team4.repeat_model329;
set team4.repeat_details329;
where original = 1;
drop
call_end_gts call_entry_flow_nm call_exit_flow_nm call_strt_gts
cust_phn_nm cust_sm_nm dnis_id filemonth fnagt_id inagt_id lob_nm cust_enrll_dt
mm_eff_dt mbsp_pgm_tier_cd sm_tier_eff_dt abandon agent_flag call_indicator
day_of_month day_of_year ivr_spec just_ivr just_spec repeat_flag sm_tier_calln;
run;
data team4.repeat_model329;
set team4.repeat_model329;
drop original;
run;
/* 2434711, 108 variables */
/* export to csv */
proc export data = team4.repeat_details329
outfile ='/gpfs/user_home/os_home_dirs/kdunca49/ds7900_spring24_team4/repeat_details329.csv'
dbms = csv replace;
run;
proc export data = team4.repeat_model329
outfile ='/gpfs/user_home/os_home_dirs/kdunca49/ds7900_spring24_team4/repeat_model329.csv'
dbms = csv replace;
run;
/************************ DATA DISCOVERY ************************/
/*****************************************************************
/* UNIVARIATE */
/* temp copy */
data details;
set team4.repeat_model329;
run;
/* minutes */
data details2;
set details;
call_length = hndl_sec_ct/60;
total_wait = tot_cust_wait_time/60;
quoted_wait = est_qtd_sec_tm/60;
run;
/* means for distribution table */
proc means data = details2 min max mean median std;
var cust_age in_delta_tenure call_length total_wait quoted_wait;
run;
/* means to assess missingness */
proc means data = details n nmiss;
```

```
run;
/* BIVARIATE */
data analysis;
set team4.repeat_model329;
run;
proc anova data = analysis;
class repeat_after;
Model tot_cust_wait_time = repeat_after;
Means repeat_after / hovtest;
run;
/* quoted wait time */
proc anova data = analysis;
class repeat_after;
Model est_qtd_sec_tm = repeat_after;
Means repeat_after / hovtest;
run;
/* call length/repeat calls */
proc anova data = analysis;
class repeat_after;
Model hndl_sec_ct = repeat_after;
Means repeat_after / hovtest;
run;
proc freq data = analysis;
tables
/* quoted wait flag/repeat calls */
proc freq data = analysis;
tables qtd_wait_flag*repeat_after / chisq
riskdiff(equal var = null cl = wald);
run;
/************************ DASHBOARD STATS ************************/
/* temp copy of repeat details dataset */
data delta;
set team4.repeat_details329;
run;
/* 2615742 */
/* set original = 1 */
data delta_original;
set delta;
where original = 1;
run;
/* 2434711 */
/********** call volume of repeat and non-repeat by month **********/
/* january */
proc sql;
select count(call_id)
from delta_original
where repeat_after = 1 & month_of_year = "01M2023";
quit;
/* 12495 repeats */
proc sql;
select count(call_id)
from delta_original
where repeat_after = 0 & month_of_year = "01M2023";
```

```
quit;
/* 179140 non-repeats */
/* february */
proc sql;
select count(call_id)
from delta_original
where repeat_after = 1 & month_of_year = "02M2023";
quit;
/* 11863 repeats */
proc sql;
select count(call_id)
from delta_original
where repeat_after = 0 & month_of_year = "02M2023";
quit;
/* 171407 non-repeats */
/* march */
proc sql;
select count(call_id)
from delta_original
where repeat_after = 1 & month_of_year = "03M2023";
quit;
/* 13299 repeats */
proc sql;
select count(call_id)
from delta_original
where repeat_after = 0 & month_of_year = "03M2023";
quit;
/* 197667 non-repeats */
/* april */
proc sql;
select count(call_id)
from delta_original
where repeat_after = 1 & month_of_year = "04M2023";
quit;
/* 12358 repeats */
proc sql;
select count(call_id)
from delta_original
where repeat_after = 0 & month_of_year = "04M2023";
quit;
/* 186043 non-repeats */
/* may */
proc sql;
select count(call_id)
from delta_original
where repeat_after = 1 & month_of_year = "05M2023";
quit;
/* 14798 repeats */
proc sql;
select count(call_id)
from delta_original
where repeat_after = 0 & month_of_year = "05M2023";
quit;
```

```
/* 210488 non-repeats */
/* june */
proc sql;
select count(call_id)
from delta_original
where repeat_after = 1 & month_of_year = "06M2023";
quit;
/* 12633 repeats */
proc sql;
select count(call_id)
from delta_original
where repeat_after = 0 & month_of_year = "06M2023";
quit;
/* 188076 non-repeats */
/* july */
proc sql;
select count(call_id)
from delta_original
where repeat_after = 1 & month_of_year = "07M2023";
quit;
/* 13362 repeats */
proc sql;
select count(call_id)
from delta_original
where repeat_after = 0 & month_of_year = "07M2023";
quit;
/* 197285 non-repeats */
/* august */
proc sql;
select count(call_id)
from delta_original
where repeat_after = 1 & month_of_year = "08M2023";
quit;
/* 13486 repeats */
proc sql;
select count(call_id)
from delta_original
where repeat_after = 0 & month_of_year = "08M2023";
quit;
/* 200467 non-repeats */
/* september */
proc sql;
select count(call_id)
from delta_original
where repeat_after = 1 & month_of_year = "09M2023";
quit;
/* 13353 repeats */
proc sql;
select count(call_id)
from delta_original
where repeat_after = 0 & month_of_year = "09M2023";
quit;
/* 188100 non-repeats */
```

```
/* october */
proc sql;
select count(call_id)
from delta_original
where repeat_after = 1 & month_of_year = "10M2023";
quit;
/* 13679 repeats */
proc sql;
select count(call_id)
from delta_original
where repeat_after = 0 & month_of_year = "10M2023";
quit;
/* 190545 non-repeats */
/* november */
proc sql;
select count(call_id)
from delta_original
where repeat_after = 1 & month_of_year = "11M2023";
quit;
/* 12958 repeats */
proc sql;
select count(call_id)
from delta_original
where repeat_after = 0 & month_of_year = "11M2023";
quit;
/* 179724 non-repeats */
/* december */
proc sql;
select count(call_id)
from delta_original
where repeat_after = 1 & month_of_year = "12M2023";
quit;
/* 13983 repeats */
proc sql;
select count(call_id)
from delta_original
where repeat_after = 0 & month_of_year = "12M2023";
quit;
/* 187502 non-repeats */
/***************** number of specialists per month ****************/
/* january */
proc sql;
select count(distinct inagt_id)
from delta_original
where month_of_year = "01M2023";
quit;
/* 4870 */
/* february */
proc sql;
select count(distinct inagt_id)
from delta_original
where month_of_year = "02M2023";
quit;
```

```
/* 4912 */
/* march */
proc sql;
select count(distinct inagt_id)
from delta_original
where month_of_year = "03M2023";
quit;
/* 4976 */
/* april */
proc sql;
select count(distinct inagt_id)
from delta_original
where month_of_year = "04M2023";
quit;
/* 5044 */
/* may */
proc sql;
select count(distinct inagt_id)
from delta_original
where month_of_year = "05M2023";
quit;
/* 5060 */
/* june */
proc sql;
select count(distinct inagt_id)
from delta_original
where month_of_year = "06M2023";
quit;
/* 4842 */
/* july */
proc sql;
select count(distinct inagt_id)
from delta_original
where month_of_year = "07M2023";
quit;
/* 4734 */
/* august */
proc sql;
select count(distinct inagt_id)
from delta_original
where month_of_year = "08M2023";
quit;
/* 4623 */
/* september */
proc sql;
select count(distinct inagt_id)
from delta_original
where month_of_year = "09M2023";
quit;
/* 4516 */
/* october */
proc sql;
select count(distinct inagt_id)
```

```
from delta_original
where month_of_year = "10M2023";
quit;
/* 4673 */
/* november */
proc sql;
select count(distinct inagt_id)
from delta_original
where month_of_year = "11M2023";
quit;
/* 4723 */
/* december */
proc sql;
select count(distinct inagt_id)
from delta_original
where month_of_year = "12M2023";
quit;
/* 4660 */
/**************** average call length (min) by month ***************/
/* set call length in minutes */
data delta2;
set delta_original;
call_min = hndl_sec_ct/60;
run;
/* average call length by month */
proc sql;
select month_of_year, avg(call_min) as average_call_length
from delta2
group by month_of_year;
quit;
/*************** average quoted wait (min) by month *****************/
/* set call length in minutes */
data delta3;
set delta2;
quoted_min = est_qtd_sec_tm/60;
run;
/* average call length by month */
proc sql;
select month_of_year, avg(quoted_min) as average_quoted
from delta3
group by month_of_year;
quit;
/******************* distribution of repeat_after ******************/
proc freq data = delta3;
tables repeat_after / nocol norow nocum;
run;
/************* top 5 intents for repeat_after by month **************/
/* sort */
proc sort data = delta3;
by month_of_year;
run;
/* frequency of intents by month */
proc freq data = delta3;
```

```
tables
baggage boarding_pass booking_upgrade book_a_ticket
cancellation card_features_benefits care_complaint change_booking change_details
checkin_issues
child_ticket companion_certificate delete_cancel_account ecredit_redemption existing_change
external_transfer flight_transfered flight_delayed flight_status fraudulent_activity
guc_ruc_upgrade irop_maintenance irop_weather login_issues medallion_question medical_fare
missed_flight missing_low_tier missing_vip name_correction notification_received
online_website other post_baggage pre_baggage promo_question refunds_request retro_credit
same_day_change schedule_change seat_selection skymiles specialty_fare ssr support
travel_requirements trip_extras ticket_upgrade upgrade / nocol nocum norow;
by month_of_year;
where repeat_after = 1;
run;
/****************** repeat_after by tier by month ******************/
/* frequency tables */
proc freq data = delta3;
tables sm_tier_call*repeat_after / nopercent norow nocum;
by month_of_year;
run;
/***************** repeat_after by age group by month ***************/
/* create age group */ data delta4;
set delta_original;
if cust_age = . then cust_age_bin = 0;
else if cust_age >=0 and cust_age < 20 then cust_age_bin = 1;
else if cust_age >=20 and cust_age < 40 then cust_age_bin = 2;
else if cust_age >= 40 and cust_age < 60 then cust_age_bin = 3;
else if cust_age >= 60 then cust_age_bin = 4;
run;
/* sort */
proc sort data = delta4;
by month_of_year;
run;
/* frequency tables */
proc freq data = delta4;
tables cust_age_bin*repeat_after / nopercent norow nocum;
by month_of_year;
run;
/****************** repeat_after by gender by month ****************/ /* frequency tables */
proc freq data = delta4;
tables sm_gender*repeat_after / nopercent norow nocum;
by month_of_year;
where sm_gender = "F" or sm_gender = "M";
run;
/***************** repeat_after by within 12 by month *****************/ /* frequency tables */
proc freq data = delta4;
tables within_12*repeat_after / nopercent norow nocum;
by month_of_year;
run;
/***************** repeat_after by shift by month ******************/ /* frequency tables */
proc freq data = delta4;
tables shift*repeat_after / nopercent norow nocum;
by month_of_year;
```

```
run;
/********************** LOGISTIC MODEL ***************************/
/*********************************************************************//* recode variables */
data model_data;
set logistic;
if mm_tier = . then mm_tier = 0;
if mm_ind0 = . then mm_ind0 = 0;
if golden360_ind = . then golden360_ind = 0;
if in_wfh_ind = "N" then in_wfh_ind_nm = 0;
if in_wfh_ind = "Y" then in_wfh_ind_nm = 1;
if ifsr_ind = "N" then ifsr_ind_nm = 0;
if ifsr_ind = "Y" then ifsr_ind_nm = 1;
if msg_opt_nm = "N" then msg_opt_nm1 = 0;
else if msg_opt_nm = "Y" then msg_opt_nm1 = 1;
else if msg_opt_nm = "" then msg_opt_nm1 = .;
/* if cll_abnd_ind = "N" then cll_abnd_ind_nm = 0; */
/* if cll_abnd_ind = "Y" then cll_abnd_ind_nm = 1; */
if callback_off_ind = "Y" then callback_off_ind_nm = 1;
else if callback_off_ind = "" then callback_off_ind_nm = 0;
if callback_requeue = "N" then callback_requeue_nm = 0;
if callback_requeue = "Y" then callback_requeue_nm = 1;
if expired = . then expired = "NM";
run;
/* drop variables for correlation and > 30% missing */
data model_data1;
set model_data;
drop
qtd_wait_flag Lngst_Hld_Cnt Tlk_Tm Spclst_Tlk_Tm Support_Call_Cnt Intr_Sec_Ct
Initial_Queue_Wait_Time Callback_Cnt Frst_Callback_Ts_ind SM_tenure mm_ind srvy_rspn cust_age
in_wfh_ind ifsr_ind msg_opt_nm callback_off_ind callback_requeue Cll_Abdn_Ind Msg_Ft_Ds
Cust_Typ_Cd InCall_Typ_Cd IN_Delta_Tenure FN_Delta_Tenure in_wfh_ind_nm expired
sm_prf_airport;
run;
/* bin continuous variables */
proc hpbin data = model_data1 output = team4.binout_delta numbin = 10 bucket WOE ;
input acw_tm est_qtd_sec_tm ivr_sec_tm hndl_sec_ct tot_cust_wait_time;
target repeat_after;
id call_id;
run;
/* copy of original removing now binned vars */
data model_data2;
set model_data1;
drop acw_tm est_qtd_sec_tm ivr_sec_tm hndl_sec_ct tot_cust_wait_time;
run;
/* sort and merge */
proc sort data = team4.binout_delta out = binout_delta;
by call_id;
run;
proc sort data = model_data2;
by call_id;
run;
/* merge */
data team4.model_merge46;
```

```
merge
model_data2 binout_delta;
by call_id;
run;
/* temp */
data model_46;
set team4.model_merge46;
run;
proc freq data = model_46;
tables sm_tier_call / nocol norow nocum missing;
run;
/* variable macros */
%let aic_var =
FN_WFH_Ind FnAgt_Loc_Nm InAgt_Loc_Nm SM_TIER_Call SM_gender SM_seniority_sort day_of_week
month_of_year within_12 BIN_Acw_Tm BIN_Est_Qtd_Sec_Tm BIN_Hndl_Sec_Ct BIN_IVR_Sec_Tm
BIN_Tot_Cust_Wait_Time Hld_Cnt MM_ind0 MM_tier Spclst_Cnt Support_Call_Hld_Sec_Ct
Support_Call_Queue_Sec_Ct Trft_Cnt baggage boarding_pass book_a_ticket booking_upgrade
callback_accpt_ts_ind callback_off_ind_nmcallback_requeue_nm cancellation
card_features_benefits care_complaint change_booking change_details checkin_issues
child_ticket companion_certificate delete_cancel_account ecredit_redemption existing_change
external_transfer flight_delayed flight_status flight_transfered fraudulent_activity
golden360_ind guc_ruc_upgrade hold_flag ifsr_ind_nm irop_maintenance irop_weather length_flag
login_issues long_hold_flag medallion_question medical_fare missed_flight missing_low_tier
missing_vipmsg_opt_nm1 name_correction notification_received online_website other post_baggage
pre_baggage promo_question refunds_request retro_credit same_day_change schedule_change
seat_selection shift skymiles specialty_fare ssr support support_flag ticket_upgrade
tot_wait_flag travel_requirements trip_extras upgrade;
/* run model */
proc hplogistic
data = model_46;
class &aic_var.;
model
repeat_after (event = '1') = &aic_var.;
selection method = stepwise (select = aic choose = aic);
run;
%let odd_var =
Hld_Cnt MM_tier Support_Call_Queue_Sec_Ct Trft_Cnt baggage boarding_pass book_a_ticket
booking_upgrade callback_accpt_ts_ind cancellation care_complaint change_booking
change_details child_ticket companion_certificate delete_cancel_account flight_delayed
flight_status flight_transfered guc_ruc_upgrade ifsr_ind_nm irop_maintenance irop_weather
login_issues medallion_question medical_fare missed_flight missing_low_tier missing_vip
name_correction notification_received online_websiteother post_baggage pre_baggage
promo_question refunds_request retro_credit same_day_change schedule_change seat_selection
shift skymiles specialty_fare ssr ticket_upgrade travel_requirements upgrade FN_WFH_Ind
FnAgt_Loc_Nm InAgt_Loc_Nm InAgt_Queue_Nm SM_gender day_of_week month_of_year within_12
BIN_Acw_Tm BIN_Est_Qtd_Sec_Tm BIN_Hndl_Sec_Ct BIN_IVR_Sec_Tm BIN_Tot_Cust_Wait_Time;
/* odds ratios */
ods output CLOddsWald = team4.odds_data416;
proc logistic data = model_46;
class
Hld_Cnt (ref = '0')
MM_tier (ref = '0')
Trft_Cnt (ref = '0')
```

```
baggage (ref = '0')
boarding_pass (ref = '0')
book_a_ticket (ref = '0')
booking_upgrade (ref = '0')
callback_accpt_ts_ind (ref = '0')
cancellation (ref = '0')
care_complaint (ref = '0')
change_booking (ref = '0')
change_details (ref = '0')
child_ticket (ref = '0')
companion_certificate (ref = '0')
delete_cancel_account (ref = '0')
flight_delayed (ref = '0')
flight_status (ref = '0')
flight_transfered (ref = '0')
guc_ruc_upgrade (ref = '0')
ifsr_ind_nm (ref = '0')
irop_maintenance (ref = '0')
irop_weather (ref = '0')
login_issues (ref = '0')
medallion_question (ref = '0')
medical_fare (ref = '0')
missed_flight (ref = '0')
missing_low_tier (ref = '0')
missing_vip (ref = '0')
name_correction (ref = '0')
notification_received (ref = '0')
ssr (ref = '0')
ticket_upgrade (ref = '0')
travel_requirements (ref = '0')
upgrade (ref = '0')
BIN_Est_Qtd_Sec_Tm (ref = '1')
shift (ref = '2')
FN_WFH_Ind (ref = 'N')
FnAgt_Loc_Nm (ref = 'ATL')
InAgt_Loc_Nm (ref = 'ATL')
InAgt_Queue_Nm (ref = 'SkyMiles Service_SSS')
sm_gender (ref = 'M')
day_of_week (ref = 'Tuesday')
month_of_year (ref = '05M2023')
within_12 (ref = 'no')
BIN_Acw_Tm (ref = '1')
BIN_Hndl_Sec_Ct (ref = '1')
BIN_IVR_Sec_Tm (ref = '1')
BIN_Tot_Cust_Wait_Time (ref = '1')
sm_tier_call (ref = 'NM');
model
repeat_after (event = '1') =
Hld_Cnt MM_tier Trft_Cnt baggage boarding_pass book_a_ticket booking_upgrade
callback_accpt_ts_ind cancellation care_complaint change_booking change_details
child_ticket companion_certificate delete_cancel_account flight_delayed
flight_status flight_transfered guc_ruc_upgrade ifsr_ind_nm irop_maintenance
irop_weather login_issues medallion_question medical_fare missed_flight
```

```
missing_low_tier missing_vip name_correction notification_received online_website
other post_baggage pre_baggage promo_question refunds_request retro_credit
same_day_change schedule_change seat_selection skymiles specialty_fare ssr ticket_upgrade
travel_requirements upgrade BIN_Est_Qtd_Sec_Tm
shift FN_WFH_Ind FnAgt_Loc_Nm InAgt_Loc_Nm InAgt_Queue_Nm sm_gender
day_of_week month_of_year within_12 BIN_Acw_Tm BIN_Hndl_Sec_Ct
BIN_IVR_Sec_Tm BIN_Tot_Cust_Wait_Time sm_tier_call/ clodds = both;
run;
/************************ FOREST PLOTS ***************************/ title height=15pt
font='Times New Roman' "Odds of Producing a Repeat Call";
ods graphics on / width = 7in height=4in;
proc sgplot
data = first_five
noborder;
yaxistable or_ci /
label = "OR (95% CI)"
location = outside
position = right
valueattrs = GraphLabelText(
family='Times New Roman'
size=15)
valuejustify = left
LABELATTRS = (
family = 'Times New Roman'
size = 15
weight = bold)
labeljustify=left;
yaxistable Intent /
location = outside
position = left
valueattrs = GraphLabelText(
family='Times New Roman'
size = 15)
labeljustify = left
valuejustify = left
LABELATTRS = (
family='Times New Roman'
size=15
weight = bold)
pad = 20;
scatter
x = or
y = Intent /


thickness = 1.9 pt)
errorcapscale = .5
markerattrs = (
symbol = CircleFilled
size = 9
color = Black);
refline 1 /
axis = x;
```

```
xaxis
type = log
logstyle = logexpand
logbase = 10
label = "Odds Ratio and 95% (*ESC*){unicode '000a'x} Confidence Interval"
labelattrs=(size=12pt weight = bold) valueattrs=(size=15pt)
min = .2
max = 1.1
values = (.5,1,2)
valueshint
;
              yaxis
display = (nolabel noticks novalues noline)
offsetmax = .05
offsetmin = .05
valueattrs = GraphLabelText(
family = 'Times New Roman'
size = 12);
run;
/************************** QLIM MODEL **************************/
/* temp dataset */
data data;
set team4.repeat_model329;
run;
/* N = 2434711, 108 variables */
/* remove highly correlated variables/too many levels (pref airport and inagt queue) */
data data2;
set data;
drop
qtd_wait_flag Lngst_Hld_Cnt Tlk_Tm Spclst_Tlk_Tm Support_Call_Cnt Intr_Sec_Ct
Initial_Queue_Wait_Time Callback_Cnt Frst_Callback_Ts_ind SM_tenure mm_ind inagt_queue_nm
SM_prf_airport;
run;
/* bin variables */
proc hpbin data = data2 output = binout numbin = 5 bucket WOE ;
input Acw_Tm IN_Delta_Tenure FN_Delta_Tenure Est_Qtd_Sec_Tm IVR_Sec_Tm Hndl_Sec_Ct
Tot_Cust_Wait_Time Support_Call_Queue_Sec_Ct Support_Call_Hld_Sec_Ct Cust_age
close_expire;
target repeat_after;
id call_id;
run;
/* copy of original removing now binned vars */
data data3;
set data2;
drop Acw_Tm IN_Delta_Tenure FN_Delta_Tenure Est_Qtd_Sec_Tm IVR_Sec_Tm Hndl_Sec_Ct
Tot_Cust_Wait_Time Support_Call_Queue_Sec_Ct Support_Call_Hld_Sec_Ct Cust_age
close_expire;
run;
/* sort */
proc sort data = data3 out = data_sort;
by call_id;
run;
proc sort data = binout out = binout_sort;
```

```
by call_id;
run;
/* merge */
data bin_merge;
merge
data_sort binout_sort;
by call_id;
run;
/* drop call_id */
data model_data;
set bin_merge;
drop call_id;
run;
/* macro for all model variables */
proc contents data = model_data;
run;
%let all_var =
BIN_Acw_Tm BIN_Cust_age BIN_Est_Qtd_Sec_Tm BIN_FN_Delta_Tenure BIN_Hndl_Sec_Ct
BIN_IN_Delta_Tenure BIN_IVR_Sec_Tm BIN_Support_Call_Hld_Sec_Ct BIN_Support_Call_Queue_Sec_Ct
BIN_Tot_Cust_Wait_Time BIN_close_expire Callback_Off_Ind Callback_ReQueue Cll_Abdn_Ind
Cust_Typ_Cd FN_WFH_Ind FnAgt_Loc_Nm Hld_Cnt IFSR_Ind IN_WFH_Ind InAgt_Loc_Nm
InCall_Typ_Cd MM_ind0 MM_tier Msg_Ft_Ds Msg_Opt_Nm SM_TIER_Call SM_gender SM_seniority_sort
Spclst_Cnt Srvy_Rspn Trft_Cnt baggage boarding_pass book_a_ticket booking_upgrade
callback_accpt_ts_ind cancellation card_features_benefits care_complaint change_booking
change_details checkin_issues child_ticket companion_certificate day_of_week
delete_cancel_account ecredit_redemption existing_change expired external_transfer
flight_delayed flight_status flight_transfered fraudulent_activity golden360_ind
guc_ruc_upgrade hold_flag irop_maintenance irop_weather length_flag login_issues
long_hold_flag medallion_question medical_fare missed_flight missing_low_tier missing_vip
month_of_year name_correction notification_received online_website other post_baggage
pre_baggage promo_question refunds_request retro_credit same_day_change schedule_change
seat_selection shift skymiles specialty_fare ssr support support_flag ticket_upgrade
tot_wait_flag travel_requirements trip_extras upgrade within_12;
/* assess missingess/levels for variables and drop as needed */
proc freq data = model_data;
tables &all_var. / nocol nocum norow missing;
run;
data model_complete;
set model_data;
if Callback_Off_Ind = "" then Callback_Off_Ind = "N"; /* create no category */
if IN_WFH_Ind = "" then IN_WFH_Ind = "M"; /* create missing category */
if FN_WFH_Ind = "" then FN_WFH_Ind = "M"; /* create missing category */
if InAgt_Loc_Nm = "" then InAgt_Loc_Nm = "NA"; /* create missing category */
if FnAgt_Loc_Nm = "" then FnAgt_Loc_Nm = "NA"; /* create missing category */
if InAgt_Queue_Nm = "" then InAgt_Queue_Nm = "Missing"; /* create missing category */
if MM_ind0 = . then MM_ind0 = 0; /* create 0 category */
if MM_tier = . then MM_tier = 0; /* create missing category (others on 1-14) */
if SM_TIER_Call = "" then SM_TIER_Call = "NA"; /* create missing category */
if SM_seniority_sort = "" then SM_seniority_sort = "NA"; /* create missing category */
drop
Callback_ReQueue /* only 16 yes */
Cll_Abdn_Ind /* all no */
InCall_Typ_Cd /* 99.7% inbound */
```

```
Msg_Ft_Ds /* 99.9% success */
Msg_Opt_Nm /* only 0.23% yes (82% no, rest missing) */
Srvy_Rspn /* 82% missing, most remaining are 5 */
golden360_ind; /* 99.4% 0 */
run;
/* macro */
%let complete_var =
BIN_Acw_Tm BIN_Cust_age BIN_Est_Qtd_Sec_Tm BIN_FN_Delta_Tenure BIN_Hndl_Sec_Ct
BIN_IN_Delta_Tenure BIN_IVR_Sec_Tm BIN_Support_Call_Hld_Sec_Ct BIN_Support_Call_Queue_Sec_Ct
BIN_Tot_Cust_Wait_Time Callback_Off_Ind SM_TIER_Call flight_delayed flight_status
Cust_Typ_Cd FN_WFH_Ind FnAgt_Loc_Nm Hld_Cnt IFSR_Ind IN_WFH_Ind InAgt_Loc_Nm
Spclst_Cnt Trft_Cnt baggage boarding_pass book_a_ticket booking_upgrade fraudulent_activity
callback_accpt_ts_ind cancellation card_features_benefits care_complaint change_booking
change_details checkin_issues child_ticket companion_certificate day_of_week
delete_cancel_account ecredit_redemption existing_change external_transfer flight_transfered
guc_ruc_upgrade hold_flag irop_maintenance irop_weather length_flag login_issues
long_hold_flag medallion_question medical_fare missed_flight missing_low_tier missing_vip
month_of_year name_correction notification_received online_website other post_baggage
pre_baggage promo_question refunds_request retro_credit same_day_change schedule_change
seat_selection shift skymiles specialty_fare ssr support support_flag ticket_upgrade
tot_wait_flag travel_requirements trip_extras upgrade;
/* variable selection */
proc hplogistic
data = team4.all_complete426;
class &complete_var.;
model
repeat_after (event = '1') = &complete_var.;
selection method = forward (select = aic choose = aic);
run;
/* macro for selected variables */
%let all_select =
BIN_Acw_Tm BIN_Est_Qtd_Sec_Tm BIN_FN_Delta_Tenure BIN_Hndl_Sec_Ct BIN_IVR_Sec_Tm
BIN_Tot_Cust_Wait_Time SM_TIER_Call flight_delayed flight_status FnAgt_Loc_Nm InAgt_Loc_Nm
Spclst_Cnt baggage book_a_ticket callback_accpt_ts_ind care_complaint change_booking
child_ticket companion_certificate day_of_week existing_change flight_transfered
guc_ruc_upgrade irop_maintenance irop_weather missed_flight missing_low_tier missing_vip
post_baggage pre_baggage same_day_change schedule_change seat_selection shift skymiles
ticket_upgrade;
/* recode binary */
proc freq data = team4.all_complete426;
tables &all_select. / nocol nocum norow missing;
run;
%let all_binary =
flight_delayed flight_status baggage book_a_ticket callback_accpt_ts_ind care_complaint
change_booking child_ticket companion_certificate existing_change flight_transfered
guc_ruc_upgrade irop_maintenance irop_weather missed_flight missing_low_tier missing_vip
post_baggage pre_baggage same_day_change schedule_change seat_selection skymiles
ticket_upgrade;
/* macro to recode variables */
%macro recode_vars(data=, out=, varlist=);
data &out;
set &data;
%let num_vars = %sysfunc(countw(&varlist));
```

```
%do i=1 %to &num_vars;
%let var = %scan(&varlist, &i);
%let new_var = &var._recode;
&new_var = 1 - &&var;
%end;
run;
%mend recode_vars;
%recode_vars(data=team4.all_complete426, out=team4.all_recode426, varlist=&all_binary.);
/* macro with recoded variables */
%let all_recode =
BIN_Acw_Tm BIN_Est_Qtd_Sec_Tm BIN_FN_Delta_Tenure BIN_Hndl_Sec_Ct BIN_IVR_Sec_Tm
BIN_Tot_Cust_Wait_Time SM_TIER_Call flight_delayed_recode flight_status_recode FnAgt_Loc_Nm
InAgt_Loc_Nm Spclst_Cnt baggage_recode book_a_ticket_recode callback_accpt_ts_ind_recode
care_complaint_recode change_booking_recode child_ticket_recode companion_certificate_recode
day_of_week existing_change_recode flight_transfered_recode guc_ruc_upgrade_recode
irop_maintenance_recode irop_weather_recode missed_flight_recode missing_low_tier_recode
missing_vip_recode post_baggage_recode;
/* model file with only repeat_after and all_recode */
data team4.all_recode_model426;
set team4.all_recode426;
keep repeat_after &all_recode.;
run;
/* N = 2434711, 37 variables */
/* qlim model */
proc qlim data = team4.all_recode_model426;
class &all_recode.;
model repeat_after = &all_recode. / discrete(d=logistic);
output out = team4.all_qlim426 marginal;
run;
/* dataset with just meff variables */
data all_means;
set team4.all_qlim426;
drop
BIN_Acw_Tm BIN_Est_Qtd_Sec_Tm BIN_FN_Delta_Tenure BIN_Hndl_Sec_Ct BIN_IVR_Sec_Tm
BIN_Tot_Cust_Wait_Time FnAgt_Loc_Nm InAgt_Loc_Nm SM_TIER_Call Spclst_Cnt baggage_recode
book_a_ticket_recode callback_accpt_ts_ind_recode care_complaint_recode change_booking_recode
child_ticket_recode companion_certificate_recode day_of_week existing_change_recode
flight_delayed_recode flight_status_recode flight_transfered_recode guc_ruc_upgrade_recode
irop_maintenance_recode irop_weather_recode missed_flight_recode missing_low_tier_recode
missing_vip_recode post_baggage_recode repeat_after;
run;
proc means data = all_means mean noprint;
var _numeric_;
output out = team4.all_means427 mean= / autoname;
run;
proc transpose data = team4.all_means427 out = team4.all_means_transpose427;
var _numeric_;
run;
data team4.all_means_transpose427;
set team4.all_means_transpose427;
if _name_ = "_TYPE_" or _name_ = "_FREQ_" then delete;
run;
proc sort data = team4.all_means_transpose427;
```

```
by descending col1;
run;
proc print data = team4.all_means_transpose427 noobs;
run;
/* proc freq for increases*/
proc freq data = team4.all_complete426;
tables bin_tot_cust_wait_time*repeat_after;
where bin_tot_cust_wait_time = 5;
run;
proc freq data = team4.all_complete426;
tables existing_change*repeat_after;
where existing_change = 0;
run;
proc freq data = team4.all_complete426;
tables flight_delayed*repeat_after;
where flight_delayed = 0;
run;
proc freq data = team4.all_complete426;
tables guc_ruc_upgrade*repeat_after;
where guc_ruc_upgrade = 0;
run;
proc freq data = team4.all_complete426;
tables change_booking*repeat_after;
where change_booking = 0;
run;
proc freq data = team4.all_complete426;
tables book_a_ticket*repeat_after;
where book_a_ticket = 0;
run;
/* proc freq for decrease */
proc freq data = team4.all_complete426;
tables flight_status*repeat_after;
where flight_status = 0;
run;
proc freq data = team4.all_complete426;
tables baggage*repeat_after;
where baggage = 0;
run;
proc freq data = team4.all_complete426;
tables care_complaint*repeat_after;
where care_complaint = 0;
run;
```

R Code: Random Forest Modeling

```{r}
# Read the packages
library(tidyverse)
library(ROCR)
library(vip)
library(caret)
```

```
library(ranger)
library(rpart)
library(xgboost)
#library(rpart.plot)
library(ggplot2)
library(ROSE)
library("pdp")
library(corrplot)
library(psych)
library(GGally)
library(ggplot2)
```
```{r}
# Read the dataset
sm_continuous423 <- read_csv("/gpfs/user_home/os_home_dirs/pbasnet2/ds7900_spring24_team4/sm_continuous423.csv")
glimpse(sm_continuous423)
```
```{r}
# Rename the dataset
model_undersample<-sm_continuous423
glimpse(model_undersample)
```

```{r}
# Check number of levels for each variable in data frame
sapply(model_undersample, function(x) length(unique(x)))
```
```{r}
# remove variable with 1 level "Cust_Typ_Cd"
model_undersample1<- model_undersample%>%
  select(-Cust_Typ_Cd)
```
```{r}
# Check missing values in each variable
colSums(is.na(model_undersample1))
# Calculate missing percentage for each variable
missing_percentage_undersample <- colSums(is.na(model_undersample1)) / nrow(model_undersample1) * 100
print(missing_percentage_undersample)
```
```{r}
# Apply listwise deletion to remove rows with any missing values total data now =1938960
complete_cases <- na.omit(model_undersample1) # original=1885789 deleted 220,669
glimpse(complete_cases)
```
```{r}
RF_data_undersample<-complete_cases
glimpse(RF_data_undersample)
```
```{r}
# Check how balanced the target variable is
target_balance_model_undersample <- table(RF_data_undersample$repeat_after)
percentage_balance_model_undersample <- prop.table(target_balance_model_undersample) * 100
cat("Target variable balance (in percentage):\n") ## Target variable balance (in percentage):
print(percentage_balance_model_undersample) #0 is 93.7% and 1 is 6.3%
```
```{r}
# change the target variable to factor
RF_data_undersample["repeat_after"] <- lapply(RF_data_undersample["repeat_after"], as.factor)
#glimpse(complete_cases)
```
```{r}
# Change the order of factor levels for the "repeat_after" variable
```

```
RF_data_undersample$repeat_after<- relevel(RF_data_undersample$repeat_after, ref = "1") # Recheck which factor level is associated with '1' and '0'
levels(RF_data_undersample$repeat_after) # total data=2434711
```
```{r}
# Check number of levels for each variable in data frame
sapply(RF_data_undersample, function(x) length(unique(x)))
```
```{r}
# Convert categorical variable to factor  using "lapply" function
cat_undersam <- c("IN_WFH_Ind","InAgt_Loc_Nm","FN_WFH_Ind",
                  "FnAgt_Loc_Nm","IFSR_Ind","SM_TIER_Call",
                  "Callback_Off_Ind","SM_gender","SM_seniority_sort",
                  "day_of_week","month_of_year","within_12")
RF_data_undersample[cat_undersam ] <- lapply(RF_data_undersample[cat_undersam], as.factor)
glimpse(RF_data_undersample)
```
```{r}
# 0 is norepeat and 1 is repeat
table(RF_data_undersample$repeat_after)# 1=120388   0= 1818572
```
```{r}
# Check and modify class levels of the outcome variable
levels(RF_data_undersample$repeat_after) <- make.names(levels(RF_data_undersample$repeat_after))
```
```{r}
# Count of target variable
table(RF_data_undersample$repeat_after) # x1 is  104880 is 6.2% and x0=1560240 is 93.8%
prop.table(table(RF_data_undersample$repeat_after))
```
```{r}
## Split the Data into a 80/20 Training/Testing Set for Undersampling the majority class in full dataset ##
set.seed(123)
p_split_undersamp<- rsample::initial_split(RF_data_undersample,prop=0.8)
train_data_undersamp <- rsample::training(p_split_undersamp )#data=486943
test_data_undersamp<- rsample::testing(p_split_undersamp )
```
```{r}
# Count of target variable
table(train_data_undersamp$repeat_after) # x1 =96387 is 6.2% and x0=1454781  is 93.8%
prop.table(table(train_data_undersamp$repeat_after))
```
Undersample the majority class
```{r}
set.seed(123)
# Identify the majority class
majority_class_undersamp <- levels(train_data_undersamp$repeat_after)[2]
majority_class_undersamp

# Get the actual sample size of the majority class
majority_size_undersamp <- sum(train_data_undersamp$repeat_after == majority_class_undersamp)
majority_size_undersamp # X0=1248305
train_under<- ovun.sample(repeat_after ~ .,data=train_data_undersamp, method="under")
train_under<- train_under$data
table(train_under$repeat_after) #    X0=83619    X1=83791  #total 192,559
```
```{r}
# Change the order of factor levels for the "repeat_after" variable for train_under
train_under$repeat_after<- relevel(train_under$repeat_after, ref = "X1") # Recheck which factor level is associated with 'X1' and 'X0'
levels(train_under$repeat_after)
```
```
```

```{r}
levels(test_data_undersamp$repeat_after)
```
```{r}
# Check number of levels for each variable in data frame
sapply(train_under, function(x) length(unique(x)))
```
```{r}
# After undersampling  var "external_transfer" has just one level so delete it
train_under1 = train_under %>%
select(-external_transfer  ) # Toatal vars 85
```
Random Forest Model
```{r}
#?ranger()
## Identify hyper-parameters
# 1.mtry (Randomly selected predictor)
# 2.splitrule
# 3.min.node size
# Specify values for hyper-parameters
# mtry --> generally start with around square-root of variables (82 parameters)
sqrt(82) # 9.2

my.mtry = c(2,4,7,8,9)
#splitrule
my.rule = "gini"
# min.node.size --> default is 1 for classification
my.nodes = c(1,3,5)
# create tuning grid

my.grid = expand.grid(mtry = my.mtry,
                      splitrule = my.rule,
                      min.node.size = my.nodes)
my.grid

#Select an appropriate evaluation metric (after oversampling dataset is imblance so use "accuracy")
my.metric = "Kappa"

#Train model over hyperparameters
set.seed(2)
rf.tune_undersamp = caret::train(repeat_after ~ ., data =train_under1,
                     method = "ranger",
                     metric = my.metric ,
                     importance = "impurity",
                     trControl = trainControl(classProbs =  T,method = "cv", number = 5) ,
                     tuneGrid = my.grid) # grid of hyperparameters
rf.tune_undersamp

rf.tune_undersamp$results %>% arrange(desc(Kappa))

## Select best hyperparameters based on evaluation metric
rf.tune_undersamp$bestTune
```

```{r}
## Use tuned model to predict test sample
prob.tune_rf_undersamp = predict(rf.tune_undersamp, test_data_undersamp,type = "prob")
head(prob.tune_rf_undersamp)
```
```{r}
## Confusion Matrix for best performing random forest model using "test sample"
pred_undersamp = predict(rf.tune_undersamp, test_data_undersamp, type = "raw") #ranger version of 'class'
```

```
conf_undersamp = caret::confusionMatrix(data = pred_undersamp, reference = test_data_undersamp$repeat_after)
conf_undersamp
# TP FN
# FP TN
t(conf_undersamp$table)
```
```{r}
# Random forest model variable importance scores were based on improvement in gini impurity
# Extract scaled variable importance scores
rf_importance_undersamp<- varImp(rf.tune_undersamp, scale = TRUE)
print(rf_importance_undersamp)
```
```{r}
# Create a data frame for Random Forest variable importance
rf_importance_undersamp_df <- data.frame( Variable = rownames(rf_importance_undersamp$importance), Importance =
rf_importance_undersamp$importance$Overall)
```
```{r}
# Sort the data frame by Importance in descending order for a meaningful plot
rf_importance_undersamp_df <- rf_importance_undersamp_df %>%
  filter(Importance >= 8)

VarImp_plot<- rf_importance_undersamp_df %>%
  ggplot(aes(x = reorder(Variable, Importance), y = Importance)) +
  geom_bar(stat = "identity", fill = "orange") +
  geom_text(aes(label = ifelse(Importance > 8, sprintf("%.1f", Importance), "")),
            size = 2.5, color = 'black', vjust = 2) +
  labs(title = "Random Forest Variable Importance (Gini Impurity)",
       x = "Variable",
       y = "Importance")+
  scale_y_continuous(breaks = seq(0, 100, by = 10))+
  theme_minimal() +
  theme(plot.title = element_text(hjust = 0.5,size=10),
        axis.text.x = element_text(angle = 45, hjust = 1,size = 6))
VarImp_plot
#ggsave("/ds7900_spring24_team4/variable_importance.png", plot = p, width = 8, height = 4)
```

```{r}
# ROCR package for ROCR package for Random forest model for test dataset "pimaIndinDiab.test"
prediction_undersamp = ROCR::prediction(prob.tune_rf_undersamp[,1],test_data_undersamp$repeat_after) #first column of pred
prob
class(prediction_undersamp)
prediction_undersamp # mainly used as an intermediate object
str(prediction_undersamp)
# Plot an ROC curve based on prediction on test
roc_undersamp= ROCR::performance(prediction_undersamp, "tpr", "fpr")
str(roc_undersamp) #measure, x.measure (y-axis is first --> measure)
plot(roc_undersamp)
```
```{r}
# Nicer plot for Random forest model
df.roc_undersamp= data.frame(fpr = as.vector(unlist(roc_undersamp@x.values)), tpr =
as.vector(unlist(roc_undersamp@y.values))) #head(df.roc_rf)
roc_plot_undersamp <-ggplot() +
  geom_line(data = df.roc_undersamp, aes(x = fpr, y = tpr, color = "Random Forest Model"), linewidth = 1) +
  geom_abline(lty = "dashed", alpha = 0.5, color = "gray25", linewidth = 1.2) + scale_color_manual(values = c("darkred",
"darkblue")) +
  ggtitle("ROC: Random Forest Model") +
  guides(color = guide_legend(title = element_blank())) +
  theme_minimal() +
  theme(plot.title = element_text(hjust = 0.5, size=10), legend.text = element_text(size = 8))
```

```
roc_plot_undersamp
```
```{r}
#AUC of ROC curve for Random Forest Model of test dataset
auc_undersamp= ROCR::performance(prediction_undersamp, measure = "auc")
str(auc_undersamp)
auc_undersamp@y.values %>% unlist()
```
```{r}
# Coordinates for the annotations
undersamp_annotation<- data.frame(x = 0.7, y = 0.5, label = "AUC: 0.651 ")
# Add annotations to the plot
roc_plot_undersamp +
  geom_text(data = undersamp_annotation, aes(x = x, y = y, label = label), color = "darkred",fontface = "bold", size = 3)
```

Python Code: Repeat flag creation, XG Boost Modeling and Phi Correlations

Repeat Flag creation

```
import pandas as pd

import numpy as np

import random

import warnings

from time import time

from tqdm import tqdm

import os

import gc

from sys import getsizeof

np.set_printoptions(formatter={'all': lambda x: f'{x:4}'})



# In[2]:

pd.set_option('display.max_rows', 500)

pd.set_option('display.max_columns', 500)

warnings.filterwarnings('ignore')
```

```
# In[3]:

df = pd.read_csv('mult_new_2.csv')

df.drop(columns = ['call_indicator'], axis = 1, inplace = True)


# In[5]:

names = df.columns

data = df.to_numpy()

print(f'The size of Pandas dataframe is {getsizeof(df)/(1024*1024)} MB')

print(f'The size of NumPy aarya is {getsizeof(data)/(1024*1024)} MB')


# In[6]:

data[:, 0] = data[:, 0].astype(int)

data[:, 6:] = data[:, 6:].astype(int)

# Set dtype of columns 3, 4, and 5 to float before assigning NaN

data[:, 3:6] = np.nan


# In[7]:

dates = np.array([np.datetime64(item) for item in data[:, 2]])

# ## Defining the labeling function

# Here are column indices:

# - phone number - 0

# - call ID - 1
```

```
# - timestamp - 2
# - original - 3
# - repeat_after - 4
# - repeat - 5
# - all intents begin from 6 and till the end


# In[8]:
def labeler(data, distance = 12):
    result = np.zeros((1, data.shape[1]))
    data_len = len(data)


    while data_len > 0:
        dates = np.array([np.datetime64(item) for item in data[:, 2]])


        first_start_call = dates[0]
        first_start_call_array = np.full_like(dates, first_start_call)
        np.set_printoptions(formatter=None)
        time_difference = dates - first_start_call_array
        window = time_difference <= pd.Timedelta(hours = distance)
        print(len(window))
        print(len(data))
        data = np.column_stack((data, window))
        the_window = data[:, -1]
```

```
    neighbors = data[the_window == True]


    # BRANCH ONE


    # call is original, has no repeats. Its original gets 1, others do not change
    # the current row gets dropped from data and loop restarts
    if len(neighbors) == 1:
        neighbors[0, 3] = 1
        print(f'There are {result.shape} in result and {neighbors.shape} shape in neighbors')
        if neighbors.shape[1] != 56:
            neighbors = np.delete(neighbors, -1, axis=1)
        result = np.vstack((result, neighbors[:, :-1]))
        data = data[1:]
        data = np.delete(data, -1, axis=1)


        # UPDATE
        print(data_len)
        data_len = len(data)
        print(data_len)


        if isinstance(data, np.ndarray):
            print(f'The are {len(data)} observations left')
        else:
```

```
            print('\n\nThe end!')
            break



        # ANOTHER BRANCH
        # There are neighbors
        else:
            print(f'The neighbors are {neighbors.shape}\n')
            neighbors[:, 6:-1] += neighbors[0, 6:-1]
            neighbors_with_2 = neighbors[(neighbors[:, 6:-1] == 2).any(axis=1)]


            # Now if neighbors_with_2 contains only 1 observation, it means the first row got added to itself
            # and there are no repeats
            # In this case we add the first row of neighbors_with_2 to the result dataframe and update dataframe data
            # to continue the loop
            # And of course the value original gets 1 for the first row of neighbors
            if len(neighbors_with_2) == 1:
                neighbors_with_2[0, 3] = 1


                print(f'There are {result.shape} in result and {neighbors.shape} shape in neighbors')
                if neighbors_with_2.shape[1] != 56:
```

```
            neighbors_with_2 = np.delete(neighbors_with_2, -1, axis=1)
        result = np.vstack((result, neighbors_with_2[:, :-1]))
        # Now drop the first row of neighbors from the data, other neighbors must be retained
        indices_to_keep = np.where(data[:, 1] != neighbors_with_2[0, 1])
        data = data[indices_to_keep]
        data = np.delete(data, -1, axis=1)
        # UPDATE
        print(data_len)
        data_len = len(data)
        print(data_len)


    # Finally, here if neighbors_with_2 contains several observations. This means first row is should have repeat_after
    # equal to 1 and all remaining columns will receive value 1 to column repeat. Once assignment is done, entire
    # dataframe neighbors_with_2 will be appended to result and then all rows from neighbors_with_2
    # will be dropped from data, and then loop restarts
    elif len(neighbors_with_2) == 0:
        continue
    else:
        neighbors_with_2[0, 4] = 1
        neighbors_with_2[1:, 5] = 1
```

```
            if neighbors_with_2.shape[1] != 56:
                neighbors_with_2 = np.delete(neighbors_with_2, -1, axis=1)
                print(neighbors_with_2[1, :])
            print(f'There are {result.shape} in result and {neighbors.shape} shape in neighbors')
            result = np.vstack((result, neighbors_with_2[:, :-1]))

            data_call_id = data[:, 1]
            neighbors_with_2_call_id = neighbors_with_2[:, 1]
            indices_to_keep = ~np.isin(data_call_id, neighbors_with_2_call_id)
            data = data[indices_to_keep]
            data = np.delete(data, -1, axis=1)

            # UPDATE
            print(data_len)
            data_len = len(data)
            print(data_len)

    zero_rows = np.all(result == 0, axis=1)
    result = result[~zero_rows]
    return(result)

# In[9]:
unique_groups = np.unique(data[:, 0])
```

```
len(unique_groups)


# In[10]:

def split_list(lst, n):
    k, m = divmod(len(lst), n)
    return (lst[i * k + min(i, m):(i + 1) * k + min(i + 1, m)] for i in range(n))
# In[11]:

split_groups = list(split_list(unique_groups, 10))


# In[12]:

def saver(num):
    unique_groups_short = split_groups[num]
    savename = 'mult2_' + str(num) + '.csv'
    return unique_groups_short, savename


# # Now keep changing number here

# In[13]:

unique_groups_short, savename = saver(0)
# Ready mult2           9, 8, 7, 6, 5, 4, 3, 2, 1, 0
# Ready mult1           0, 1, 2, 3, 4, 5, 6, 7, 8, 9
# Troubled
# # Now running

# In[14]:
```

```
start = time()
result_arrays = []
for index, group in tqdm(enumerate(unique_groups_short), total = len(unique_groups_short)):
    print(f'We are at iteration {index}\n')
    group_data = data[data[:, 0] == group]
    print(f'There are {len(group_data)} call IDs!')
    labels = labeler(group_data)
    if labels.shape[1] != 55:
        print(labels)
        break
    result_arrays.append(labels)
    print(f'There are {len(result_arrays)} dataframes')

print(f'It took {time() - start} seconds for {len(unique_groups_short)} phone numbers')
final_result = np.vstack(result_arrays)
final_df = pd.DataFrame(final_result, columns=names)
final_df.to_csv(savename, index = False)
final_df.head(10)

# ## Cleaning everything
# In[15]:
get_ipython().run_line_magic('reset', '-f')
from IPython.display import clear_output
```

```
for i in range(50):
    clear_output(wait=False)
    print('Output cleared.')
```

XGBoost models

```
# In[1]:
import pandas as pd, numpy as np, warnings
import xgboost as xgb
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, roc_curve,
auc, confusion_matrix
import matplotlib.pyplot as plt

# In[2]:
pd.set_option('display.max_rows', 500)
pd.set_option('display.max_columns', 500)
warnings.filterwarnings('ignore')

# In[3]:
df_base = pd.read_sas('base_complete.sas7bdat')
df_nm = pd.read_sas('nm_complete.sas7bdat')
df_hvc = pd.read_sas('hvc_complete.sas7bdat')
```

```
# ## Building 50 models for High-Value customers
# In[4]:
def build_xgboost_model(df, dependent_var):
    X = df.drop(columns=[dependent_var])
    y = df[dependent_var]

    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=42,
stratify=y)

    model = xgb.XGBClassifier(scale_pos_weight=sum(y == 0) / sum(y == 1))

    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    y_pred_proba = model.predict_proba(X_test)[:, 1]

    accuracy = accuracy_score(y_test, y_pred)
    precision = precision_score(y_test, y_pred)
    recall = recall_score(y_test, y_pred)
    f1 = f1_score(y_test, y_pred)
    conf_matrix = confusion_matrix(y_test, y_pred)

    fpr, tpr, _ = roc_curve(y_test, y_pred_proba)
```

```
    roc_auc = auc(fpr, tpr)


    plt.figure()
    plt.plot(fpr, tpr, color='darkorange', lw=2, label='ROC curve (area = %0.2f)' % roc_auc)
    plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
    plt.xlim([0.0, 1.0])
    plt.ylim([0.0, 1.05])
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title('Receiver Operating Characteristic')
    plt.legend(loc="lower right")
    plt.show()

    print(f'The accuracy is {accuracy:.2f}, the precision is {precision:.2f}, recall is {recall:.2f}, F1
score is {f1:.2f} and confusion is {conf_matrix}')


    return {
        'model': model,
        'accuracy': accuracy,
        'precision': precision,
        'recall': recall,
        'f1_score': f1,
        'confusion_matrix': conf_matrix,
```

```
        'auc':roc_auc
    }


# In[5]:
all_accuracy, all_recall, all_precision, all_f1, all_auc = [], [], [], [], []
# In[15]:
the_df = df_nm
the_intents = the_df.loc[:, 'baggage':'upgrade']
the_intents = list(the_intents.columns)
 # In[16]:
# Start a loop
counter = 1
the_models = {}
for intent in the_intents:
    print(f'\nWe are working with intent {intent} which is {counter}-th in list of intents')
    counter += 1
    df = the_df[the_df[intent] != 0]
    df.drop(columns = the_intents, inplace = True, axis = 1)


    if len(df) != 0:
        df = df.select_dtypes(include=['number'])
        if len(df['repeat_after'].value_counts()) == 2:
            try:
```

```
            results = build_xgboost_model(df, 'repeat_after')
            the_models[intent] = results
            all_accuracy.append(results['accuracy'])
            all_recall.append(results['recall'])
            all_precision.append(results['precision'])
            all_f1.append(results['f1_score'])
            all_auc.append(results['auc'])
        except:
            print(f'The model for intent {intent} failed - not enough observations!\n')
 # In[17]:
print(counter)
# 48 + 49 + 48 = 145
# In[10]:
hvc_models = the_models
# In[14]:
base_models = the_models
# In[18]:
nm_models = the_models
 # In[19]:
performances = pd.DataFrame({
    'Accuracy': all_accuracy,
    'Recall': all_recall,
    'Precision': all_precision,
```

```
    'F1 Score': all_f1,
    'AUC': all_auc
})
# In[20]:
avg_perf = performances.mean()
avg_perf.to_excel('avg_performances.xlsx')
# In[21]:
print(avg_perf)
```

Phi Correlations

```
import pandas as pd, numpy as np, matplotlib.pyplot as plt, warnings
from scipy.stats import ttest_ind
from tabulate import tabulate
from scipy.stats import chi2_contingency
# In[5]:
pd.set_option('display.max_rows', 500)
pd.set_option('display.max_columns', 500)
warnings.filterwarnings('ignore')
# In[7]:
df_base = pd.read_sas('base_complete.sas7bdat')
df_nm = pd.read_sas('nm_complete.sas7bdat')
df_hvc = pd.read_sas('hvc_complete.sas7bdat')
```

```
df_all = pd.read_csv('repeat_model329.csv')

# In[7]:

# Exclude date and object data types

df_nm = df_nm.select_dtypes(exclude=[object, 'datetime'])

df_base = df_base.select_dtypes(exclude=[object, 'datetime'])

df_hvc = df_hvc.select_dtypes(exclude=[object, 'datetime'])

df_all = df_all.select_dtypes(exclude=[object, 'datetime'])

# In[8]:

df_all

# # Running for all dataframes

# ### Try Cramer's V for skymile tier and repeat_after

# In[9]:

the_df = df_all

the_name = 'all_'

# In[10]:

varbin = []

phicorr = []

varnum = []

mean1 = []

mean2 = []

pvals = []


for column in the_df.columns:

    if column != 'repeat_after':
```

```
    if the_df[column].nunique() == 2:
        contingency_table = pd.crosstab(the_df['repeat_after'], the_df[column])
        chi2, p, dof, expected = chi2_contingency(contingency_table)
        phi_coefficient = np.sqrt(chi2 / the_df.shape[0])

        print(f"Phi coefficient between 'repeat_after' and '{column}': {phi_coefficient}")
        print("Contingency table:")
        print(tabulate(contingency_table, headers='keys', tablefmt='psql'))
        print('\n\n\n')
        varbin.append(column)
        phicorr.append(phi_coefficient)

    elif the_df[column].nunique() > 2:
        group0 = the_df[the_df['repeat_after'] == 0][column]
        group1 = the_df[the_df['repeat_after'] == 1][column]

        t_stat, p_value = ttest_ind(group0, group1, equal_var=False)
        mean_group0 = group0.mean()
        mean_group1 = group1.mean()
        print(f"Average of '{column}' for group 0 (repeat_after = 0): {mean_group0}")
        print(f"Average of '{column}' for group 1 (repeat_after = 1): {mean_group1}")
        print(f"T-test p-value between groups in '{column}': {p_value}")

        varnum.append(column)
```

```
        mean1.append(mean_group0)
        mean2.append(mean_group1)
        pvals.append(p_value)

        plt.figure(figsize=(6, 4))
        the_df.boxplot(column=column, by='repeat_after')
        plt.title(f'Boxplot of {column} by repeat_after')
        plt.suptitle('')
        plt.xlabel('repeat_after')
        plt.ylabel(column)
        plt.show()

binaries = pd.DataFrame({'Binary variable': varbin, 'Phi Correlation': phicorr})
ttests = pd.DataFrame({'Numeric variable':varnum, 'Group 0 mean':mean1, 'Group 1
mean':mean2, 'P-values':pvals})
binaries.to_excel(the_name + 'analysis_binaries.xlsx')
ttests.to_excel(the_name + 'analysis_ttests.xlsx')
# ## Outputting to Excel
# In[15]:
import pandas as pd
import numpy as np
from scipy.stats import chi2_contingency, ttest_ind
import matplotlib.pyplot as plt
from tabulate import tabulate
```

```
# Create an Excel writer
writer = pd.ExcelWriter('analysis_results.xlsx', engine='xlsxwriter')

for column in the_df.columns:
    if column != 'repeat_after':
        # Shorten the worksheet name if it's too long
        sheet_name_chi2 = f'Chi2_{column}'[:31]
        sheet_name_ttest = f'TTest_{column}'[:31]

        if the_df[column].nunique() == 2:
            # Perform the chi-squared test
            contingency_table = pd.crosstab(the_df['repeat_after'], the_df[column])
            chi2, p, dof, expected = chi2_contingency(contingency_table)
            phi_coefficient = np.sqrt(chi2 / the_df.shape[0])

            # Write the results to the Excel
            df_output = pd.DataFrame({
                'Phi coefficient': [phi_coefficient],
                'P-value': [p]
            })
            df_output.to_excel(writer, sheet_name=sheet_name_chi2, index=False)

            # Also write the contingency table
```

```
            contingency_table.to_excel(writer, sheet_name=sheet_name_chi2,
 startrow=len(df_output)+2)

        elif the_df[column].nunique() > 2:
            # Perform the T-test
            group0 = the_df[the_df['repeat_after'] == 0][column]
            group1 = the_df[the_df['repeat_after'] == 1][column]
            t_stat, p_value = ttest_ind(group0, group1, equal_var=False)

            # Calculate means
            mean_group0 = group0.mean()
            mean_group1 = group1.mean()

            # Write the results to the Excel
            df_output = pd.DataFrame({
                'Mean Group 0': [mean_group0],
                'Mean Group 1': [mean_group1],
                'T-test P-value': [p_value]
            })
            df_output.to_excel(writer, sheet_name=sheet_name_ttest, index=False)

            # Generate and save a plot
            plt.figure(figsize=(6, 4))
            the_df.boxplot(column=column, by='repeat_after')
```

```
        plt.title(f'Boxplot of {column} by repeat_after')
        plt.suptitle('')
        plt.xlabel('repeat_after')
        plt.ylabel(column)
        plt.savefig(f'{column}_boxplot.png')
        plt.close()

# Save the Excel file
writer.close()
# In[16]:
import os
import pandas as pd
import numpy as np
from scipy.stats import chi2_contingency, ttest_ind
import matplotlib.pyplot as plt
from tabulate import tabulate

# Define the path for the Excel file
file_path = 'analysis_results.xlsx'
# Create an Excel writer
writer = pd.ExcelWriter(file_path, engine='xlsxwriter')

for column in the_df.columns:
    if column != 'repeat_after':
```

```
        sheet_name_chi2 = f'Chi2_{column}'[:31]
        sheet_name_ttest = f'TTest_{column}'[:31]

        if the_df[column].nunique() == 2:
            contingency_table = pd.crosstab(the_df['repeat_after'], the_df[column])
            chi2, p, dof, expected = chi2_contingency(contingency_table)
            phi_coefficient = np.sqrt(chi2 / the_df.shape[0])

            df_output = pd.DataFrame({
                'Phi coefficient': [phi_coefficient],
                'P-value': [p]
            })
            df_output.to_excel(writer, sheet_name=sheet_name_chi2, index=False)
            contingency_table.to_excel(writer, sheet_name=sheet_name_chi2,
startrow=len(df_output)+2)

        elif the_df[column].nunique() > 2:
            group0 = the_df[the_df['repeat_after'] == 0][column]
            group1 = the_df[the_df['repeat_after'] == 1][column]
            t_stat, p_value = ttest_ind(group0, group1, equal_var=False)

            mean_group0 = group0.mean()
            mean_group1 = group1.mean()
```

```
        df_output = pd.DataFrame({
            'Mean Group 0': [mean_group0],
            'Mean Group 1': [mean_group1],
            'T-test P-value': [p_value]
        })
        df_output.to_excel(writer, sheet_name=sheet_name_ttest, index=False)

        plt.figure(figsize=(6, 4))
        the_df.boxplot(column=column, by='repeat_after')
        plt.title(f'Boxplot of {column} by repeat_after')
        plt.suptitle('')
        plt.xlabel('repeat_after')
        plt.ylabel(column)
        plt.savefig(f'{column}_boxplot.png')
        plt.close()

# Finalize and save the Excel file
writer.close()

# Print the absolute path of the file
full_path = os.path.abspath(file_path)
print("Excel file has been saved at:", full_path)
```