

Vrije Universiteit Amsterdam

CCmath BV



Master Thesis

Prediction of chat handling times and simulation of chat processes

Author: Hamid Abdelloui 2593193

1st supervisor: Joost Berkhout

daily supervisor: Alex Roubos CCmath BV

2nd reader: Ronald Meester

*A thesis submitted in fulfillment of the requirements for
the Master of Science degree in Business Analytics. The template for this thesis is based
on the VU Computer Science department standard format.*

January 17, 2022

Abstract

Nowadays companies have started to use chat systems alongside traditionally phone calls and e-mails to serve customers. These chat systems provide a live chat environment that has the advantage that it has the low threshold of e-mails, while also allowing the query to be resolved as fast as with phone calls. It is also advantageous for contact centers as these systems save costs compared to phone calls as they allow agents to serve multiple customers simultaneously.

However, the models that are created by the workforce management company CCmath require the average handling time (AHT) as a function of the concurrency. The more accurate this AHT input per concurrency level is, the more accurate the prediction of the required number of agents per time unit is. More accurate predictions prevent or diminish overstaffing and understaffing, and consequently lower operational costs.

This handling time can be derived directly when we want to know the handling time for a concurrency of one, but there is a phenomenon called censoring when we want to know the AHT for higher levels of concurrency. The handling times are censored because the concurrency changes during the chat, so we have no clear measurements of the handling time if a certain level of concurrency would persist. We also investigate the effect of concurrency on a quality of service measure, the sojourn time. An additional problem to be tackled in this thesis is the lack of data. To combat this a simulation was created that simulates the entire chat process in-depth to be able to generate new data.

To be able to determine the handling times per level of concurrency we have to either use aggregates or survival analysis techniques. In this thesis the method linear regression was used in the former situation. For survival analysis Kaplan-Meier, Cox regression with time-varying covariates and finally Bayesian survival analysis were applied. All of these models will yield a configuration, which is the AHT per level of concurrency according to that model. Two experiments are done: one for a medium sized contact center (experiment 1) and one for a small contact center (experiment 2). In both experiments we train using simulated data, but in experiment 1 the configurations are tested on real life data whereas experiment 2 tests on data obtained from the same simulation model.

In this thesis we find the Bayesian survival model to be the best scoring model, both in terms of accuracy and consistency. However, when obtaining configurations for the sojourn time, this model did not converge. In those cases the linear regression model is the best performing model. Based on these insights, a prototype is created in R that incorporates those models and chooses the best option based on the type of data the user provides.

Contents

1	Introduction	6
2	Problem statement and context	9
2.1	CCmath	11
2.2	Erlang Excel Addins	12
2.2.1	Chat model	13
3	Related Work	16
4	Data	18
4.1	Data Analysis	19
4.2	Factors	21
4.3	Data Wrangling	22
4.3.1	Counting process data structure	22
5	Models for estimating average handling and sojourn times in chats	25
5.1	Linear Regression	25
5.2	Survival Analysis methods	26
5.2.1	Kaplan-Meier	27
5.2.2	Cox Regression	30
5.2.3	Fitting the coefficients using Newton's Method	31
5.2.4	Bayesian Survival Analysis	33
6	Simulation of data	36
6.1	Details of the simulation	36

6.2	Arrival Process	36
6.3	Service Process	37
6.4	Tracking the customer through its lifetime	38
6.5	Verification	38
7	Experimental Setup	40
7.1	Simulating data	40
7.2	Training the models to get configurations	41
7.3	Testing the configurations on new data	42
8	Results	43
8.1	Configuration testing results	43
8.1.1	Experiment 1: Configurations for medium sized contact center . . .	45
8.1.2	Experiment 2: Configurations for small contact center	45
8.1.3	Linear regression results	46
8.1.4	Kaplan-Meier curves	47
8.2	Simulation results	48
9	Conclusion and recommendations	50
10	Manual of the prototype	53
10.1	Steps the user goes through	53
10.2	Example of running the program	54
	References	56
10.3	Graphs and Figures	60
10.4	Data Wrangling	60
10.5	Code of the simulation	63

1

Introduction

Many companies use call centers to service customer queries (inbound) or to reach out to customers (outbound). These call centers employ ‘agents’ or ‘customer service representatives’ working in shifts. The incoming calls are routed to an available agent. Inbound call centers have to hire a certain number of agents to ensure that calling customers are helped in a timely manner. In 2019 it was estimated that there were approximately 3 million agents employed in the United States. Although this number is expected to decline by 1% over the next ten years, it will remain an important industry [1]. Nowadays, many call centers have evolved into so-called contact centers which also allow customers to interact through different channels, most notably: chat. These chat systems provide customers with an instant messaging environment to interact with agents live.

An example of such a system is shown in Figure 1.1 on the website of bol.com, which is one of the largest online retailers in the Netherlands. Here, when the user visits the site, a chat window is immediately opened in the bottom right. First the user is helped by a chatbot where the user is asked to specify the problem or request they have. In Figure 1.1 among others the user can choose for *delivery information*, *broken product* and *product information*. In case the problem can be solved without human interaction, the bot can provide the needed information. Otherwise, the customer is put into the queue to be helped by an actual agent.

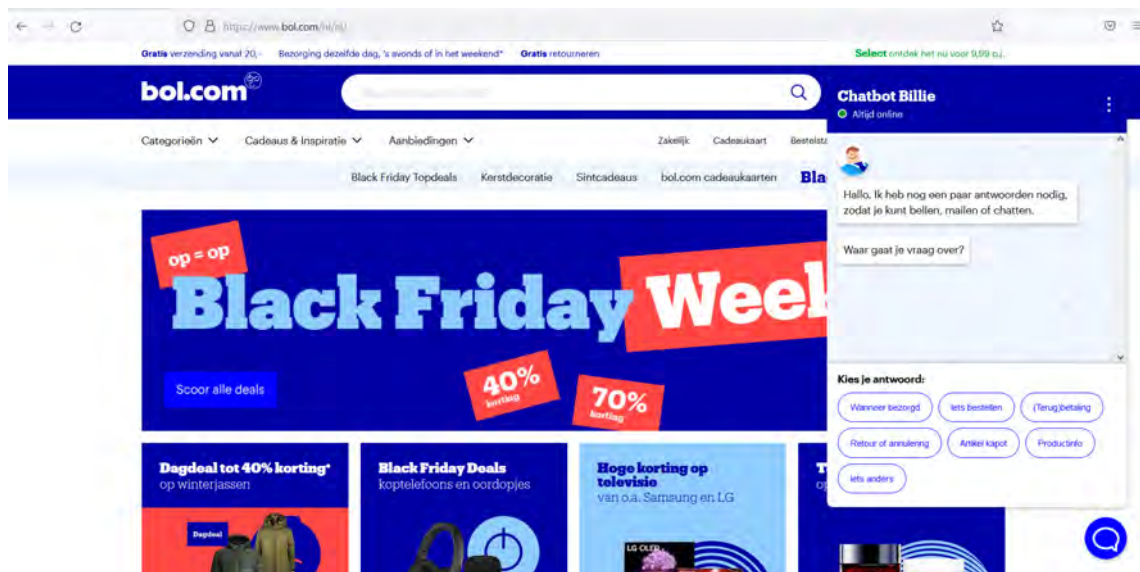


Figure 1.1: An example of a chat prompt on the website bol.com.

The availability of chat opens up completely new opportunities and considerations. Chat provides customers with more direct interactivity than emails, while also giving the customers more convenience compared to a phone call. It is also advantageous for the contact center as it requires less working hours because agents can handle multiple customers simultaneously compared to calls, where agents can service one customer at a time. Additionally, if one agent is idle, while another agent is helping many customers, the idle agent can pick up one of the conversations of the busy agent. Also, AI engines are available to assist the agent to even further increase productivity.

A company can save up to \$26 per chat instance compared to phone calls depending on the software and the applications used [2]. The figure given for calls is a cost of up to \$30 per call, whereas a chat instance can go for any number between \$4 and \$10 [2]. The usage of email costs approximately the same as using chats [3].

Chat systems also have some downsides. Research suggests that communication where the voice is involved like (video) calls creates stronger social bonds than communication means with no voice (chat, e-mail) [4]. Another possible issue is that people who are less

1. INTRODUCTION

sophisticated with technology are left out of customer support if phone calls are (mostly) replaced with live chat interactions. Furthermore, highly relevant to this thesis is the possibility that the concurrency has an effect on an agent's handling time. For instance, an agent cannot immediately reply to a customer's message because he is busy typing a reply to another customer's query.

The remainder of this thesis is organized in nine sections and an appendix. First, Section 2 defines the problem as well as the context, such as the company and the products that this thesis will improve. In Section 3 we will discuss previous research related to this thesis. In Section 4 the available data sets will be explained and analyzed. Section 5 contains an overview of the methods that will be applied. Additionally the simulation model that the methods are trained on is explained in Section 6. The experimental setup on two contact center sizes will be elaborated on in Section 7. In Section 8 the results of the models are presented. These results and their consequences will be discussed in Section 9. Finally, the appendix will contain relevant figures and tables and the code of the simulation. The prototype will be delivered alongside this thesis.

2

Problem statement and context

As previously mentioned the possibility of concurrency in chat will affect handling times. Because this "masks" the actual handling time of a single chat, the handling times are unknown and have to be predicted. The concurrency can change over the course of the chat, which prevents us from directly determining the handling time as a function of concurrency. The phenomenon that we cannot know the time-to-event because of the inability to follow the subject until the event happens or interference from an external variable is called censoring. In this thesis the interference happens because of concurrency, because a new chat is started during a chat we will not know how long the chat would have took with only the other chat. As a visual example we have Figure 2.1, where we see the jobs an agent does and the concurrency at each time point. The initial job starting at time 0 is 'interfered' by the jobs coming in at times 1 and 3. If we have detailed data that tracks the start and end times of the chat sessions per agent, we can determine the handling time if no other jobs would have come in. But what if only the job starting at time 1 arrived? In that case the approximation of the handling time becomes non-trivial.

In this thesis methods to calculate the handling times for chats will be compared and analyzed. The research question is: How do we calculate handling times for chats in contact centers? Additionally, we are interested in the quality of service as a function of concurrency as a second research question. There has, to my knowledge, been no previous work solely dedicated to predicting handling times for chat. Methods used in this thesis are linear regression, Kaplan-Meier, Cox regression and a Bayesian hazard model. Because there

2. PROBLEM STATEMENT AND CONTEXT

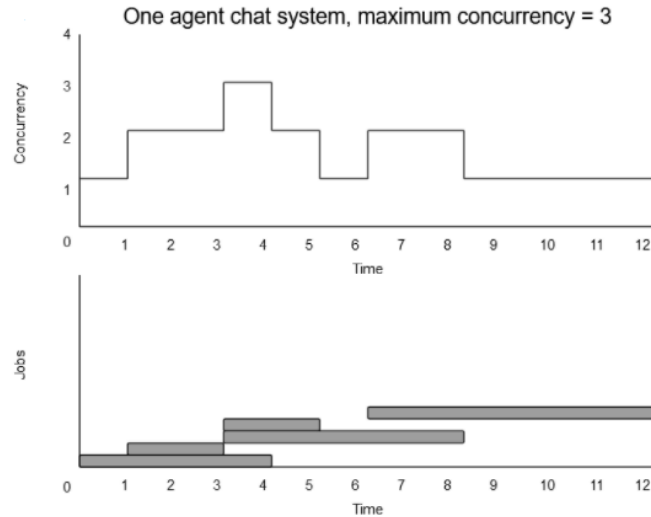


Figure 2.1: Example of the concurrency and corresponding jobs in a one agent system.

is no large scale in-detail data available, a simulation model based on the real-life situation is created to get data to train the model on. There is only data for a medium sized contact center for twelve days, which will be analyzed in the data analysis section and used to test on the configuration derived from the simulated data that mimics this real-life dataset.

With the results of the research in this thesis, the handling time input for the Excel add-in (see Section 2.2) for chats will then be automated, and more accurately determined. This results in more accurate staffing predictions. Sub-optimal staffing causes either too many agents or too few agent. In case there are not enough agents, the quality of service decreases. On the other hand, too many agents means unnecessarily high labor cost. Better staffing reduces the probability of getting into those situations as well as reducing the severity if the contact center gets into those situations. The remaining part of this section will elaborate on CCmath, the industry and where the results of this thesis will be applied.

2.1 CCmath

CCmath is a workforce management (WFM) company founded in 2005 by researchers from the Vrije Universiteit Amsterdam. WFM is defined as the process that aims to maximize performance levels, in particular activities involving managing a workload. In Figure 2.2 an overview of the WFM cycle is shown. An improvement in one of the steps means that the next step will be performed better as well (i.e., better planning leads to better schedules, better schedules leads to easier traffic management etc.). Here, the cell colored red (planning) is where this thesis will contribute.

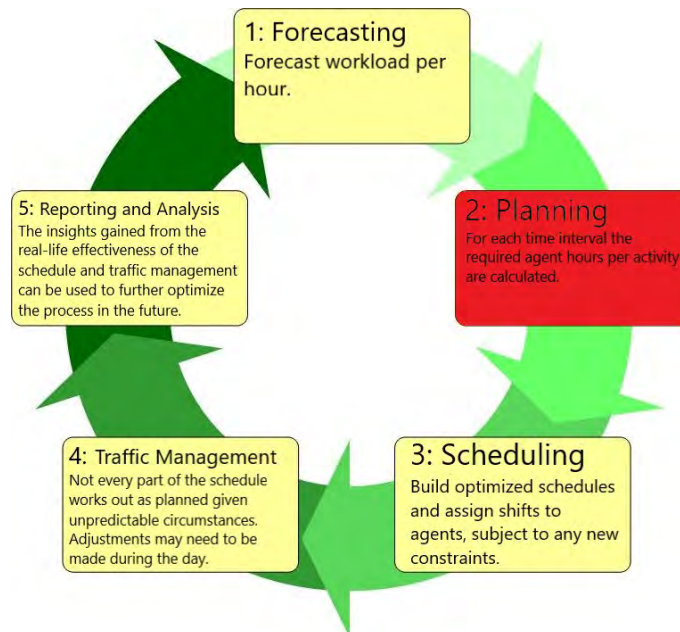


Figure 2.2: An overview of the WFM cycle.

The advanced mathematical modeling made by CCmath allows clients to make their WFM processes more efficient. Some of the products offered by CCmath are CCforecast, CCschedule and the Erlang add-ins. CCforecast uses advanced AI to offer best-in-class forecasts. An example of what CCforecast looks like is shown in Figure 2.3a. Here we see in orange the forecast, and in blue the historical values. CCschedule is scheduling software for the workforce. With the help of CCschedule, agents can be scheduled correctly and

2. PROBLEM STATEMENT AND CONTEXT

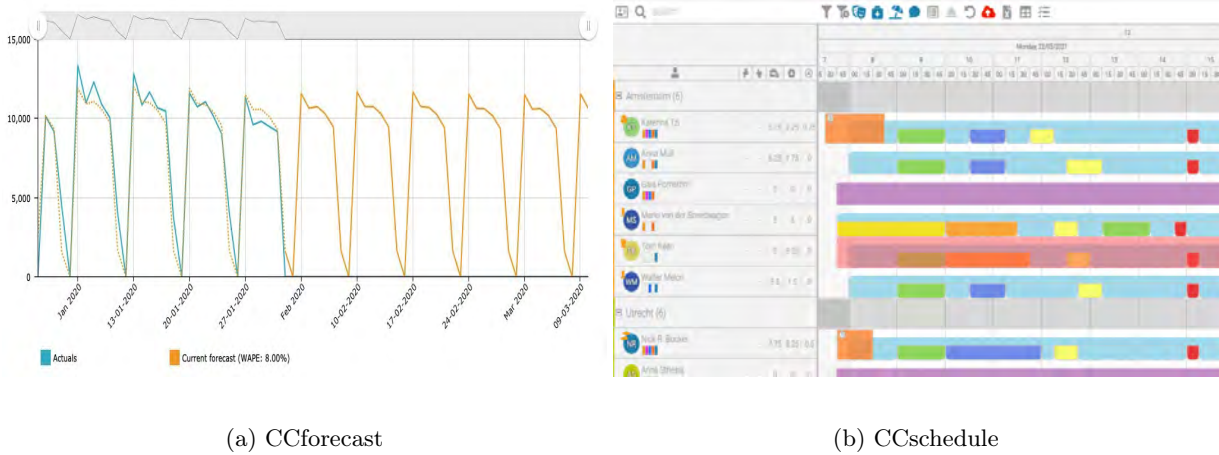


Figure 2.3: An overview of the software offered by CCmath.

fast automatically using advanced algorithms and AI. In Figure 2.3b we see how such an optimized schedule looks. The rows contain the agents and the different colors represent the tasks done by that agent throughout the shift. The Erlang Excel add-in that is used to calculate the number of agent hours needed will be explained in detail in Section 2.2.

2.2 Erlang Excel Addins

The add-in allows you to calculate all needed metrics in Erlang-C, Erlang-X, blending and chat models. Erlang C is a formula that calculates the number of agents needed given the arrival rate, handling time and the required service level. Erlang-X expands upon the Erlang-C model with queue abandonment after an exponentially distributed amount of time. The blended model combines inbound and outbound calls, where agents can work on outbound calls when they are idle and the service level is satisfied by a certain threshold. This suite of models provides all functions and parameters needed for contact centers to accurately predict capacity. These models can be optimized for Occupancy, Service Level, Abandonment, Patience, Redials, Concurrency and Average Speed of Answer (ASA). The functions as they can be chosen in the Excel worksheet are shown in Figure 2.4.

2.2 Erlang Excel Addins

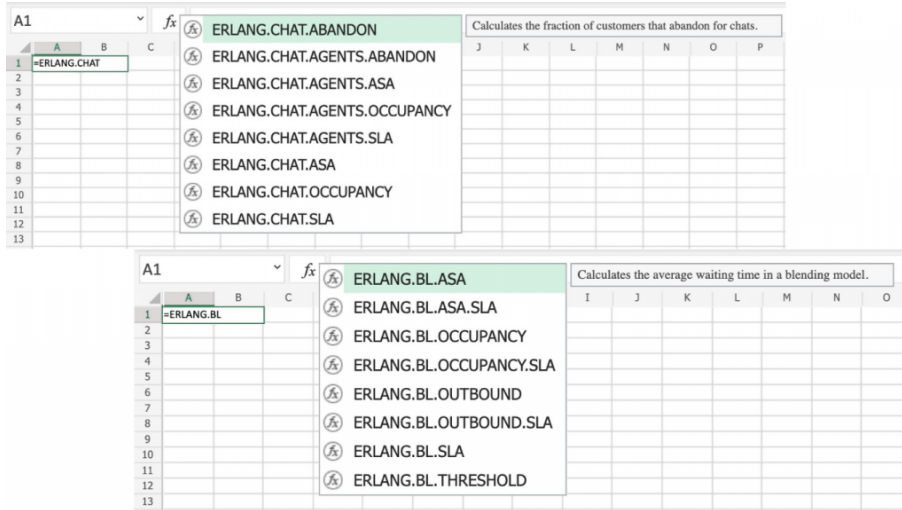


Figure 2.4: The Erlang add-in functions in Excel.

2.2.1 Chat model

The add-in contains a chat model that allows for agents to work on multiple chats in parallel. In this model there is an extra parameter that contains per concurrency level what the average handling time (AHT) is (i.e., [300,400,500] as input means that the agent can at most do three chats simultaneously, and for 1, 2 and 3 chats the handling times are 300, 400 and 500 seconds respectively).

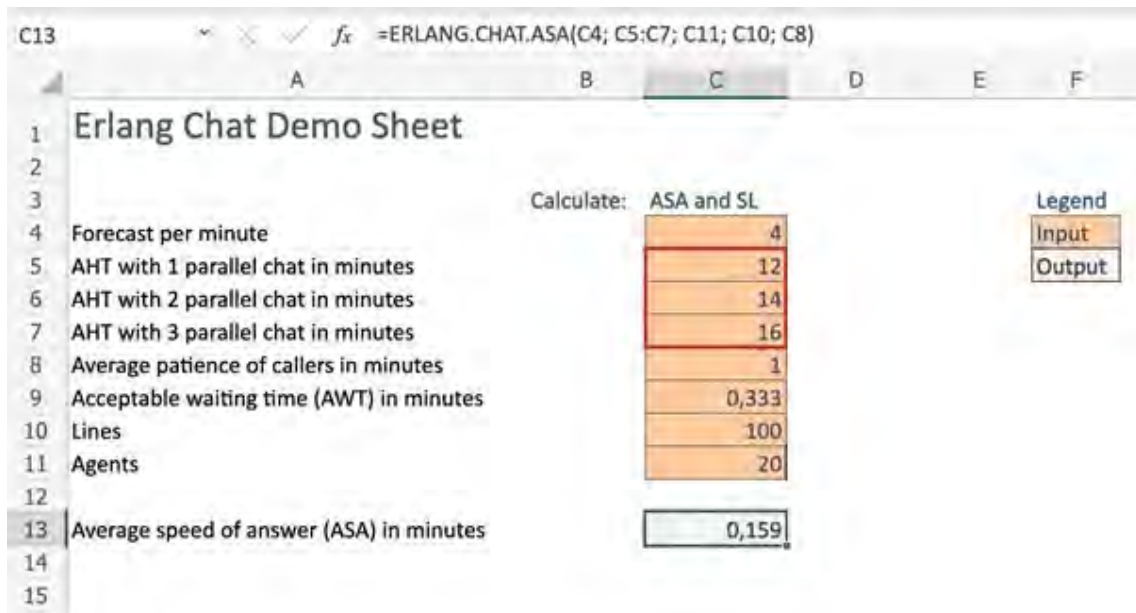
Example walkthrough Chat model

Figure 2.5 contains an example of an Excel sheet that uses the Erlang Chat model. The function used is `ERLANG.CHAT.ASA`, which calculates the average speed of answer in minutes for chats. The input cells (colored in orange) can be found in C4:C11. Please note: the input values that are expressed in units of time, all have to be expressed in the same unit. Table 2.1 explains all the input values needed.

In this example thus, we have a chat center with 4 arrivals per minute, an AHT of 12, 14 and 16 minutes for 1, 2 and 3 parallel chats respectively, an average patience of 1 minute and finally 100 lines for 20 agents. The function then calculates that the ASA for this contact center is equal to 0.159 minutes. There are identical functions that calculate all

2. PROBLEM STATEMENT AND CONTEXT

kinds of performance measures like service level, occupancy etc. as previously mentioned. Additionally, there is the function `ERLANG.CHAT.AGENTS.<performance measure>`, here `<performance measure>` can be for instance: ASA, service level or any of the other possible performance measures. This function calculates what the optimal number of agents is, such that the objective for the performance measure is satisfied. Determining C5:C7 would be done using the tool created by this thesis. Currently this is guessed by the customers themselves, but as described in this thesis, we will now have a data driven method.



The screenshot shows an Excel spreadsheet with the following data:

	Calculate:	ASA and SL
4	Forecast per minute	4
5	AHT with 1 parallel chat in minutes	12
6	AHT with 2 parallel chat in minutes	14
7	AHT with 3 parallel chat in minutes	16
8	Average patience of callers in minutes	1
9	Acceptable waiting time (AWT) in minutes	0,333
10	Lines	100
11	Agents	20
13	Average speed of answer (ASA) in minutes	0,159

The formula bar at the top shows: `=ERLANG.CHAT.ASA(C4; C5:C7; C11; C10; C8)`

Legend:
Input (orange)
Output (grey)

Figure 2.5: Example of what the Erlang add-in looks like.

Cell	Description
C4	The forecast of the number of arrivals per unit of time (here minutes).
C5:C7	The average handling times in minutes for each level of concurrency. The value in C5 is for a concurrency of one, C6 two simultaneous chats etc. The number of rows given as input decides the maximum concurrency for an agent.
C8	Average time callers are willing to wait in the queue in units of time (here minutes).
C9	The maximum allowed waiting time. This value can be used as input in place of C8. This means that the user has the choice to either use the patience (C8) or work with a maximum waiting time (C9).
C10	The number of lines is the limit on the total number of customers that can be in the system at the same time.
C11	The number of chat agents the contact center employs.

Table 2.1: Description of all the input cells needed for the Chat model from Figure 2.5.

3

Related Work

There has been previous work on adapting queueing models for call centers for live chats in contact centers. Enqvist and Svensson have adapted classical queueing models for chats [5]. The queue and service are modelled as a Markov process. The arrival process is modeled as a Poisson process with arrival rate λ , independently and identically distributed based on the day of the week and the interval. Customers that arrive have to either wait in line, or are routed to the agent that is busy with the fewest customers. The service rate is exponential and based solely on the number of customers the agent is serving simultaneously up to a maximum number to be chosen by the user. This queueing model comes down to a M/G/c/ ∞ queue. The proposed service rate is estimated with the parametric function $f(n)$, which gives the handling time for n customers simultaneously:

$$f(n) = \begin{cases} 0, & \text{when } n < 1. \\ na(1 - \frac{1}{1+be^{c(d-n)}}), & \text{when } n \geq 1. \end{cases} \quad (3.1)$$

Here, $n \in \mathbb{R}$ is the continuous version of the number of customers per agent, and a, b, c and d are non-negative model parameters. The desired shape (non-increasing service rate per customer) is achieved with this function. The parameters are to be fitted from data, which the authors mention to be non-trivial.

Enqvist and Svensson later expand upon this model by adding agent groups, routing rules and finding the stationary distribution of the Markov process. The routing is determined based on the performance measure and the concurrency levels. An important

part of the paper focuses on setting the planned level of maximum concurrency. For this, a good quality of service (QoS) measure needs to be determined. Traditionally, two QoS measures used are Average Speed of Answer (ASA) and Traffic Service Factor (TSF). The first one is the expected time the client has to wait on service to begin, whereas the second one gives the fraction of clients that start their service within a given time. For a chat system these measures can be made zero by opening an infinite number of chats, which will have a negative effect on the handling times. This suggests that a good QoS measure should include both a waiting time component and a handling time component. One such measure is the average headcount process (i.e., the number of clients in the system at a given time t) and equivalently expected sojourn time. The advantage of using the average sojourn time is that it is easy to use and Little's law may be applied. However it can be the case that an individual customer will have a longer waiting time, the tail events [6].

Two studies have looked at why customers choose for live chat over calls or e-mail [7]. The most important reason given is convenience, because the customer was under the impression it would be the fastest way to get what they needed. Other reasons given are familiarity, curiosity and positive recommendations. Other users prefer not to ask for help in person.

Other studies have looked at the possible levels of concurrency. According to [8] agents can do at least three chats simultaneously without performance degradation. Higher levels of concurrency have not been researched here, but it could well be a number higher than three. In [9] this number is put at ten or more. In the experiment from [8], the ASA of phone calls is almost four times as high as the ASA of chats because of the ability to multitask. The length of the chat conversation however is on average 19.4 minutes compared to 11.4 minutes for phone calls. The first call resolution (the percentage of customers that are helped in one call or chat, without the need to later call back or start a new instance) is equal at 90% for both. Given these numbers the economic and service advantage of chat support systems is made evident.

4

Data

For this thesis I was provided with a dataset from a real-life chat contact center. The dataset covers a period of 12 days, from November 26th 2011 to December 8th 2011. This dataset can be used to facilitate the creation of the simulation, for estimations of the chat process parameters like arrivals and censored agent handling times. Another dataset provided covers one day, but also includes focus time. Focus time is the total time the agent actually had the chat window open.

How sophisticated your models and simulation can be is limited by the data availability. The default data situation assumed in this thesis is on the chat level. For each chat the agent does, the start time, end time and agent name are known. Additionally, it can be useful to have additional data and factors like skill, customer name, focus time, waiting time, day of week and time of day. The linear regression model that will be introduced can be used with aggregated data, where the sum of handling times, agents present and the number of sessions per time interval suffices. There is also the ideal case where the entire chat process is tracked, which allows for more precise determination of the handling time. However, this advanced data was not available for this thesis and is therefore not included in the analysis. Another problem is that time spent on a task related to a chat is not counted or can be attributed to the wrong chat. For instance, an agent that has to create a case file based on a chat that took place earlier will not be counted as time spent on that chat, but can have effect on any other chats the agent is doing at that time.

Modeling in this data situation is omitted from this thesis.

Before any modeling is done, we will discuss some insights from the data first in Section 4.1. Some factors and their effect on handling time is analyzed in Section 4.2. Finally, Section 4.3 consists of data wrangling, which is the final step before the models and the simulation. Data wrangling is the transformation of raw data into data applicable as input for the models.

4.1 Data Analysis

Figure 4.1 contains a visualization of the shifts one agent does. The figure shows the number of customers that are serviced by this agent per hour, for the twelve days in the dataset. As can be seen in the graph there is a big difference in the number of customers the agent helps in an hour. We can also see what shifts the agent does.



Figure 4.1: Illustrative example of what the week of an agent looks like.

4. DATA

In Figure 4.2 it can be seen that there is fluctuation in arrivals between the days of the week. Most notably, traffic approximately halves during the weekend days (Friday, Saturday and Sunday). When we want to model the arrival process, splitting the week and weekend is therefore a straightforward differentiation.

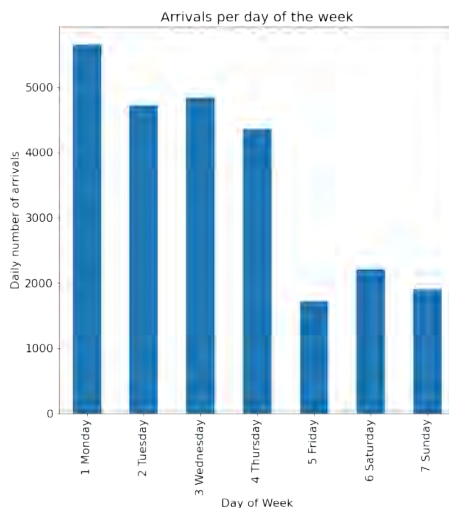


Figure 4.2: Number of arrivals per day of the week.

In Figure 4.3, the quantiles of the probability distribution of the handling times are plotted against the quantiles of the exponential distribution. The resulting plot shows a line approximately following $y = x$, slightly right-skewed. We can conclude that the exponential distribution is a reasonable assumption¹.

Counting all the chats ongoing at the time of pickup of a new chat we see that larger levels of concurrency occur less often as can be seen in Table 4.1. Based on this insight it is therefore important to minimize the bias caused by this skewed distribution.

¹Here, the chats are not split by concurrency. Nevertheless it shows that for general modeling, the exponential distribution is a good choice.

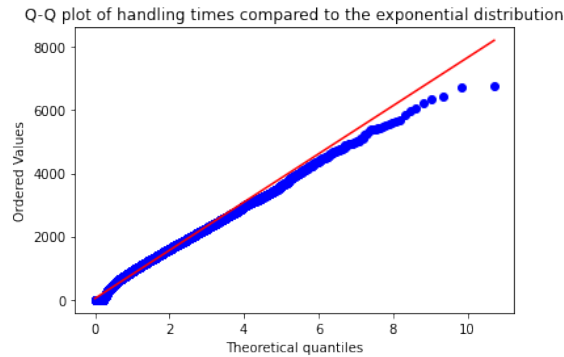


Figure 4.3: QQ-plot of the handling time against the exponential distribution.

Concurrency at pickup	Occurrence
0	5861
1	6906
2	12310
3	324
4	8

Table 4.1: Errors of the linear regression model per level of concurrency.

4.2 Factors

To get an idea of predictability of the available factors, the correlations between the factors are calculated in Table 4.3. A summary of what the factors mean can be found in Table 4.2.

Factor	Explanation
Waiting_Time_on_Queue	The time the customer spent in the queue (abbreviated as WTOQ).
Total_Time	The total time the customer spent in the system.
Handle_Time	Time from start service until departure (abbreviated as HT).
hr_round	The time rounded (i.e. 6:42 will become 7, where 6:29 will be 6).
shift_length	Total number of hours the current shift of the agent will/has lasted.
serviced	Total number of customers the agent has already helped in this shift.

Table 4.2: Explanation of the factors.

4. DATA

	WTOQ	total_time	HT	hr_round	shift_length	serviced
WTOQ	1.00	0.37	0.02	0.03	0.03	0.21
total_time	0.37	1.00	0.94	0.03	0.05	0.04
HT	0.03	0.94	1.00	0.02	0.05	-0.04
hr_round	0.03	0.03	0.02	1.00	0.18	0.21
shift_length	0.03	0.05	0.05	0.18	1.00	0.31
serviced	0.21	0.04	-0.04	0.21	0.31	1.00

Table 4.3: Correlations between the different factors.

In Table 4.3 it can be seen that shift duration has the highest correlation with the handling time outside of the total time. A possible explanation is that in a longer shift, the customer will do more chats and will therefore be able to do more "long handling time" customers. Customers serviced has a significant negative effect on waiting time. On the opposite, the number of customers serviced seems to have a minor positive effect on handling time. Here, we can say that after having helped a few customers the agent gets into a flow and becomes slightly faster. The hour seems to have almost no effect on handling time.

4.3 Data Wrangling

The raw data will be transformed before it can function as input for the methods. This can be applied to both real-life data as well as data that will be derived from the simulation to be created in section 6. Most importantly, some models require pre-processing to read the data.

4.3.1 Counting process data structure

In this section the transformation of the dataset that has data per chat, to an interval notation. The Cox regression model as well as the Bayesian model require the covariate values to be defined over the duration of the chat. Because we know all chats an agent does, we can determine at each time point how many customers the agent is helping. Every action during the chat that leads to a change in concurrency creates a new row with the start and end time of the interval with the covariate values during that interval. The

pseudo-code for this algorithm is included in the appendix in Algorithm 1. The algorithm is clarified using an example:

Example counting process transformation

The algorithm can be explained using an example. Consider the dataset as shown in Table 4.4, where each row represents a chat.

ID	Agent Name	Time Pickup	Time Departure
1	B	1	6
2	B	2	3
3	B	3	7
4	B	2	5

Table 4.4: Example of a dataset to convert into a counting process style.

Walking through the algorithm, we first take the row with ID=1. Births then becomes a subset of Dataset 4.4 where the pickup time is between 1 and 6, yielding ID's: 1, 2, 3 and 4. Deaths similarly is where the Departure time is between 1 and 6, yielding ID's: 1,2 and 3. A column of 1's is added to the births dataframe and a column of 0's to the deaths dataframe to later be able to distinguish them. Births and deaths are then concatenated into one dataframe, shown in Table 4.5.

Index	Agent Name	Time Pickup	Time Transition	Birth
0	B	0	1	1
1	B	1	1	1
2	B	1	2	1
3	B	2	2	1
4	B	2	4	0
5	B	4	5	0

Table 4.5: The intermediate dataframe.

Table 4.5 adds the column with the pickup times replicated starting from row 1, this column is called transition.

4. DATA

Now, the concurrency can be calculated. For the first row it simply adds one to the concurrency at pickup for that interval. After that, for each row where the birth column is 1, it adds one chat concurrent to the value in the previous row. Similarly, if the birth column is 0 (death), one chat is subtracted. In the end, we will have the resulting dataframe as in Table 4.6. This same process will then be repeated for the other rows, also for all other agents. These resulting tables are all concatenated, and can then be used as input for the Cox Time-Varying model.

Agent Name	Start Interval	End Interval	Concurrency
B	0	1	1
B	1	2	3
B	2	4	3
B	4	5	2

Table 4.6: The final resulting dataframe.

5

Models for estimating average handling and sojourn times in chats

In this report multiple methods to calculate the handling (and sojourn) time will be compared. As a simple method we will first apply a linear regression on aggregated data. Secondly, a Kaplan-Meier (KM) estimator will be used to calculate the handling times. Because KM does not allow for time-varying regressors, a Cox Regression will be applied. Finally, because there is a good prior available (exponential), Bayesian Survival Analysis is the final method that will be compared.

5.1 Linear Regression

The first and most simple method is a least squares linear regression model created in Excel with concurrency as the predictor and the average handling time as the dependent variable. Each hour the minutes of all agents are summed up, so that each hour counts as an instance such that the predictor c , representing the concurrency rate in the hour, is obtained:

$$c = \frac{\sum \text{Handling times in hour}}{\sum \text{Agent's time in hour}} \quad (5.1)$$

Here, the numerator is the sum of all the handling times logged for all agents in that hour. The denominator is the sum of agent's time of all agents in that hour. The agent's

5. MODELS FOR ESTIMATING AVERAGE HANDLING AND SOJOURN TIMES IN CHATS

time is the time the agent was logged in, ready and not idle in that hour. The average handling time (AHT) in the hour as the dependent variable is obtained as follows:

$$AHT = \frac{\sum \text{Handling times in hour}}{\sum \text{Number of Sessions in hour}} \quad (5.2)$$

Here, the denominator is the total number of sessions done by the agents in that hour (i.e., 10 agents doing 3 sessions in an hour each, will amount to 30 session done in that hour). The model has the following form:

$$AHT = \beta_0 + \beta_1 c + \epsilon \quad (5.3)$$

Here, β_0 and β_1 are the coefficients, c are the resulting values from Equation 5.1, and ϵ is the random error term.

Advantages:

- This method is applied to aggregated data. If only summarized data is available it is the only possible method. In case the data is more detailed, it is always possible to aggregate and also apply the method
- It is very easy to apply and can be done in Excel or any programming language.

Drawbacks:

- It can only learn (log-)linear functions, it would otherwise have to be extended to a generalized linear model.
- Cannot incorporate censored data

5.2 Survival Analysis methods

Survival analysis is a group of techniques applied when we want to know the time until an event happens. Most commonly the amount of time until the event happens is referred to as the survival time. This can be literal survival as in healthcare studies, but in the case

of this thesis it is the handling time. Many different methods would be available if the event occurred for all participants, but for some participants it is possible that the true time of the event is lost. This phenomenon is called censoring and can happen because of a few reasons, namely: the individual has not (yet) experienced the event, is lost for follow-up or a different event happens that makes follow-up impossible. For the handling times the last problem is the case because of concurrency. As the concurrency changes to a different value whenever the agent opens another chat, the handling time if that level of concurrency would persist is unknown [10].

This survival time is represented in a survival function, which can be interpreted as the proportion of participants that have not yet had the event happen (e.g. in this case ending the chat). Let T denote the survival time, then the survival function $S(t)$ is defined as: $S(t) = P(T > t) = 1 - F(t)$, where $F(t)$ is the distribution function of T . In this thesis you will also see the hazard function $\lambda(t)$ used which is derived from the survival function in the following manner: $S(t) = \exp(-\int \lambda(t))$. The hazard function is also known as the instantaneous failure rate and denotes the proneness to failure as a function of the age of the individual. The survival function, cumulative distribution function and the hazard function are mathematically equivalent, so if one of the functions is known, all three can be derived [11].

Survival Analysis will be applied here because we are interested in the duration until an event happens, namely: the departure of the customer. Because there is censoring, it becomes clear that survival analysis techniques are applicable to the problem we want to solve. These methods also allow us to choose the point from which the departure time is to be predicted, so we can use them when predicting handling time as well as sojourn time.

5.2.1 Kaplan-Meier

The statistical method Kaplan-Meier is used to estimate the time-to-event based on a factor. First introduced in the Journal of the American Statistical Association, it offers a method to analyze the survival of a subject when censoring takes place [12]. The technique is used to estimate the probability of (metaphorical) ‘death’ occurring during the lifetime

5. MODELS FOR ESTIMATING AVERAGE HANDLING AND SOJOURN TIMES IN CHATS

of a chat, patient, customer or other observable object [13]. An observation is censored when measurements are stopped before the time of the event. This can happen because the event occurs after the time window being observed or another variable affects the event. In this report the latter issue is the case, because a chat is opened with another customer, censoring the handling time the customer would have had if the agent only focused on their current chats.

We want to construct a survival function $p(t)$, where at each point t_i after the i th event took place, the probability of surviving past t is defined. The Kaplan-Meier estimator as a step function of the survival function is defined as follows:

$$\hat{p}(t) = \prod_{i=1, t_i \leq t} \left(1 - \frac{d_i}{n_i}\right) \quad (5.4)$$

Here, d_i is the number of deaths (i.e., the number of chats that have ended) that occurred at time t_i , and n_i is the number of individuals that are known to have survived up until just before that point (i.e., the number of chats that were still active at that point in time).

Example

An example of a Kaplan-Meier curve is shown in Figure 5.1. Here it can be seen that the survival decreases over time as expected. The treatments can be compared here side by side to come to the conclusion that treatment B is more adequate to extend survival if that is the goal notwithstanding other consequences.

From the survival curve $p(t)$, we can derive the median survival time for each level of concurrency. That is, the time since pick-up at which exactly 50% of the customers have ended their chat. This is simply done by calculating the inverse of the survival function in a dataframe. It is then sorted and searched for the value closest to the 0.5th percentile. Here it takes, as median survival time, the value that comes right after where the 0.5th percentile would be placed. Note that this gives the median and not the mean. However, it has been shown that the mean and median are not significantly different in Kaplan-Meier through a log rank test ($P = 0.08$) [15]. If we take Figure 5.1 as an example, in the

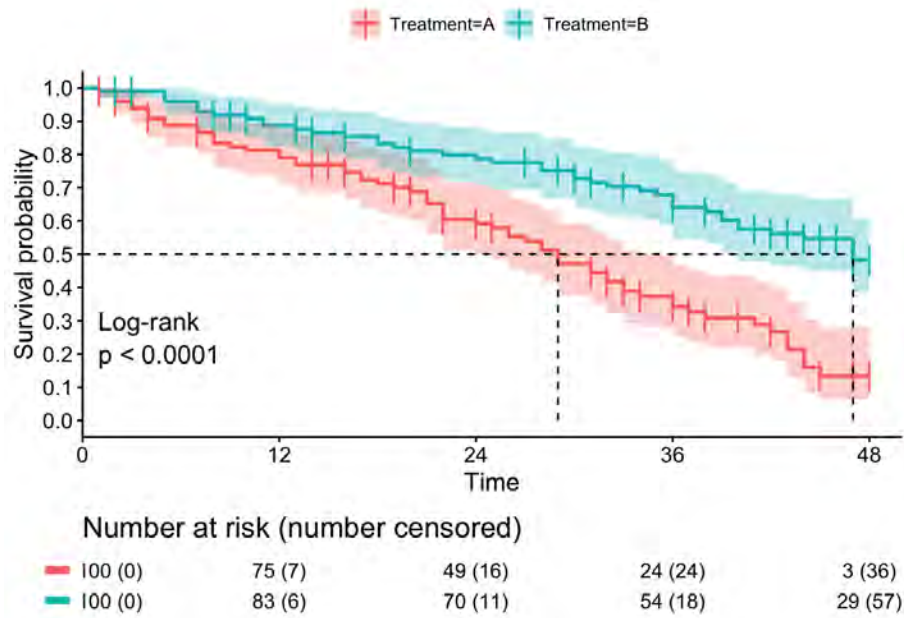


Figure 5.1: Example of a Kaplan-Meier survival function (from [14]).

dataframe with the survival function, the values are only given at the points where there is a change in survival. As can be seen in Figure 5.1, the survival times from a survival probability of approximately 0.47 to 0.51 are the same. That would mean that we look up the value closest to 0.50 in the dataframe. We take the first value we find larger than 0.50, which would be approximately 0.51. Here we find the median survival time to be 28 hours.

Advantages:

- Requires more detailed data than linear regression. Namely, the start and end times for each chat.
- Can incorporate censored data
- Allows for intuitive visualization of the survival curve.
- Kaplan-Meier is non-parametric, and thus requires fewer assumptions.

5. MODELS FOR ESTIMATING AVERAGE HANDLING AND SOJOURN TIMES IN CHATS

- It is one of the most popular methods for survival analysis [16].

Drawbacks:

- It can not incorporate time-dependent variables.
- It can only be evaluated per covariate value. Meaning: for each level of concurrency a new Kaplan-Meier estimator is required. This can make the method inconvenient in case there are many covariate values.
- As the model yields a survival function, there is no clear cut way to get to the mean. In this case the median is derived as a placeholder.

5.2.2 Cox Regression

Cox regression (also known as proportional hazards regression) is a method to investigate the effect of multiple variables upon the time-to-event of survival data. The method can be applied when the event is censored, either by measurement stopping before the death event, or external variables interfering which causes the event to be expedited or delayed. This method has been used in healthcare to predict events like deaths and diseases where there is a variable that changes over time such as medicine dosages [17, 18]. The model is used in this thesis for the same reason as concurrency is a time-varying variable: during the call it changes every time a chat is finished or a new customer is assigned to the agent. The model is adapted for time dependent covariates. A time dependent covariate is a covariate that may change over the observation period, which is the case for concurrency during a chat.

A distinction is made between internal and external covariates [19]. Internal covariates are time measurements on the subject, and are typically the output of stochastic processes. External covariates on the other hand are outside of the subject being observed, but can effect the failure process. External covariates can be further divided into external defined covariates and external ancillary covariates. External defined covariates are known before measuring starts (i.e., medicine dosages or water pressure). External ancillary processes are external stochastic processes. Concurrency in chats can be seen as a external ancillary

covariate, as arrival of other customers is an external process, but can effect failure.

The hazard function $\lambda(t|\mathbf{z})$ of the proportional hazards model is the probability of failure over time. The hazard function is defined in the original paper by Cox as follows [20]:

$$\lambda(t|\mathbf{z}) = \exp(\mathbf{z}\beta')\lambda_0(t) \quad (5.5)$$

where the variable t represents time and $\beta' = (\beta_1, \beta_2, \dots, \beta_p)$ are regression coefficients. $\lambda_0(t)$ here represents the baseline hazard. The baseline hazard is the hazard for an individual independent of the covariates and coefficients. Let z be a $p \times 1$ vector representing observations on p covariates in total. For the j th individual at time t we have $\mathbf{z} = (z_{1tj}, z_{2jt}, \dots, z_{pjt})$.

The model is extended for time varying covariates as follows [21]:

$$\lambda(t|Z(t)) = \exp(\beta'x + \gamma'zg(t))\lambda_0(t) \quad (5.6)$$

where β' are the coefficients for the fixed variables and γ' for the time varying covariates. $Z(t)$ represents the covariates:

$$Z(t) = [x_1, x_2, \dots, x_p, z_1g(t), z_2g(t), \dots, z_qg(t)], \quad (5.7)$$

The median is derived from the survival function in the same way as in the Kaplan-Meier: finding the 0.5th percentile in the inverse survival function. The coefficients are fitted using Newton's Method, which will be explained in section 5.2.3.

5.2.3 Fitting the coefficients using Newton's Method

To fit the coefficients β' , the Newton-Raphson iterative scheme is used. At each step n , the guess for β' , is updated according to the following formula [22]:

$$\beta_{n+1} = \beta_n - \frac{\lambda(\beta_n)}{\lambda'(\beta_n)} \quad (5.8)$$

5. MODELS FOR ESTIMATING AVERAGE HANDLING AND SOJOURN TIMES IN CHATS

This process is repeated until $\beta_{n+1} - \beta_n < \epsilon$, where ϵ is a pre-defined threshold for convergence. A visual representation of this scheme on another function can be seen in Figure 5.2.

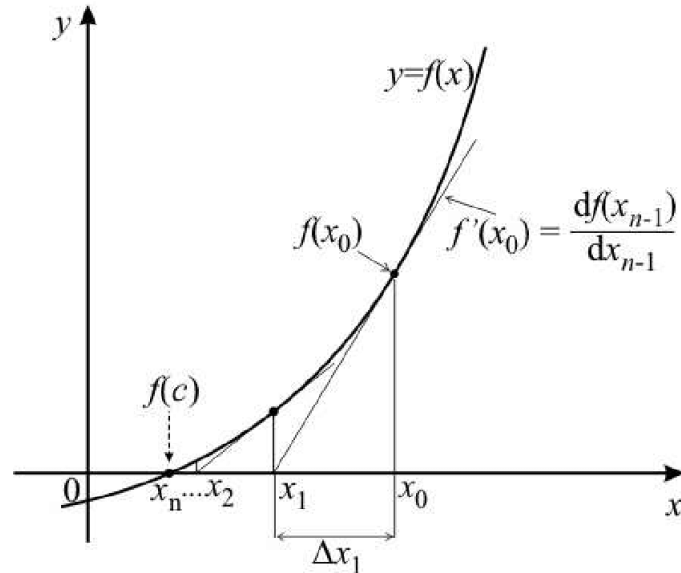


Figure 5.2: Illustrative example of what the Newton-Raphson scheme looks like.

Short Example Newton-Raphson scheme

Say we have the function $y = x^2 - 3$. We first need the derivative, which is $y' = 2x$. As initial guess, let us take $x = 2$. Filling in:

$$x_{new} = 2 - \frac{2^2 - 3}{2 * 2} = 1.75 \quad (5.9)$$

Now, as x_{old} we take 1.75:

$$x_{new} = 1.75 - \frac{1.75^2 - 3}{2 * 1.75} \approx 1.73 \quad (5.10)$$

One of the actual roots of the function is $\sqrt{3} \approx 1.73$, so we see that it comes very close after just two iterations. The difference between the first and second iteration is ~ 0.02 , if

we would have put this value as our stopping threshold, the algorithm would now terminate.

Advantages:

- If the initial guess in Newton's Method is too far off, it is possible it will not be able to converge.
- It is non-parametric.
- If the proportional hazards assumption is violated in the non-time varying Cox model, this model can be used [23].
- Can incorporate censored data.

Drawbacks:

- It does not allow for visualization of the survival curve.
- It requires the data to be transformed into a counting process.
- As the model yields a survival function, there is no clear cut way to get to the mean. In this case the median is derived as a placeholder.

5.2.4 Bayesian Survival Analysis

The previously mentioned Cox model is the most popular model for survival analysis, whereas a Bayesian approach is a novel approach [24]. When a good prior is available it has been shown to outperform the Cox Regression model [25]. Bayesian survival analysis can be done in two ways: either using a hazard scale or accelerated failure time. The hazard scale representation simply means that a hazard function is obtained: the rate of failure over time, which is the probability of the chat ending per time interval given that you have survived until that point. In the accelerated failure representation the covariate either accelerates or decelerates the life expectancy of an individual [26]. Because those models tend to be closely aligned, and the hazard model can be directly compared to the Cox model a hazard scale representation will be used with the hazard function $h_i(t)$:

5. MODELS FOR ESTIMATING AVERAGE HANDLING AND SOJOURN TIMES IN CHATS

$$h_i(t) = h_0(t)exp(\eta_i(t)) \quad (5.11)$$

where $h_0(t)$ is the baseline hazard equivalent to the baseline hazard in the Cox Model, and $\eta_i(t)$ is the linear predictor for individual i at time t . In our case, the linear predictor here incorporates time-varying covariates. The hazard function $h_0(t)$ represents the probability of failure over time similar to the Cox model.

The package in R used called `rstanarm` allows the distribution of the baseline hazard as a parameter (i.e. exponential, Weibull, Gompertz). However, as the differences between the setting of this parameter are small, the standard option: M-splines is chosen. The M-spline model in `rstanarm` is the method used by default. It makes use of splines to model the baseline hazard. Splines are piecewise defined polynomials on a set interval that is split in subintervals at knots. The "M" indicates that it is strictly positive. We get:

$$h_i(t) = \sum_{l=1}^L \gamma_l M_l(t, \mathbf{k}) exp(\eta_i(t)) \quad (5.12)$$

Here $M_l(t, k)$ denotes the l^{th} ($l = 1, \dots, L$) basis term for a cubic M-spline function evaluated at knot locations $k = k_1, \dots, k_J$. The variable γ_l is the l^{th} M-spline coefficient [27].

The median is derived from the survival function in the same way as in the Kaplan-Meier: finding the 0.5th percentile in the inverse of the survival function.

Advantages:

- As it requires data in exactly the same format as the Cox model, it is easy to use it when also applying the Cox model.
- The many different possible distributions allow it to fit a wide range of types of data.

Drawbacks:

- In the instance where the model was trained to predict quality of service (through sojourn time) using concurrency, the model was not able to converge.

5.2 Survival Analysis methods

- There is currently a lack of implementations of Bayesian Survival Analysis. The available packages `rstanarm` in R is the only one available.
- As the model yields a survival function, there is no clear cut way to get to the mean. In this case the median is derived as a placeholder.

6

Simulation of data

As there was no real world in-chat data available, this data was obtained with simulation. In the results section the results of the models will be compared using the simulated data to determine the optimal method. The simulation is done in Python with the help of Discrete Event Simulation (DES) using the package SimPy.

6.1 Details of the simulation

An entire day is simulated in seconds, with one agent and a stochastic in chat process. Time passes in the simulation with the use of the function in SimPy `env.timeout(x)` for x seconds, simulating the parts of the service process. Only when this function is called is the current time updated. The running time can be extended to have more robust results. As the system may take some time to reach equilibrium a warm-up period may be added. Similarly, at the end of the simulated period, the system has incomplete data. This can be helped by adding a cool-down period.

6.2 Arrival Process

The arrival process is modeled as a Poisson process. The inter arrival times are based on real data from the medium sized contact center, and are split by hour of the day. Every customer that arrives is either directly assigned to service if the agent is not doing the maximum number of chats simultaneously, otherwise the agent is put into an arrival

queue. Customers from this queue will be put into service whenever the agent becomes available again.

6.3 Service Process

The service process is split up into multiple stages, designed to replicate the actual service process. During service the chat goes through the stages as shown in Figure 6.1. The service alternates between customer and agent actions. Every interaction goes through steps 1 to 6, but after step 6 another interaction is generated with a certain probability each time. This probability can be manually changed, but is fixed for a run. At each step a duration is drawn from a uniform distribution with minimum and maximum values as shown in Figure 6.1.

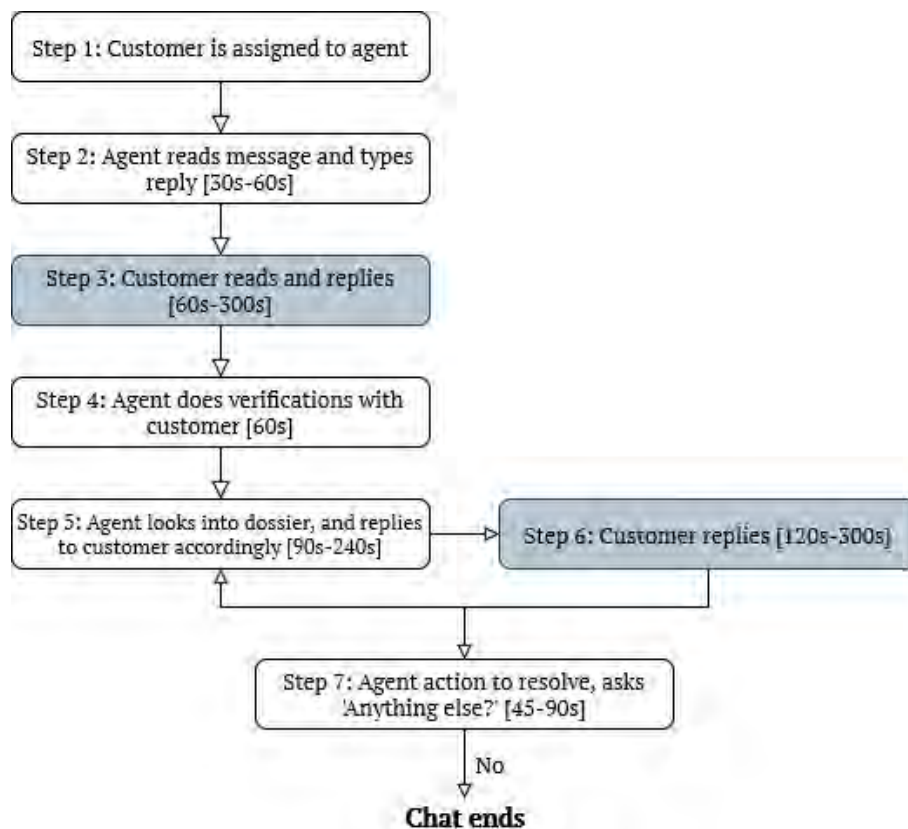


Figure 6.1: Steps gone through during the service of a customer.

6.4 Tracking the customer through its lifetime

Out of the simulation we would like to have a dataset with for each simulated customer, we have their arrival time, the concurrency at their arrival and at pick-up, the pick-up time and the departure time. This way, we end up with a dataframe that contains the same information as well as being in the same format as the dataset provided as described in the Data section. The customer is tracked as follows:

- At arrival, the arrival time and the concurrency at that time is obtained from the environment and written to the output dataframe.
- Once the customer is picked up by the agent, the time this happens as well as the concurrency is put into the dataframe. With the pick-up time, the waiting time can be determined anytime.
- Everytime the agent is focused on that particular chat, the time that is generated from the uniform distribution is noted. With this data, we can see how concurrency affects the agent's ability to handle a chat timely.
- Once the customer has gone through service and is let go, the departure time is noted. Now, we also know how long service took.

6.5 Verification

To verify the simulation, it is first written to mimic an Erlang Blocking model. Testing with different settings of arrival rates and service times, the resulting blocking probabilities are compared to the analytical Erlang B results. For each setting, the simulation is run for 40 runs, the results are then averaged to account for variability. An overview can be seen in Table 6.1.

From the results as shown in Table 6.1, we can conclude that the simulation accurately mimics the Erlang B for all settings. The largest deviation is for the contact center with few arrivals and fast service time with a 0.67 percent point higher blocking probability. With these small differences it can therefore be used to model the chat process.

Run	Arrival rate	Service time	Blocking simulation	Blocking Erlang B
1	24p/h	340 s	69.55%	69.40%
2	24p/h	90 s	37.15%	37.50%
3	6p/h	340 s	35.66%	36.17%
4	6p/h	90 s	13.71%	13.04%

Table 6.1: Resulting values of the verification of the simulation.

7

Experimental Setup

To compare the models two experiments are done. In experiment 1 the medium sized contact center is replicated in a simulation and will be tested on the real data set. Experiment 2 takes simulated data of a contact center with less traffic and tests the configuration on other data obtained from running the same simulation. Configurations refer to the handling and sojourn times given per level of concurrency after running the model. First the simulation settings are presented in Section 7.1. Then the method and settings of training the models are shown in Section 7.2. Section 7.3 explains how the configurations will be tested.

7.1 Simulating data

The simulation is ran for 42 hours (or equivalently 151200 seconds) not including a warm up and cool down period, both 1800 seconds. The interarrival time is set at 450 seconds, based on the arrival rate derived from the medium sized contact center dataset. The service process is described in Section 6. The maximum number of customers that the agent can service simultaneously is four. This number is based on discussions with an industry expert. At step 6 in the service process, the probability of the chat continuing is set to 10% each time. These values are slightly varied where the arrival rate is varied between 300 and 800, the service process is between 0.6x and 1.2x the speed shown in 6.1 and the probability of another query from the customer between 0 and 25%. These values

7.2 Training the models to get configurations

are chosen based on the values of the real data from the medium sized contact center. These settings are then run to each represent a day.

7.2 Training the models to get configurations

Because of the availability of only one real word data per chat, the models are trained on the simulation model from 6. Firstly, experiment 1 is a simulation modeled to mimic the arrivals and service in the medium sized contact center as discussed above. Secondly, in experiment 2 the models are trained on a contact center with less traffic, namely on average 1300 seconds as inter-arrival time. With these results we will be able to confirm that my approach works for more than one contact center size. The models are trained with the hyper-parameters as shown in Table 7.1.

Model	Configuration
Linear Regression	NA
Cox Proportional Hazards	Time varying covariates, penalizer = 0.2
Kaplan-Meier	NA
Bayesian Survival Analysis	M-Splines, chains = 2

Table 7.1: The models and the configurations.

The penalizer P for the Proportional Hazards model is used in the penalty term. The penalty term ensures that the coefficients do not blow up in size after fitting. The penalty term is equal to:

$$P(S(\beta')L + \frac{1}{2}\beta'^2(1 - L)) \tag{7.1}$$

with

$$S(x) = \frac{1}{1.5}(\log(1 + e^{-1.5x}) + \log(1 + e^{1.5x})) \tag{7.2}$$

Here, β' are the coefficients in the Cox Regression, L is the L1-ratio as a constant and S is the softabs function. The L1-ratio is between zero and one, and determines the combination of L1 and L2 regularization to be used. If the ratio is equal to zero, it is equivalent

7. EXPERIMENTAL SETUP

to L1 regularization, as the ratio grows more L2 regularization is added while the weight of L1 regularization is decreased until only L2 regularization remains at a ratio of 1. The difference between the two methods of regularization is that L1 adds absolute weighted values, whereas L2 takes the squared value of the coefficients. The function $S(x)$ is an approximation of the absolute value of the coefficients and is implemented to avoid slow conversion. This approximation has an approximation accuracy bound of at most $2\frac{\log 2}{1.5}$. The penalizer is set to 0.2 in my approach, as this penalty succeeds in keeping the coefficients small.

The Bayesian model in R is run with 2 chains with 2000 iterations of which 1000 warm-up iterations and 1000 sampling. In case this number of chains and iterations is not enough, the model gives a warning and the number of chains and iterations can be increased until convergence.

7.3 Testing the configurations on new data

To see how well the models yields good parameters, the parameters derived from experiment 1 will be tested on the real-life contact center data, whereas in experiment 2, the configurations will be tested on data simulated with the same parameters as the train data. The performance of the configurations derived from all models and contact centers will be judged using the median relative absolute error. This way, the results will be interpretable no matter the values of the handling and sojourn times and no matter the expertise of the end user. Predicting 200 higher from a handling time of 200 gives an error of 100% whereas for a handling time of 1600, being 200 off only adds an error of 12.5%.

8

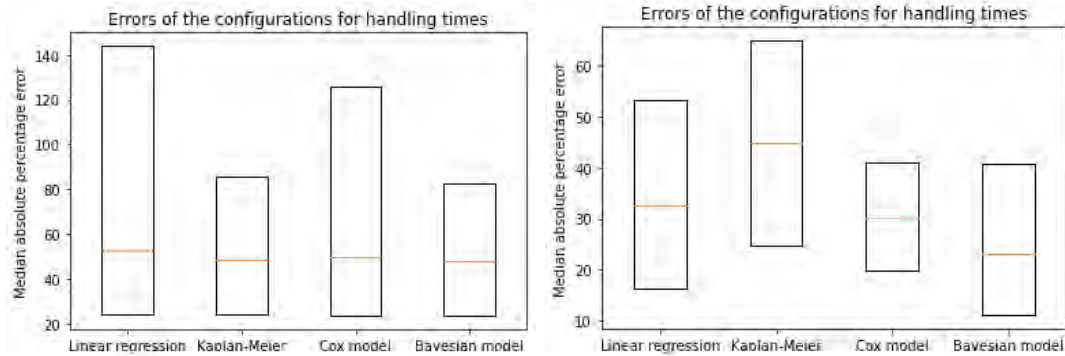
Results

This section will contain the results of the experiments and the simulation. The performance of the resulting configurations is compared. Configurations refer to the handling and sojourn times given per level of concurrency after running the model. First the errors of the resulting configurations will be compared in Section 8.1. Then an overview of the configurations is shown for experiment 1 both for handling time and sojourn time in section 8.1.1. The same overview is given for experiment 2 in Section 8.1.2. As the linear regression and Kaplan-Meier models give interesting visual results these are explained in Sections 8.1.3 and 8.1.4. The visual results of the Cox regression and the Bayesian model showed no noteworthy differences, they can be found in the appendix in Figures 10.6 and 10.7. Finally, we have a look at what happened during the simulation run in Section 8.2.

8.1 Configuration testing results

In Figure 8.1, the distribution of the relative absolute error when predicting the handling times is shown per model for both experiments. We also see in Figure 8.1 that for the medium sized contact center in experiment 1 all models are around the same median, but the linear regression and the cox regression the spread is significantly larger. The Bayesian model performs the best for both contact center, both when it comes to the median as well as the maximum error. The Kaplan-Meier is on-par with the Bayesian model for the medium sized contact center, but is the worst performing model for the small contact center in experiment 2.

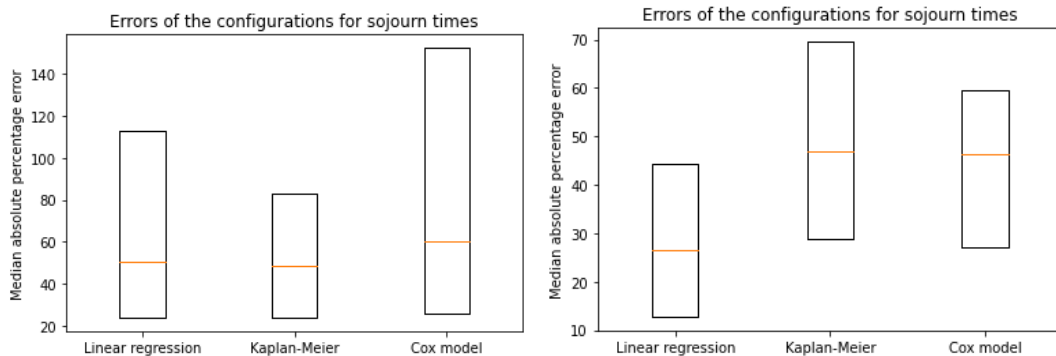
8. RESULTS



(a) Medium sized contact center

(b) Small contact center

Figure 8.1: Deviation for both contact center sizes when predicting handling times.



(a) Medium sized contact center

(b) Small contact center

Figure 8.2: Deviation for both contact center sizes when predicting sojourn times.

When we look at quality of service in Figure 8.2, we see a different picture. Here for medium sized contact centers, the Kaplan-Meier has the lowest median and spread, but in the small contact center it performs by far the worst. In both cases the linear regression model has a low median, and a relatively low spread. The Bayesian model is missing from the graph as it did not converge to any solution.

8.1.1 Experiment 1: Configurations for medium sized contact center

The configuration derived from a simulated medium sized contact center and tested on real life data is shown in Table 8.1. There are no results for the Bayesian model when predicting sojourn time as the model did not converge. Increasing the iterations caused running time issues.

Medium sized contact center	Predicted handling time per level of concurrency				Predicted sojourn time per level of concurrency			
Model	1	2	3	4	1	2	3	4
Linear Regression	1034	1198	1361	1525	748	1201	1654	2108
Kaplan-Meier	610	971	1565	3730	922	1337	1935	2533
Cox Time-varying Regression	949	1107	1326	1652	1656	1675	1680	1687
Bayesian Survival Analysis	627	940	1254	1881	-	-	-	-

Table 8.1: Resulting configurations of the handling and sojourn times per concurrency level for all the models in seconds.

Surprisingly, the models result in very different configurations. All models show an increasing handling and sojourn time for higher levels of concurrency. For sojourn time the Cox model shows very small increases for the levels of concurrency.

8.1.2 Experiment 2: Configurations for small contact center

The configuration derived from a simulated small contact center and tested on other simulated data with the same settings is shown in Table 8.2. Again, there are no results for the Bayesian model when predicting sojourn time as the model did not converge to any solution. Here too increasing the iterations caused running time issues.

Here, the Kaplan-Meier estimator shows some unexpected results, for a concurrency of two, the predicted handling time is almost half the handling time where concurrency is one. None of the other models have a decrease here. Also, for the Kaplan-Meier it predicts a lower sojourn time than its predicted handling time, where sojourn time should be equal to or larger than the handling time.

8. RESULTS

Small contact center	Predicted handling time per level of concurrency				Predicted sojourn time per level of concurrency			
Model	1	2	3	4	1	2	3	4
Linear Regression	866	1235	1605	1974	730	1321	1912	2503
Kaplan-Meier	1200	462	903	2423	448	843	1930	3728
Cox Time-varying Regression	444	547	708	1015	626	653	683	714
Bayesian Survival Analysis	514	1028	1028	2056	-	-	-	-

Table 8.2: Resulting configurations of the handling and sojourn times per concurrency level for all the models for the small contact center.

8.1.3 Linear regression results

In Figures 8.3 and 8.4, the linear regression models fitted are shown. For the medium sized contact center, the regression for the handling time has an R^2 of 76.0 and 96.7 for sojourn time. For the small contact center the model has a good fit as well, it reaches an R^2 of 71.5 for the handling time and 95.1 for the sojourn time. The high score for quality of service are consistent when looking at the median relative absolute difference, as the linear regression scores best there.

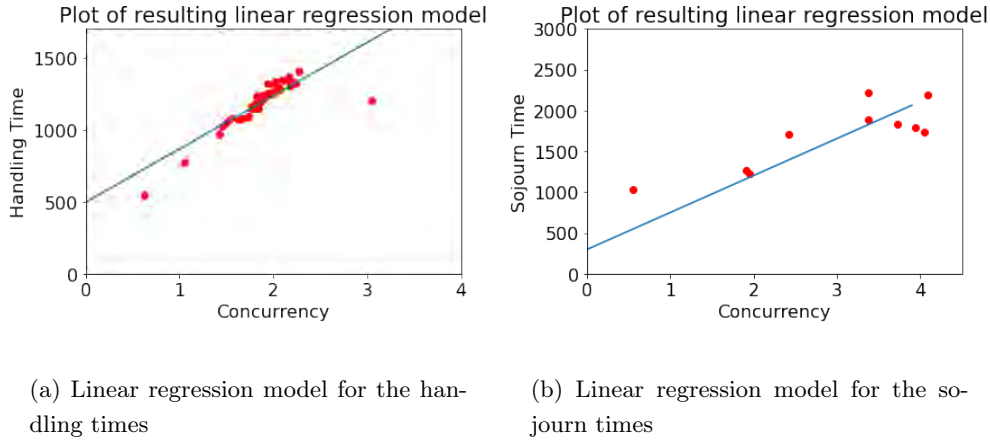


Figure 8.3: Resulting linear regression models for medium sized contact center.

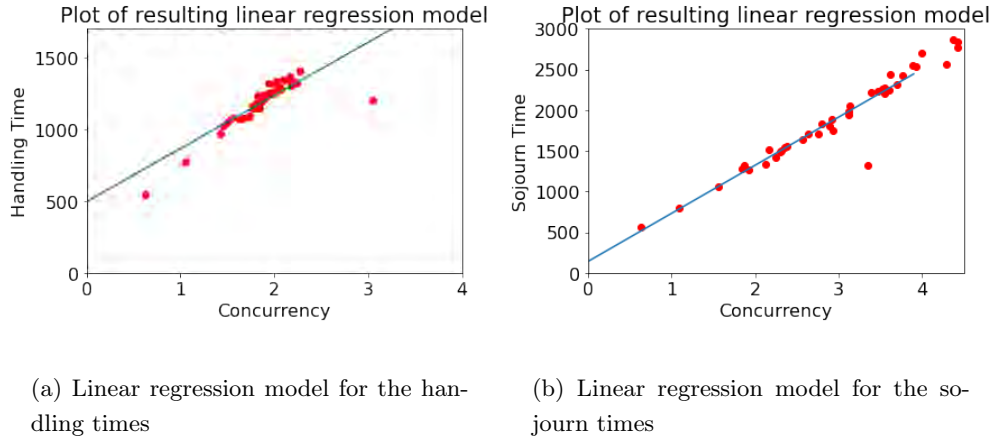


Figure 8.4: Resulting linear regression models for small contact center.

8.1.4 Kaplan-Meier curves

The Kaplan-Meier curves for both contact center sizes are shown in Figures 8.5 and 8.6. The curve can be read as the situation where the agent is serving that number of customer simultaneously for the entire duration of the chat. If we take Figure 8.5a as an example, when the agent is helping four customers at the same time, after 1700 seconds the probability of any chat to still continue is 50%. For both contact centers simulated, there seems to be an abnormality where concurrency is equal to one, as the handling time drops off slower than the higher concurrency levels, in Figure 8.6a even longer than with a concurrency of three.

This phenomenon is not observed for quality of service, where a higher concurrency shows a higher sojourn time, consistent with the other models. The difference between the levels of concurrency is bigger for the small contact center. Especially between concurrency levels two and three and three and four, the difference is substantial.

8. RESULTS

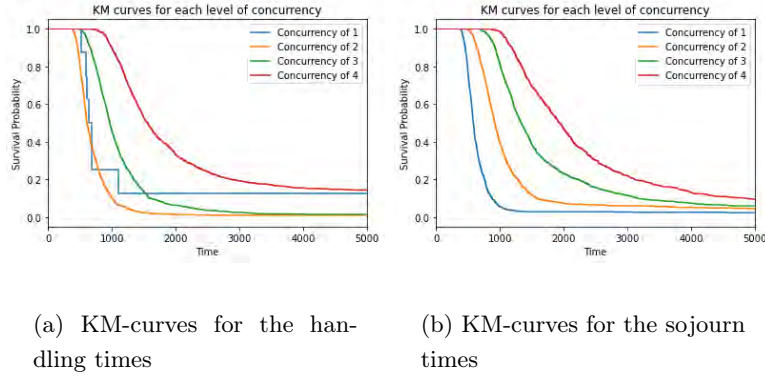


Figure 8.5: Resulting KM-curves for small contact center for each level of concurrency.

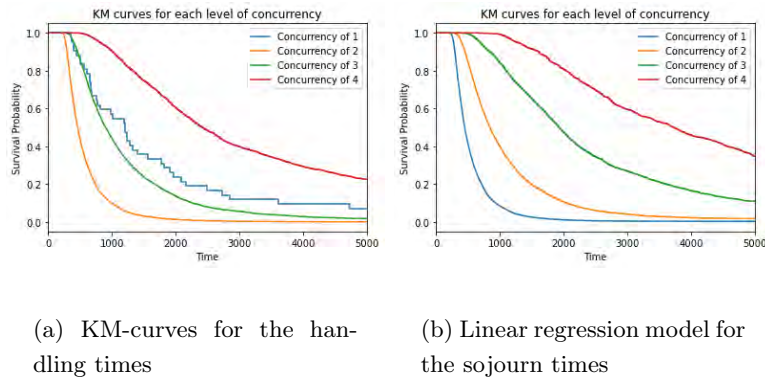


Figure 8.6: Resulting KM-curves for small contact center for each level of concurrency.

8.2 Simulation results

The concurrency during the simulation run with the default settings can be seen in Figure 8.7. On average it is equal to 2.6 chats simultaneously. At times the agent was mostly occupied for some hours [~ 50000 - ~ 80000] and some hours the agent was mostly idle [~ 30000 - ~ 50000]. Most of the duration the occupancy moved up and down in waves.

In Figure 8.8 we see the "actual" handling time. Meaning, the time the agent was actually occupied with each particular customer. If the agent is doing an action related

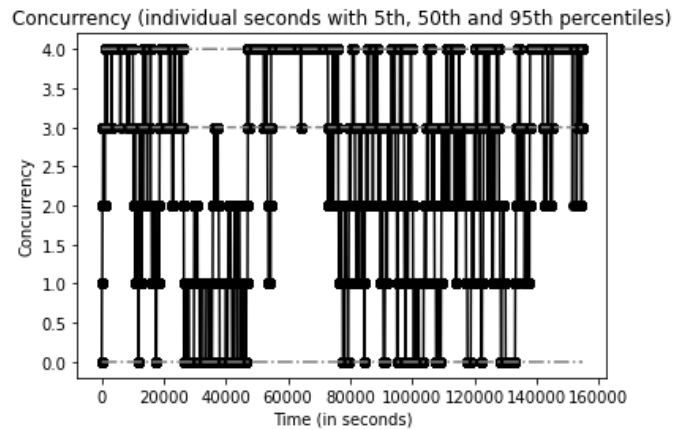


Figure 8.7: Concurrency during the simulation run.

to another chat, this time is not counted as handling time for that customer. The values for these "handling times" are plotted for all customers during the running time of the simulation. To be able to compare the handling time we use, the time between pick-up and departure, is plotted alongside it. Note that these values are closest to each other when the agent is mostly idle, whereas the biggest difference is reached when the agent is fully occupied.

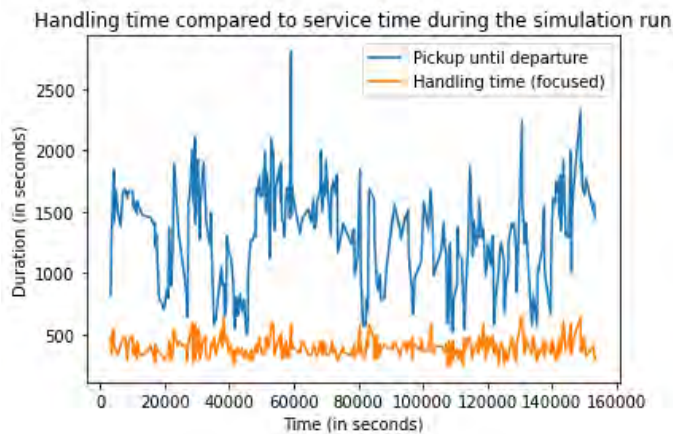


Figure 8.8: Handling times and chat duration during the simulation run.

9

Conclusion and recommendations

The main problem this thesis aims to solve is to find a method that can automate the determination of the handling times of agents in chats as a function of concurrency. This ultimately will be used in the chat model, where the handling times are added as input. In the chat model, the service rate can then be set to these values depending on the number of chats the agent is doing. To find a method that best works, different methods are compared. As we want to have the time-to-event, namely the departure, censoring models are applied. To accommodate aggregated data, linear regression is also included.

To predict handling time, the best and most consistent performing model is the Bayesian survival model. For both contact center sizes it has the lowest median and the lowest maximum median error. For this reason it is the model that will be implemented into the prototype for the handling time calculator as will be explained in Section 10. Additionally, as for some contact centers, only aggregated data is available, the linear regression is also implemented. Surprisingly, when we want to predict sojourn time, linear regression is the best option.

A possible concern to be raised is the large variation in outcomes from the different models. This can be explained by outliers and a lack of real data. As was shown in the Data section, larger levels of concurrency occur less often. Not surprisingly, this is also where the disagreement between the models is the largest. With more real world data it

is to be expected that the configurations will stabilize.

The linear regression model shows a good fit for handling time prediction and an excellent fit for sojourn time prediction. However, when we look at the errors for the handling times, it performs sub-par. When we look at the graph, this can be explained with the fact that higher levels of concurrency occur less often for handling times. If we split up the errors by level of concurrency, this hypothesis is confirmed. Where a concurrency of 1 and 2 gives a median absolute percentage error of respectively 50% and 54%, this goes up to 73% for concurrency level 3 and 63% for concurrency level 4 (see Appendix 10.1).

Both for predicting handling and sojourn time, the Kaplan-Meier shows inconsistent results. Especially for the small contact center the model performs the worst. Inspecting the KM curves, we see an unexpected result when predicting handling times. Namely, for a concurrency of one, the duration of the chats drops off slower than for a concurrency of two. In the case of the small contact center, it shows a longer duration than a concurrency of three. For the large errors for the small contact center when predicting sojourn times, a possible explanation is the large gaps between the levels of concurrency. Unlike the medium contact center where the prediction is quite close to each other, here we observe a substantial difference between the levels of concurrency.

The Cox model did not perform well in most tests, and was never the best performing model. In most cases it performed worse than the easy to implement linear regression. The values for the configurations for sojourn time are very close to each other, indicating that the model is not able to distinguish between the levels of concurrency. A possible explanation for this is the use of a time-varying covariate to represent concurrency in the Cox model.

The Bayesian model performs best for handling times, but is not able to give results for quality of service. With the described setting of 2 chains and 2000 iterations, the sampling has a running time of hours without results. Reducing the number of iterations runs within a reasonable time frame but the Monte Carlo Markov Chains do not converge.

9. CONCLUSION AND RECOMMENDATIONS

The usage of simulation to create data works, but can never perfectly replicate real-life data. Ideally, with more real data availability the methods created can be properly assessed. Otherwise it is a good idea to expand upon the simulation with more precise arrivals and handling. Arrivals could be made with variable arrival rate using rejection sampling. Additionally, the handling can be made more sophisticated, where every step of the chat process can be adjusted based on input from contact center experts. The current simulation creates one agent instance, but it is possible to add new agents, to come even closer to the real life situation.

Based on the insights from the results the prototype will be created using the Bayesian survival analysis model to predict handling times in case there is detailed data as it was the best performing model both in terms of median and extreme errors. In case only aggregated data is available the linear regression will be used as it is the only available model for such data.

An opportunity for future research is looking at customer satisfaction as a performance measure. In this paper, we only look at handling and sojourn times, but it is to be expected that more concurrency for the agents corresponds with lower customer satisfaction. For the prototype, building it into the addin with a VBA script would make it easy for customers to make the addin predict those handling times automatically in real-time.

10

Manual of the prototype

As a final product of this thesis a prototype is created and run from R so that handling time predictions can be made on future data. The linear regression model is built in for aggregated data and the Bayesian survival model for data provided per chat. The Bayesian model is chosen as it showed the best and most consistent performance in Section 8. First in Section 10.1 the steps the user goes through will be given with a short explanation. Then in Section 10.2 the program will be shown with figures of an example.

10.1 Steps the user goes through

Step 1: The user is asked whether the data is aggregated or the data is given per chat.

Step 2: The user is asked what the maximum concurrency is.

Step 3: A window is opened where the user can choose the data file to use as input for the model.

Step 4: Because each data file can have different column names, the user is asked to give a list with the equivalents.

Step 4a: In case the user has aggregated the following columns are needed:

- Handling time sum: The total sum of the handling times of the chats the agents did in that time period.

10. MANUAL OF THE PROTOTYPE

- Busy time: The total time the agents were occupied in that time unit. When the agent takes a break this is not counted for instance.
- Sessions: The total number of customers helped in that time unit.

Step 4b: In case the user has data on the chat level the following columns are required:

- Agent name: This is the name of the agent.
- Pickup: The time the agent first replied to the customer.
- Departure: The time the chat is ended.

Step 5: The model gives the handling times per level of concurrency based on the data. These values can then be copied to Excel to run the Excel chat addin.

10.2 Example of running the program

```
PS C:\Users\Hamid\Downloads\Chat\Prototype> python prototype.py
Type "aggregated" if the data is aggregated or type "per_chat" if the data is given per chat instance
```

Figure 10.1: The two possible options are presented to the user.

```
PS C:\Users\Hamid\Downloads\Chat\Prototype> python prototype.py
Type "aggregated" if the data is aggregated or type "per_chat" if the data is given per chat instance
aggregated
```

Figure 10.2: In this example we have aggregated data.

10.2 Example of running the program

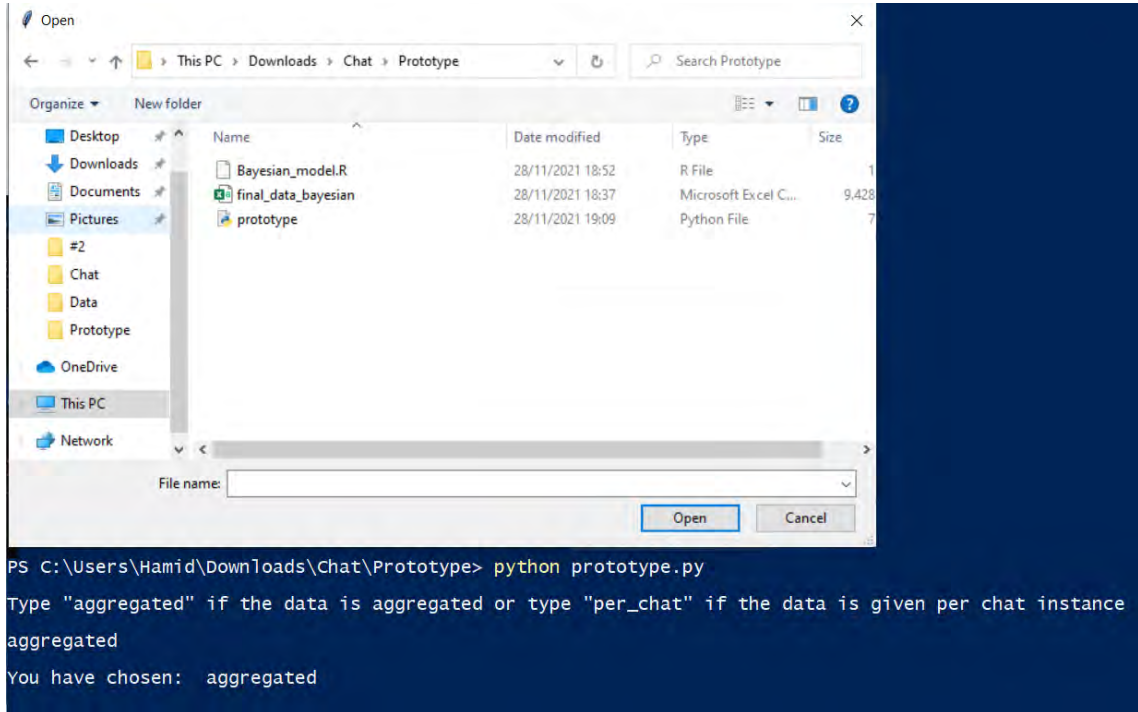


Figure 10.3: A window where the user can choose the file with the data is prompted.

```
Please give the names of the columns in the following format: handling_time_sum,busy_time,sessions. Please note that these values should be for multiple agents per time interval of your choice.
HT_sum,busy_time,sessions
```

Figure 10.4: Here the user types in the required equivalent column names, so that they can be correctly mapped in the model.

```
1 1.0 791.764668
2 2.0 799.217308
3 3.0 806.669949
4 4.0 814.122589
PS C:\Users\Hamid\Downloads\Chat\Prototype>
```

Figure 10.5: Resulting handling times per level of concurrency. The program is finished and the user can now copy the values to Excel.

References

- [1] *Customer Service Representatives*: <https://www.bls.gov/ooh/office-and-administrative-support/customer-service-representatives.htm>. (accessed: 20.07.2021). 6
- [2] K. BANNAN. *Breakaway (A Special Report) — Chatting Up a Sale: Online shopping sites use ‘real-time rooms’ to aid customers*. Wall Street Journal: Eastern Edition, 2000. 7
- [3] T. TEZCAN. *Design and control of customer service chat systems*. University of Rochester, Simon Graduate School of Business, 2011. 7
- [4] N. EPLEY A. KUMAR. *It’s surprisingly nice to hear you: Misunderstanding the impact of communication media can lead to suboptimal choices of how to connect with others*. Journal of Experimental Psychology, 2021 Mar;150(3): 595-607, 2020. 7
- [5] G. SVENSSON P. ENQVIST. *Chat Based Contact Center Modeling - System Modeling, Parameter Estimation and Missing Data Sampling*. ICORES2017, 2017. 16
- [6] G. SVENSSON P. ENQVIST. *A State Dependent Chat System Model*. Proceedings of the 8th International Conference on Operations Research and Enterprise Systems - Volume 1: ICORES, 2019. 17
- [7] J. SALAMON M. L. MATTESON AND L. BREWSTER. *A Systematic review of research on Live Chat Service*. Reference User Services Quarterly, vol. 51, no. 2, pp. 82–10, 2011. 17

REFERENCES

- [8] R. BHOSE R. MUKHERJEE S. GÜVEN G. PINGALI Z. SHAE, D. GARG. *Efficient Internet Chat Services for Help Desk Agents*. IBM T.J. Watson Research Center, 2007. 17
- [9] I. EL-MOUGHRABI A. ELMORSHIDY, M. M. MOSTAFA AND H. AL-MEZEN. *Factors Influencing Live Customer Support Chat Services: An Empirical Investigation in Kuwait*. Journal of Theoretical and Applied Electronic Commerce Research: Vol 10, Issue 3, p63-76, 2015. 17
- [10] S.B. LOVE T.G. CLARK, M.J. BRADBURN AND D.G. ALTMAN. *Survival Analysis Part I: Basic concepts and first analyses*. Cancer Research UK/NHS Centre for Statistics in Medicine, Institute of Health Sciences, University of Oxford, British Journal of Cancer (2003) 89, 232 – 238, 2003. 27
- [11] O. T. GO E. T. LEE. *Survival analysis in public health research*. Annu. Rev. Public Health. 1997. 18:105–34, 1997. 27
- [12] P. MEIER E.L. KAPLAN. *Nonparametric Estimation from Incomplete Observations*. Journal of the American Statistical Association, 1958. 27
- [13] N. COOMBS WILLIAM N. DUDLEY, R. WICKHAM. *An Introduction to Survival Statistics: Kaplan-Meier Analysis*. Journal of Advanced Practitioner in Oncology: 7(1), 2016. 28
- [14] *Kaplan Meier curves: an introduction: <https://towardsdatascience.com/kaplan-meier-curves-c5768e349479>*. (accessed: 21.11.2021). 29
- [15] NEELY J. G. PANIELLO-R. C. VOELKER C. C. NUSSENBAUM B. WANG E. W. RICH, J. T. *A practical guide to understanding Kaplan-Meier curves*. Otolaryngology–head and neck surgery : official journal of American Academy of Otolaryngology-Head and Neck Surgery vol. 143(3), 331-336, 2010. 28
- [16] G. MARINOS A AND D. KYRIAZIS. *A Survey of Survival Analysis Techniques*. Proceedings of the 14th International Joint Conference on Biomedical Engineering Systems and Technologies (BIOSTEC 2021) - Volume 5: HEALTHINF, pages 716-723, 2021. 30

REFERENCES

- [17] S. MATHOULIN-PELISSIER C. A. BELLERA, G. MACGROGAN. *Variables with time-varying effects and the Cox model: Some statistical concepts illustrated with a prognostic factor study in breast cancer*. BMC Medical Research Methodology: Article 20, 2010. 30
- [18] S. BOECK ET AL. *Application of a Time-varying Covariate Model to the Analysis of CA 19-9 as Serum Biomarker in Patients with Advanced Pancreatic Cancer*. American Association for Cancer Research, 2010. 30
- [19] JASON P. FINE PETER C. AUSTIN, A. LATOUCHE. *A review of the use of time-varying covariates in the Fine-Gray subdistribution hazard competing risk regression model*. Stat Med, 2019. 30
- [20] D. R. COX. *Regression Models and Life-Tables* . Journal of the Royal Statistical Society. Series B (Methodological) Vol. 34, No. 2, pp. 187-220, 1972. 31
- [21] M. E. PIETERSE Z. ZHANG, J. REINIKAINEN. *Time-varying covariates and coefficients in Cox regression models*. Ann Transl Med 6(7), 2018. 31
- [22] W. J. GILBERT. *Generalizations of Newton's Method*. Fractals, Vol. 9, No. 3: 251-262, 2001. 31
- [23] *Proportional Hazards Assumption: https://lifelines.readthedocs.io/en/latest/jupyter_notebooks/Proportional_hazards_assumption.html*.(accessed : 24.10.2021).33
- [24] V. LÉVY S. CHEVRET L. BIARD, A. BERGERON. *Bayesian survival analysis for early detection of treatment effects in phase 3 clinical trials*. Contemp Clin Trials Commun. 2021 Jan 9, 2021. 33
- [25] M. TURE I. KURT OMURLU, K. OZDAMAR. *Comparison of Bayesian survival analysis and Cox regression analysis in simulated and breast cancer data sets*. Expert Systems with Applications Volume 36, Issue 8, October 2009, Pages 11341-11346, 2009. 33
- [26] M. MAHMOODI K. MOHAMMAD H. ZERAATI K. H. NAIENEI A. ZARE, M. HOSSEINI. *A Comparison between Accelerated Failure-time and Cox Proportional Hazards Models in Analyzing the Survival of Gastric Cancer Patients*. Iran J Public Health Aug, 44(8), 2015. 33

REFERENCES

- [27] J. B. NOVIK S. L. BRILLEMANN, E. M. ELCI. *Bayesian Survival Analysis Using the rstanarm R Package*. <https://arxiv.org/pdf/2002.09633.pdf>, 2020. 34

Appendix

10.3 Graphs and Figures

Concurrency	Median relative absolute error
1	0.5
2	0.54
3	0.73
4	0.63

Table 10.1: Errors of the linear regression model per level of concurrency.

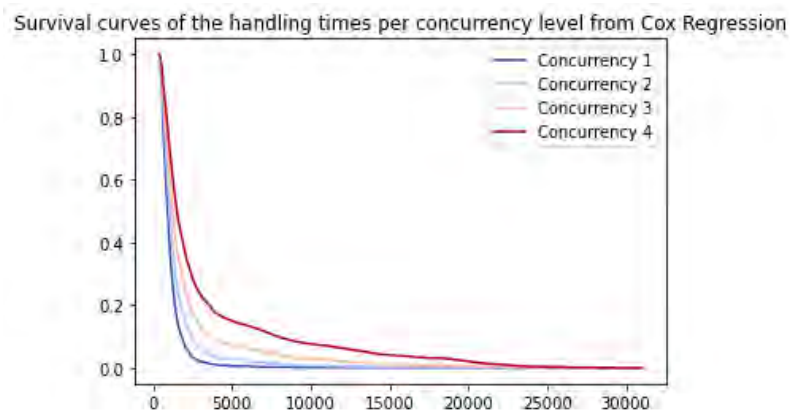


Figure 10.6: Resulting plot from the time-varying Cox regression.

10.4 Data Wrangling

Algorithm 1 Interval conversion Algorithm

Input: Dataframe *df* where each chat is included with its start and end time.

Output: Dataframe *final* where each chat is split up into intervals every time concurrency changes. Every interval corresponds to a concurrency level during the chat.

- 1: *factors* = factors to use as predictive variables (i.e. concurrency)

 - 2: **for** *row* in *df* **do**
 - 3: *births* = *df* where *Agent-Name* is equal to *row.Agent*, *Pick-Up-time* is later than or equal to *row.Pickup* and *Pickup-time* is smaller than or equal to *row.Departure*.
 - 4: *deaths* = *df* where *Agent-Name* is equal to *row.Agent*, *Departure-time* is later than or equal to *row.Pickup* and *Departure-time* is earlier than or equal to *row.Departure*.
 - 5: Add to *births* column with 1's, and to *deaths* column with 0's.
 - 6: Concatenate the *births* and *deaths* dataframes into dataframe *concat*

 - 7: *result* = *concat*[0:length(*concat*) -1]
 - 8: *result.Transition* = *concat*[1:length(*concat*)]

 - 9: *result* = *concurrency(result)*(*)
 - 10: **end for**

 - 11: *final.append(result)*
-

REFERENCES

Survival curve per concurrency level from the Bayesian survival model.

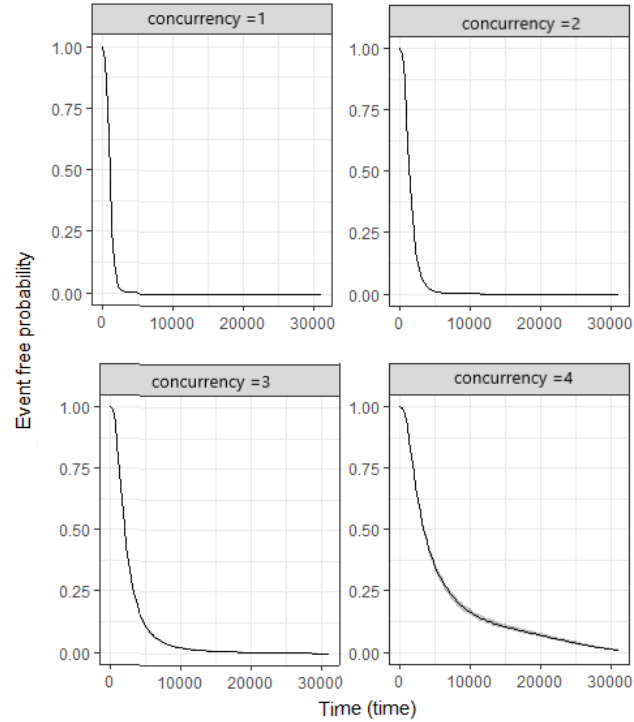


Figure 10.7: Resulting plot from the Bayesian survival model.

1: **Function** concurrency(*)

Input: dataframe *result*

Output: dataframe *result* with column concurrency

2: **for** *interval* in *result* **do**

3: **if** *interval*.Index = 0 **then**

4: *result*[row:*interval*.Index,col:'concurrency'] = *interval*.concurrent-at-pickup+1

5: **else if** *interval*.birth = 1 **then**

6: *result*[row:*interval*.Index,col:'concurrency'] =*result*[row:*interval*.Index-1,col:concurrency]+1

7: **else if** *interval*.birth = 0 **then**

8: *result*[row:*interval*.Index,col:'concurrency'] =

result[row:*interval*.Index-1,col:concurrency]-1

9: **end if**

```
10: end for
```

10.5 Code of the simulation

```
import simpy
import random
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
# Graphing and global tracking adapted from bed occupancy model pythonhealthcare.org

class var():
    # Global variables
    concurrency = 0 # concurrency
    iat = 450 #default inter arrival time
    percentage = .1 #default percentage value for continuation of the chat at step 6.
    speedup = 1 # bigger or smaller than 1 slower or faster respectively by that factor

    q = []
    handling_times = {} #track the focus time
    servicetodep = {} #track the service time
    output = pd.DataFrame()
    #output dataframe, will be per chat the arrival, pickup, departure etc.

    tasks = {0:(30,60),2:(60,60),3:(60,200)} #list of tasks
    for i in np.arange(5,100,2):
        tasks[i] = (60,200)
    cust_tasks = {1:(60,300),4:(120,300)}
    for i in np.arange(6,100,2):
        cust_tasks[i] = (120,300)

    audit_time = []
    audit_conc = []
    audit_interval=1 # global concurrency measured at this interval
```

REFERENCES

```
def arrival_process(env,var):
    while True:
        #customer types first message (30-90 seconds)
        yield env.timeout(random.uniform(30,90))

        var.output.loc[i-1,'Arrival'] = env.now #log time arrival
        var.output.loc[i-1,'concurrency_time_calling'] = var.concurrency
                                                    # log concurrency at arrival

        if var.concurrency < 4:                    #if agent available to add chat
            var.concurrency += 1
            store.put((i,0))
            var.servicetodep[i] = env.now
        else:                                       #add to arrival queue
            var.q.append((i,0))
        next_customer =random.expovariate(1/var.iat) #sample next arrival time
        #print('Next arrival in ',next_customer,' seconds')
        yield env.timeout(next_customer) # wait for next arrival

def arrival_blocked(env,var):
    while True:
        #if customer can be assigned, do. else wait 0.5 seconds.
        if (var.concurrency < 4) & bool(var.q):
            var.concurrency += 1
            cust = var.q.pop(0)
            #print(cust,' entered after queued')
            store.put(cust)
            var.servicetodep[cust[0]] = env.now
        yield env.timeout(0.1)

def service(env,var):
    while True:
        # greet + first reply
        nxt = yield store.get()
```

```

if nxt[1] == 0:
    var.output.loc[nxt[0]-1,'Pickup'] = env.now
    var.output.loc[nxt[0]-1,'concurrent_at_pickup'] = var.concurrency

if (nxt[1] == 0) | (nxt[1] == 2) | (nxt[1] == 3) |
    ((nxt[1] > 4) & (nxt[1] % 2 != 0)):

    lb =var.tasks[nxt[1]][0]
    ub =var.tasks[nxt[1]][1]) * (1/var.speedup)
    duration = random.uniform(lb,ub)

    yield env.timeout(duration)

    if nxt[0] in var.handling_times:
        var.handling_times[nxt[0]] += duration
    else:
        var.handling_times[nxt[0]] = duration

    store.put((nxt[0],nxt[1]+1))
elif (nxt[1] == 1) | (nxt[1] == 4) |
    ((nxt[1] > 4) & (random.random() > (1-var.percentage))):
    cust_act.put((nxt[0],nxt[1]))
else:
    var.concurrency -= 1
    var.servicetodep[nxt[0]] = env.now - var.servicetodep[nxt[0]]
    var.output.loc[nxt[0]-1,'Departure'] = env.now
    #print('Customer',nxt[0],'leaving at time: ',env.now)

def cust_action(env,var):
    while True:
        cust = yield cust_act.get()

        #print('Customer',cust[0],'does following task ',cust[1],'at time: ',env.now)

```

REFERENCES

```
    lower = var.cust_tasks[cust[1]][0]
    upper = var.cust_tasks[cust[1]][0]
    yield env.timeout(random.uniform(lower,upper) * 1/(var.speedup))
    store.put((cust[0],cust[1]+1))

def audit_concurrency(env,delay): # add concurrency at that time to dataframe to chart
    yield env.timeout(delay)
    while True:
        var.audit_time.append(env.now)
        var.audit_conc.append(var.concurrency)
        yield env.timeout(var.audit_interval)

def build_audit_report(): # all values measured are combined in a single dataframe
    audit_report=pd.DataFrame()
    audit_report['Time']=var.audit_time
    audit_report['Concurrency']=var.audit_conc
    audit_report['Median_conc']=audit_report['Concurrency'].quantile(0.5)
    audit_report['Conc_5_percent']=audit_report['Concurrency'].quantile(0.05)
    audit_report['Conc_95_percent']=audit_report['Concurrency'].quantile(0.95)
    return audit_report

def chart(): # plot of the concurrency over time
    plt.plot(var.audit_dataframe['Time'],
             var.audit_dataframe['Concurrency'],
             color='k',marker='o',linestyle='solid',
             markevery=1,label='Concurrency')

    plt.plot(var.audit_dataframe['Time'],
             var.audit_dataframe['Conc_5_percent'],
             color='0.5',linestyle='dashdot',
             markevery=1,label='5th percentile')

    plt.plot(var.audit_dataframe['Time'],
             var.audit_dataframe['Median_conc'],
             color='0.5',linestyle='dashed',
```

```
        label='Median')

plt.plot(var.audit_dataframe['Time'],
         var.audit_dataframe['Conc_95_percent'],
         color='0.5',linestyle='dashdot',
         label='95th percentile')

plt.xlabel('Time (in seconds)')
plt.ylabel('Concurrency')
plt.title('Concurrency (individual seconds with 5th, 50th and 95th percentiles)')
print(var.audit_dataframe['Concurrency'].describe())
plt.show()

def reset_values(): # After each run, all environment variables are emptied/set to zero.
    var.handling_times= {}
    var.servicetodep = {}
    var.q = []
    var.concurrency = 0
    var.output = pd.DataFrame()

# Initialise environment
train_df = pd.DataFrame()
for iat in np.arange(300,500,100):
    for speedup in np.arange(0.6,1.2,0.1):
        for perc in np.arange(.15,.25,0.05):
            # set var.iat and var.speedup and var.percentage
            var.iat, var.speedup, var.percentage = iat, speedup, perc

            # Initialize environment
            env=simpy.Environment()
            # Initialise processes
            store = simpy.Store(env, capacity=1)
            cust_act = simpy.Store(env)
            env.process(arrival_process(env,var))
            env.process(arrival_blocked(env,var))
```

REFERENCES

```
env.process(service(env,var))
env.process(cust_action(env,var))
env.process(audit_concurrency(env,delay=20))

# Start simulation run
env.run(until=43*60*60)
#number of seconds in 42 hours (working day) + warm up 30m + cool down 30m

# Process data
if('Departure' in var.output.columns):
    #drop cool down and warm up
    var.output =
    var.output.drop(var.output[(var.output['Arrival'] < 1800) |
    (var.output['Departure'] > 153000)].index)
    var.output['iat'], var.output['speedup'], var.output['perc']
    = iat, speedup, perc

    train_df = train_df.append(var.output)
    # Set all values to default (empty/zero)
    reset_values()
# Build audit table and plot
var.audit_dataframe=build_audit_report()
chart()
```