TECHNICAL UNIVERSITY OF MUNICH

TUM Data Innovation Lab

A Company's Digital Twin

Authors	Sagarika Kathuria, Jieyi Zhang, Frederik Wenkel, Pooreumoe Kim
Mentor(s)	Simon Brand, Anton Kurz (Celonis SE)
Co-Mentor	Laure Vuaille (Department of Mathematics)
Project lead	Dr. Ricardo Acevedo Cabra (Department of Mathematics)
Supervisor	Prof. Dr. Massimo Fornasier (Department of Mathematics)

Abstract

A Digital Twin of an organization is a data-based model of assets or processes. It enables users to analyze and optimize their product, service or process in a digital instance. For example, a sophisticated model allows people to visualize products performing used by actual people in real-time or address issues before they turn into real-world problems. Therefore, it has become increasingly attractive for organizations to utilize this technique to achieve their business excellence. There are various ways of implementing a Digital Twin. In this work, we present how we build a Digital Twin model based on the existing process mining log of an IT Service Management Process provided by Celonis. This model is developed using Markov Processes to simulate cases based on the observations of the original system. Furthermore, we investigate the number of cases and their throughput times with Data Preprocessing, Kernel Density Estimation and Time Series Prediction Methods. Based on our model, we try to answer some what-if questions come up by our internal clients, including 'What if the first level automation rate is improved?' and 'What if my number of tickets goes up or down?'. The company can thus simulate the effects of possible improvements and facilitate the users' decision making.

Contents

1 Motivation

Today, many organizations such as companies or even governments have a strong interest in long term predictions of their organization's development. To guarantee success, companies must adapt fast to changes of their environment. In the oil industry for example, the volatility of the main resource poses an omnipresent uncertainty on the companies future and in the financial industry the consequences of financial crisis are hard to estimate in advance. Of course, solutions have been found which account for uncertainty of external influences and get reasonable forecasts. Such forecasts, however, are usually restricted to a set of specific key performance indicators and fail to capture the organization's behavior in its entirety.

But how about a digital representation of our organization where we can fast forward and see how the performance is influenced given different scenarios. One could just try all possible scenarios in terms of tuples of company strategy and outer influences and select a strategy which maximizes expected success in the models. This idea is usually referred to as a Digital Twin. In the past, organizations' structures have been too complex to be fully understood with the techniques in place but with new emerging data processing technologies such as Process Mining, our understanding of processes has made way for the creation of such Digital Twins.

2 Process Mining at Celonis

The term Digitalization has influenced enterprises from all industrial sectors for many years. Today, almost every process within a company leaves its footprints in the company's digital environment. This "idle" data contains a lot of information concerning problems and inefficiencies of processes. This sparked the development of a whole new branch of companies which developed software to grasp this information.

Celonis for example, with its software *Celonis Process Mining* has been extremely successful in this field. Since the foundation in 2011, Celonis has grown rapidly and was even named the fastest growing German technology developing company. Less than ten years after the foundation, Celonis runs offices all around the globe and works with corporations of all kinds, many of them with strong global magnitude.

When an organization decides to use Celonis Process Mining for a process, the raw data which is provided by the organization is structured in several objects, which will serve as the data base, the so-called *Data Model* in the Celonis environment.

The main object is the *Activity Table*. It contains all recognized process flows from the customers data which are referred to as *cases*. Each of these cases consists of a number of *activities* which characterize the process flow. Attached to every activity, there is an *event time*, representing the time of the start of the activity, a *sorting* and a *category*. The sorting will determine the ordering of activities with exactly the same event time, while the category can be used to indicate the executing department. To distinguish different cases within the Activity Table, every case has its own *case key*.

Apart from the Activity Table, there can be several other tables. They store additional information which enriches our very general knowledge from the Activity Table with additional attributes.

With the Data Model in place, Celonis Process Mining offers a variety of elaborate tools and key performance indicators (KPIs) to visualize and understand the data, a so-called *Analysis*. In an Analysis it is easy to locate bottlenecks or inefficiencies within

_CASE_KEY	ACTIVITY_EN	EVENTTIME	CATEGORY_NAME	_SORTING
201807010	Status: New	2018.7.2 3:08	Category 1	10
201807010	Status: Open	2018.7.2 3:14	Category 1	10
201807010	Change assignee	2018.7.2 3:14	Category 1	10
201807010	Status: On Hold	2018.7.2 3:49	Category 1	10
201807010	Status: Open	2018.7.2 6:00	Category 1	10
201807010	Status: Closed	2018.7.2 12:06	Category 1	10
201807011	Create Ticket	2018.7.1 18:21	Category 2	10
201807011	Change priority	2018.7.1 18:21	Category 2	10
201807011	Status: Closed	2018.7.1 21:39	Category 2	10

Figure 1: Extract from an Activity Table

the process. On the one hand there are many standardized objects like an overview page, process explorers (Figure 2), variant explorers and standard KPIs. On the other hand, the Analysis is highly customizable with a large variety of tables, diagrams, charts and an editor for the creation of custom KPIs.

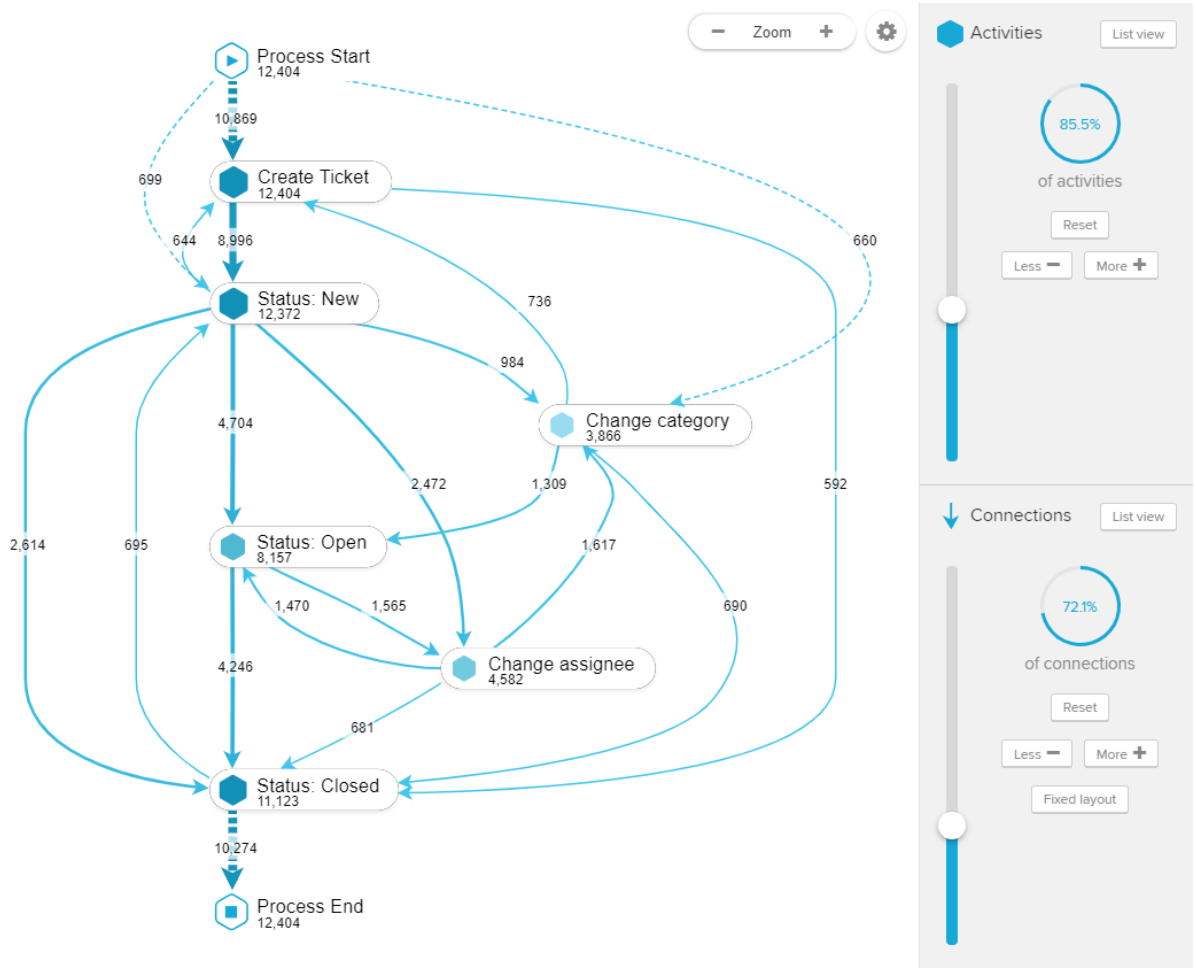


Figure 2: Process explorers allow us to oversee the interaction between activities with adjustable complexity

2.1 Incident Management at Celonis

For our project, Celonis provided us data from their internal incident management department as sample data. The data describes the incident management process Happyfox at Celonis. When support services are requested by a client of Celonis, it opens an incident and determines the priority of the incident. Furthermore, it notifies the incident priority and ticket number, which enables an accurate placement of subsequent queries within the incident management system for the users. It uses commercially reasonable efforts to respond to the incident within a negotiated initial response time and to subsequently acknowledge the incident and provide a resolution. An incident resolution may consist of a fix, workaround, service availability or any other solution that is deemed reasonable. An incident is dropped and subsequently closed if the user has not responded within five business days to an attempt to collect additional information required to resolve the incident.

This data is structured as a Data Model as introduced in the section before. The cases in the Activity Table represent each the treatment of an incident ticket. For an example, please refer to Figure 1. The set of activities is

$$\begin{aligned} A = \{&\text{'Change assignee', 'Change category', 'Change priority',} \\ &\text{'Create Ticket', 'Status: Closed', 'Status: New',} \\ &\text{'Status: On Hold', 'Status: Open', 'Status: Solved'}\}. \end{aligned}$$

Apart from the activities and corresponding event times, the categories play an important role for our purposes. They indicate the department where the activity is executed. Note that the Activity Table only shows the category at the process end and we have to use two other Tables to trace the category history back. Those tables are the Tickets Table which contains additional information for the individual incident tickets and the Updates Table which stores the change history of some attributes like the category.

3 Digital Twin Concept

So far we have talked primarily about process mining and how it is done at Celonis. Let us turn now to a concept which is at the very center of this Project, the concept of a *Digital Twin*.

Our understanding of a Digital Twin is that of a counterpart of a process in terms of a digital model. However, it is not supposed to be an exact copy of the original process. While it should behave similarly under the assumptions of the initial system, it is nevertheless a dynamic object which adapts to changes of its environment. Such a model is a powerful tool because it gives insights into the longtime future behavior of a process conditioned on the setting around the process. This can be very useful for predictions of the future performance of a process given different scenarios.

It can give valuable insights into what-if questions which most organizations would like to answer. What happens if I increase or decrease certain resources within my organization? What happens if demand for my product goes up or down?

We want to stress the importance of Process Mining for this concept. The fundamental understanding of the inner dynamics of a process are vital to create the desired counterpart of the process and moreover indicate how it can gain independence from the original process. Therefore, Process Mining can be seen as an enabler of Digital Twins.

4 Data Exploration

4.1 Input Description

4.1.1 Data Set

In this work, we build our Digital Twin model using the data set of Celonis Happyfox ITSM. Happyfox is a software as a service (SaaS) support platform, they offer help desk ticketing system for over 12,000 enterprises around the world. In our original data model, there are three data tables which contains the information for each ticket. A ticket is described as a single process consisting of different activities in sequence. The relevant features about each ticket and each event are recorded in different tables. Finally, these tables are mapped together using ticket ID. Given these data as input, the Celonis process mining analysis tool allows us to visualize and analyze the ITSM process performances. The detailed description about each data table is demonstrated in the following part.

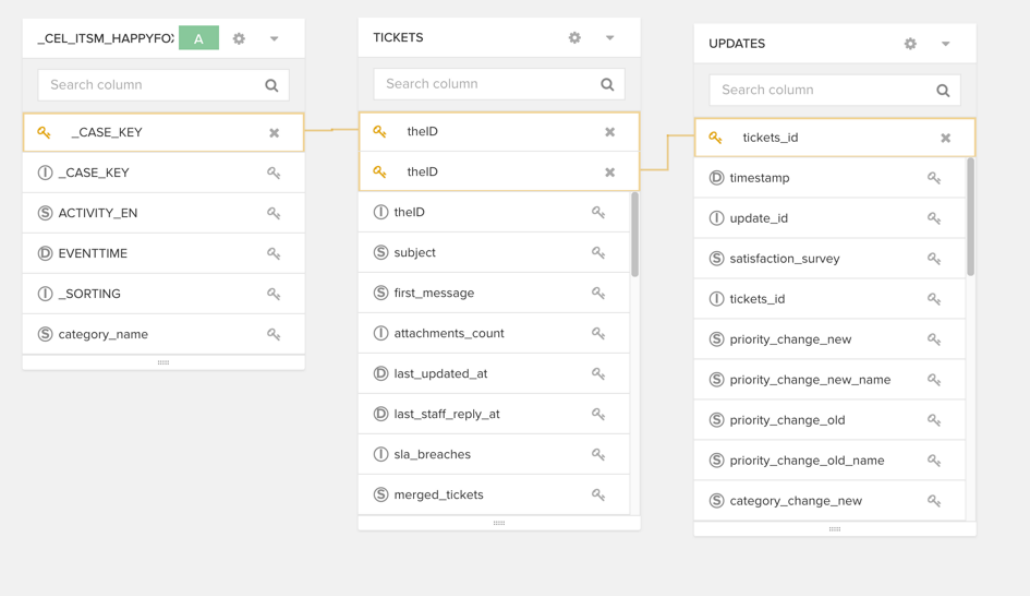


Figure 3: Celonis Happyfox ITSM Data Model. From left to right: activity table, case table and update table. In case table and update table, only part of the features is listed.

The data model structure is presented in Figure 3. The table in the middle named *TICKETS* is the case table in our work. The primary key of this data table is the field called *theID*. The IDs correspond to the ticket ID in the Celonis ITSM, and the values are unique. In our data model, we have 12,404 entries in total recording the tickets generated between 4th August 2017 and 17th July 2018.
There are 41 feature fields except for the ticket ID, and they are all listed in Figure 4. These fields include the necessary information of each ticket, such as the subject and attachments. Besides, the particular additional attributes, for example, category, assignment, priority, status, and users, are also recorded in other columns. The exploratory analysis results are presented in the next section.
The table on the left side is the activity table, *_CEL_ITSM_happyfox_ACTIVITIES*. This table has 66,371 entries and contains the log of each activity during the ticket processing. As it is shown in Figure 3, the foreign key, namely the case key, corresponds the

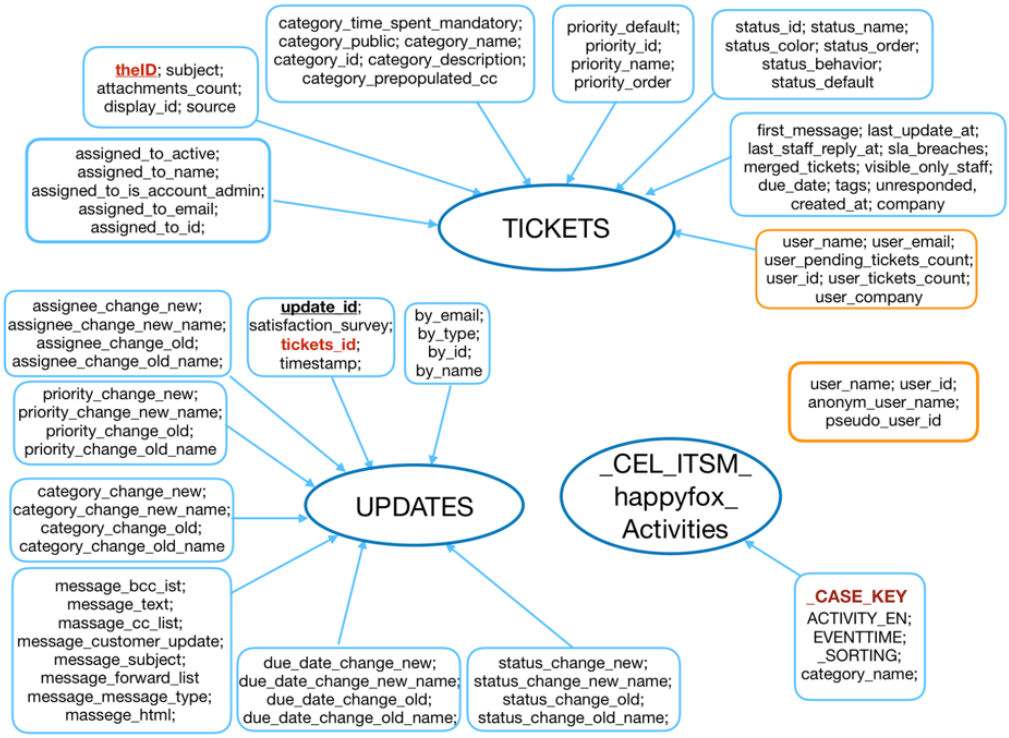


Figure 4: Celonis Happyfox ITSM Data Model with All Features. The features for each table are connected to the table names, respectively. Within each table, the features are grouped by their categories. The primary keys are marked with underscore lines and the foreign keys are marked in red. The users are anonymized and replaced with the yellow table for pseudo users on the right

ticket ID in the case table. The field, *ACTIVITY_EN*, is the activity name in English. The typical activities that we have observed include "Create Ticket", "Status: New", and "Status: Closed". For each event (activity), the timestamp recorded when this event happened in the system, and normally, the sequence of each activity in a process is determined by the event timestamp. If two events in a single process in the data set have the same timestamp, then their sequence is set based on the attribute called *_SORTING*. The column, *category_name*, recorded the final category of each step. Moreover, this information is relevant to the first level service in our what-if questions.

The table called UPDATES on the left side in Figure 1 records the updates about the tickets. It has 58,295 update entries, and the primary key of this table is called *update_id* (see Figure 4). Each entry contains information, such as the time when this update happens and which type of change it has made, and the entries are mapped with the case table through the ticket IDs. The fields are grouped according to the type and listed in Figure 4. The change types including changing assignee, category and due date.

The data are stored in the model mainly as string, number, Boolean and date time. There are PQL functions(Process Query Language) in Celonis analysis tool and Celonis Python API that allows us to transform and compute with this data.

4.1.2 ITSM Ticket Processing

As we have described in the last section, the processing of a ticket including several activities and their timestamps and sorting values determine their sequence in the process. The sequence is sorted ascending first by time, and if several events have the same timestamp, they will be sorted based on the sorting values.
Each ticket in the ITSM system has its own process, and each type of process is called variant. Different tickets can have different variants and some tickets can share a same kind of process. In Celonis ITSM system, we name the variant, which happens most frequently in our data set, as the happy path. By computing the difference of the timestamp of the activity and its previous activity in each process, we get the throughput time of each step. We name ts_{act} as the timestamp of a event and $ts_{act\_previous}$ as the timestamp of the previous activity, the throughput time T_{tp} can be expressed as

$$T_{tp} = ts_{act} - ts_{act_{previous}} \tag{1}$$

Similarly, the total throughput time for each process is defined as

$$T_{total\_tp} = ts_{last\_act} - ts_{first\_act} \tag{2}$$

Where ts_{last_act} and ts_{last_act} are the timestamps of the first and the last activity in the process.
In Figure 5, we show an example of the happy path in our original data set in Celonis Process Explorer. The nodes between "Process Start" and "Process End" represent the activities or events in this variant and their happening sequence are demonstrated by the arrows. The time cost value in days of each step is presented on the connecting arrows and the number of cases flowing through each node is presented under the names of the activity. We can observe that in the happy path, the total throughput time $T_{total\_tp} = 1day$, and all the cases following happy path are closed cases, namely ending with "Status Closed". The total number of different variants in our Happyfox ITSM data set is 1,850 and around 20% of our cases follow the happy path presented in Figure 3.

4.2 Anonymization of Data

The data which we deal with contains the name and user id of the person who created the incident ticket. As this project's content will be publicly accessible it was important to make the data anonymous. For this purpose we implemented a anonymization function in Python. The input is a table with two columns. They contain the actual user name and user id. Note that user names as well as user ids might possibly appear twice. Two users might have the same name for example. The pair of them however is always unique. Now we create vectors with the same length as the just mentioned table which just contain the numbers $\mathbb{N} \cup \{0\}$ in ascending order. We randomly shuffle both vectors separately and join one as the pseudo id and one as the pseudo name to the table. For the pseudo name we append the string 'name' in front of the number. This will map our input table to a table with four columns like in Figure 6.

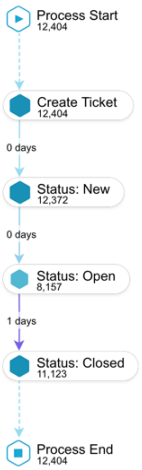


Figure 5: A variant example in the Celonis Process Explorer. (The node "Process Start" and "Process End" are appended to each variant as a default.) The time unit can be changed to hours, minutes or seconds easily in the explorer.

$$\begin{bmatrix} 100 & \text{Sagarika Kathuria} \\ 100 & \text{Sagarika Kathuria} \\ 200 & \text{Jieyi Zhang} \\ 300 & \text{Bluemountain Kim} \\ 300 & \text{Frederik Wenkel} \\ 400 & \text{Frederik Wenkel} \end{bmatrix} \mapsto \begin{bmatrix} 100 & \text{Sagarika Kathuria} & 2 & \text{name0} \\ 200 & \text{Jieyi Zhang} & 0 & \text{name3} \\ 300 & \text{Bluemountain Kim} & 3 & \text{name1} \\ 300 & \text{Frederik Wenkel} & 3 & \text{name2} \\ 400 & \text{Frederik Wenkel} & 1 & \text{name2} \end{bmatrix}$$

Figure 6: Example of an application of the anonymization function

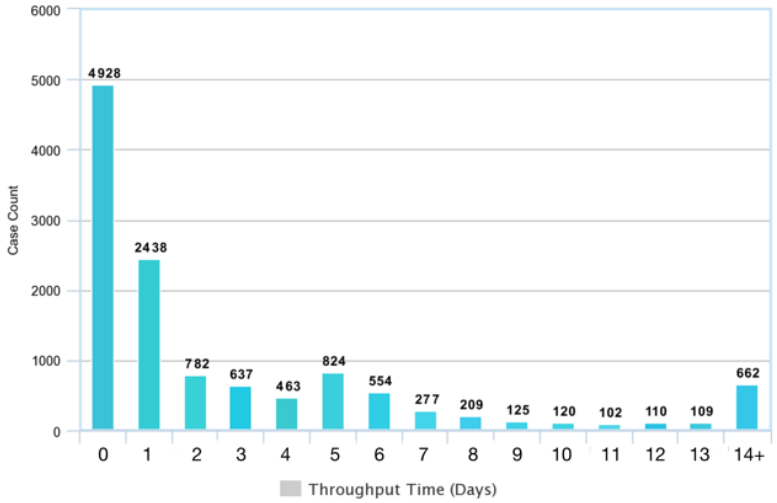
Note that different tuples of id and name are preserved, while duplicated tuples are discarded. The table which is created as an output can be uploaded to Celonis Process Mining to be joined to the Tickets Table using the original user information. Our observations indicate, that the discrepancy between the number of user names an user ids was only caused by users having the same name. Therefore, it is sufficient to use the user id as a key to map the table to our Data Model.

4.3 Exploratory Analysis

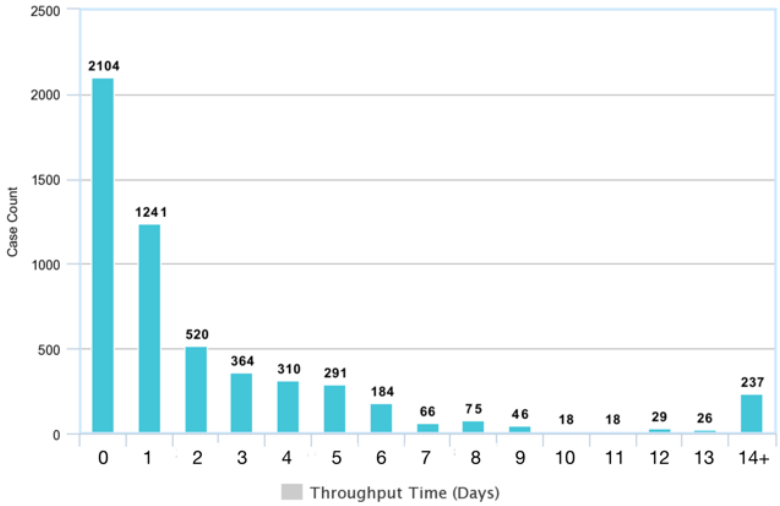
In this section, we presented part of our initial analysis which is relevant to constructing our Digital Twin model.

4.3.1 Throughput Time Analysis

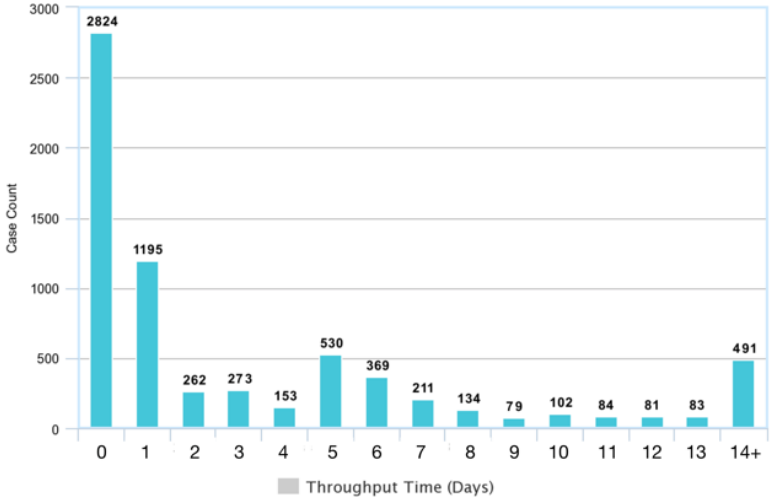
Total Throughput Time Distribution The total throughput time distribution of all cases is presented in Figure 7a. We observed that it has spike at 0 day with a long tail distribution. Intuitively, this distribution looks similar to a Poisson distribution or exponential distribution. It will benefit our modelling process if we can parameterize the distribution of total throughput time.
Therefore, we have tried to fit the distribution to some possible typical distribution. To this end, we fit our data to different types of distributions with the model fitting function



(a) Total throughput time computed with all data



(b) Total throughput time computed with data generated in 2017



(c) Total throughput time computed with data generated in 2018

Figure 7

in SciPy Stats[5]. Then test whether the fitted distribution is the same as the original distribution of our data. To test the goodness of fit of the distribution with respect to the data, we apply Kolmogorov–Smirnov test (KS test). This test is based on a distance between the empirical distribution function of the data and the cumulative distribution function (CDF) of the reference distribution. In this test, the null hypothesis is two distributions follow the same distribution. If the p-value is greater than 0.05, then the null hypothesis holds.
The test results are listed in the following:

Distribution	p-value	Is passed
Poisson Distribution	4.675e-10	no
Exponential Distribution	0.0	no
Birnbaum-Saunders	0.0	no

In the results, none of tests has passed. This suggests that we need to find some non-parametric methods to simulate the throughput time.
To observe whether the total throughput time changes over time, we separate the dataset into two which are cases created in 2017 and tickets created in 2018 respectively. It can be observed that in the graph of 2018, there is a spike on throughput time 5 days, which is consistent with the distribution of all cases. However, in 2017, this spike does not exist. Therefore, we come to a conclusion that the total throughput time distribution changes from 2017 to 2018 and the latest data distribution are more similar to the distribution of the all cases.
This observation also indicates that the change of throughput time distribution of each step can cause the change of total throughput time distribution. This is one of major reasons why we choose non-parametric method to simulation the throughput time in this work.

Outliers and Anomalies In our dataset, we observe some cases with total throughput time up to more than 200 days. These outliers have been plotted in Figure 8. The median is indicated by the red vertical line that runs down the center of the box. In the boxplot above, the median is significantly close to 0 and the distribution is skewed right, while the maximum total throughput time is 276 days. This observation indicates that there are some anormal cases in our dataset and there can be some bottlenecks in the process as well.

However, in the following sections, we introduce the Digital Twin model which empirically simulates not only the normal cases but also some anormal cases. Therefore, we will not filter out the outliers in our preprocessing part.

4.3.2 Category-based Analysis

In Celonis, there are 21 ticket categories in total. Their names and counts are listed in Figure 9. We notice that in our dataset, only 5 categories are included, they are ServiceDesk, Internal IT, System Alert, Cloud Instances, and SalesForce. There portions are plotted in Figure 10. Our Digital Twin model will simulate the case categories based

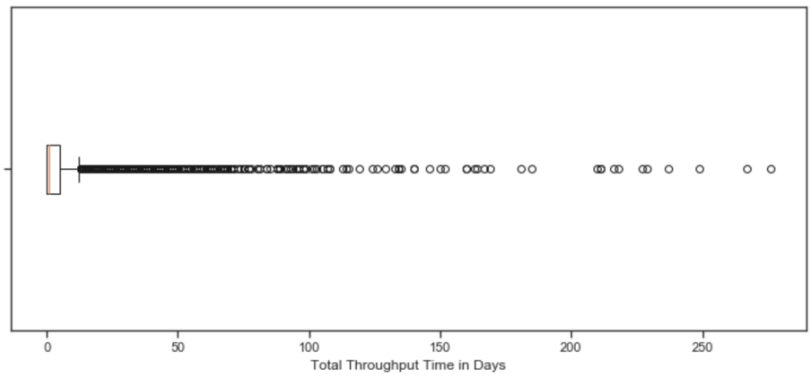


Figure 8: Boxplot of total throughput time

on the category percentages.

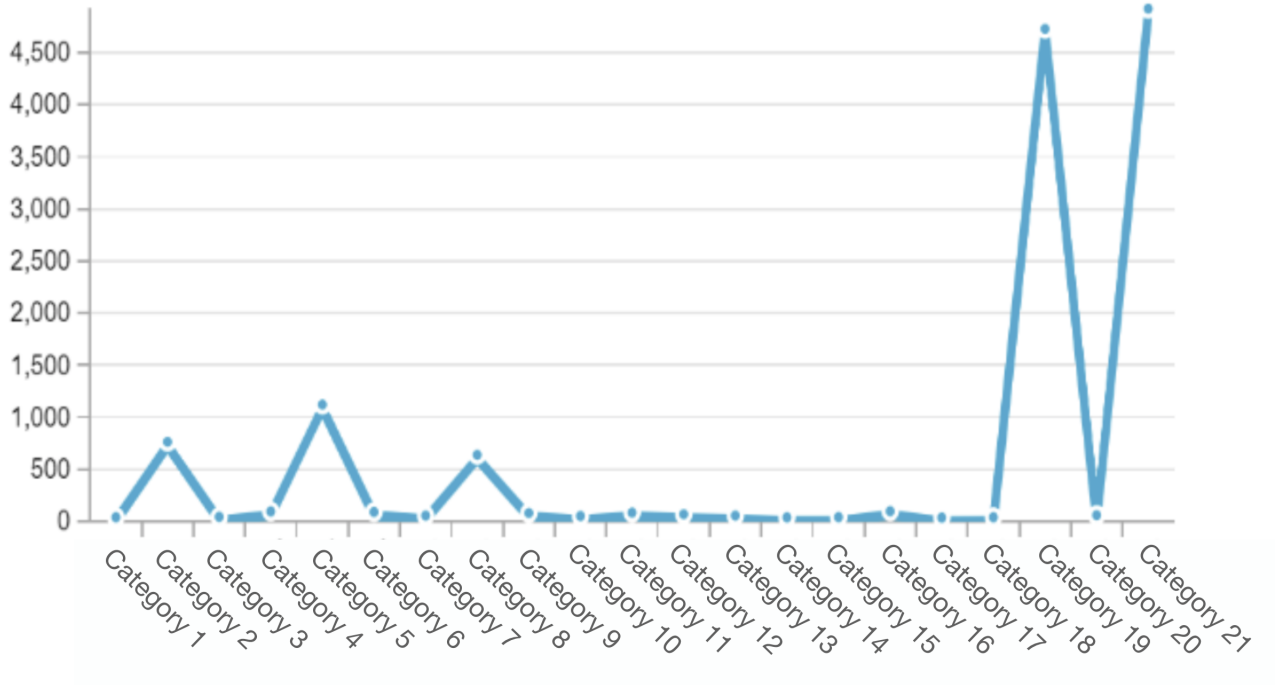


Figure 9: Distribution of Categories (anonymized)

4.3.3 Ticket Number Analysis

A Time Series is said to be stationary if its statistical properties such as mean, variance remain constant over time. But why is it so important? Most of the Time Series models work on the assumption that the TS is stationary. Intuitively, we can say that if a Time Series has a certain behavior over time, there is a very high probability that it will follow the same in the future.

Stationarity is defined using a very strict criterion. However, for practical purposes we can assume the series to be stationary if it has constant statistical properties over time, i.e. the following:

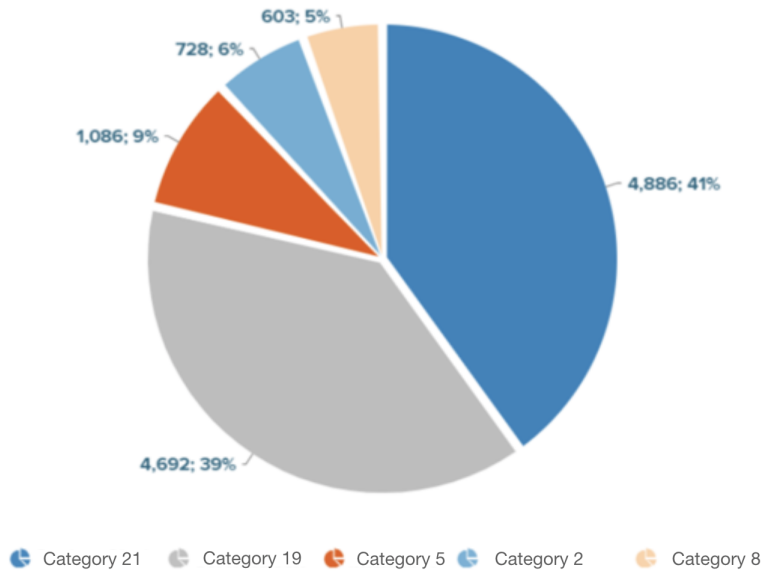


Figure 10: Category Percentages

1. constant mean
2. constant variance
3. an autocovariance that does not depend on time.

More formally, we can check stationarity using the following:

1. **Plotting Rolling Statistics:** We can plot the moving average or moving variance and see if it varies with time. By moving average(variance) we mean that at any instant 't', we will take the average(variance) of the last year, i.e. last 12 months [4].

2. **Dickey-Fuller Test:** This is one of the statistical tests for checking stationarity. Here the null hypothesis is that the time series is non-stationary. The test results comprise of a test statistic and some critical values for difference confidence levels. If the 'Test Statistic' is less than the 'Critical Value', we can reject the null hypothesis and say that the series is stationary [4].

The data used to analyze the number of tickets is the count of tickets based on the date of creation provided using Celonis API can be visualized in Figure 11.

Figure 12 shows the results using Plotting Rolling Statistics and Dickey-Fuller Test gives insight that the series is not stationary. Note that the test statistic is more than the critical value, hence we cannot reject the null hypothesis and say that the series is non-stationary.

There are typically two major reasons behind non-stationary of data in Time Series:

1. **Trend** – varying mean over time.

2. **Seasonality** – variations at specific time-frames. For e.g. a tendency of increase

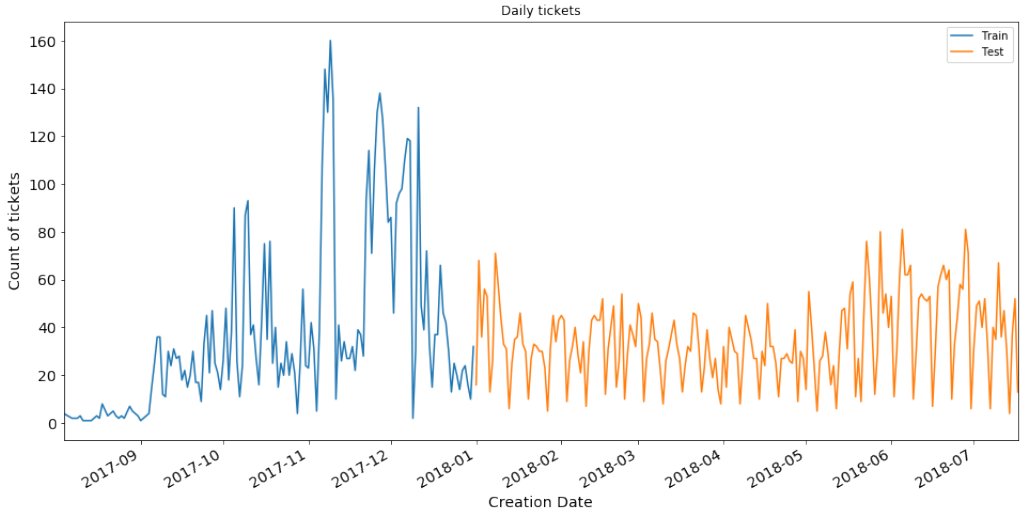


Figure 11: Count of Tickets Based on Creation Date

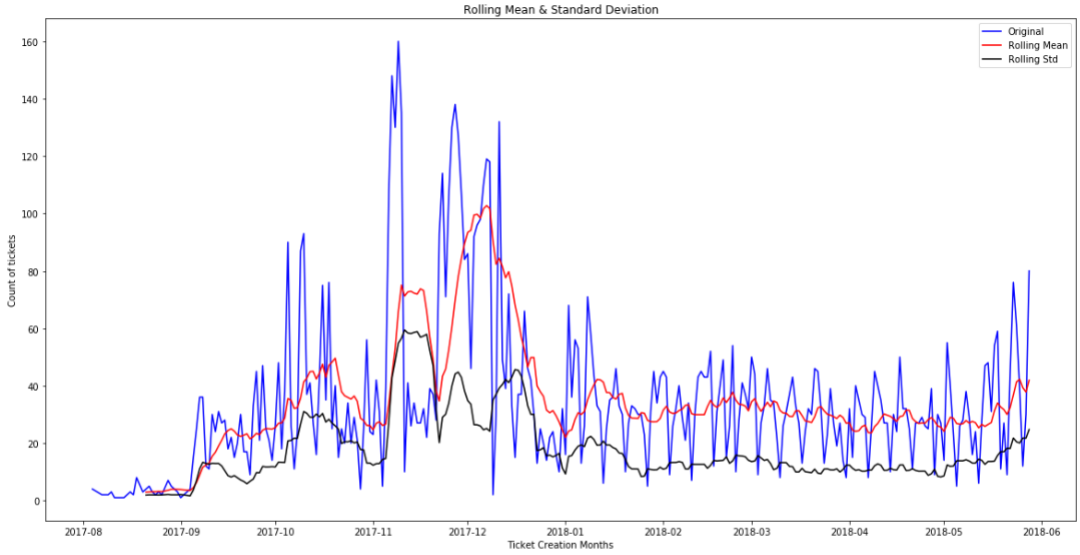
or decrease in the number of tickets in certain months.

Our aim is to model or estimate the trend and seasonality in the series and reduce it by applying transformations on count of tickets per creation date to get a stationary series. Then statistical forecasting techniques can be implemented. The final step would be to convert the forecasted values into the original scale by applying trend and seasonality constraints back. One of the ways to reduce trend can be transformation. We can apply a transformation which penalizes higher values more than smaller values. Applying a log-transform on our data leads to the Time Series shown in Figure 13.

Note that the series is more stationary than before and that the Test Statistic is less than the Critical Value, hence we can reject the null hypothesis and say that the series is non-stationary. This transformation technique was further used in Time Series Prediction Methods stated in section 5.2.4.

4.4 Limitations

Out Digital Twin model depicts the ITSM ticket processing and it allows us to test changes before implementing them across the operations. This product should be able to help to answer relevant what-if questions such as "what if we spread our team globally" or "what if we automate the first level service". Although the current dataset contains a lot of different aspects relevant to the what-if questions, there are still some limitations that we come across while building our model.
First, in this dataset, no automation data is recorded which makes it difficult to validate the automation simulation results from our Digital Twin model. Secondly, the service level of each event is not recorded directly in the data tables. To answer what-if question like “What if I bring together 2 second levels”, we need more complete information.




```
Results of Dickey-Fuller Test:
Test Statistic                  -2.608286
p-value                          0.091297
#Lags Used                      13.000000
Number of Observations Used    274.000000
Critical Value (1%)             -3.454444
Critical Value (5%)             -2.872147
Critical Value (10%)            -2.572422
dtype: float64
```

Figure 12: Count of Tickets Based on Creation Date

5 Simulation

5.1 General Approach

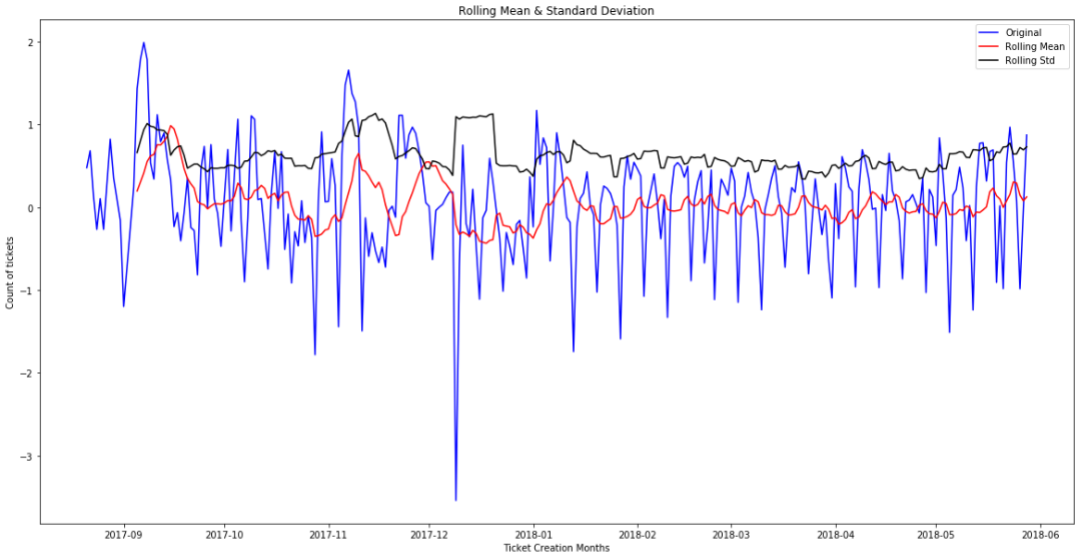
For modeling the Digital Twin of a process we envisioned two major characteristics of the twin. On the one hand, it should behave similarly to the initial process. This attribute could be described with the term "family resemblance". On the other hand, we do not wish to create an exact copy of the initial system. Instead, we want to be able to "raise" the Digital Twin differently. This enables us to observe how the twin would develop under slightly changed conditions.

In order to implement the just mentioned principles, we first track the behavior of the initial system statistically. The Digital Twin will be set up as a simulation according to these empirical observations so that we can ensure the "family resemblance". Meanwhile, we carefully choose the parameters of this simulation in a way such that they give us the opportunity to influence certain habits of the twin.

The goal of a Digital Twin is often to investigate specific what-if scenarios. In this case, the choice of parameters is very important because they must reflect the scenario in a comprehensive way.

5.2 Methodology

In this section we want to introduce the main concepts which are used to create the Digital Twin model.




```
Results of Dickey-Fuller Test:
Test Statistic                  -4.284331
p-value                          0.000473
#Lags Used                      13.000000
Number of Observations Used    263.000000
Critical Value (1%)             -3.455461
Critical Value (5%)             -2.872593
Critical Value (10%)            -2.572660
dtype: float64
```

Figure 13: Count of Tickets Based on Creation Date

5.2.1 Case Simulation

We begin with the simulation of the activity flows of the cases. It is the core of the Digital Twin model as most other attributes are simulated according to this simulation's results.

Ordinary Markov Process The first approach was to model the activity flows by a first order *Markov Process*. Considering a system consisting of a set of k states or activities $A = \{a_1, \ldots, a_k\}$, the probability to go from state x_n to x_{n+1} only depends on the current state, i.e.

$$P\left(X_{n+1} = x_{n+1} | X_0 = x_0, \ldots, X_n = x_n\right) = P\left(X_{n+1} = x_{n+1} | X_n = x_n\right),$$

where $x_i \in A$ for all $i \in [n+1]$.

This model possesses a convenient representation in terms of a stochastic matrix $P \in \mathbb{R}^{k \times k}$,

$$\begin{array}{c} \\ a_1 \\ a_2 \\ \vdots \\ a_k \end{array} \begin{array}{c} \begin{array}{cccc} a_1 & a_2 & \ldots & a_k \end{array} \\ \begin{pmatrix} p_{11} & p_{12} & \ldots & p_{1k} \\ p_{21} & p_{22} & \ldots & p_{2k} \\ \vdots & \vdots & \ddots & \\ p_{k1} & p_{k2} & \ldots & p_{kk} \end{pmatrix} \end{array} =: P.$$

Note that as a stochastic matrix, all elements are larger or equal than zero and the rows sum up to one. In this matrix we store all transition probabilities between the states, i.e.

$$p_{ij} := P\left(X_{n+1} = x_j | X_n = x_i\right).$$

If the current state of the process is activity a_j, the simulation will consider the j-th row of the matrix for the transition probabilities to the next state. To randomize the choice

of the next activity, we split the unit interval $[0,1]$ into k ordered intervals I_i, $i \in [k]$, each of the length of the probability to go to the state i, i.e.

$$\begin{aligned}[0,1] &= [t_0, t_1] \cup (t_1, t_2] \cup \cdots \cup (t_{k-2}, t_{k-1}] \cup (t_{k-1}, t_k] \\ &= [0, t_1] \cup (t_1, t_2] \cup \cdots \cup (t_{k-2}, t_{k-1}] \cup (t_{k-1}, 1] =: I_1 \cup \cdots \cup I_k,\end{aligned}$$

such that $(t_i - t_{i-1}) = p_{ji}$ for $i \in [k]$. Now, we draw a uniformly on $[0,1]$ distributed random number r and assign the next state of the process flow to the unique state i with $r \in I_i$.

To deduce the matrix in our model we consider all cases within the activity table of the given Data Model and only look at the activity flow, Overall we look at 12404 cases with in total 66371 activities. We extract the cases one by one and add one activity 'Start' at the beginning and one activity 'End' after the last activity. This conveniently embeds the problem to find the probabilities to start or end in a certain state into the algorithm which deduces the transition probabilities between the states. The transition matrix is initialized as a quadratic matrix $P \in \mathbb{R}^{d \times d}$ of dimension $d = |A| + 2$ with all entries being zero. Now, for every case, we observe every transition between two activities $\ldots x_k x_{k+1} \ldots$ where $x_k = a_i$ and $x_{k+1} = a_j$. In this case we add "+1" to the corresponding matrix entry of this activity transition $p_{ij} \mapsto p_{ij} + 1$. When this procedure is completed for all activity transitions of all cases, P will contain the absolute count of every transition between two activities. Note that the last row of the matrix is untouched because the row represents the transitions starting at 'End' whereof there are of course non. To avoid complications, we set the bottom right value of the matrix to one. The final step is to divide every row by the sum of its entries. Now, the matrix contains the transition probabilities between all pairs of activities.

If we have a look at the matrix in Figure 14 the first column and last row give us a first sanity check. It signifies that we never go to the state 'Start' and never leave the state 'End'. If we want to observe the performance of this method, we want to know how similar the cases are which are created with the algorithm. In order to do this, we just create the same number of cases as we have in our input and compare the cases.

$$P = \begin{pmatrix}
0 & 0.01 & 0.05 & 0 & 0.88 & 0 & 0.06 & 0 & 0 & 0 & 0 \\
0 & 0.05 & 0.3 & 0.02 & 0.02 & 0.12 & 0 & 0.1 & 0.27 & 0.11 & 0.02 \\
0 & 0.12 & 0.02 & 0.05 & 0.18 & 0.17 & 0.01 & 0.03 & 0.32 & 0.02 & 0.06 \\
0 & 0.04 & 0.01 & 0 & 0 & 0.1 & 0.73 & 0.03 & 0.05 & 0.03 & 0 \\
0 & 0.02 & 0.02 & 0.15 & 0 & 0.05 & 0.73 & 0 & 0.03 & 0 & 0 \\
0 & 0 & 0.01 & 0 & 0 & 0 & 0.06 & 0 & 0.03 & 0 & 0.89 \\
0 & 0.19 & 0.08 & 0.02 & 0.05 & 0.21 & 0 & 0.02 & 0.37 & 0.02 & 0.05 \\
0 & 0.06 & 0.01 & 0.01 & 0 & 0.15 & 0 & 0 & 0.41 & 0.33 & 0.04 \\
0 & 0.15 & 0.02 & 0.01 & 0.01 & 0.42 & 0.01 & 0.2 & 0 & 0.17 & 0.01 \\
0 & 0 & 0 & 0 & 0 & 0.49 & 0.07 & 0 & 0.18 & 0 & 0.25 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1
\end{pmatrix}$$

Figure 14: Matrix of Markov Process rounded on two digits

As a first indication we compare the average length of the cases and the absolute activity frequencies. The average length of the real process and of the simulation (both ≈ 7.35) lie near with a small relative error ($<0.05\%$). The relative errors of the activity

frequencies are below 3% for all activities. These low relative errors are reasonable because this aspect of the input is obviously easily captured by this model.

A more insightful attribute to look at, is how often the most frequent variants from the input appear in the simulation.

Most frequent variants	**Input**	**Model**	**Rel. error**
1	2483	1087	56%
2	1551	1444	7%
3	459	234	49 %

Table 1: Count of most frequent variants of model with Markov Process

As seen in Tabel 1 our model fails to reliably predict the main paths of the original system. We try to reduce the input data for our next model (Model 2) and restrict it to the 19 most common variants. The goal is to reduce the influence of outliers in the data. Under this assumption we are considering 6915 cases with overall 28201 activities

Most frequent variants	**Input**	**Model 2**	**Rel. error**
1	2483	1665	33%
2	1551	1621	5%
3	459	158	66%

Table 2: Count of most frequent variants of Model 2

Unfortunately, this does not yield a significant improvement as we can see already in Table 2. However, if we take the average over the relative errors of the variant frequency of the three most frequent variants weighted with the total number of occurrences of the variants in the input data, we see an improvement from 38% to 27%.

The best way to simulate the activity flow given our tools so far turns out to be by splitting the input data into groups according to specific attributes and then individually treating those groups. In order to do this, we group the cases from the input according to their first activity. Then, we deduce the transition matrix for every such group. For the simulation, we randomly determine the first activity of a case proportional to the frequency in the input data and then simulate the case with its group's Markov Process.

We performed this technique on both the complete and reduced input data and tested it by simulating for every starting activity the number of cases which we have in the input data. We refer to them as Model 3 with full input and Model 4 with reduced input. Both are upgrades compared to the original simulation.

Most frequent variants	**Input**	**Model 3**	**Rel. error**	**Model 4**	**Rel. error**
1	2483	1380	50%	1859	25%
2	1551	1574	1%	1799	16%
3	459	261	44%	183	60%

Table 3: Count of most frequent variants of Model 3 and Model 4

This can be seen also if we look again at the average relative error of the variant frequency for the three most frequent variants weighted with the total number of occurrences in the input data. In Table 4, the error decreases from the top-left to the bottom-right.

That indicates, that both approaches which were stated above can improve the quality of our simulation. The patterns in Tables 1 through 4 can be observed as well considering more variants but we refer to the three most common variants for simplification.

	Full Input	**Reduced Input**
Markov Process	38%	27%
Markov Process with Grouping	32%	25%

Table 4: Weighted average of relative error of counts of variants

It is important to note that using a first order Markov Process implies the assumption that the assignment of the next state only depends on the current state. In fact, the assignment of activities is also influenced by the state history and the position of the activity within a process flow . While this problem could be improved by grouping cases with specific characteristics and separately treating those groups as shown above, we explored alternative approaches as well.

In the following, we will abbreviate $P(X = x)$ as $P(x)$.

Linear Additive Markov Process While searching for a simulation technique which would address the dependence of the state assignment on the state history, we discovered the *Linear Addaptive Markov Process* which is also referred to as LAMP. The idea is taken from the paper [6] and we adapted the technique to fit our requirements.

Like in the ordinary Markov Process we use a stochastic matrix P to store the transition probabilities between the states. In vast contrast to the ordinary Markov Process though, we will take into account the state history up to a memory of $M \in \mathbb{N}$ and not only consider the current state x_n. In particular we include a stochastic vector w of dimension M which expresses how much we value the history. With probability w_i, we proceed as if we were in state x_{n-i+1} and determine the next state accordingly using P.

This procedure can be expressed as follows.

$$P(x_{n+1}|x_0, \ldots, x_n) = \sum_{i=1}^{M} w_i \cdot P\left(x_{\max\{n-i+1,0\}}, x_{n+1}\right)$$

Note that the intuitive choice to couple this technique with the matrix of the ordinary Markov process and add a reasonable w is not possible. Just consider two activities, which in the initial system never occur after each other, but often with one activity in between. For $w_2 > 0$, this activity transition would have a positive probability of occurrence in the LAMP model which is of course undesirable.

In fact, the parameters w and P for LAMP have to be learned.

The paper [6, p. 415f] suggests to choose as loss function the negative log likelihood of one single case. This would obviously cause strong overfitting on the inserted case. Therefore, we use a batch of multiple cases for the loss function and sum over their negative log likelihoods. For a batch of size B and cases $c_1, \ldots, c_B \in A^{\mathbb{N}}$ with lengths $L_1, \ldots, L_B \in \mathbb{N}$, the loss function has the shape

$$L(w, P, c_1, \ldots, c_B) = -\sum_{k=1}^{B} \left[\sum_{l=1}^{L_k} \log\left(\sum_{i=1}^{M} w_i \cdot P\left(x^k_{\max\{l-i,0\}}, x^k_l\right) \right) \right]. \tag{3}$$

We remark that this function is individually convex in w and P. In the log we have an affine function in w, P which is thus concave. If we couple the log(.) with the minus sign,

we get a convex function. The concatenation of a convex function with an affine function is convex again according to [2, p. 79]. The convexity follows because the sum of convex functions is convex.

The individual convexity suggests to minimize the loss function alternatingly in w and P. This task is quite laborious, due to the parameters being stochastic. Let us observe the minimization in w. We fix P and the batch of cases c and get the constrained optimization problem

$$\operatorname{argmin}_w L(w, P, c) \quad \text{subject to} \quad w \geqslant 0, \quad \mathbb{I}^T w = 1,$$

with $\mathbb{I}$ a one vector of dimension M. The matrix P can be treated analogously by considering the rows separately and performing block descent as stated in [6, p. 416].

We used the Python package *scpiy optimize* [5] in order to find the solutions. The optimization routine is set up as follows in Python. We hand a number of iterations, batch size and the desired memory to a function which will perform the routine. This function will internally call the function *scipy.optimize.minimize*. We will perform block descent for every row of the matrix P separately, followed by a minimization with regard to w. These operations happen on the same batch of activity flows. This concludes one iteration and we will repeated this until the improvement of the iteration falls below a threshold or the number of iterations is reached.

We want to mention, that the optimization usually improves over some iterations but then tends to go to unfeasible solutions. The iteration will return to feasible solutions but the loss goes back to values near the starting level. This phenomenon is more likely the bigger the batch size is chosen. It leads to a trade off because we want batch sizes of size 30 to 50 to have a good mix of process flows within the batch on the one hand. On the other Hand, our algorithm exhibits more violations of the constraints then.

This problem could be improved by normalizing the terms of the sums in equation 3. The loss function L becomes then

$$L(w, P, c_1, \ldots, c_B) = -\sum_{k=1}^{B} \frac{1}{B} \left[\sum_{l=1}^{L_k} \frac{1}{L_k} \log \left(\sum_{i=1}^{M} w_i \cdot P\left(x^k_{\max\{l-i,0\}}, x^k_l \right) \right) \right].$$

We decided however not to use this alternative approach primarily because the deduced activity simulations of two different pairs of parameters w, P could vary significantly even if they were trained with the same training parameters (number of iterations, batch size, memory).

A different way to receive w and P could potentially be found using the KKT conditions and the water-filling problem [2, p.245f].

5.2.2 Category Simulation

The categories are assigned to every activity of a case using two functions. We model the flow of categories again as a Markov Process and distribute the starting category according to the distribution of first categories in the input. The first function assigns a starting category to the first activity. The probability to pick a certain category is proportional to the distributions of starting categories in the input. This category is assigned to every activity until the activity 'Change category' appears. Here we look at the corresponding row of the transition matrix of the current activity and assign the new one analogously to the simulation of the activity simulation.

Similar considerations like before have to be done about the independence from the category history. Due to time restrictions we were not able to incorporate those. The main issue is, that cases are generally influenced by the underlying category. The throughput times are affected and it also influences the flow of activities. According to our research to improve the activity flow simulation, a sequential change of the simulation would be the easiest way to combat the issue. Note that the current simulation structure is

"activities" → "first category" → "following categories".

We simulate all activities of a case first and then add the category information. Instead we could proceed as follows:

"first category" → "activities 1" → "next category" → "activities 2" → ...

Here we determine the starting category first according to the relative occurrence in the input. Then, "activities 1" is a Markov Process which simulates the activities. It is only trained with case information up to the first category change with underlying first category and we stop if the simulation reaches the activity 'Change category'. Now, the next category is determined with the Markov Process for categories, followed by "activities 2". This is again a Markov Process for activities, which is trained with the case information beginning with the activity 'Change category' together with the new assigned category. This ends when we reach a category change again and we go back to the step "next category".

The tools for this procedure are all implemented. Only the data must be separated correctly. We remark that the complexity will grow. Our current model uses only to transition matrices for the activity and category simulation. The new one requires far more. For an input with n different categories we need $2n + 1$ matrices.

5.2.3 Throughput Time Simulation

We will now discuss the algorithm that we used to simulate the throughput time between arbitrary activities. We decided to mainly use throughput time from 'Status: New' to 'Change assignee'. This is because 'Change assignee' is the most frequent one among the activities which are not on 'Happy Path'[1]. Thus we can acquire the enough number of data to be divided into 'train', 'validation' and 'test' set, as well as analyze the cases out of Happy Path.

Kolmogorow-Smirnow-Test We would repeatedly apply Kolmogorow-Smirnow-Test (KS-Test) to compare two distribution. Its null and alternative hypotheses are stated below.

H_0: $F_X(x) = F_Y(x)$
(The random variables X and Y have the same probability distribution)

H_1: $F_X(x) \neq F_Y(x)$
(The random variable X has a different probability distribution from Y)

Therefore, if we acquire p value higher than 0.05, we cannot reject H_0 and make decision that the given two data sets have same distribution.

[1]Recall that the path is the top frequent variant made up of sequential activities. They are 'Create Ticket', 'Status: New', 'Status: Open', and 'Status: Closed'.

Modeling by parametric way After data exploration, we recognized that throughput time distribution in the business process typically has non-negative, skewed distribution. We started from parametric method such as *Poisson, Exponential, Gamma* and other theoretical distribution in order to fit model to data. The first model has formed by *Poisson* on the ground that it is general theoretical distribution to express the probability of a given number of events occurring in a fixed interval of time. We generated random numbers from *Poisson* distribution with $\lambda = \bar{X}$ which is sampled mean time. Then we apply KS-Test to compare the numbers with original data. Unfortunately, KS-Test suggests 0.0 p-value which shows that we cannot describe throughput time with *Poisson*. Futhermore, *Exponential* and *Gamma* give similar result. They are not matched to the real distribution. The result has been suggested in Table 5 where $\bar{X}$ means the sample mean, while S^2 does sample variance. $\hat{\lambda}$, $\hat{\theta}$, $\hat{k}$ are estimators of parameters for each . To summarize, we are not able to adopt these parametric to simulate the throughput time distributions, according to KS-Test.

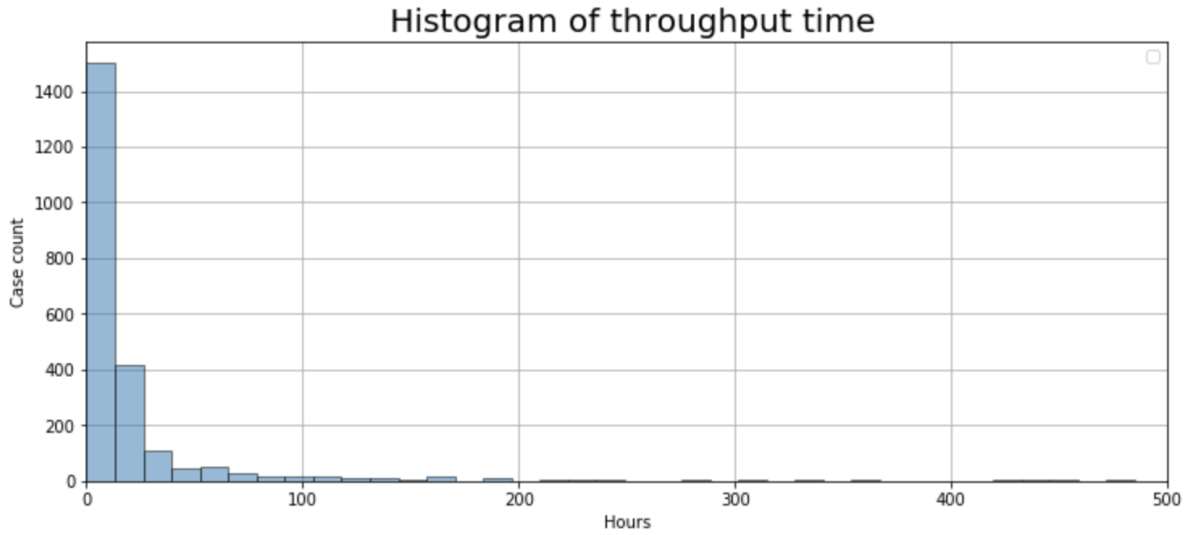


Figure 15: Throughput time from 'Status: New' to 'Change assignee' in hours.

Distribution	KS-statistic	p-value	Parameter
Poisson	0.6768	0.0	$\hat{\lambda} = \bar{X}$
Exponential	0.4403	0.0	$\hat{\lambda} = 1/\bar{X}$
Gamma	0.9957	0.0	$\hat{\theta} = S^2/\bar{X}, \quad \hat{k} = \bar{X}/\theta$

Table 5: KS-Test result with three parametric distribution.

Bins-method The results have shown that we need other methods to model the throughput time. Therefore we did create naive way to mimic the distribution of histogram, called *'Bins-method'*. First, we separated the data into train(80%) and test set(20%). Second, the train data set is splitted into 30 bins: $\{B_1, B_2, ..., B_{30}\}$. In detail, B_{30} has the biggest 3.33% data. B_{29} has next 3.33% and so on. Therefore B_1 is the set of the minimal 3.34% data. Next, we generate random number by $Uni(min(B_i), max(B_i))$ within each bean. And finally, apply KS-Test for the test set and the generated data. Thereby we have

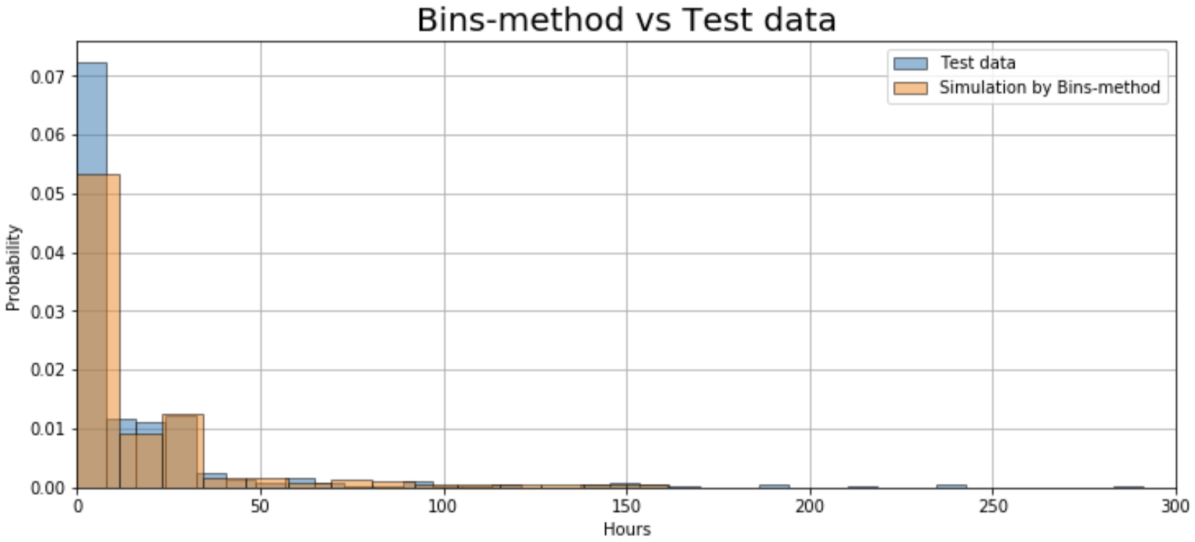


Figure 16: Simulation by *Bins-method*.p-value for KS-Test is 0.4575

succeeded to simulate the same distribution with the test set. On average, the KS-Test between the distribution by *Bins-method* and the test-set outputs higher p-value than 0.05. The average p-value has been 0.438 after 1,000 rounds.

Introduction to Kernel Density Estimation (KDE) *Bins-method* is somehow a naive way because it provides uniformly random numbers. This is a drawback because uniform distribution does not describe outliers enough. Kernel Density Estimation(KDE) is our alternative way to compensate this drawback. It is a non-parametric way to estimate the probability density function(PDF) of a random variable.

$$\hat{f}(x) = \frac{1}{nh}\Sigma_{i=1}^{n} K\left(\frac{x - X_i}{h}\right)$$

The above formula is called the Kernel estimator. Here h is the smoothing parameter. In this paper, we call it a bandwidth. The function K is called the kernel, and it controls the weight given to the observations X_i at each point x based on their proximity. Thus, we expect KDE would compensate the weak points of *Bins-method* and calculate smooth PDF.

Figure 17 shows simulated PDF by KDE method. In this example, the kernel is uniform function and bandwidth is 1. Among total throughput time data, 80% has been randomly sampled for training and 20% for the test. The simulated distribution has become much smoother compared to *Bins-method* in Figure 16, 17.

Kernel selection for Digital Twin The type of kernels and size of bandwidth is important parameter for KDE. Which kernels and bandwidth should we choose? From now, we would test each kernel and discuss selection of bandwidth for Digital Twin. In order to implement this task, it is necessary to see how different Python packages perform regarding KDE. Table 6 explains three types of packages in Python. When testing kernels, Statsmodels is the best choice which offers 8 types of kernels. We can

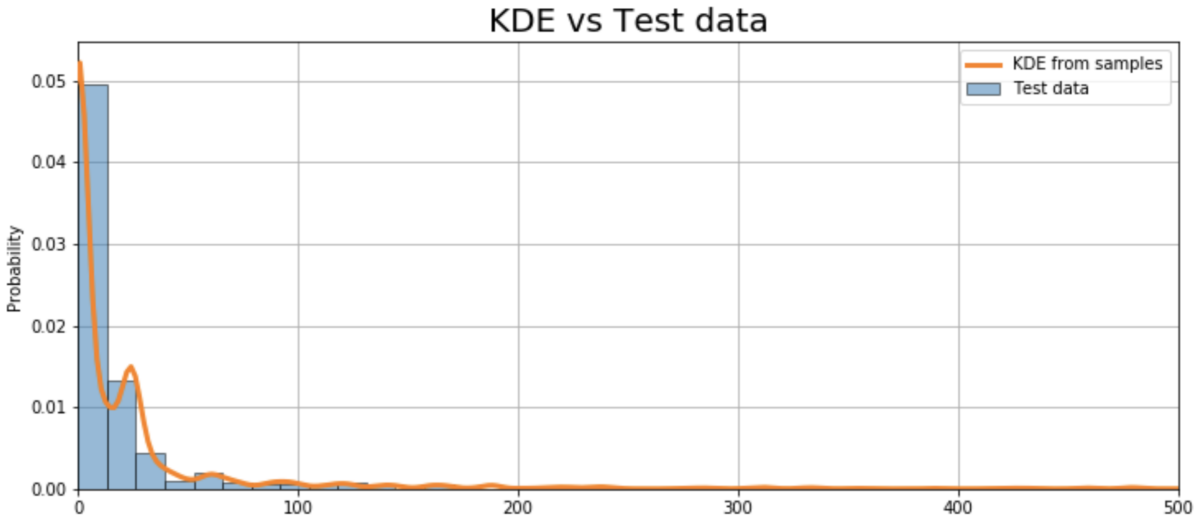


Figure 17: KDE for the throughput time

ignore the performance of bandwidth selection in that we will not use it. In other words, we will select bandwidth according to KS-Test, and this somehow requires manual coding.

Here, we would test the performance of kernels by simulating same train and test data. In a bid to implement this task, we need to generate random number. Unfortunately, Statsmodels does not provide sampling function. Therefore we have coded random number generator $G(.)$ manually by inverse Cumulative Distribution Function(CDF) and uniform function.

$F_X^{-1}(x) : [0, 1] \rightarrow T\{t_1, t_2, ..., t_n\}$ where T is throughput time dataset.

$G(x_1, x_2) = Uni(F_X^{-1}(x_1), F_X^{-1}(x_2)) = s$

where s is simulated throughput time and $Uni(t_i, i_j)$ generates random number uniformly within the interval $[t_i, t_j]$

This method has drawbacks. First, $G(.)$ approximates true distribution by uniform function. Secondly, it cannot generate random value greater than $max(T)$

	Available Kernels	Bandwidth Selection	Random Number Geration
Scipy	One(Gaussian)	Scott and Silverman	Available
Statsmodels.KDEUnivariate	8	Scott and Silverman	Not available
Scikit-Learn	6	Not built-in	Available

Table 6: Comparison of KDE in Python packages.

In a bid to know the best kernel for Digital Twin, We have tested the all 8 kernels with default size of bandwidth, 1. For each kernel, we trained KDE model with train data and generate 100 random data. Then we compare test-set and the simulated data by KS-Test. After implementing this procedure 1,000 times, we calculated average P-value as well as standard deviation. Table 7 describe this result. It seems that different kernels lead little difference when they have default bandwidth, 1.
If we tilt the bandwidth, can we differentiate them? In table 7, Gaussian, uniform, cosine2, and cosine kernels display the most highest p-value somehow. Next step is to

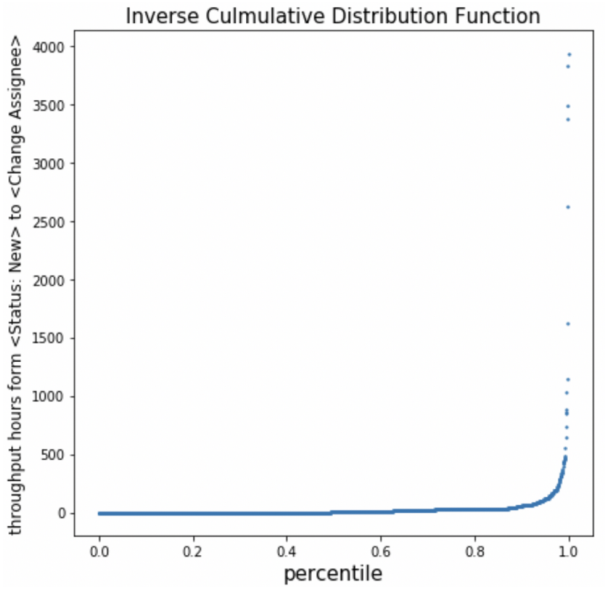


Figure 18: inverse CDF

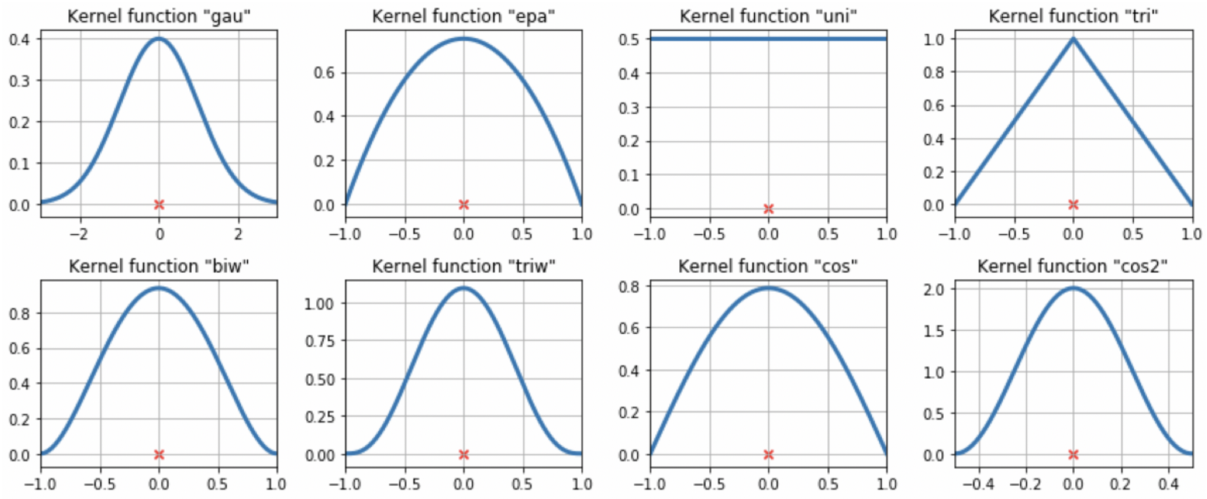


Figure 19: 8 kernels in Statsmodels package

test them while tilting them(Figure 20). Note that Gaussian kernel has long tail, so it is more likely to generate negative value. This happens especially when bandwidth is higher than 0.1.

In order to take a closer look, we have quantified how many negative values occurred. Table 8 means that Gaussian may generate negative time value when bandwidth is equal to 1. In Digital Twin simulation, the range of grid search would be from 1 to 1e-4. In this range, the difference among kernels is not observed, except Gaussian.

Then which kernel should be adopted? Recall that the Statsmodel package does not have internal sampling method, so the function is made by hand. Therefore, to calculate more accurate result, we have chosen uniform kernel on the ground that Sklearn package include internal sampling function for this. Meanwhile, uniform kernel has a shortage that it does not describe continuous PDF properly. However, we simulate time data with specific unit such as hours, minutes or seconds. This means that actually original PDF has discrete form. Therefore we have simulated throughput time by using uniform kernel

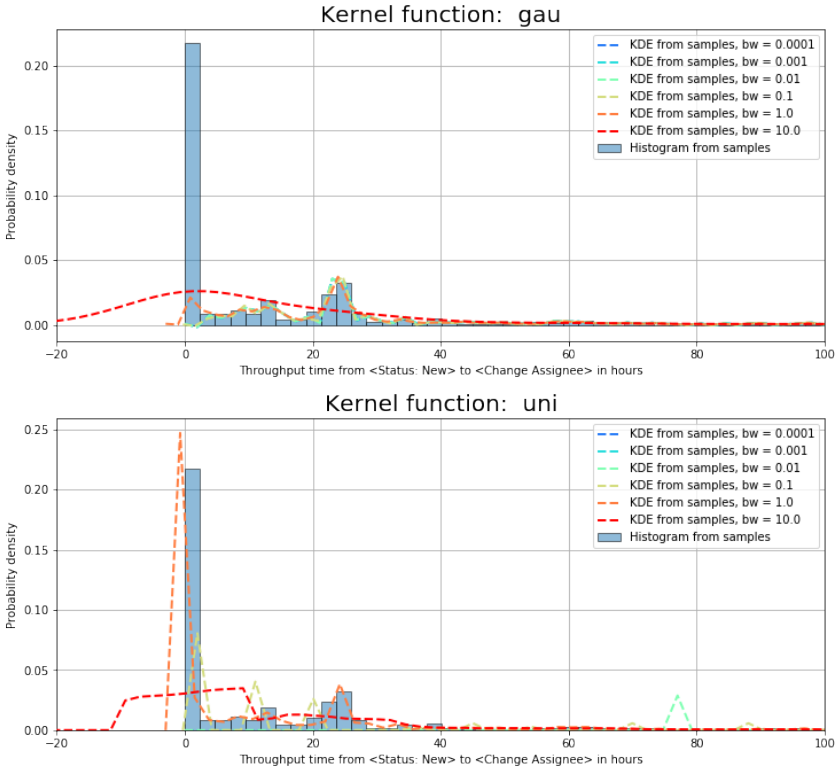


Figure 20: KDE with tilted bandwidths

and SKlearn[2].

Kernel	gau	cos2	cos	triw	biw	tri	epa	uni
Average	0.66809	0.66366	0.66557	0.65411	0.64872	0.66313	0.65425	0.6621
Standard deviation	0.30096	0.30992	0.30718	0.31251	0.31709	0.30823	0.31093	0.30595

Table 7: Average and Standard deviation of P-value for KS-test

Tunning bandwidth for Digital Twin So far, we have researched what the optimal kernel for throughput time simulation is. In this section, we would discuss to search optimal bandwidth, the other important parameter for KDE. We start to tune from 0.1 as a default value for this parameter. This is because we have observed larger bandwidth greater than 0.1 probably allows negative time value. There are three way of tuning. First one is to implement KDE with constant bandwidth. Second method is grid search. The last one is a grid search with cross validation.

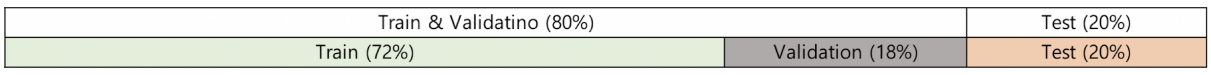


Figure 21: Design of dataset

Figure 21 reports how train, validation and test datasets are designed as well as comparison of simulated data to test set.

[2]The uniform kernel is named as 'tophat' in Sklearn package

Kernel	gau	cos2	cos	triw	biw	tri	epa	uni
10	0.147432	0.120265	0.136335	0.110505	0.10969	0.101762	0.118282	0.060792
1	0.002171	0	0	0	0	0	0	0
1.00E-01	0	0	0	0	0	0	0	0
1.00E-02	0	0	0	0	0	0	0	0
1.00E-03	0	0	0	0	0	0	0	0
1.00E-04	0	0	0	0	0	0	0	0

Table 8: Probability where KDE generates negative value

Basically *KDE with constant bandwidth(0.1)* work as follows:

1) Train: split data into train and test-set by the ratio of 80 to 20. Then separate train set again by 80 to 20 to create validation set. Next, train distribution model with KDE. This time, 'tophat'(equal to uniform) kernel was used.

2) Validation: internally validate the model and repeat to train until p-value becomes higher than 0.25 in KS-Test. Generates random numbers with KDE.

3) Test : This step is not included in actual Digital Twin model, because test is only available after final model has been made. KS-Test works again to evaluate the constant bandwidth

We have repeated 1) - 3) and acquired p-value of 0.5608 on average in step 3). It takes 4.006 seconds to run this code.

Gridsearch method work by following three steps:

1) Train: This time keep changing size of bandwidth while training. Train and test data are organized as same above. In other words, gridsearch strategy is to search one bandwidth among $\{1, 10^{-1}, 10^{-2}, 10^{-3}, 10^{-4}\}$ which outputs the highest p-value.

2) Validation: implement KDE with the selected bandwidth. Repeat 1) and 2) stages until p-value is higher than 0.25

3) Test (not in Digital Twin): Apply KS-Test for the simulated data and test set.

After running this procedure 1,000 rounds, the result has shown the mean of p-value in step 3) is 0.5986, and the code takes 12.5433 seconds to run 1,000 times.

The final experiment has been implemented by *gridsearch with cross-validation*.

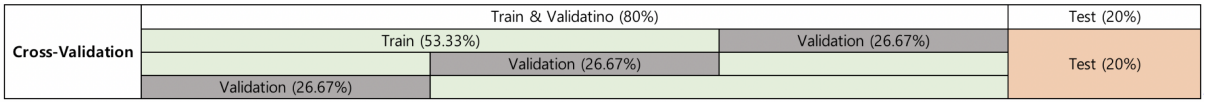


Figure 22: Design of dataset to for cross validation

1) Train: after dividing train and test set, we created three pairs of train and validation set. See Figure 22. This time, we again picked the bandwidth among $\{1, 10^{-1}, 10^{-2}, 10^{-3}, 10^{-4}\}$. On the other hand, the bandwidth gives the highest *average* p-value across the three data set. As a result it requires timely expensive cost of computation. Train the KDE model which has this p.

2) Validation: KS-Test for validation set and created number by KDE.

3) Test (not in Digital Twin): Apply KS-Test for the simulated data and test set.

Its average p-value in 3) step is 0.5804. It runs 21.5929 seconds for 1,000 round.

Bandwidth	Running Times(1,000 rounds)	Avg(p-value)	Std(p-value)
0.1 (Constant)	4.0061	0.5608	0.3396
Gridsearch	12.5433	0.5986	0.3187
Cross-Validation	21.5929	0.5804	0.3241

Table 9: Comparison of ways to tune bandwidth. Std means standard deviation

Running time strongly and obviously depends on strategy of tuning parameter. In contrast, we can hardly differentiate the above three method with respect to p-value. Thus we have experimented with smaller data and the result is mentioned in Table 10. The given data set comes from throughput time from 'Change priority' to 'Status: Solved'. It has only 68 cases which dominates only 1% of the total. As we have observed so far, we rarely tell the difference except running time.
At Model Training section, we deal with this issue again. We would compare the final Digital Twin model while only changing three KDE methods. They do not show statistical difference.

Dependency test In real world, throughput time between activities are not independent. If sequential activities happen, they affect each other. Because we did not take this factor into account in our model, we need to test the dependency. If we find dependency, this means we cannot avoid error.

We have used Pearson's chi-squared test. This is statistical test applied to sets of categorical data to evaluate how likely it is that any observed difference between the sets arose by chance[3]. By this test, we can assess Independence of variables. In this case,

Bandwidth	Avg(p-value)	Std(p-value)
0.1 (Constant)	0.3522	0.1989
Gridsearch	0.3680	0.2061
Cross-Validation	0.3594	0.2063

Table 10: Comparison of ways to tune bandwidth. Std means standard deviation

0 ~ 1976	1976 ~3952	3952 ~ 5928	5928 ~ 7903	over 7903
174	36	53	214	65

Figure 23: Count of throughput time from 'Status: On Hold' to 'Status: Solved'

0 ~ 3982	3982 ~ 7963	7963 ~ 11945	11945 ~ 15926	over 15926
107	38	41	21	20

Figure 24: Count of sequential throughput time: 'Status: Solved' to 'Status: Closed'

we have following null and alternative hypotheses

H_0: there is no difference between the distributions.

H_1: there is a difference between the distributions

'no difference' means the given distribution is consistent regardless of counter variable. If rejected, then we can decide the variables affect each other, so they are dependent. The test statistics Z is defined as

$Z = \sum \chi^2 = \sum \frac{(O-E)^2}{E}$ where O is Observed and E is Expected value.

In this section, we test the dependency of two kinds of throughput time. One is time from 'Status: On hold' to 'Status: Solved'. The other one is time from 'Status: Solved' to 'Status: Closed'. This is because they are bottle of the given process. As a result, they have high throughput time so that the data satisfies Chi-squared assumption. Details about the assumptions are followed later.

In order to apply this method, we need to change our time data as nominal(categorical) form. We have generated 5 bins at interval of $d = max(T)/5$. Here T is a dataset without top 10% values. For example,

data in $[0, d)$ belongs to the first bin
data in $[d, 2d)$ belong to the second bin
data in $[2d, 3d)$ belong to the second bin
data in $[3d, 4d)$ belong to the ninth bin
data greater than $4d$ belong to tenth bin

Refer to (Figure 23 and 24). In the figures, we used minutes as time unit in order to split time in detail.

Next, we count sequential throughput time. With contingency matrix, we have expressed count for the sequential activities. For instance, if there is on case which takes 1,000 minutes from 'Status: On Hold' to 'Status Solved' and takes 3,000 minutes from 'Status: Solved' to 'Status: Closed', then it we add count (1, 1). See Figure25. Even without Chi-squared test, we can directly see the dependency in the Figure. In other words, one variable happens more frequently when count of the other goes up. We applied Chi-squared test in R and acquired a almost zero p-value(Table 11).

Pearson's Chi-squared test requires to satisfy following assumptions[7]. Before conclusion, check whether they have been all fulfilled.

Status: On Hold' to 'Status: Solved'

Status: Solved' to 'Status: Closed'	0 ~ 1976	1976 ~ 3952	3952 ~ 5928	5928 ~ 7903	over 7903
0 ~ 3982	170	32	12	1	1
3982 ~ 7963	1	1	38	85	1
7963 ~ 11945	1	1	1	89	1
11945 ~ 15926	1	1	1	18	1
over 15926	1	1	1	21	61

Figure 25: Contingency matrix: frequency of throughput time from 'Status: On Hold' to 'Status: Solved' to 'Status: Closed'

Pearson's Chi-squared test		
χ^2	Degree of freedom	p-value
849.23	16	2.2e-16

Table 11: Chi-squared test result by 'stats' package in R

1) The data in the cells should be frequencies: we have generated contingency matrix whose elements represent count(frequency).

2) The levels (or categories) of the variables are mutually exclusive: it is holds because each variable has unique counter value.

3) Each subject may contribute data to one and only one cell in the χ^2.

4) The study groups must be independent. This means that a different test must be used if the two groups are related: this assumption is satisfied because each cases are independent in activity tables.

5) There are 2 variables, and both are measured as categories, usually at the nominal level: we have splitted data and create nominal values.

6) The value of the cell expecteds should be 5 or more in at least 80% of the cells, and no cell should have an expected of less than one: this is why we selected the bottle neck process. Currently every expected value is higher than 1 and 88% elements of contingency matrix are bigger than 5. See the matrix in Figure 26.

Therefore, all the assumptions hold.

Status: On Hold' to 'Status: Solved'

Status: Solved' to 'Status: Closed'	0 ~ 1976	1976 ~ 3952	3952 ~ 5928	5928 ~ 7903	over 7903
0 ~ 3982	69.34317	14.34686347	21.12177122	85.28413284	25.904059
3982 ~ 7963	40.45018	8.36900369	12.32103321	49.74907749	15.110701
7963 ~ 11945	29.85609	6.177121771	9.094095941	36.7195572	11.153137
11945 ~ 15926	7.062731	1.461254613	2.151291513	8.686346863	2.6383764
over 15926	27.28782	5.645756458	8.311808118	33.56088561	10.193727

Figure 26: Expected Contingency matrix: 80% of the contingency matrix should have expectations more than 5 for Pearson's Chi-squred

To summarize, we adopt actual data and test if the throughput time violates our

independence assumption. According to χ^2-Test, there are dependency. Furthermore, our data satisfied required assumptions for the test. As a result, there might be inevitable error, because our current model is built on independence assumption.

5.2.4 Time Series Prediction Methods : Number of Cases Prediction

Understanding the Problem Statement

To predict the number of tickets at a specific date so as to simulate the data for a given date in our model, we have used Time Series approach to generate the predictions. Dataset comprises of 11 months of data (Oct 2017-July 2018) and using this data we have to forecast the number of cases or tickets depending on the time span of prediction period stated.

Characteristics of Time Series

The analysis of experimental data that have been observed at different points in time leads to new and unique problems in statistical modeling and inference. The obvious correlation introduced by the sampling of adjacent points in time can severely restrict the applicability of the many conventional statistical methods traditionally dependent on the assumption that these adjacent observations are independent and identically distributed. The systematic approach by which one goes about answering the mathematical and statistical questions posed by these time correlations is commonly referred to as time series analysis. The impact of time series analysis on scientific applications can be observed in diverse fields. For example, many familiar time series occur in the field of economics, where we are continually exposed to daily stock market quotations or monthly unemployment figures. In medicine, blood pressure measurements traced over time could be useful for evaluating drugs used in treating hypertension.

Time Series Statistical Models

The primary objective of time series analysis is to develop mathematical models that provide plausible descriptions for sample data. In order to provide a statistical setting for describing the character of data that seemingly fluctuate in a random fashion over time, we assume a time series can be defined as a collection of random variables indexed according to the order they are obtained in time. For example, we may consider a time series as a sequence of random variables, x1, x2, x3, . . ., where the random variable x1 denotes the value taken by the series at the first time point, the variable x2 denotes the value for the second time period, x3 denotes the value for the third time period, and so on. In general, a collection of random variables, xt, indexed by t is referred to as a stochastic process. In this text, t will typically be discrete and vary over the integers t = 0, 1, 2, ..., or some subset of the integers. The observed values of a stochastic process are referred to as a realization of the stochastic process.

Method 1: Naive Approach

Many a times we are provided with a dataset, which is stable throughout its time period. If we want to forecast the number of tickets for the next day, we can simply take the last

day value and estimate the same value for the next day. Such forecasting technique which assumes that the next expected point is equal to the last observed point is called Naive Method.

$$\hat{y}_t + 1 = y_t$$

Method 2: Simple Average

When we are provided with a data which might vary by a small margin throughout its time period, but the average at each time period remains constant. In such a case we can forecast the number of tickets of the next day somewhere like the average of all the past days. Such forecasting technique which forecasts the expected value equal to the average of all previously observed points is called Simple Average technique.

$$\hat{y}_t + 1 = \frac{1}{t}\sum_{i=1}^{t} y_i$$

Method 3: Moving Average

In order to use the previous stated simple average method, we must use the mean of all the previous data but using all the previous data is usually not the right approach. For example, using the tickets of the initial period would highly affect the forecast for the next period. Therefore, as an improvement over simple average, we will take the average of the tickets for last few time periods only. Clearly, the reasoning here is that only the recent values matter. Such forecasting technique which uses window of time period for calculating the average is called Moving Average technique. Calculation of the moving average involves what is sometimes called a "sliding window" of size n.

Using a simple moving average model, we forecast the next values in a time series based on the average of a fixed finite number 'p' of the previous values. Thus, for all i >p

$$\hat{y}_l = \frac{1}{p}(y_{i-1} + y_{i-2} + y_{i+3} \cdots + y_{i-p})$$

An advancement over Moving average method is Weighted moving average method. In the Moving average method as seen above, we equally weigh the past 'p' observations. But we might encounter situations where each of the observation from the past 'p' impacts the forecast in a different way. Such a technique which weighs the past observations differently is called Weighted Moving Average technique.

A weighted moving average is a moving average where within the sliding window values are given different weights,so that more recent points matter more. Instead of selecting a window size, it requires a list of weights (which should add up to 1). For example, if we choose [0.40, 0.25, 0.20, 0.15] as weights, we would be giving 40%, 25%, 20% and 15% to the last 4 points respectively [8].

$$\hat{y}_l = \frac{1}{p}(w_1 y_{i-1} + w_2 y_{i-2} + w_3 y_{i+3} \cdots + w_p y_{i-p})$$

Method 4: Simple Exponential Smoothing

We can observe that both Simple average and Weighted moving average are completely opposite. We would need something between these two extremes approaches which consider all the data while weighing the data points differently. For example, it may be sensible to attach larger weights to more recent observations than to observations from the distant past. The technique which works on this principle is called Simple exponential smoothing.

Forecasts are calculated using weighted averages where the weights decrease exponentially as observations come from further in the past, the smallest weights are associated with the oldest observations:

$$\hat{y}_{t+1|t} = \alpha y_t + \alpha(1-\alpha)y_{t-1} + \alpha(1-\alpha)^2 y_{t-2} + \dots$$

where $0 \leq \alpha \leq 1$ is the smoothing parameter.
The one-step-ahead forecast for time t+1, is a weighted average of all the observations in the series $y_1, \dots, y_t$. The rate at which the weights decrease is controlled by the parameter α. The expected value $\hat{y}$ is the sum of two products: αy_t and $(1-\alpha)\hat{y}_{t-1}$
Hence, it can also be written as:

$$\hat{y}_{t+1|t} = \alpha y_t + (1-\alpha)\hat{y}_{t|t-1}$$

Essentially, we achieve a weighted moving average with two weights: α and $1-\alpha$. Since $1-\alpha$ multiplied by the previous expected value $\hat{y}_t - 1$ makes the expression recursive, this method is called Exponential. The forecast at time t+1 is equal to a weighted average between the most recent observation y_t and the most recent forecast $\hat{y}_t|t-1$ [8].

Method 5: Double Exponential Smoothing or Holt's Linear Trend method

Sometimes the models stated above don't work well on data with high variations. If we use any of the above methods, it will not consider this trend. Here, trend is the general pattern of tickets that we observe over a period.

Although each one of these methods can be applied to the trend as well. E.g. the Naive method would assume that trend between last two points is going to stay the same, or we could average all slopes between all points to get an average trend, use a moving trend average or apply exponential smoothing. But we need a method that can map the trend accurately without any assumptions. Such a method that considers the trend of the dataset is called Holt's Linear Trend method.

Series decomposition will help us as we obtain two components: intercept (i.e. level) l and slope (i.e. trend) b. We have learnt to predict intercept (or expected series value) with our previous methods; now, we will apply the same exponential smoothing to the trend by assuming that the future direction of the time series changes depends on the previous weighted changes. As a result, we get the following set of functions:

$$Level\ l_t = \alpha y_t + (1-\alpha)(l_{t-1} + b_{t-1})$$
$$Trend\ b_t = \beta(l_t - l_{t-1}) + (1-\beta)b_{t-1}$$
$$Forecast\ \hat{y}_t + 1|t = l_t + b_t$$

The first one describes the intercept, which, as before, depends on the current value of the series. The second term is now split into previous values of the level and of the trend. The second function describes the trend, which depends on the level changes at the current step and on the previous value of the trend. In this case, the β coefficient is a weight for exponential smoothing. The final prediction is the sum of the model values of the intercept and trend [8].

Method 6: Triple Exponential Smoothing or Holt Winter's Method

The idea is to add a third component - seasonality. This means that we should not use this method if our time series is not expected to have seasonality. Seasonal components in the model will explain repeated variations around intercept and trend, and it will be specified by the length of the season, in other words by the period after which the variations repeat. For each observation in the season, there is a separate component; for example, if the length of the season is 7 days (a weekly seasonality), we will have 7 seasonal components, one for each day of the week.

The Holt-Winters seasonal method comprises the forecast equation and three smoothing equations — one for the level l_t, one for trend b_t and one for the seasonal component denoted by s_t, with smoothing parameters α, β and γ.

$$Level\ l_t = \alpha(y_t - S_{t-s}) + (1-\alpha)(l_{t-1} + b_{t-1})$$
$$Trend\ b_t = \beta(l_t - l_{t-1}) + (1-\beta)b_{t-1}$$
$$Seasonality\ S_t = \gamma(\gamma_t - l_t) + (1-\gamma)S_{t-s}$$
$$Forecast\ \hat{y}_t + k = l_t + S_{t+k+s}$$

where s is the length of the seasonal cycle, for $0 \leq \alpha \leq 1$, $0 \leq \beta \leq 1$ and $0 \leq \gamma \leq 1$.

The intercept now depends on the current value of the series minus any corresponding seasonal component. Trend remains unchanged, and the seasonal component depends on the current value of the series minus the intercept and on the previous value of the component. Also note that the component is smoothed through all the available seasons; for example, if we have a Monday component, then it will only be averaged with other Mondays. Now that we have the seasonal component, we can predict not just one or two steps ahead but an arbitrary k future step ahead [8].

Method 7: ARIMA

It stands for Autoregressive Integrated Moving average. While exponential smoothing models were based on a description of trend and seasonality in the data, ARIMA models aim to describe the correlations in the data with each other. An improvement over ARIMA is Seasonal ARIMA. It considers the seasonality of dataset just like Holt Winter's method [8].

Forecast Quality Metrics: Imply how to measure the quality of our predictions

Our predictions have been tested using the following metric:

Root mean squared error (RMSE): It is the square root of the average of squared differences between prediction and actual observation.

$$RMSE = \sqrt{\frac{1}{n}\sum_{j=1}^{n}(y_j - \hat{y}_j)^2}$$

Since the errors are squared before they are averaged, the RMSE gives a relatively high weight to large errors. This means the RMSE should be more useful when large errors are particularly undesirable.

Display of Plots based on Methods stated above - Data has been aggregated on a daily basis to explain the different methods. The plots can be visualized in Figure 27, Figure 28, Figure 29, Figure 30, Figure 31 and Figure 32.

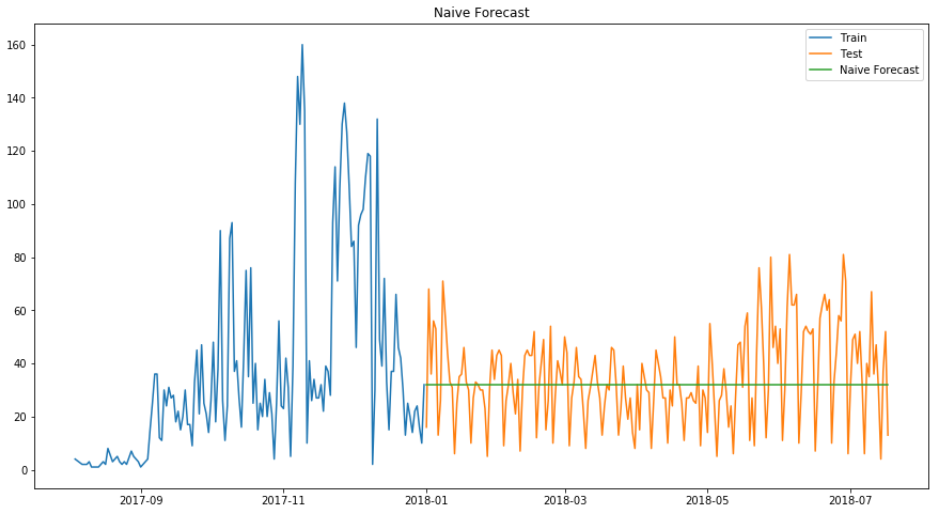


Figure 27: Naive Approach- Creation Date Vs Count of Tickets

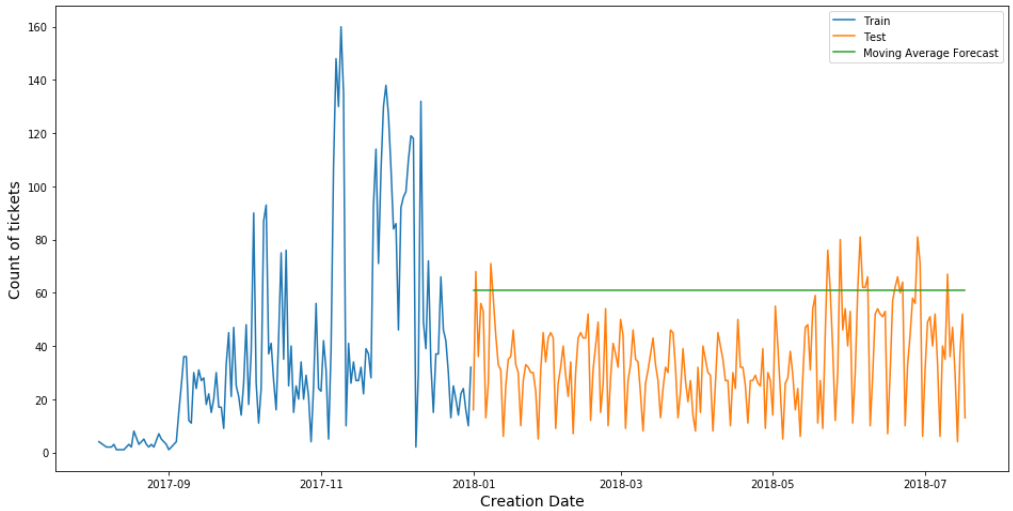


Figure 28: Moving Average - Creation Date Vs Count of Tickets

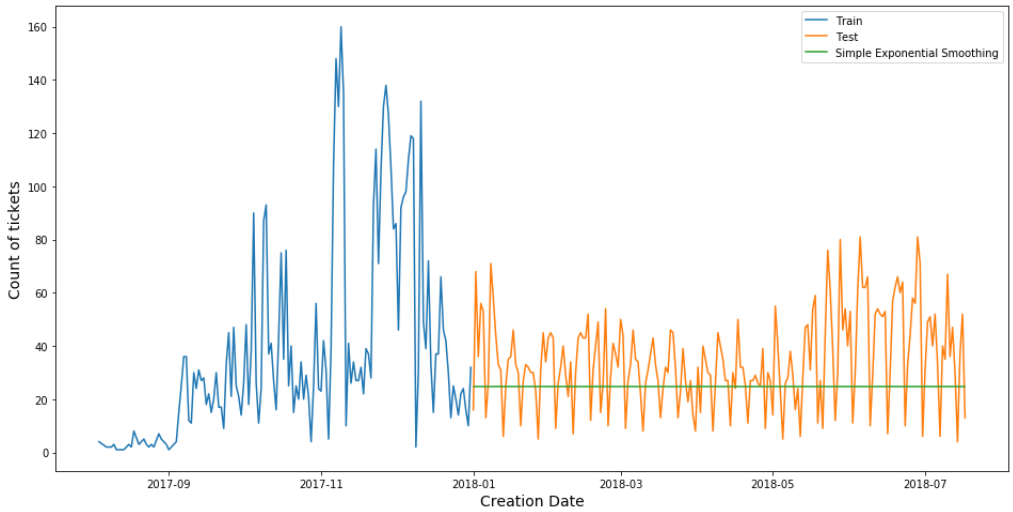


Figure 29: Simple Exponential Smoothing - Creation Date Vs Count of Tickets

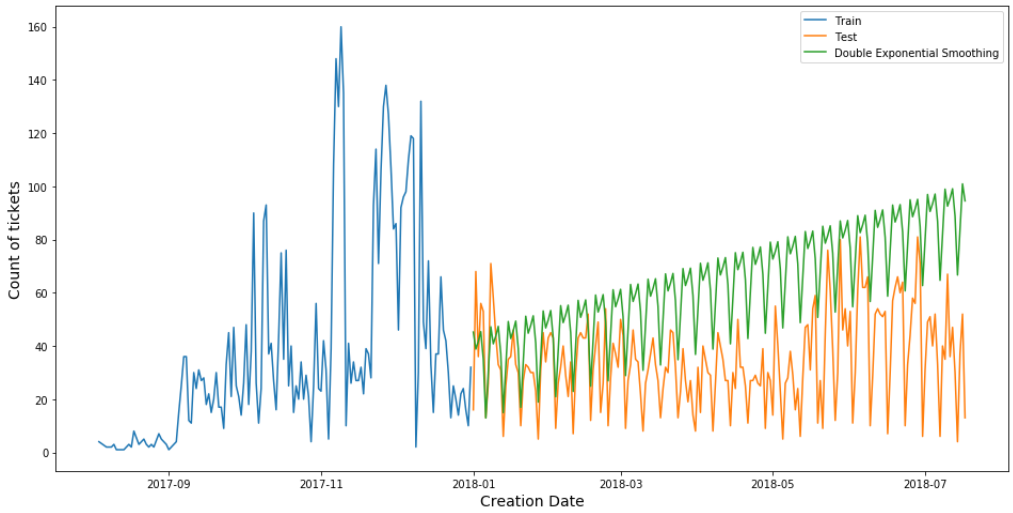


Figure 30: Double Exponential Smoothing - Creation Date Vs Count of Tickets

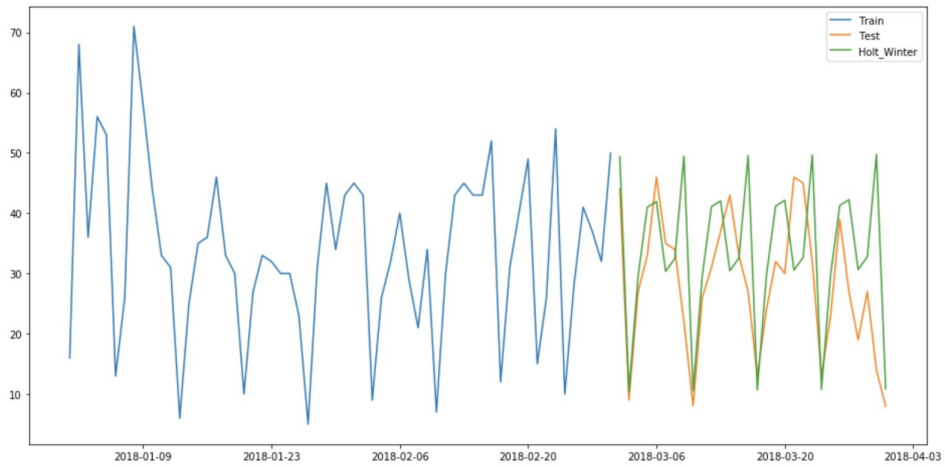


Figure 31: Holt Winter - Creation Date Vs Count of Tickets

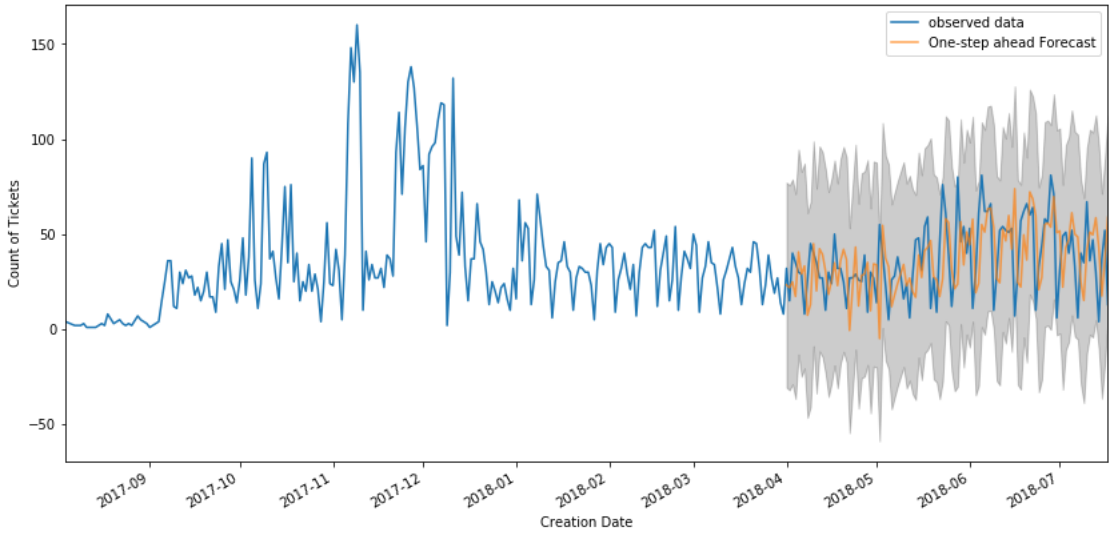


Figure 32: ARIMA - Creation Date Vs Count of Tickets

5.3 Model Training

5.3.1 Different KDE Methods

We have discussed three different methods to select bandwidth in the section of throughput time. Even though we did recognize that there is little difference among them regarding simulating throughput time, we should measure final Digital Twin models. This is because Digital Twin is built on combination of mathematical models including Marcovchain or Time-series modeling. Thus correlation might occur and affect the final model while training. We have trained 10 Digital Twin models per each KDE methods(constant bandwidth, grid search, additional cross validation). Thus we have trained total 30 models. All of them are made by same train set. Time range of the train data is from '2018-01-01' to '2018-02-28'. By using Analysis of Variance (ANOVA), we test if their outputs are significantly different. Data for the test is introduced in table 12 and 13. Each element in the tables represents total average throughput time from 'Process start' to 'Process End'.

Method	model 1	model 2	model 3	model 4	model 5	model 6	model 7	model 8	model 9	model 10	model 11	model 12	model 13	model 14	model 15
bandwidth = 0.1	118	133	146	131	109	122	119	131	131	121	138	122	120	105	121
gridsearch	142	142	112	120	114	129	142	126	131	125	124	121	135	122	137
cross validation	109	116	116	125	126	116	117	152	110	122	116	119	129	118	101

Table 12: Throughput time from 'Process start to End'. Model 1 - 15

Method	model 16	model 17	model 18	model 19	model 20	model 21	model 22	model 23	model 24	model 25	model 26	model 27	model 28	model 29	model 30
bandwidth = 0.1	122	141	139	119	139	115	116	129	133	130	124	139	121	115	119
gridsearch	153	115	116	130	115	128	140	111	120	127	111	120	128	154	130
cross validation	133	125	139	99	125	112	118	125	144	134	109	107	131	141	150

Table 13: Throughput time from 'Process start to End'. Model 15 - 30

Before we actually implement ANOVA, two assumptions should be satisfied: Independence of observations, Normality, Equality (or "homogeneity") of variances[1] [9]

Independence of observations All Digital Twins are created independently. Whenever Digital Twin model is created, Markov chain, KDE method and Time-series model work independently.

Normality Data within the group should have normal distribution. In our data, type of method defines a group. Therefore, we applied KS-Test.

H_0: Data in a group has same distribution with $N(\bar{X}, S^2)$.

H_1: Data in a group does not have the $N(\bar{X}, S^2)$ distribution.

Test results of three group are suggested at Table 14. Regarding all groups, we cannot reject H_0 due to higher p-value than 0.05. As a result, we have decided normality assumption holds

Equality of variances It is important for each group have same distribution. This is due to the point that ANOVA compares between-group variation to within-group variation. For this purpose, we have used Levene test. Our null and alternative hypothesis

Method	$p-$value
bandwidth = 0.1	0.3267
gridsearch	0.7408
cross validation	0.05485

Table 14: KS-Test result for three methods of time simulation

are explained below:

$H_0 : \sigma_1^2 = \sigma_2^2 = \sigma_3^2$

$H_1 : \sigma_i^2 \neq \sigma_j^2$ for at least one group pair i, j

By lawstat package in R, we directly tested. The p-value shows that we cannot reject H_0. Thus, *Equality of variances* also holds.

Test Statistic	$p-$value
0.83067	0.4392

Table 15: Levene test result

ANOVA At last, we applied ANOVA. The result allows us to decide that three methods actually do not make difference considering total throughput time. Based on this analysis, we finish our comparison of Digital Twins from distinct throughput time simulation. Last but not least, we recommend use constant bandwidth(0.1) as it is most fast one.

F-value	$p-$value
0.839	0.362

Table 16: ANOVA by stats package in R

5.3.2 Time Series Model

Final Results In Table 17, we can see the comparison between different methods of Time Series applied to the data set containing creation dates of tickets and their counts. We can see the best result is generated using Holt Winter's Method, also known as Triple Exponential Smoothing, based on the RMSE value generated. The model was able to successfully approximate the initial time series, capturing the daily seasonality, overall downwards trend, and even some anomalies. The model deviations captured reacts quite sharply to changes in the structure of the series but then quickly returns the deviation to the normal values, essentially "forgetting" the past. This feature of the model allows us to quickly build anomaly detection systems, even for noisy series data, without spending too much time and money on preparing the data and training the model.

Model	RMSE Value
Naive Approach	34.51
Simple Average	19.18
Moving Average	20.02
Simple Exponential Smoothing	31.08
Double Exponential Smoothing	20.59
Holt Winter's / Triple Exponential Smoothing	12.15
ARIMA	22.56

Table 17: Comparison Table of various Methods with RMSE Values

5.3.3 Model Validation

Ensemble Model As we present in the exploratory analysis section, the distribution of throughput time significantly changed from 2017 to 2018, and the one generated in 2018 is closer to the distribution of all cases. Hence, we tend to use more information from recent data. To predict the activity chains, throughput time and the number of cases more accurately, we use training dataset which covers more historical data for case number prediction and uses more recent data for training the Markov process model including the KDE part.
We build three models whose training data timespans are different from each other. After validation, we observe that these models tend to perform better in different aspects. Therefore, we use an ensemble model in our work to simulate the Digital Twin. The prediction of the ensemble model is the average of the three models we trained.
The training data and parameters settings which are chosen based on their performance in the last section are presented in the table below. In the table, the values represent the timespan of the training data. For example, "2 months" means we use cases generated since two months ago until the starting date of prediction. "All" means all the data in since 2017-08-04 until the starting date of prediction.

Model	Markov Process	KDE (bandwidth=0.1)	Holtwinter (Period=7 days)
Model 1	2 months	2 months	All
Model 2	2 months	2 months	4 months
Model 3	1 month	1 month	4 months

As we introduce in the previous section, we have data which are generated from August 2017 to July 2018, we validate our model by comparing the accuracy of predictions for May and June. We also compared long–term and short-term predictions, namely predicting the following week and the following month. The results indicate that the one-month prediction achieves better accuracy, therefore, we would suggest using our Digital Twin model for predicting performance in the following month.
In Figure 33, we list all the validation results. The red ones are the best two results in each column and the bold results are the most accurate results in each column. As it is shown in the tables, before ensemble, different models tend to perform better in different columns. And the ensemble model out-perform the other ones in general.

	Cases per Day	Events per Day	Avg Total TP Time	Trimmed Avg Total TP Time	Sample Size	Standard Dev. TP Time
Celonis Data	16	105	116 Hours	81 Hours	1100	9.08
Model 1	11	75	134 Hours	94 Hours	864	10.26
Model 2	12	71	103 Hours	72 Hours	841	8.00
Model 3	13	83	95 Hours	69 Hours	820	6.99
Ensemble	12	76	111 Hours	78 Hours	842	

(a) Simulated Values Comparison (Prediction for May 2018)

	Cases per Day	Events per Day	Avg Total TP Time	Trimmed Avg Total TP Time	Sample Size
Rel. Error1	31.25%	28.57%	15.52%	16.05%	**21.45%**
Rel. Error2	25%	32.38%	11.20%	11.11%	23.55%
Rel. Error3	**18.75%**	**20.95%**	18.10%	14.81%	25.45%
Rel. Error Ensemble	25%	27.62%	**4.31%**	**3.7%**	23.45%

(b) Simulated Values Validation (Prediction for May 2018)

	Cases per Day	Events per Day	Avg Total TP Time	Trimmed Avg Total TP Time	Sample Size	Standard Dev. TP Time
Celonis Data	31	173	85 Hours	66 Hours	1432	5.88
Model 1	17	109	122 Hours	88 Hours	1240	9.29
Model 2	19	120	121 Hours	87 Hours	1645	9.19
Model 3	20	125	109 Hours	78 Hours	1645	8,39
Ensemble	19	118	117 Hours	84 Hours	1510	

(c) Simulated Values Comparison (Prediction for June 2018)

	Cases per Day	Events per Day	Avg Total TP Time	Trimmed Avg Total TP Time	Sample Size
Rel. Error1	45.16%	36.99%	43.53%	33.33%	13.40%
Rel. Error2	38.71%	30.64%	42.35%	31.82%	14.87%
Rel. Error3	**36.67%**	**27.75%**	**28.24%**	**18.18%**	14.87%
Rel. Error Ensemble	38.71%	31.79%	37.65%	27.27%	**5.45%**

(d) Simulated Values Validation (Prediction for June 2018)

Figure 33: Model Validation Results

6 Application of the Digital Twin Model

Digital Twin model depicts the business operating model. Many organizations are using it to coordinate the critical interdependencies between people, processes and IT within complex digital business transformations. It provides enterprises with the following benefits to achieve business excellence:
1. Make the companies more manageable and more adaptive to change
2. Design and implement digital business transformation roadmaps
3. Test changes before implementing them across the operations
By tuning the parameters and changing the input to the Digital Twin model, users can exploit the potential opportunities, for example improving their working efficiency while reducing the required resources. In the following sections, we present two what-if questions that we can answer given the current dataset.

6.1 What if the first level service is automated?

In the first application case, we demonstrate how our model can predict what will happen if the first level service in the ITSM process is automated. In the Celonis IT service process, the first-level service can be defined as the actions that are processed by the IT Service Desk, which is recorded in the category attribute in our dataset.
To simulate what Celonis can benefit from the automation of the first level service, we reduce the relevant throughput time to zero and run the model to simulate the changes that could happen in May 2018. In the table in Figure 34, we can observe a significant re-

	Cases per Day	Events per Day	Avg Total TP Time	Trimmed Avg Total TP Time	Sample Size	Standard Dev. TP Time
Celonis Data	16	105	116 Hours	81 Hours	1100	9.08
Model 1	15	95	39 Hours	19 Hours	864	5.23
Model 2	14	98	36 Hours	15 Hours	820	5.49
Model 3	17	107	35 Hours	19 Hours	820	5.05
Ensemble	15	100	37 Hours	18 Hours	835	
Percentage of Change	-6.25%	-4.76%	-68.10%	-78.75%%	-24.09%	

Figure 34: Simulation Result for First Level Automation

duction in throughput time after applying automation to first level service, which means that in May 2018, if we automate the first level service, the average total throughput time would have likely been reduced by around 68%. However, the case numbers are not influenced. This could be caused by our independent assumption that the throughput time and the number of cases or events are not dependent on each other.

Potential improvement The number of events or cases processed every day should be dependent on how the colleagues at the IT Service Desk allocate their working hours. Since the first level service is automatically finished, the colleagues could spend more time working on other activities. To build a more sophisticated model, we need to add these additional relevant features and reflect their correlations in the structure.

6.2 What if the number of cases increase or decrease?

Naive prediction With Digital Twin, we can simulate more or less cases. We expect to see what happen if the number of cases variate. Unfortunately, our model depends on independence assumption. This means that even if the number of ticket increases, there would be no change for activity frequency and throughput time distribution. Therefore, we introduce outlook of the given what if questions, and suggest potential improvement based on current achievement.

Naive approach Keeping in mind the limitation, we have simulated the what-if question. Currently we are able to observe how the cases per day, events per day and frequency between activities change. Their change is exactly proportional to the ratio of increase or decrease. Refer to Table 18. For more advanced analysis rather naive, we need to upgrade our model.

Change of ticket number	−50%	−20%	0%	+20%	+50%	+100%
Cases per day	3	4	4	5	6	9
Events per day	18	26	29	32	39	53
Frequency of 'Status: New' to 'Change assignee'	38	27	35	42	57	78

Table 18: Naive analysis: what if the number of ticket increase or decrease

Potential improvement As displayed in figure 35, the number of cases per day and average throughput time is correlated. We tried to apply their relation to for the time simulation, but did not implement. We have observed there is a possibility of regression model. We actually built the simple model, but decided it does not have enough accuracy for Digital Twin. We leave it as potential improvement in the future.

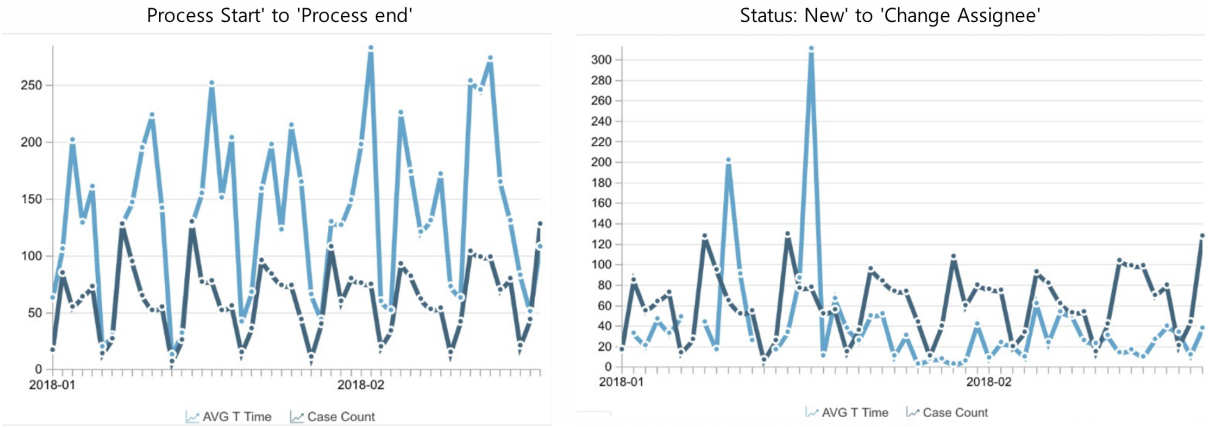


Figure 35: Data observation: dark line refers case count, and light one is average throughput time

7 Discussion and Conclusion

To summarize, we have discussed and created a Digital Twin model. First, we introduced concepts of process mining and Digital Twin. Next, we described the Happyfox ITSM Data Model and how it organized. After exploring the data, we give a more in-depth description of each techniques we have applied. It is built grounded on three simulations: the sequence of activities, throughput times and the number of tickets, under the independent assumptions. After training the model with our preprocessed dataset, we validate the simulation results which suggest that the Digital Twin model can be applied to a one-month prediction. By changing the input or tuning the parameter in the Digital Twin model, we answer several what-if questions which can help enterprises exploit potential opportunities for improvement. We hope that the virtual twin helps companies to simulate and predict challenges of business.

8 Outlook

Based on the Digital Twin model we have built so far, we suggest some outlook for future work.
The first point of outlook is about independence assumption. In this work, we assume that the activity sequence and throughput time simulation do not depend on the number of the ticket. To build a more sophisticated model, we introduced possible methods such as LAMP. We also suggested how throughput time simulation and the number of the tickets or events per day might influence each other.
Secondly, we can solve more what-if questions once there is more relevant data available. For example, we can then answer questions such as "What if I come up with an FAQ?" and "Would a chatbot help me?" What's more, the automation application can also provide a 24/7 IT service. These techniques can help with giving suggestions to frequently asked questions, so the customers get the response immediately. The potential improvement of the business operating model can then be observed using the output of the company's Digital Twin.

References

[1] David Ray Anderson, Dennis J Sweeney, and Thomas Arthur Williams. *Statistics for business and economics*. Arden Shakespeare, 1996.

[2] Stephen Boyd and Lieven Vandenberghe. *Convex optimization*. Cambridge university press, 2004.

[3] Narinder Kaur Gosall and Gurpal Singh Gosall. *The doctor's guide to critical appraisal*. PasTest Ltd, 2012.

[4] Aarshay Jain. A comprehensive beginner's guide to create a time series forecast. https://www.analyticsvidhya.com/blog/2016/02/time-series-forecasting-codes-python, 2016.

[5] Eric Jones, Travis Oliphant, Pearu Peterson, et al. SciPy: Open source scientific tools for Python, 2001–. [Online; accessed <today>].

[6] Ravi Kumar, Maithra Raghu, Tamás Sarlós, and Andrew Tomkins. Linear additive markov processes. In *Proceedings of the 26th International Conference on World Wide Web*, pages 411–419. International World Wide Web Conferences Steering Committee, 2017.

[7] Mary L McHugh. The chi-square test of independence. *Biochemia medica: Biochemia medica*, 23(2):143–149, 2013.

[8] Gurchetan Singh. 7 methods to perform time series forecasting. https://www.analyticsvidhya.com/blog/2018/02/time-series-forecasting-methods, 2018.

[9] George W Snedecor. Cochran, 1967. statistical methods. *Iowa State University Press, Ames, Iowa*.