KU LEUVEN

FACULTY OF ECONOMICS AND BUSINESS

FACULTY OF ECONOMICS AND BUSINESS

Master's Thesis

From Web Log to Process Data

Elisa Van der Perre

R0636635

Ilayda Caliskan

R0831711

Thesis submitted to obtain the degree of

MASTER OF BUSINESS AND INFORMATION SYSTEMS

Major: Data Science

Promoter: Prof. Dr. Johannes De Smedt

Tutor: Jari Peeperkorn

Academic year 2021-2022

Contents

Abstract

The main goal of this thesis is to explore how customer journey maps can be uncovered from a web log collected via Google Analytics. We explore trace clustering as a means to reduce the complexity that is present in web logs. More specifically, multiple trace clustering techniques were applied to a web log resulting in cluster-level process models that reveal useful information about the customers journey.

This is relevant as customer journey information is essential when making strategic marketing decisions. While the current state-of-the art mainly relies on web analytics tools that abstract users to an overly-simplistic level, the repeatable framework we propose allows much more fine-grained forms of analyses based on customer segmentation.

Keywords: Process mining, Trace Clustering, Representation Learning

Acknowledgements

This thesis was made under the competent guidance of Prof. dr. Johannes De Smedt and our supervisor, Jari Peeperkorn. We are thankful to them for generously sharing their insights, during many iterations, and for their patience while prodding us on.

While this thesis marks one of the highlights of (this phase of) our studies, it also marks the end of it. It seems a good moment to express our gratitude to our parents, family and friends who have supported us all along this exciting journey.

1 Introduction

Electronic commerce (e-commerce) is an emerging market that stands for the transaction of goods and services through electronic communications. While the general public has only become acquainted with e-commerce in the 2000s, it has been in existence since the 1970s. (58) Especially in the last decade, as the newest phenomenon of the global economy, the advantages of e-commerce and the specifics of its functioning have gained importance. (51) Particularly after the Covid outbreak, fundamental changes in consumers' as well as retailers' behaviors are observed. (27) Prior to the Covid lockdowns, only a small minority of retailers had digital capabilities mature enough to provide customers an omnichannel experience. (29) But during the pandemic, both consumers and retailers have started to fully realize the potential and importance of e-commerce. (27) The increasing importance of e-commerce companies every year and their rise to become global and quasi-omnipotent players in the international arena confirms this. (22) In order to be successful in this ever-evolving field, retailers need to exploit data, data analytics and machine learning tools when making crucial strategic decisions. (67) (53)

According to the literature, the key to profitable e-commerce retailers strategies is using the large volumes of clickstream data collected on their websites in an effective way. (67) The log of consumers navigating on a website is called clickstream data. (50)(45) Despite the fact that such data contains significant information about customers' online activity, (41)(26)(40)(8)(9) it is rarely used to get insights that might guide marketing decisions. (45)(13)(39) This is due to the fact that clickstream data is a type of big data, and therefore, it is characterized by high volumes that are difficult to analyze. (12) (54) Nevertheless, clickstream data can provide important insights to help with the marketing operations if the right data analytics approaches are used. (45)(12)(67)

Consumer segmentation is one of the key applications of analyzing clickstream data in marketing research. (12) (64) Consumer segmentation refers to the partition of consumers into buyer groups with unique qualities and habits that may need separate goods or marketing mixes. (28) (42) (67) In this thesis, we propose a repeatable framework to segment visitors according to the similarities and differences of their navigation behavior on a website.

In addition to consumer segmentation, we present a reusable framework to analyze the customer journey of each consumer segment we have discovered from the clickstream data. Websites are an ideal setting for investigating the notion of customer journey mapping since every action taken by users are documented in web logs, a tabular format that can be utilized for a variety of analyses. (57) Currently, many businesses rely on web analytics tools to extract important insights from data; however, such programs present an overly simplistic perspective of the customer journey, leading to an erroneous understanding of user behavior on the website. (10) Specifically, they narrowly identify the preceding and subsequent pages in the path for each page the consumers visited, offering an abstracted picture of the relationship between pages, exit points, and significant pathways followed by customers. (10) While these flows are helpful to get a general understanding of the user journey on the website by displaying the key pathways that visitors take, these tools have important limitations. (57) Each website visitor follows its own unique path, and if visitors are not segmented, it is not possible to find a one-size-fits-all customer journey path without any information loss. They are incapable of providing end-to-end process maps that can display and clarify the choices and loops between activities. They also do not allow the investigation of processes from various perspectives (e.g.: time constraints, bottlenecks, or relations between resources). (57)

The restrictions mentioned above can be efficiently addressed using process mining approaches. (60) In this thesis, we explore customer journeys by using process mining on the weblogs. Process mining works with event logs, which is a sequential format perfect for describing customer journeys. Therefore, process mining appears to be a good fit for this challenge. (6) Another advantage is that, dealing with predicted and actual models is fundamental to the process mining framework, by capturing the causality and routes of user interactions that lead to desired results, such as purchasing a product. (44) Finally, by extracting a process that describes user behavior, it is possible to find useful insights and to compare different behavioral clusters' processes using process mining. (57) With the features of process mining, website owners can choose which paths of the process apply to which customer segment through recommendations; declutter and organize the website with customer segments and their customer journeys in mind. (57)

The main contribution of this thesis is two fold: by creating a re-usable framework that consists of machine learning and process mining approaches, it is possible to *(i)* segment and *(ii)* analyze website visitors. The methodology we propose is to use each visitor's visited pages when searching for similar traces. Since encoding this feature results in high dimensional data, we reduce the dimensionality by using representational learning techniques. Then we use clustering techniques to explore visitor segments. The traces of these segments are then used as inputs to process mining algorithms that build consumer journey paths that can be analyzed.

All in all, e-commerce retailers are on the rise, and to survive in this market, they need to utilize machine learning approaches to clarify consumers as valuable and potentially valuable based on insights from records of how consumers browse. (53) By applying the proposed methodologies, e-commerce retailers, can increase their online visit/purchase rate, (11) minimize losses and realize revenue growth. (2) (67)

2 Preliminaries

This section introduces the essential theoretical components and literature background needed to understand the methodology.

2.1 Customer Journey

The customer journey is a graphical and process-oriented tool to analyze people's experiences. The customer journey covers the entire experience of being a customer instead of merely concentrating on one particular stage in the customer's process. It starts when the customer needs a product or service and continues until it is purchased. (33) The organization's goal is to design an optimal journey to maximize customer and organization values. Businesses can compare the experience of customers across all the channels with the experience that is relative to the company's expectations. (33) Customers have started demanding a more personalized experience. Therefore customer journey analysis can be conducted to understand this demand and improve the customer's experience. (30)

2.1.1 Customer Journey Maps (CJM)

The Customer Journey Map (CJM) depicts the key stages a customer encounters when interacting with a business or service in a linear, time-based manner. (33) CJMs have a structure that reflects the customers' cognitive, emotional, and behavioral impulses, according to Wolny and Charoensuksai. (65) Understanding how to improve the customer experience is the critical objective of tracking the customer journey. (30)

The Customer Journey Map's key elements can be summarized as follows (6):

- Customer
- Journey: shows at least one route of the customer.
- Goal: the organization's aim while mapping the customer journey.
- Touchpoint: the point in time at which customers interact with the organization.
- Timeline: period of time between the first and the last touch points.
- Channel: the method utilized by the customer to engage with the touchpoint.

With these elements, all potential organizational touchpoints that customers might experience during the service exchange process are listed by CJM. Senior management can collaborate with cross-functional team members to use strategies that encourage service innovation by clearly grasping client touchpoints. The strategies seek to improve interactions between customer service providers and consumers, by enhancing the customer experience linked to each touchpoint. (47)

As described in the previous paragraphs, customer journey maps are notably useful for organizations. However, manually creating customer journeys takes a lot of time and domain expertise. For this reason, process mining techniques are used to discover customer journeys from datasets (logs). These techniques automate, accelerate and augment the entire process and allow greater focus on the analysis of the journey. (57)

2.1.2 Web Usage Mining Techniques in Customer Journeys

Poggi et al. define web usage mining as collecting, measuring, and analyzing the data from a web server. (44) The goal of web usage mining is to gather meaningful user access information so that websites may be improved to meet user needs, provide better user service, and gain greater economic rewards. Log records from the web server contain valuable information (URL, IP address, time, and others) to apply mining techniques. Organizations are able to acquire new consumers and boost the popularity of their sites by analyzing and discovering the log. The outcomes of online data mining can be used to restructure websites for quick and straightforward consumer access, to enhance links and navigation, to increase ad revenue through clever advertisements, increase website conversion rates through improved site layout, and to track the effectiveness of websites. (52)

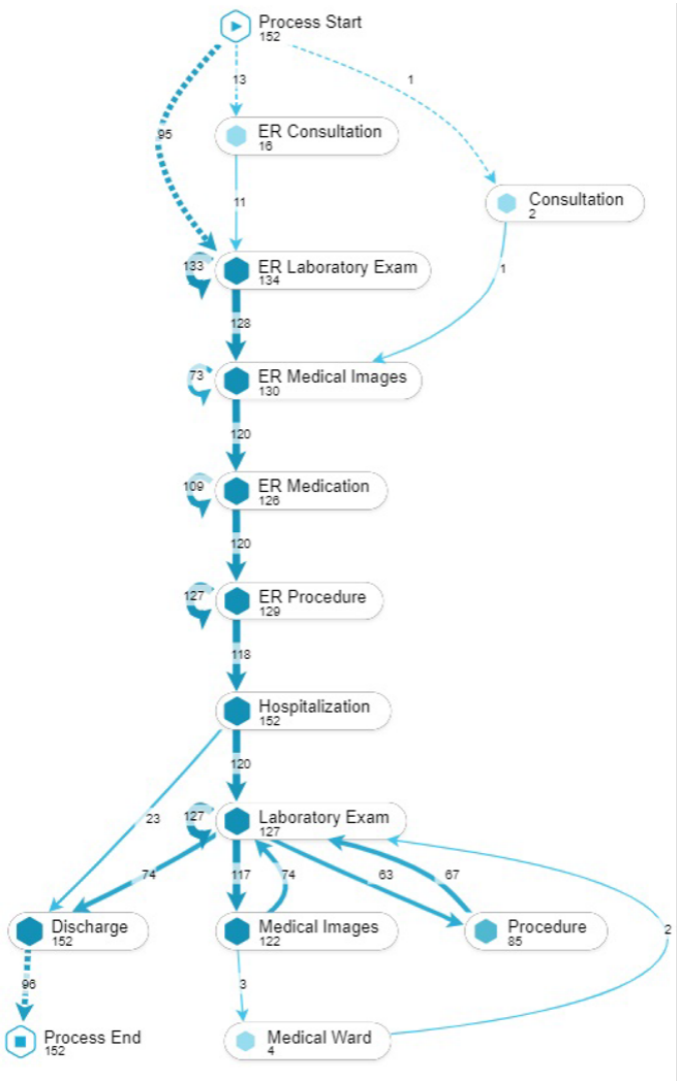


Figure 1: An example of a customer journey map discovered by the Celonis platform. (4)

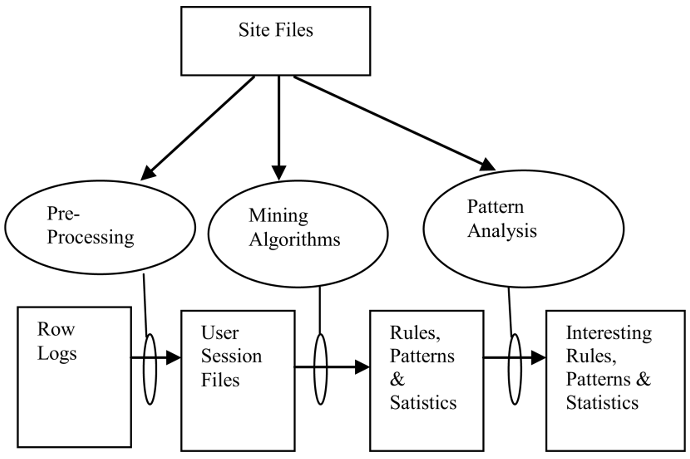


Figure 2: High-level web usage mining (52)

Data mining methods, including classification, clustering, and association rules are applied effectively in the field of web usage mining. (52) These methods summarize data sequences into characterizations of sessions, which are a collection of high-level data from the user's navigational activities. (24) In order to present the user's navigational activities in a higher-level abstraction, customer behavior model graphs (CBMG) can be utilized. (36) CBMG uses a k-means clustering algorithm to express the probability matrix of the possible path transitions from a state. (23) For this case a webpage is considered as a state.

To create customer behavior model graphs, commercial tools like Google Analytics are usually not enough. The behavior flow feature in Google Analytics helps to analyze weblogs. However, it does not extract the user's behavior at an abstraction level that is appropriate to understand the actual critical path that consumers take. These tools focus more on the frequency of page visits rather than on the process itself. (57)

2.2 Process Mining

As described in (59), process mining can be seen as the link between a process model and the actual event data generated by the modeled process. As can be seen in figure 3, process mining starts from an event log and aims to discover, monitor, and improve the associated process.

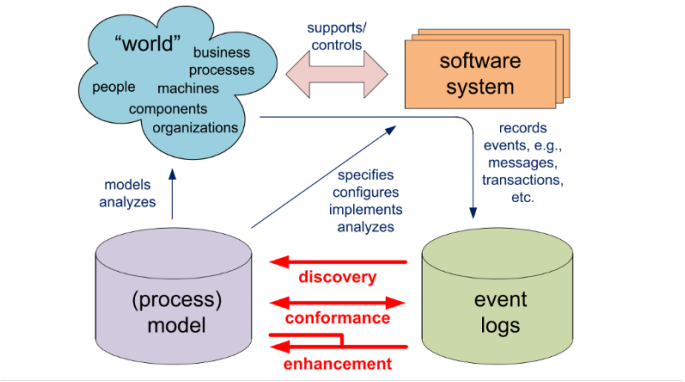


Figure 3: Positioning of the three main types of process mining: discovery, conformance and enhancement (59)

2.2.1 Event Logs

Process mining techniques use a specific data format: an event log. It can be seen as a particular view of the event data available for a process and can be defined as follows:

"Definition (Event log): Given an event log $L = e_1, e_2, e_3, \ldots$, *each event* $e_j = (c_j, a_w, t)$ *must refer to an activity* a_w *from the activity set* $a_w = a_1, a_2, \ldots, a_{|A|}$, *to a timestamp* t, *and a case* c_j, *which uniquely identifies the user who performs the action. Thus, an event log can be seen as a collection of cases, and a case can be seen as a trace of events."* (57)

This structure can also be seen in figure 4, distinguishing the following components:

- A **process** consists of cases, each case belonging to precisely one process.
- A **case** consists of events, with each event relating to precisely one case. Events within a case should be ordered.
- An **event** can have associated attributes.

An example of an event log belonging to a process of handling compensation requests is illustrated in figure 5. We can distinguish four process instances, also called cases. Each case consisting of an ordered collection of events. Additional properties like timestamp, activity resource and cost are stored per event.

2.2.2 Trace Clustering

When the underlined process is less structured and more flexible, traditional process discovery algorithms tend to generate spaghetti-like models. An approach to overcome this is by creating clusters that correspond to a coherent set of process instances in which each be represented by a process model. (57) Figure 6 this concept. Furthermore,

- The bag of activities clustering methods are appropriate when the focus of the customer journey analysis is to discover how the behavior of users changes in relation to different web page contents. (1)

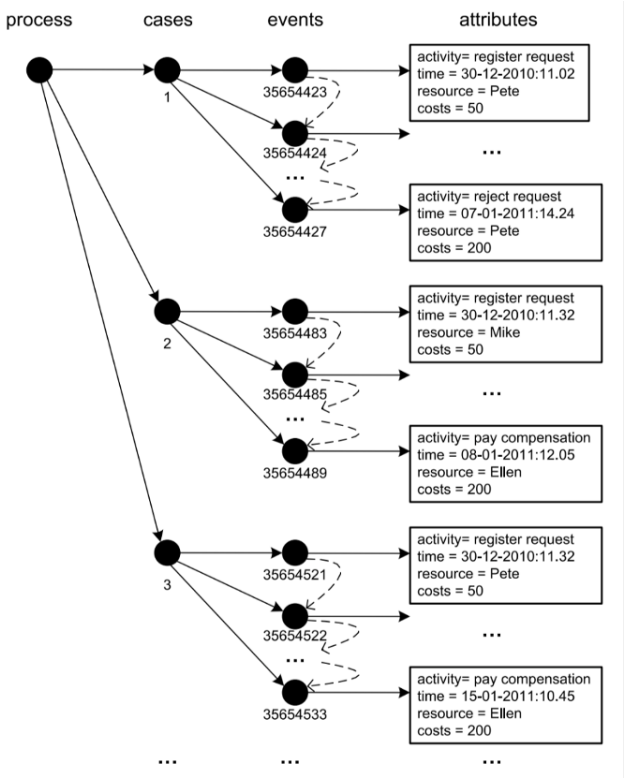


Figure 4: Structure of an event log (59)

- Trace clustering methods based on sequence distances are appropriate when the target of the customer journey analysis is to discover the paths followed by clusters of different users. (1)
- Some manual filters can be used to answer particular questions, related to the domain in which the customer journey lives. (1)

2.2.3 The Bag of Activities Clustering Methods

Traces are encoded in a vectors with a dimension equal to the total number of activities. Each value in the vector indicates how many times the activity appears in the trace. This matrix can then be used as input for various algorithms like k-means or hierarchical clustering. (1)

"Bag of activities" clustering methods do not consider the order in which the activities were executed. "Sequence-based distances" methods can be used to overcome this shortcoming.

2.2.4 Sequence-based Distances Methods

Sequence distances are used to cluster traces, also focusing on the order of the web pages visited rather than solely on which web pages were visited. (1) In this context, distance is defined as, the minimum of insertions, deletions and substitutions needed to convert one trace to the other. Different weights can be assigned to each operation. These weights can be manually assigned or can be found by optimizing some evaluation metric. (1)

2.2.5 Manual Filters

Personalized sub-logs can be created based on some specific attribute(s). It is, for example, possible to compare the process models of users visiting a website via their smartphone and users navigating the website via a personal computer. Or, it could be interesting to compare visitors behavior based on the country from which the user is visiting the website. (1)

2.2.6 Process Mining Applied to a Web Log

There are several important key differences between the characteristics of web logs and event logs, which should be considered when applying process mining techniques on web logs.

Case id	Event id	Properties				
		Timestamp	Activity	Resource	Cost	...
1	35654423	30-12-2010:11.02	register request	Pete	50	...
	35654424	31-12-2010:10.06	examine thoroughly	Sue	400	...
	35654425	05-01-2011:15.12	check ticket	Mike	100	...
	35654426	06-01-2011:11.18	decide	Sara	200	...
	35654427	07-01-2011:14.24	reject request	Pete	200	...
2	35654483	30-12-2010:11.32	register request	Mike	50	...
	35654485	30-12-2010:12.12	check ticket	Mike	100	...
	35654487	30-12-2010:14.16	examine casually	Pete	400	...
	35654488	05-01-2011:11.22	decide	Sara	200	...
	35654489	08-01-2011:12.05	pay compensation	Ellen	200	...
3	35654521	30-12-2010:14.32	register request	Pete	50	...
	35654522	30-12-2010:15.06	examine casually	Mike	400	...
	35654524	30-12-2010:16.34	check ticket	Ellen	100	...
	35654525	06-01-2011:09.18	decide	Sara	200	...
	35654526	06-01-2011:12.18	reinitiate request	Sara	200	...
	35654527	06-01-2011:13.06	examine thoroughly	Sean	400	...
	35654530	08-01-2011:11.43	check ticket	Pete	100	...
	35654531	09-01-2011:09.55	decide	Sara	200	...
	35654533	15-01-2011:10.45	pay compensation	Ellen	200	...
4	35654641	06-01-2011:15.02	register request	Pete	50	...
	35654643	07-01-2011:12.06	check ticket	Mike	100	...
	35654644	08-01-2011:14.43	examine thoroughly	Sean	400	...
	35654645	09-01-2011:12.02	decide	Sara	200	...
	35654647	12-01-2011:15.44	reject request	Ellen	200	...
...	...	...	...	...	...	...

Figure 5: An example showing a fragment of an event log (59)

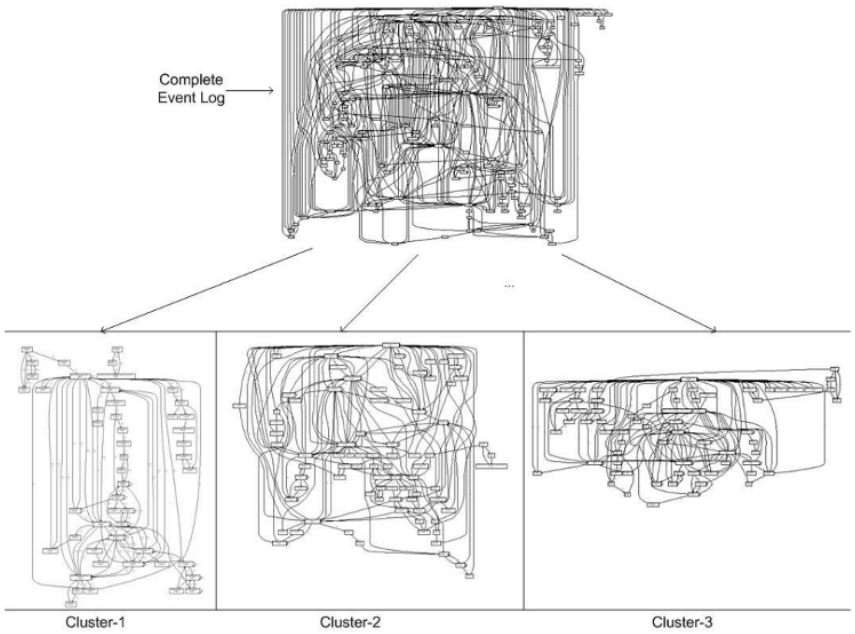


Figure 6: Example of trace clustering (7)

1. **Noise**: Web logs are typically noisier than event logs. Noise in web logs can for example be

caused by a dropped connection for users roaming over a mobile network, by an expired web session, by web proxies and caches causing requests not to be recorded, etc. (44)

2. **Number of activities**: The number of web pages present in a web log is typically higher than the number of distinct activities present in an event log. (44) (1)

3. **Event types**: Standard process mining techniques usually treat all events equally. For a web log, it is possible to distinguish between different event types. For example, a distinction can be made between input events, such as string input or copy/paste, and navigational events, such as scrolling or clicking a URL. (1)

4. **Flexible control flow**: The interaction of a user with the website is a poorly structured process, as it is possible for the user to perform activities in an arbitrary order and revisit previously executed activities. Usually, there are no clear control-flow relations between the activities, which leads to a high number of unique trace variants. (1)

5. **Timing**: The average web navigation takes only several minutes, while a business process can span days. (44)

6. **Log size**: As web logs capture data at a high resolution, the size of a web log typically greatly exceeds the size of a conventional event log. (1)

These differences in characteristics clearly state the vast contrast in complexity between the two logs. To effectively analyze web logs using process mining techniques, these techniques should therefore be able to handle complex, high-volume, and high-cardinality data. Traditional process mining techniques tend to lack these capabilities as they were designed to analyze more conventional end-to-end processes.

3 Research Question

This thesis intends to further explore the use of data-driven methods to discover customer journey maps starting from a web log. In particular, two specific research questions are addressed:

- How to aggregate web logs and convert them into event logs enabling to approach web logs from a process mining perspective?
- How can process mining techniques be exploited in the context of customer journey analysis?

4 Literature Review

This thesis focuses on converting web logs into event logs enabling the approach of web logs from a process mining perspective. The consulted sources can be roughly divided into four categories:

- Processing weblogs
- Processing event logs
- Converting weblogs to event logs
- Event abstraction.

A key recent paper (67) discusses how e-commerce weblogs are processed. The authors aim to segment visitors who surf the e-commerce site by using a dataset, which includes

- the number of past visits, the total number of pages viewed, and revenue measures as numerical data
- the time per page and visit duration measures as continuous data
- the used device, traffic source, cart opened, and purchase made measures as categorical data.

For this purpose, Partitioning Around Medoids (PAM) algorithm is used to construct clusters with Gower distance as the similarity measure. The dataset and the necessity of clustering are similar to our dataset and research objective.

Makanju, Zincir-Heywood, and Milios (2009) present a different method to cluster event logs using iterative partitioning. (32) IPLoM (Iterative Partitioning Log Mining) starts by partitioning the data by token count (tokens are defined as data point attributes, in this case, the individual words on each line). It assumes that most cluster instances have the same token length; thus, the resulting segments of this heuristic are likely to contain instances of different sets with the same token count. The output of the first step is log data, separated into clusters with the same number of tokens. In the second step, the token position with the least number of unique values is found and further split into the clusters containing the most similar tokens. In the final partitioning step, two token positions with the highest occurrence are selected. If bijection is observed between the two sets of tokens, log messages with these token values at their corresponding token positions are divided into a new partition. After the third step, clusters have been formed, and discovering cluster descriptions (line formats) are left for the final step. In this stage, it is assumed that the log messages in each cluster are produced using the same line format. If the tokens have only one value, the line format is considered to be that constant value. If more than one value is observed in the clusters, the line format is represented using wildcard values. (32)

R. Pérez-Castillo, Weber, Pinggera et al. (2011) propose a technique to acquire event logs from traditional information systems, which have no logging feature. (43) It creates event log files "according to the MXML (Mining XML) format (20), which is used by the process mining tool ProM (61) " (43) when it is executed in the dynamic analysis step. The acquired event logs are mined with the ProM tool to discover business processes. With the conducted case study, the research shows that the proposed technique is qualified to obtain event logs to discover business processes. (43)

B. Fazzinga et al. (2018) define a framework to describe the process behavior in terms of activities and events; to discover a control-flow model from a sequence of low-level events. (16) The paper

aims to support analysts, finding reasoning more opportune at the abstraction level of activities rather than low-level events. The proposed framework is based on modeling the event log formation with an appropriate Hidden Markov Model (HMM). HMM retrieves the statistics of precedence relationships between the hidden activities resulting in the event logs. The retrieved statistics are put in the Heuristics Miner algorithm to model the process in terms of activities. The process model is enriched by probabilistic information on the mapping between activities and events discovered with HMM. (16) The first step of the framework is learning a stochastic model, which is able to reproduce a new generation of the input logs of events. This stochastic model is recognized as a Hidden Markov Model, and while the observations in the model stand for the low-level events, underlying hidden states encode the activities. In the second stage, statistics are extracted from the model resulting in the first step. Next, a process model is produced by inputting the statistics into the Heuristics Miner algorithm. At last, the resulting model is enhanced by adding the probability distribution of the low-level events corresponding to the activities. In the end, a two-level process model originating from a low-level event log is produced. (16)

In "Business Process Mining from E-Commerce Web Logs", the authors try business process mining techniques on e-commerce weblogs. (44) Their analysis starts by classifying URLs into logical tasks. They classify pages by their semantics and additionally present Simple K Means and Expectation Maximization algorithms for automating page classification. If the number of clusters is unknown, automating the page classification results in a large percentage of errors. As a next step, the dataset is saturated by removing sessions that did not purchase. The authors then apply three process mining algorithms (Knowledge-based Miner, Heuristic Miner, and Fuzzy Miner), and compare and contrast the differences based on the results on their dataset. The paper resembles our thesis; however, classification of URLs could not be performed on our dataset due to the anonymization of the website. Events captured by an information system do not directly correspond to occurrences of meaningful activities. Abstraction methods are required to transform low-level events into high-level activities.

In "From Low-Level Events to Activities - A Pattern-Based Approach", a supervised abstraction method is introduced based on behavioral activity patterns. (34) This enables the method to deal with noise, reoccurring and concurrent behavior, and with shared functionality. First, activity patterns are encoded using domain knowledge on how high-level activities are translated into low-level events. The set of all activity patterns is composed of an integrated abstraction model, which should reflect the behavior for all high-level activities. The abstraction model is aligned with the low-level events from the event log using existing alignment techniques, which establish a mapping between log traces and process traces. (34) This alignment is used to build the abstracted event log in which the events correspond to the instantiations of recognizable activities. Two quality measures are defined for evaluating the abstraction mapping: fitness and matching error. After applying well-known out-of-the box process mining tools on both the original event log and the abstracted event log, it can be concluded that the results obtained by using the abstracted event log reveal insights that cannot be obtained by using the original event log.

Since the experimental data of this thesis, only has numbers as page paths, event abstraction is the most important point in our thesis. To get ideas about converting low-level activities into high-level activities, event abstraction techniques have been researched in the literature. In their article, "Event-log abstraction using batch session identification and clustering", De Leoni, Massimiliano, and Safa Dündar introduced an abstraction method that does not need domain knowledge. (15) They start with an event log, and they divide the log into batch sessions in a way that the time interval between the last event of one session and the first event of the following session is greater than a user-defined threshold. These sessions are then clustered by using DBSCAN and K-Means clustering techniques. For each cluster, cluster centroids are shown using heat maps, and these maps are then used while creating the abstracted event log.

The literature survey written in 2021 "Event abstraction in process mining: literature review and taxonomy" by van Zelst et al. categorize event abstraction techniques as supervised and unsupervised. (63) The focus of our thesis is on unsupervised techniques. A first relevant article about this matter in the survey is by Bose and van der Aalst (2009). (7) The authors of the paper utilized the concept of coherent behavioral sub-sequences. With this technique, the authors discover maximal repeats and abstract those to higher-level activities, without discovering clusters. The second relevant article mentioned in the survey is by Günther et al. (2009). It introduces "trace segmentation" to create coarse-granular events. (20) This technique assigns each event or trace

segment to a cluster until all events become part of the root set of the hierarchical cluster set. Another technique by Folino et al. (2015), uses predictive clustering trees to cluster low-level events. (18) Frequently, local process models have also been used to discover process models that describe only a portion of the observed fine-grained trace and then use it as a basis for abstraction. (35) Also, word-embedding techniques have been utilized by grouping semantically similar names to map fine-granular events into coarse-granular events. (49) For our case study, using similar names would not help because the URLs are anonymized. Alharbi et al. (2018) utilize hidden Markov models to learn high-level activities. (3) In a final relevant example from this literature survey, Rehse and Fettke (2019) group co-occuring events by computing the "spatial proximity" between fine-granular activities until a hierarchical clustering of events is computed. (46)

In Abb et al. (2022), the authors cluster traces for user behavior mining. (1) They first encode traces as feature vectors and use representation learning techniques to reduce the dimensionality of the feature matrix. The feature matrix is embedded using Isomap (56) , Autoencoder (25) , and Word2Vec (38) (37). Then the three standard clustering algorithms (k-means, agglomerative hierarchical clustering, and DBSCAN) are applied in combination with each embedding technique. The paper collates the embedding techniques and clustering algorithms. According to the research, Isomap with DBSCAN and Autoencoder with DBSCAN resulted in the highest silhouette scores.

We will test whether their results and findings can be generalized to a web context by using PCA embedding and K Means clustering techniques on the anonymized case study dataset. Our thesis will also utilize embedding techniques on the feature matrix, and after clustering, we will apply process mining techniques to the largest clusters.

The publication by Terragni, A. (2018) "Analyzing Customer Journey with Process Mining: from Discovery to Recommendations" was a great inspiration. (57) To discover customer journey from a weblog, the weblog needs to be mapped into an event log. As in Poggi et al. (44) and Terragni, timestamps from weblogs can be used as is in the event log, with the user identifier as the case id, and page paths as actions. Terragni defines actions by handcrafting rules for the URLs, additionally he explains how to use unsupervised clustering as a way to map URLs into high-level activities. The next step in the paper is to filter the data from bots, a small number of events, unnecessary events etc. Once the event log is at hand and filtered, process discovery algorithms can be applied to discover the process model. The study continues with trace clustering techniques to discover behavioral patterns in the customer journey. The first trace clustering technique encodes traces into vectors and uses the matrix as an input to the clustering algorithms with different similarity measures. The second trace clustering uses sequence-based distances, such as edit distances. The third trace clustering approach is shown as filtering the logs depending on the interest. (57) The study by Terragni continues with decision point mining, bottleneck analysis, and recommender systems, but these are beyond the scope of our work.

5 Dataset

The dataset used in our experimentation is an anonymized web log from an unknown e-commerce website. The log contains information about the visitors' interaction with the website during the first and the third of August 2017. It is standard Google Analytics data consisting of 589.084 records, originating from 30.475 different visitors in 31.734 different sessions. The 22 attributes present in the dataset are all typical Google Analytics fields. Figure 7 displays an excerpt from the dataset. Due to limited computational resources, it was opted to use this rather (small) sample.

```
   visitNumber     visitId  visitStartTime      date  visits  hits  pageviews  \
0            2  1501552553      1501552553  20170801     1.0     2        1.0
1            2  1501552553      1501552553  20170801     1.0     2        1.0
2           13  1501569067      1501569067  20170801     1.0     2        2.0
3           13  1501569067      1501569067  20170801     1.0     2        2.0
4           84  1501605097      1501605097  20170801     1.0     2        2.0

   timeOnSite  bounces  browser  ... isMobile screenResolution  fullVisitorId  \
0       692.0      1.0   Chrome  ...    False         1440x900   8.106460e+18
1       692.0      1.0   Chrome  ...    False         1440x900   8.106460e+18
2        26.0      NaN    Opera  ...    False        2560x1440   7.015010e+18
3        26.0      NaN    Opera  ...    False        2560x1440   7.015010e+18
4        36.0      NaN  Firefox  ...    False         1440x900   6.623020e+18

  channelGrouping  hitNumber    time  hour  minute  pagePath  screenDepth
0          Direct          1       0     2      55         0            0
1          Direct          2  691661     3       7         0            0
2         Organic          1       0     7      31         1            0
3         Organic          2   25866     7      31         2            0
4           Email          1       0    17      31         3            0

[5 rows x 22 columns]
```

Figure 7: Extract from our dataset

5.1 Features

In order to better comprehend the analyzed dataset, we briefly describe the main features and their characteristics. Figure 8 lists all 22 features present in the dataset. It are all standard Google Analytics fields defined in (55) and can be grouped into the following categories:

- Time: date, hour, minute
- User/Session/Hit: visitNumber, visitId, visitStartTime, hits, pageviews, timeOnSite, bounces, fullVisitorId, visits, hitNumber
- Browser/Network: browser, browserVersion, BrowserSize
- Device: isMobile, screenResolution
- Traffic Source: channelGrouping
- Content: pagePath
- App Tracking: screenDepth.

It is important to consider the 3 levels present in the User/Session/Hit category. More precisely:

- User-level: fullVisitorId
- Session-level: visitNumber, visitId, visitStartTime, hits, pageviews, timeOnSite, bounces
- Hit-level: visits, hitNumber.

User-level attributes are equal for all records belonging to the same user. In the same way, session-level attributes are equal for all records belonging to the same session.

```
visitNumber
visitId
visitStartTime
date
visits
hits
pageviews
timeOnSite
bounces
browser
browserVersion
browserSize
isMobile
screenResolution
fullVisitorId
channelGrouping
hitNumber
time
hour
minute
pagePath
screenDepth
```

Figure 8: Feature names

5.1.1 fullVisitorId

The fullVisitorId field shows a different identifier for each unique visitor. In total, there are 30.475 unique visitors present in the dataset. It is interesting to look to the distribution of both the number of records (hit-level) and the number of sessions (session-level) per visitor.

- Hit-level: The number of records per visitor in the dataset ranges from a minimum of 2 to a maximum of 639 records per visitor. 75% of the visitors have less than 21 records present in the dataset.
- Session-level: The number of sessions per visitor ranges from a minimum of 1 to a maximum of 4 visits per visitor. For 29.276 visitors out of 30.475 (96%), only 1 visit was recorded.

Note: Session-level and User-level almost match for this particular dataset.

5.1.2 date

As already mentioned earlier, the dataset consists of data drawn from two days. The distribution of the data over the two dates can be described on hit, session and user-level.
On hit-level, the distribution of the records over the two dates is as follows:

- 01/08/2017: 251.442 records
- 03/08/2017: 337.642 records.

In terms of sessions, the following distribution is observed:

- 01/08/2017: 18.590 sessions
- 03/08/2017: 13.144 sessions.

And for visitors:

- 01/08/2017: 18.058 visitors
- 03/08/2017: 12.417 visitors.

5.1.3 pagePath

Each pagePath value refers to a web page on the website. For this dataset, the web pages are anonymized to numbers for privacy reasons. The most visited web page corresponds to page path number 7 which most likely refers to the homepage. Figure 9 displays the 20 most visited pagePath values and the corresponding number of visits to the web page.

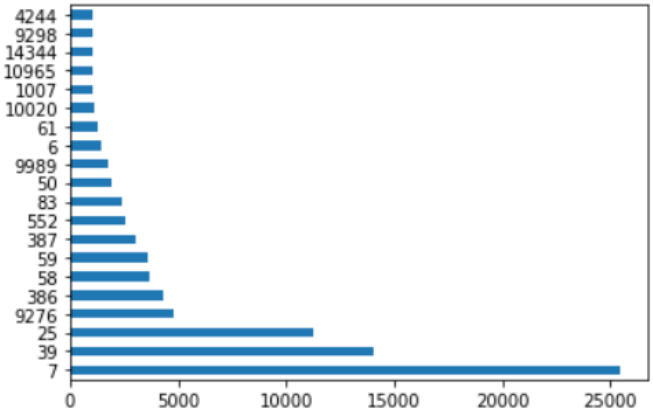


Figure 9: Distribution of the pagePath feature

5.1.4 hour

There are two peaks in the number of records recorded in the dataset, one at 13:00 and one at 21:00. Also, as expected, there is less traffic during nighttime hours.

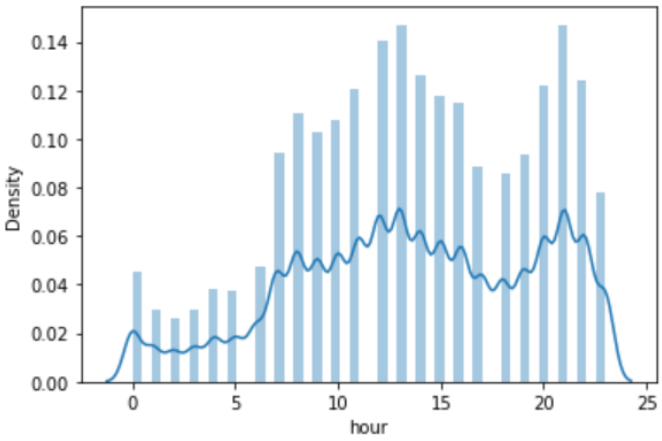


Figure 10: Distribution of the hour feature

5.1.5 Other features

Other features which were not used in our experiments but could be interesting for future work are:

- **visitNumber** also called sessionCount in (55), is an integer representing the session index for a user. The average of 38 means the average visitor present in the dataset visited the website 38 times. 75% of the visitors visited the website less than 24 times. It should be noted that this count does not start from 1 for our dataset and we do not have further information about the time span of this count. Most sessions, 9.635 out of 31.734 (30,36%), are first-time visits.
- **visitId** is an identifier for a session but only unique for a user. For a completely unique session identifier the combination of the visitId and fullVisitorId should be used. The new feature 'sessionId' was added to the dataset. As already mentioned, there are 31.734 unique sessions present in the dataset. The number of records per session ranges from a minimum of 2 to a maximum of 500 records per session. 75% of the sessions consist of less than 20 records.
- **visitStartTime** is the timeStamp of a session expressed in POSIX time.
- **hits**: Hits in Google Analytics is a metric that counts all interactions of the visitor with the website.(55) These interactions include clicking on the website, submitting a form, playing

a video, or downloading from the website. Each record in the dataset corresponds to one hit. The hits feature indicates the number of hits made per session, or in other words, it indicates the number of records per session. As already mentioned under visitId, the number of records per session ranges from a minimum of 2 to a maximum of 500 records per session. 75% of the sessions consist of less than 20 records.

- **timeOnSite** measures the total time spent on the website per session in seconds. (55). The average of 360,22 implies that a visit lasts on average 360,22 seconds, or 6 minutes. The maximum total time of a session is 10.663 seconds, or approximately 3 hours. 75% of the sessions take less than 6 minutes and 11 seconds.

- **browser**: The distribution of browsers utilized by the visitors is shown in figure 11. Out of the 30.475 visitors present in the dataset, more than half accessed the website using the browsers Chrome and Safari.

Chrome	12422
Safari	9650
Internet Explorer	2149
Firefox	1861
Safari (in-app)	1406
Edge	907
Android Webview	885
Samsung Internet	773
Opera	147
Amazon Silk	94
Android Browser	91
YaBrowser	46
Maxthon	13
Opera Mini	8
UC Browser	6
BlackBerry	6
Coc Coc	4
Puffin	4
Mozilla Compatible Agent	1
(not set)	1
MRCHROME	1

Figure 11: Distribution of browser feature

- **isMobile**: Slightly more than half of all visitors accessed the website using their mobile devices.
  - True: 15.837 visitors
  - False: 14.638 visitors.

- **channelGrouping**: indicates the traffic source of the visitor.
  - Organic: 26,74% of the visitors navigated from unpaid search sources like Google.
  - Direct: 17,71% of the visitors navigated directly by typing in the URL.
  - Email: 16,44% is traffic from links clicked in emails.
  - ...

- **time**: The time in milliseconds after the previous hit. The first hit of the session has a time value equal to 0. The average time between hits is 508,23 seconds, or 8 minutes and 28 seconds. The maximum time between hits is approximately 3 hours. In 75% of the cases, time does not exceed 9 minutes 54 seconds.

6 Experimental Setting

The goal of the experimental setting is to investigate which steps could be taken to make it possible to approach a web log from a process mining perspective and how this could be exploited in the context of customer journey analysis.

Starting from a Google Analytics web log, irrelevant complexity was filtered out in the preprocessing stage. (Section 6.1) The requirements needed to perform process mining on an event log were investigated and the resulting principles were then translated to a web log context, technically enabling to perform process mining techniques on the web log. (Section 6.2) Due to the differences in characteristics between a web log and an event log, discussed in 2.2.6, performing process discovery techniques on the web log resulted in spaghetti-like process models (as expected). In section 6.3, we therefore propose to apply trace clustering to the web log to reduce complexity. In fact, multiple trace clustering approaches were experimented with and compared to each other. In section 6.4, process discovery techniques were then performed on the best trace clustering results resulting from section 6.3.

6.1 Preprocessing

As our goal is to get insights into the control flow present in the web log, it is helpful to filter out records which do not contribute to this goal and, such, add unnecessary complexity to the dataset. For example, it was beneficial to filter out pagePath values which were only visited very rarely, or sessions consisting of only 1 record.

6.2 Web Log to Event Log Mapping

As first mentioned in section 2.2.1, process mining techniques require a specific data format, that of an event log. To transform our dataset to an appropriate format, we investigated which additional conditions needed to be fulfilled. The minimal requirements needed to perform process mining on an event log are described in (59) as:

1. Any event should be relatable to a case and an activity, and,
2. events within a case are ordered.

These requirements are equivalent to requiring an event log to contain at least these three features with the following characteristics:

- a caseId, which should uniquely identify the process instance,
- an activity, referring to a distinct step in the process, and,
- a timestamp, enabling to order the events within a process instance.

In order to apply process mining techniques to our dataset, concepts like process, process instance, activity and timestamp should therefore first be "translated" to the context of a web log.

Process A web log contains sequential data describing the interactions between visitors and the website. It is thereby possible to say it describes the web navigation behavior of the visitors on a particular website, or even more precisely, the path visitors take on the specific website. It is essential to consider that multiple levels are present in this described process. The process namely consists of multiple sessions, and each session consists of multiple records.

Process instance, caseId In this context, a process instance can be seen as the interactions of a specific visitor with the website, or in other words, the path a particular visitor takes on the website. Therefore, as caseId, which should uniquely identify a process instance, a feature can be used that uniquely identifies a visitor, such as the fullVisitorId feature.

Activity A process instance can be seen as a trace of executed activities. A process instance in the context of a web log, the path a visitor takes on a website, can be seen as a sequence of visited web pages. Therefore, the pagePath feature, which describes the visited webpage, can be used as the translation for the activity feature. In our approach, we used a one-to-one translation of the pagePath attribute to the activity attribute, but depending on the use case, other more interesting translations can be made. In (44), the pagePath values (the URLs) were first abstracted in higher-level tasks before being used as translation for the activity feature.

Timestamp The web log features date, hour, and minute can be combined as the translation for the timestamp.

When using the fullVisitorId feature as caseId, the pagePath feature (or as in (44), a manipulation of the pagePath feature) as activity, and a combination of the date, hour and minute feature as timestamp, it is technically possible to perform process mining techniques on a web log.

6.3 Trace Clustering

Due to the differences in characteristics between a web log and an event log (as discussed in section 2.2.6), applying process discovery algorithms directly to an event log created from a web log as described in section 6.2 will most likely result in spaghetti-like process models. Figure 12 gives an example of a spaghetti-like process model, clearly illustrating that no insights can be gained from it.

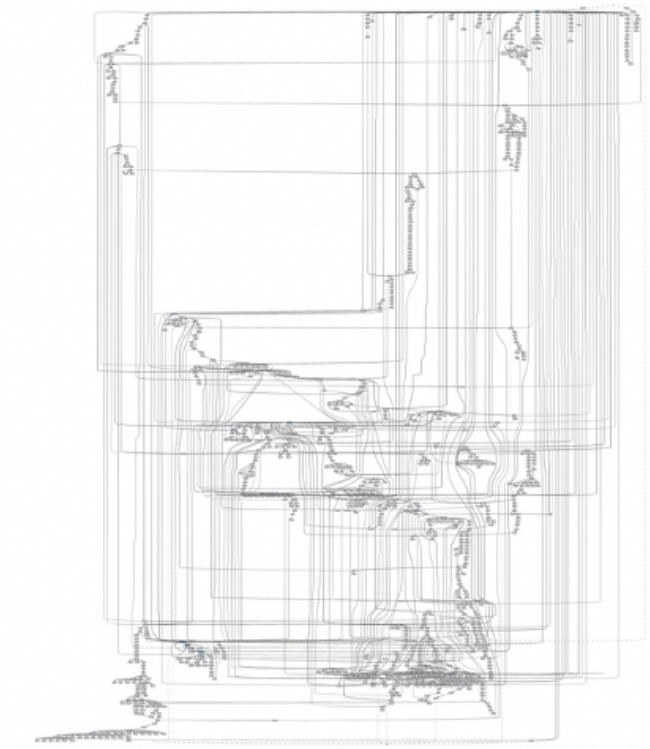


Figure 12: An Example of a Spaghetti Model

One possibility to reduce complexity is abstracting the activities to higher-level ones, as proposed in (44). However, this will reduce the level of detail at which the visitors' behavior can be analyzed. In other words, discovering a process model from an abstracted web log can give a clear view of which high-level activities were executed. However, it does not provide insights into how the visitor interacted with the website on web page level.

Another possibility to reduce complexity while retaining the level of detail present in the web log, is trace clustering. In this paper, we investigate whether trace clustering is suitable to handle the aforementioned characteristics of a web log.

Trace clustering divides the event log into smaller batches with more similar traces. The ideal outcome of applying process discovery to those more coherent sub-logs is a series of simpler, easier-to-understand cluster-level process models that together give a better perspective of the entire log. (66)

6.3.1 Trace Representation

Trace representation refers to which features are chosen to assess the similarity between traces and how the features are encoded. As we want to cluster traces with the same navigation behavior, using the control-flow feature pagePath in the representation makes sense. Other context features such as visitNumber, timeOnsite, hits, browser, isMobile, channelGrouping, and time could also be interesting to explore.

In our approach, only the feature pagePath was used in the representation. As the pagePath feature is a categorical feature, the bag-of-feature representation was used as encoding method, resulting in feature vectors with dimensions equal to the total number of pagePath values present in the dataset. Each value in the vector indicates how many times the pagePath appears in the trace. The dataset contains many pagePath values, resulting in a very high-dimensional and sparse feature space. Therefore, embedding techniques could be used to reduce the dimensionality, making it more likely for clustering algorithms to find a good partitioning.

6.3.2 Representation Learning

Representation learning techniques are used to embed high-level dimensional data in a lower dimensional vector space while preserving as much information as possible. We experimented with the following three embedding techniques: PCA and t-SNE.

PCA PCA (Principal Component Analysis) (31) is a widely used linear feature reduction technique. It projects data onto a lower dimensional subspace while retaining most of the variance among the data points.

t-SNE To test a non-linear representation learning technique on our trace matrix, t-SNE was chosen. t-distributed Stochastic Neighbor Embedding (t-SNE) (62) is a non-linear dimensionality reduction technique based on probability distributions with random walks on neighborhood graphs to find the structure within the data, especially well suited for the visualization of high-dimensional datasets.

6.3.3 Clustering

In this section, we briefly explain the used clustering algorithms: K-Means and Density-based spatial clustering of applications with noise (DBSCAN). Both clustering techniques were used on the different representation learning results from section 6.3.2. In section 6.3.4, all representation-cluster combinations are evaluated and compared to each other.

K-means K-means clustering (23) divides the dataset into k clusters using an iterative optimization strategy. More specifically, it minimizes the sum of SSE over the k clusters by shifting the cluster centers. The sum of squared errors (SSE) of a cluster is the sum of squared distances between the points belonging to the cluster and the cluster's center.

When choosing k, the trade-off between the number of clusters and the usefulness of the corresponding cluster-level process models should be made. In particular, models mined from too few large clusters will still be difficult to understand. In contrast, too many clusters will result in simpler but less expressive models, which are also useless. The goal is to find the number of clusters for which the corresponding cluster-level process models are comprehensive but not simplified too much that they become trivial. (1)

DBSCAN Density-based spatial clustering of applications with noise (DBSCAN) is, in contrast to k means clustering, a non-parametric clustering algorithm. DBSCAN computes nearest neighbor graphs and creates arbitrary-shaped clusters. K-means clustering, on the other hand, typically generates spherical-shaped clusters.

6.3.4 Evaluation Clustering

In this section, all representation-cluster combination were evaluated using the Davies-Bouldin Index and Silhouette Score.

Davies-Bouldin Index The Davies-Bouldin Index (DBI) (14) is a clustering algorithm evaluation measure. It calculates the average similarity of each cluster to the cluster most similar to it. The lower the index, the lower the average similarity and the better the clustering algorithm performed.

Silhouette Score The silhouette coefficient (48) is also a clustering algorithm evaluation measure. It measures intra-cluster cohesion and inter-cluster separation. A silhouette score close to 1 indicates well-separated clusters, whereas a score close to -1 indicates poorly separated and overlapping clusters.

6.4 Process Mining

In this section, process discovery techniques, mainly the Disco process mining tool, were applied to the cluster outcomes. This resulted in a series of simpler, easier-to-understand cluster-level process models that together give a better perspective of the entire web log.

The Disco process mining tool (17) utilizes the Disco Miner algorithm; an improved version of the Fuzzy Miner. With its advanced features such as seamless simplification of processes and accentuating frequent events and paths, the Fuzzy Miner algorithm introduced the “map metaphor” to the process mining field. It is a customizable process discovery algorithm, that can handle unstructured processes. Also, Fuzzy Miner can discover multiple models at different levels of detail via its parameters. (19) (21)

7 Experimental Evaluation

7.1 Preprocessing

The following two filters were applied to the web log:

Filter 1: Remove rare pagePath values. The relative frequency of each pagePath value was calculated, and all records referring to a pagePath value with a lower relative frequency than 0.01% (1) were filtered out. This resulted in the removal of 45.256 rare pagePath values. The total number of pagePath values was thereby reduced from 46.520 to 1.264. At the same time, removing those records resulted in a decrease in the number of unique visitors from 30.475 to 14.087.

Filter 2: Remove all sessions that consist of only 1 record. After applying this second filter, the filtered web log still contains 107.653 records from 12.554 different visitors referring to 1.264 different pagePath values.

	Original Dataset	After filter 1	After filter 2
Total amount of records	589.084	109.259	107.653
Amount of pagePath values	46.520	1.264	1.264
Amount of fullVisitorId values	30.475	14.087	12.554
Amount of sessionId values	31.734	14.496	12.890

Table 1: Filtering Results

7.2 Web Log to Event Log mapping

The following mapping was used:

- the fullVisitorId attribute was used as caseId,
- the pagePath attribute was used as activity, and
- a combination of attributes date, hour and minute was used as timestamp.

7.3 Trace Clustering

As expected, applying process mining to the resulting event log resulted in the spaghetti-like process model displayed in Figure 13.

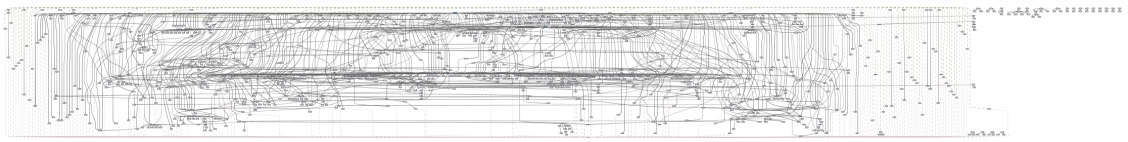


Figure 13: Spaghetti Model generated from the mapped event log

7.3.1 Trace Representation

Only the feature pagePath was used in the trace representation, resulting in 12.554 feature vectors with 1.264 dimensions. Each value in a vector indicates how many times the pagePath appears in the trace. The dataset contains 1.264 pagePath values, so the resulting feature space is very high-dimensional and sparse.

7.3.2 Representation Learning

We experimented with two representation learning techniques, PCA and t-SNE, to reduce the dimensionality of the feature matrix.

Principal Component Analysis (PCA) Using PCA, the dataset's dimensionality was reduced from 1.264 to 2, 10, 100, and 394. The principal component numbers (2, 10, 100, and 394) have been selected to experiment and compare the explained variances. Table 2 shows the explained variances for the different principal component numbers. PCA allows to specify the maximum amount of variance that will be given up while reducing the dimensionality of a dataset. After selecting a maximum loss of 1%, PCA could reduce the dimensionality to 394.

number of components	2	10	100	394
Explained variance	42,4%	66,9%	91%	99%

Table 2: PCA results

Figure 14 illustrates a scatterplot of the result of PCA with a principal component number equal to 2, or in other words, the result of mapping the feature matrix from 1.264 to 2 dimensions.

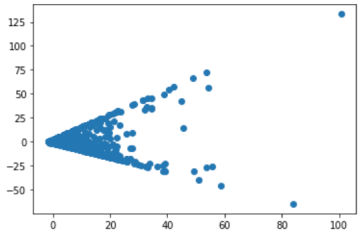
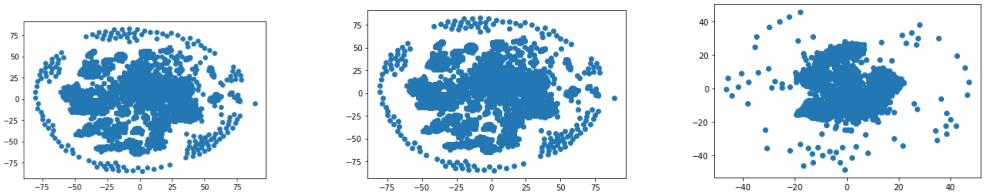


Figure 14: Scatterplot result PCA with 2 components.

t-distributed Stochastic Neighbor Embedding (t-SNE) There is no specific rule for determining the optimal values for the parameters like perplexity. Therefore, we repeatedly performed t-SNE with the following values for the perplexity parameter: 50, 100, and 500. Figure 15 illustrates the results.



(a) perplexity = 50 (b) perplexity = 100 (c) perplexity = 500

Figure 15: t-SNE with respectively perplexity = 50, 100 and 500

7.3.3 Clustering

7.3.3.1 K-means Clustering

The Silhouette score was used to find the optimal k for all the different representation learning results. As it does not make sense to have too many cluster-level process models for our use case, only scenarios up to k equal to 25 were investigated. Table 3 shows the optimal K and the corresponding Silhouette score for each PCA result. Table 4 shows the same, but now for the t-SNE results. The following observations can be made from the tables: the optimal K is overall lower for the PCA embedded datasets than for the by t-SNE embedded datasets. Also, the Silhouette scores for PCA are all higher than those for t-SNE, indicating overall better clusters were found for the PCA embedded results.

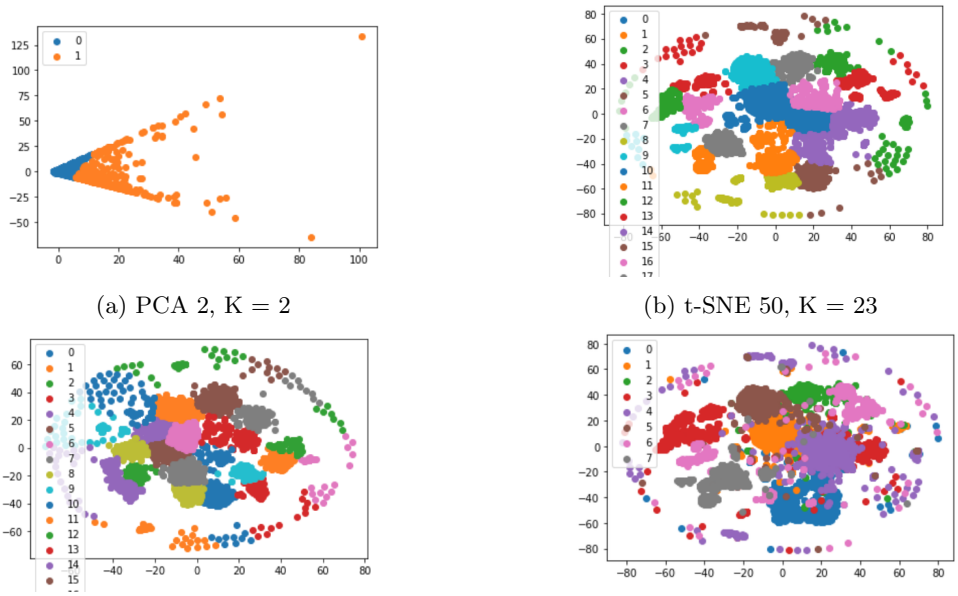
It is possible to visualize the clusters found for PCA 2, t-SNE 50, t-SNE 100, and t-SNE 500 as, for these cases, the trace matrix was embedded in the 2-dimensional space. Figure 16 illustrates the result.

	PCA 2	PCA 10	PCA 100	PCA 394
Optimal K	2	2	2	2
Silhouette Score	0,86	0,74	0,69	0,67

Table 3: Optimal K and Corresponding Silhouette Scores

	t-SNE 50	t-SNE 100	t-SNE 500
Optimal K	23	25	8
Silhouette Score	0,51	0,54	0,55

Table 4: Optimal K and corresponding Silhouette Scores



(a) PCA 2, K = 2

(b) t-SNE 50, K = 23

(c) t-SNE 100, K = 25

(d) t-SNE 500, K = 8

Figure 16: Scatterplots cluster results using K-means for PCA 2, t-, t-SNE 100, t-SNE 500

7.3.3.2 DBSCAN Clustering

DBSCAN requires two parameters, ϵ and $minPts$.

- $minPts$ should be 2$*$ number of dimensions of data.
- ϵ depends on distance function and should be as small as possible.

To determine the optimal ϵ parameter, the k-nearest neighbor method was used to calculate the average distance of every data point to its k-nearest neighbors in the input dataset (resulting from PCA and t-SNE). After creating the k-NN distance plot, the "knee" of the curve is found for the optimal value of ϵ. In Figure 17, the knee occurs at approximately 1. This can be translated as the points below 1 belong to a cluster, and points above 1 are noise or outliers (noise points will have higher kNN distance). (5) The k-NN distance is plotted for every input matrix resulting from PCA and t-SNE, to determine the optimal ϵ value before applying DBSCAN clustering.

Tables 5 and 6 illustrate for each dataset the number of clusters found and the corresponding Silhouette score. Again we observe that the Silhouette scores are overall higher for the PCA embedded datasets than for the t-SNE embedded datasets.

	PCA 2	PCA 10	PCA 100	PCA 394
Amount of clusters	14	3	2	1
Silhouette Score	0.834	0.697	0.238	0.341

Table 5: Number of found clusters and corresponding Silhouette Scores

	t-SNE 50	t-SNE 100	t-SNE 500
Amount of clusters	104	51	4
Silhouette Score	-0.098	-0.280	-0.339

Table 6: Number of found clusters and corresponding Silhouette Scores

Again, the found clusters for PCA 2, t-SNE 50, t-SNE 100, and t-SNE 500 were visualized as, for these cases, the trace matrix was embedded in the 2-dimensional space. Figure 18 illustrates the result.

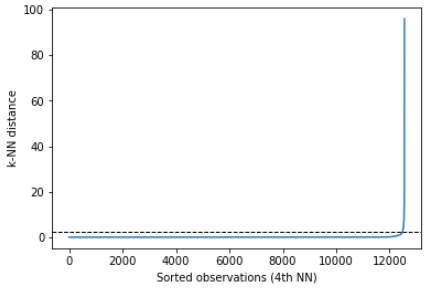
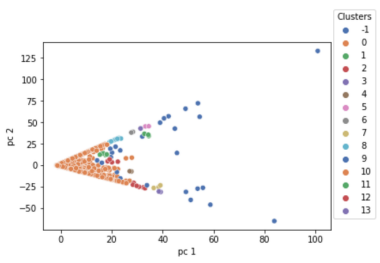
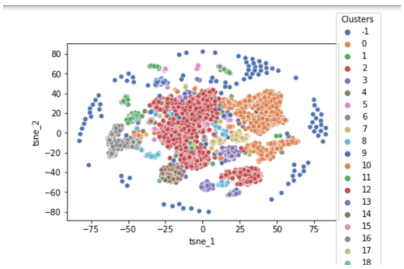


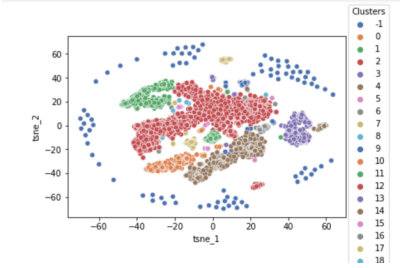
Figure 17: K-NN distance plot calculated on PCA with 2 components



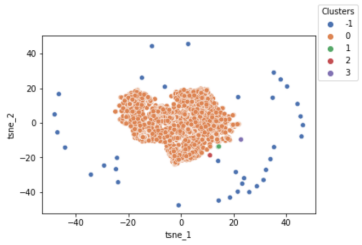
(a) PCA 2, K = 2



(b) t-SNE 50, K = 23



(c) t-SNE 100, K = 25



(d) t-SNE 500, K = 8

Figure 18: Scatterplots cluster results using DBSCAN for PCA 2, t-, t-SNE 100, t-SNE 500

7.3.4 Evaluation Clustering

In this section, all representation-clustering combinations were evaluated using the Davies-Bouldin Index. Per representation-clustering combination, these are the parameter values that led to the best Davies-Bouldin Index value.

- For the PCA-K-means combination: PCA with 2 PCs and K-means with K=2.
- For the t-SNE-K-means combination: t-SNE with perplexity=500, K-means with K=25.
- For the PCA-DBSAN combination: PCA with 10 PCs.
- For the t-SNE-DBSAN combination: t-SNE with perplexity=100.

Table 7 displays the Davies-Bouldin Index for all representation-cluster combinations. Also, all Silhouette scores, which were already calculated in the previous sections, are displayed in the table.

The best Silhouette score and the best Davies-Bouldin Index value were highlighted. In the next section, these two cluster results were used to discover cluster-level process models from.

Representation, Clustering	Silhouette Score	Davies-Bouldin Index
PCA 2, K=2	0,86	0,931
PCA 10, K=2	0,74	1.203
PCA 100, K=2	0,69	1,347
PCA 394, K=2	0.67	1.351
t-SNE 50, K=23	0,51	0.652
t-SNE 100, K=25	0,54	0.605
t-SNE 500, K=8	0,55	0.578
PCA 2, DBSCAN	0.83	1.571
PCA 10, DBSCAN	0.70	1.407
PCA 100, DBSCAN	0.24	3.051
PCA 394, DBSCAN	0.34	3.890
t-SNE 50, DBSCAN	-0.098	3.250
t-SNE 100, DBSCAN	-0.280	2.810
t-SNE 500, DBSCAN	-0.339	4.529

Table 7: Evaluation Clustering Results

7.4 Process Mining

Process discovery, particularly the Disco process mining tool, was applied to the following two cluster results:

- best Silhouette Score: the Discovered clusters uing the representation-clustering combination K-means with K=2 and PCA with 2 components.
- best Davies-Bouldin Index: the Discovered Clusters using K-means K=8 after t-SNE 500.

7.4.1 Discovered Clusters using K-means K = 2 after PCA with 2 Components

7.4.1.1 Cluster 1

- Cluster 1 with 17.5% activities

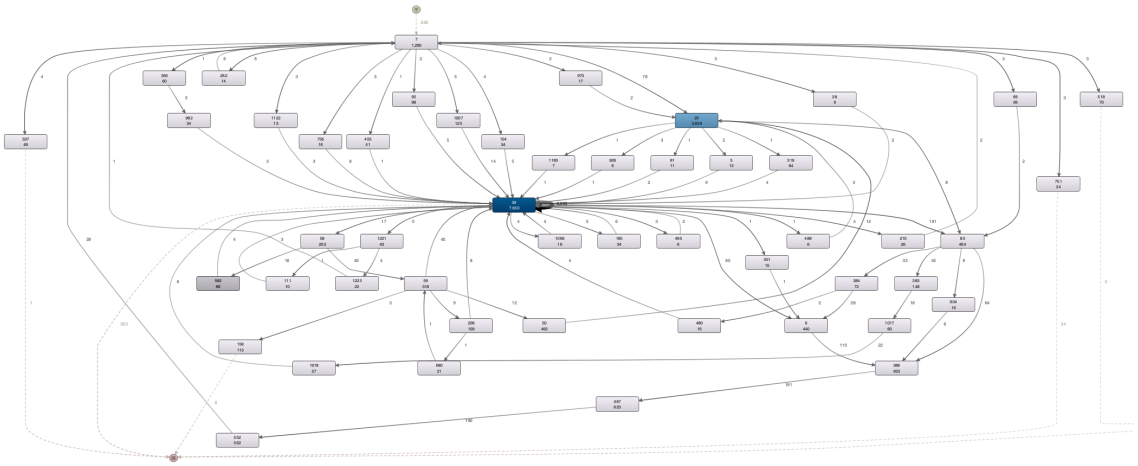


Figure 19: Process map mined from the largest cluster with 17.5% activities. This cluster has 88.327 events, 12.030 cases and 1.209 activities. The most popular page path for this cluster is the number 7. With 1079 cases, visiting page 7 twice is the most popular trace for this cluster.

- Cluster 1 with 84.6% activities

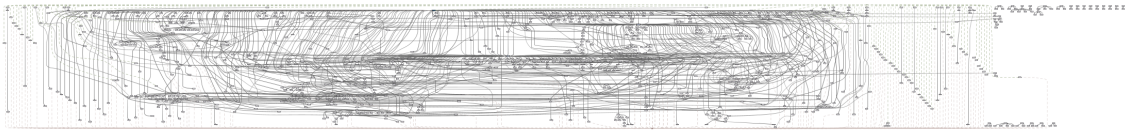


Figure 20: Process map mined from the largest cluster discovered with K-means, K=2 with 84.6% (maximum percentage) activities.

7.4.1.2 Cluster 2

- Cluster 2 with 2.8% activities

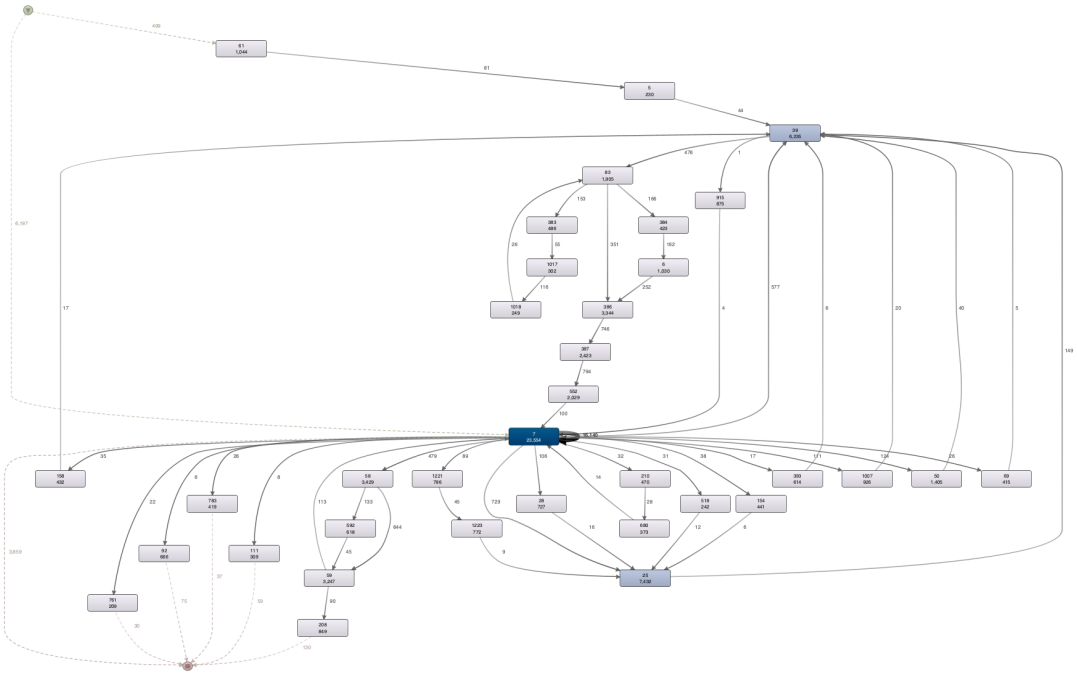


Figure 21: Process map mined from the second cluster discovered with K-means, K=2 with 2.8% activities. This cluster has 19.406 events, 531 cases and 229 activities. The most popular page path for this cluster is the number 39. With 12 cases, visiting page 39, 12 or 14 times is the most popular trace for this cluster.

- Cluster 2 with 100% activities

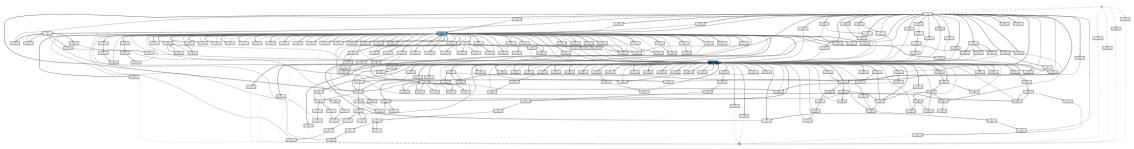


Figure 22: Process map mined from the second cluster discovered with K-means, K=2 with 100% (maximum percentage) activities.

7.4.2 Discovered Clusters using K-means K=8 after t-SNE 500

- Largest Cluster with 13.9% activities

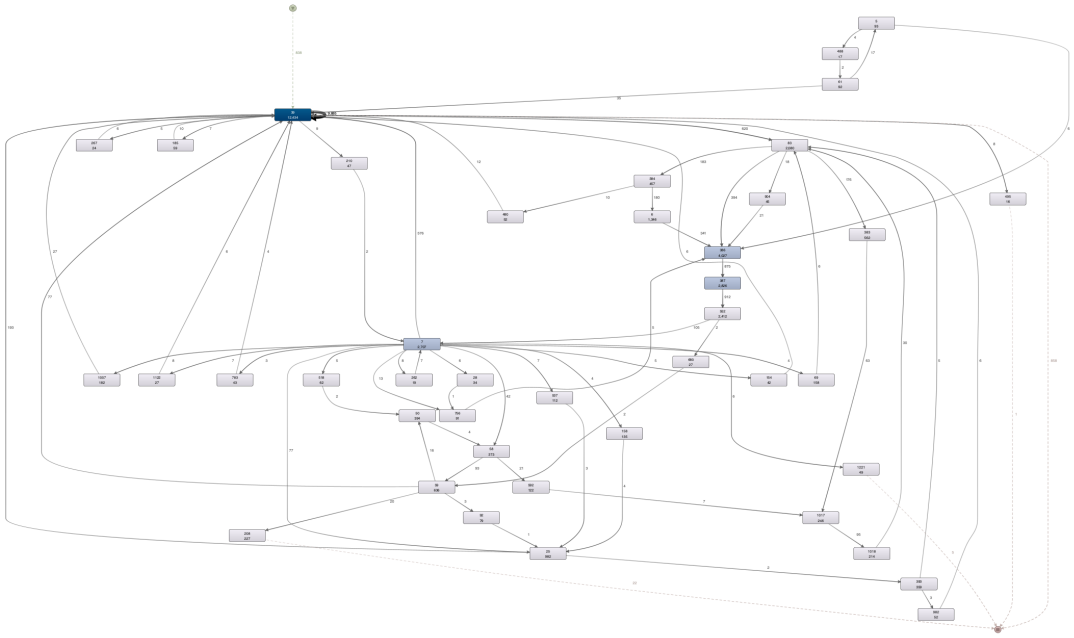


Figure 23: Process map mined from the largest cluster discovered with K-means, K=8 with 13.9% activities. This cluster has 36.519 events, 2.165 cases and 288 activities. The most popular page path for this cluster is the number 39.

- Largest Cluster with 100% activities

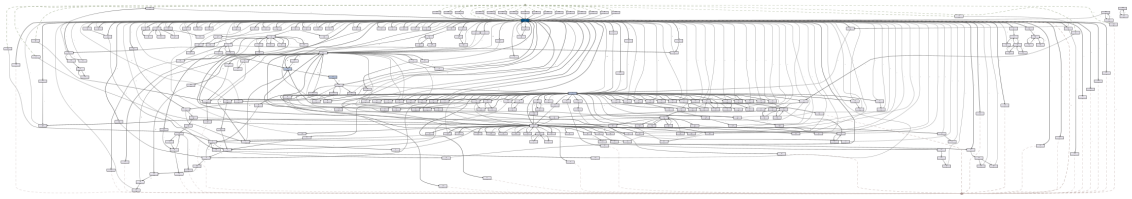


Figure 24: Process map mined from the largest cluster discovered with K-means, K=8 with 100% activities.

8 Conclusions

This paper applied process discovery techniques, particularly the Disco process mining tool, to analyze web visitor behavior expressed in weblogs. To do so, we compared the web navigation process described in a web log to a standard business process described in an event log. The following key differences uncover the vast contrast in complexity between the two processes:

- First, the number of possible activities which can be executed during the web navigation process is much higher than for a typical business process.
- Second, the control flow in which the activities can be executed is much looser.
- Finally, extra complexity in the analysis is added by the huge size of a web log and the extra noise present in the web log.

In order to apply process mining techniques to a web log, this complexity should somehow be reduced. In this paper, we have therefore explored the applicability of trace clustering to reduce this complexity. After applying trace clustering techniques to the web log, it is not clear if meaningful cluster-level process models were discovered from the web log. However, after the time we spent investigating and applying different techniques, we believe this is not necessarily related to the framework used.

We are confident that the steps we followed are common sense first steps, and we still believe trace clustering and the framework we used are promising. More precisely, the framework suggests reducing complexity in three stages:

- The preprocessing stage should focus on eliminating the noise in a web log.
- In the web log to event log mapping stage, the number of possible activities should be reduced.
- In the trace clustering stage the complexity due to the flexible control flow should be handled by splitting up the dataset into subsets with more similar control flows.

As the dataset on which the framework was applied was anonymized, we had difficulties reducing the complexity in the "web log to event log mapping" stage. Without any information about the web pages or the website structure, it is, for example, impossible to map the pagePath values (URLs) to higher-level URLs, as presented in (44).

As it is, we still believe our approach is promising and could be the basis of a repeatable framework.

9 Future Work

As mentioned in the conclusions, our main suggestion for future work is to try and apply the framework to a richer dataset for which some information is available about the URLs' content or the site's basic structure. This enables to reduce the complexity of the web log both by reducing the number of possible activities, and by reducing the flexible control flow through trace clustering.

Repeating the experimental setting and comparing it to our results should shed light on the generalizability of our findings and lead to a more comprehensive understanding of whether trace clustering can be beneficial in this context.

10 References

References

[1] Luka Abb, Carsten Bormann, Han van der Aa, and Jana R Rehse. Trace clustering for user behavior mining. 2022.

[2] Shahriar Akter and Samuel Fosso Wamba. Big data analytics in e-commerce: a systematic review and agenda for future research. *Electronic Markets*, 26(2):173–194, 2016.

[3] Amirah Alharbi, Andy Bulpitt, and Owen A Johnson. Towards unsupervised detection of process models in healthcare. In *MIE*, pages 381–385, 2018.

[4] Michael Arias, Eric Rojas, Santiago Aguirre, Felipe Cornejo, Jorge Munoz-Gama, Marcos Sepúlveda, and Daniel Capurro. Mapping the patient's journey in healthcare through process mining. *International Journal of Environmental Research and Public Health*, 17(18):6586, 2020.

[5] Renesh Bedre. Dbscan in python (with example dataset). Available at `https://www.reneshbedre.com/blog/dbscan-python.html` (2022/08).

[6] Gaël Bernard and Periklis Andritsos. A process mining based model for customer journey mapping. In *Forum and doctoral consortium papers presented at the 29th International Conference on Advanced Information Systems Engineering (CAiSE 2017)*, volume 1848, pages 49–56. CEUR Workshop Proceedings, 2017.

[7] RP Bose and Wil MP van der Aalst. Trace clustering based on conserved patterns: Towards achieving better process models. In *International Conference on Business Process Management*, pages 170–181. Springer, 2009.

[8] Wouter Buckinx and Dirk Van den Poel. Customer base analysis: partial defection of behaviourally loyal clients in a non-contractual fmcg retail setting. *European journal of operational research*, 164(1):252–268, 2005.

[9] Randolph E Bucklin, James M Lattin, Asim Ansari, Sunil Gupta, David Bell, Eloise Coupey, John DC Little, Carl Mela, Alan Montgomery, and Joel Steckel. Choice and the internet: From clickstream to research stream. *Marketing letters*, 13(3):245–258, 2002.

[10] Surajit Chaudhuri, Umeshwar Dayal, and Vivek Narasayya. An overview of business intelligence technology. *Communications of the ACM*, 54(8):88–98, 2011.

[11] Stephanie Chevalier. Leading online only fashion retailers visited and shopped from in the uk 2021. Available at `https://www.statista.com/statistics/1178854/online-fashion-retailer-consumers-purchasing-in-the-uk/` (2022/08).

[12] Tsan-Ming Choi, Hing Kai Chan, and Xiaohang Yue. Recent development in big data analytics for business operations and risk management. *IEEE transactions on cybernetics*, 47(1):81–92, 2016.

[13] Tsan-Ming Choi, Stein W Wallace, and Yulan Wang. Big data analytics in operations management. *Production and Operations Management*, 27(10):1868–1883, 2018.

[14] David L Davies and Donald W Bouldin. A cluster separation measure. *IEEE transactions on pattern analysis and machine intelligence*, (2):224–227, 1979.

[15] Massimiliano de Leoni and Safa Dündar. Event-log abstraction using batch session identification and clustering. In *Proceedings of the 35th Annual ACM Symposium on Applied Computing*, pages 36–44, 2020.

[16] Bettina Fazzinga, Sergio Flesca, Filippo Furfaro, and Luigi Pontieri. Process discovery from low-level event logs. In *International Conference on Advanced Information Systems Engineering*, pages 257–273. Springer, 2018.

[17] Fluxicon. Disco. Available at `https://fluxicon.com/blog/2012/05/say-hello-to-disco/` (2022/08).

[18] Francesco Folino, Massimo Guarascio, and Luigi Pontieri. Mining multi-variant process models from low-level logs. In *International Conference on Business Information Systems*, pages 165–177. Springer, 2015.

[19] Christian W Gunther and Anne Rozinat. Disco: Discover your processes. *BPM (Demos)*, 940(1):40–44, 2012.

[20] Christian W Günther, Anne Rozinat, and Wil MP Van Der Aalst. Activity mining by global trace segmentation. In *International Conference on Business Process Management*, pages 128–139. Springer, 2009.

[21] Christian W Günther and Wil MP Van Der Aalst. Fuzzy mining–adaptive process simplification based on multi-perspective metrics. In *International conference on business process management*, pages 328–343. Springer, 2007.

[22] Solomiya Ohinok1 Maryana Gvozd, Yaroslav Kis, Yuriy Fedun, and Galyna Kopets. S&p 500 index as an applied intelligent system for analyzing the development of e-commerce.

[23] John A Hartigan and Manchek A Wong. Algorithm as 136: A k-means clustering algorithm. *Journal of the royal statistical society. series c (applied statistics)*, 28(1):100–108, 1979.

[24] Sergio Hernandez, Pedro Alvarez, Javier Fabra, and Joaquin Ezpeleta. Analysis of users' behavior in structured e-commerce websites. *IEEE Access*, 5:11941–11958, 2017.

[25] Geoffrey E Hinton and Ruslan R Salakhutdinov. Reducing the dimensionality of data with neural networks. *science*, 313(5786):504–507, 2006.

[26] Eric J Johnson, Wendy W Moe, Peter S Fader, Steven Bellman, and Gerald L Lohse. On the depth and dynamics of online search behavior. *Management science*, 50(3):299–308, 2004.

[27] Tomoya Kawasaki, Hisayuki Wakashima, and Ryuichi Shibasaki. The use of e-commerce and the covid-19 outbreak: A panel data analysis in japan. *Transport Policy*, 115:88–100, 2022.

[28] Philip Kotler and Gary Armstrong. Principles of marketing (16th global edition). 2013.

[29] Danielle Lecointre-Erickson, Safaa Adil, Bruno Daucé, and Patrick Legohérel. The role of brick-and-mortar exterior atmospherics in post-covid era shopping experience: a systematic review and agenda for future research. *Journal of Strategic Marketing*, pages 1–15, 2021.

[30] Katherine N Lemon and Peter C Verhoef. Understanding customer experience throughout the customer journey. *Journal of marketing*, 80(6):69–96, 2016.

[31] Andrzej Maćkiewicz and Waldemar Ratajczak. Principal components analysis (pca). *Computers & Geosciences*, 19(3):303–342, 1993.

[32] Adetokunbo AO Makanju, A Nur Zincir-Heywood, and Evangelos E Milios. Clustering event logs using iterative partitioning. In *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 1255–1264, 2009.

[33] Riccardo Mangiaracina, Gianluca Brugnoli, et al. The ecommerce customer journey: A model to assess and compare the user experience of the ecommerce websites. *The Journal of Internet Banking and Commerce*, 14(3):1–11, 1970.

[34] Felix Mannhardt, Massimiliano de Leoni, Hajo A Reijers, Wil MP Van Der Aalst, and Pieter J Toussaint. From low-level events to activities-a pattern-based approach. In *International conference on business process management*, pages 125–141. Springer, 2016.

[35] Felix Mannhardt and Niek Tax. Unsupervised event abstraction using pattern abstraction and local process models. *arXiv preprint arXiv:1704.03520*, 2017.

[36] Daniel A Menascé, Virgilio AF Almeida, Rodrigo Fonseca, and Marco A Mendes. A methodology for workload characterization of e-commerce sites. In *Proceedings of the 1st ACM conference on Electronic commerce*, pages 119–128, 1999.

[37] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.

[38] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. *Advances in neural information processing systems*, 26, 2013.

[39] Wendy W Moe. Buying, searching, or browsing: Differentiating between online shoppers using in-store navigational clickstream. *Journal of consumer psychology*, 13(1-2):29–39, 2003.

[40] Wendy W Moe and Peter S Fader. Dynamic conversion behavior at e-commerce sites. *Management Science*, 50(3):326–335, 2004.

[41] Alan L Montgomery, Shibo Li, Kannan Srinivasan, and John C Liechty. Modeling online browsing and path analysis using clickstream data. *Marketing science*, 23(4):579–595, 2004.

[42] Ana Mosquera, Emma Juaneda Ayensa, Cristina Olarte Pascual, and Yolanda Sierra Murillo. Omnichannel shopper segmentation in the fashion industry. *Journal of Promotion Management*, 25(5):681–699, 2019.

[43] Ricardo Perez-Castillo, Barbara Weber, Jakob Pinggera, Stefan Zugal, Ignacio García-Rodríguez de Guzmán, and Mario Piattini. Generating event logs from non-process-aware systems enabling business process mining. *Enterprise Information Systems*, 5(3):301–335, 2011.

[44] Nicolas Poggi, Vinod Muthusamy, David Carrera, and Rania Khalaf. Business process mining from e-commerce web logs. In *Business process management*, pages 65–80. Springer, 2013.

[45] Orit Raphaeli, Anat Goldstein, and Lior Fink. Analyzing online consumer behavior in mobile and pc devices: A novel web usage mining approach. *Electronic commerce research and applications*, 26:1–12, 2017.

[46] Jana-Rebecca Rehse and Peter Fettke. Clustering business process activities for identifying reference model components. In *International Conference on Business Process Management*, pages 5–17. Springer, 2018.

[47] Mark S Rosenbaum, Mauricio Losada Otalora, and Germán Contreras Ramírez. How to create a realistic customer journey map. *Business horizons*, 60(1):143–150, 2017.

[48] Peter J Rousseeuw. Silhouettes: a graphical aid to the interpretation and validation of cluster analysis. *Journal of computational and applied mathematics*, 20:53–65, 1987.

[49] David Sánchez-Charles, Josep Carmona, Victor Muntés-Mulero, and Marc Solé. Reducing event variability in logs by clustering of word embeddings. In *International Conference on Business Process Management*, pages 191–203. Springer, 2017.

[50] Daniel Schellong, Jan Kemper, and Malte Brettel. Clickstream data as a source to uncover con-sumer shopping types in a large-scale online setting. 2016.

[51] Vladimir Simakov. History of formation of e-commerce enterprises as subjects of innovative entrepreneurship. *Three Seas Economic Journal*, 1(1):84–90, 2020.

[52] Brijendra Singh and Hemant Kumar Singh. Web data mining research: a survey. In *2010 IEEE International Conference on Computational Intelligence and Computing Research*, pages 1–10. IEEE, 2010.

[53] Ryan Skinner. Asos uses machine learning to understand customer value. Available at `https://www.forrester.com/blogs/machine-learning-asos-customer-value/` (2022/08).

[54] Qiang Su and Lu Chen. A method for discovering clusters of e-commerce interest patterns using click-stream data. *electronic commerce research and applications*, 14(1):1–13, 2015.

[55] Supermetrics. Google analytics field reference. Available at `https://supermetrics.com/docs/integration-google-analytics-fields/` (2022/08).

[56] Joshua B Tenenbaum, Vin de Silva, and John C Langford. A global geometric framework for nonlinear dimensionality reduction. *science*, 290(5500):2319–2323, 2000.

[57] Alessandro Terragni and Marwan Hassani. Analyzing customer journey with process mining: From discovery to recommendations. In *2018 IEEE 6th International Conference on Future Internet of Things and Cloud (FiCloud)*, pages 224–229. IEEE, 2018.

[58] Yan Tian and Concetta Stewart. History of e-commerce. In *Encyclopedia of e-commerce, e-government, and mobile commerce*, pages 559–564. IGI Global, 2006.

[59] Wil Van Der Aalst. *Process mining: data science in action*, volume 2. Springer, 2016.

[60] Wil MP Van Der Aalst, Hajo A Reijers, and Minseok Song. Discovering social networks from event logs. *Computer Supported Cooperative Work (CSCW)*, 14(6):549–593, 2005.

[61] Wil MP Van der Aalst, Boudewijn F van Dongen, Christian W Günther, Anne Rozinat, Eric Verbeek, and Ton Weijters. Prom: The process mining toolkit. *BPM (Demos)*, 489(31):2, 2009.

[62] Laurens Van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. *Journal of machine learning research*, 9(11), 2008.

[63] Sebastiaan J van Zelst, Felix Mannhardt, Massimiliano de Leoni, and Agnes Koschmider. Event abstraction in process mining: literature review and taxonomy. *Granular Computing*, 6(3):719–736, 2021.

[64] Kuang-Wei Wen and Kuo-Fang Peng. Market segmentation via structured click stream analysis. *Industrial Management & Data Systems*, 2002.

[65] Julia Wolny and Nipawan Charoensuksai. Mapping customer journeys in multichannel decision-making. *Journal of Direct, Data and Digital Marketing Practice*, 15(4):317–326, 2014.

[66] Fareed Zandkarimi, Jana-Rebecca Rehse, Pouya Soudmand, and Hartmut Hoehle. A generic framework for trace clustering in process mining. In *2020 2nd International Conference on Process Mining (ICPM)*, pages 177–184. IEEE, 2020.

[67] Melina Zavali, Ewelina Lacka, and Johannes de Smedt. Shopping hard or hardly shopping: Revealing consumer segments using clickstream data. *IEEE Transactions on Engineering Management*, pages 1–12, 2021.

ThesisCodeFinal

August 21, 2022

1 5. Dataset

[1]:
```
# importing "test3 3" dataset
import pandas as pd
df = pd.read_csv ('test3 3.csv')

# shows how many rows (records) and columns (features) the dataset consists of.
print("shape: ", df.shape)

# shows that the dataset is a composite of data from two different days: 1/8
 ↪and 3/8.
print("unique dates: ", df['date'].unique())

# Adding sessionId to the dataset.
df["sessionId"] = df["visitId"] + df["fullVisitorId"]

#shows the amount of sessions present in the dataset.
print("unique sessions: ", len(df['sessionId'].unique()))

# shows the amount of unique visitors present in the dataset.
print("unique visitors: ", len(df['fullVisitorId'].unique()))

# shows the five last records of the dataset.
print("first 5 records: ")
print(df.head())

print(df.describe())
print(df.info())
```

```
shape:  (589084, 22)
unique dates:  [20170801 20170803]
unique sessions:  31734
unique visitors:  30475
first 5 records:
   visitNumber     visitId  visitStartTime      date  visits  hits  pageviews  \
0            2  1501552553      1501552553  20170801     1.0     2        1.0
1            2  1501552553      1501552553  20170801     1.0     2        1.0
2           13  1501569067      1501569067  20170801     1.0     2        2.0
```

```
3           13  1501569067      1501569067  20170801     1.0     2        2.0
4           84  1501605097      1501605097  20170801     1.0     2        2.0

   timeOnSite  bounces  browser  … screenResolution fullVisitorId  \
0       692.0      1.0   Chrome  …          1440x900  8.106460e+18
1       692.0      1.0   Chrome  …          1440x900  8.106460e+18
2        26.0      NaN    Opera  …         2560x1440  7.015010e+18
3        26.0      NaN    Opera  …         2560x1440  7.015010e+18
4        36.0      NaN  Firefox  …          1440x900  6.623020e+18

   channelGrouping hitNumber    time hour  minute  pagePath  screenDepth  \
0           Direct         1       0    2      55         0            0
1           Direct         2  691661    3       7         0            0
2          Organic         1       0    7      31         1            0
3          Organic         2   25866    7      31         2            0
4            Email         1       0   17      31         3            0

      sessionId
0  8.106460e+18
1  8.106460e+18
2  7.015010e+18
3  7.015010e+18
4  6.623020e+18

[5 rows x 23 columns]
         visitNumber       visitId  visitStartTime          date    visits  \
count  589084.000000  5.890840e+05    5.890840e+05  5.890840e+05  588831.0
mean       43.853977  1.501691e+09    1.501691e+09  2.017080e+07       1.0
std       133.743666  8.696889e+04    8.696774e+04  9.892368e-01       0.0
min         1.000000  1.501540e+09    1.501542e+09  2.017080e+07       1.0
25%         2.000000  1.501599e+09    1.501599e+09  2.017080e+07       1.0
50%         6.000000  1.501740e+09    1.501740e+09  2.017080e+07       1.0
75%        29.000000  1.501768e+09    1.501768e+09  2.017080e+07       1.0
max      7468.000000  1.501801e+09    1.501801e+09  2.017080e+07       1.0

                hits      pageviews     timeOnSite  bounces  fullVisitorId  \
count  589084.000000  587566.000000  574995.000000  13035.0   5.890840e+05
mean       70.616805      18.415269    1071.057750      1.0   4.608522e+18
std        84.775264      21.088153    1345.228051      0.0   2.651525e+18
min         2.000000       1.000000       1.000000      1.0   8.933250e+13
25%        18.000000       5.000000     192.000000      1.0   2.303820e+18
50%        41.000000      11.000000     560.000000      1.0   4.607640e+18
75%        90.000000      24.000000    1443.000000      1.0   6.918160e+18
max       500.000000     208.000000   10663.000000      1.0   9.223290e+18

           hitNumber          time           hour         minute  \
count  589084.000000  5.890840e+05  589084.000000  589084.000000
mean       35.955833  5.082300e+05      13.566963      29.583188
```

```
std        53.182172  8.419655e+05       5.926740      17.316847
min         1.000000  0.000000e+00       0.000000       0.000000
25%         6.000000  3.756900e+04       9.000000      15.000000
50%        17.000000  1.753960e+05      14.000000      30.000000
75%        43.000000  5.936828e+05      19.000000      45.000000
max       500.000000  1.066318e+07      23.000000      59.000000

            pagePath  screenDepth     sessionId
count  589084.000000     589084.0  5.890840e+05
mean    14079.189980          0.0  4.608522e+18
std     12328.729738          0.0  2.651525e+18
min         0.000000          0.0  8.933400e+13
25%      4301.000000          0.0  2.303820e+18
50%     11060.000000          0.0  4.607640e+18
75%     19887.000000          0.0  6.918160e+18
max     46519.000000          0.0  9.223290e+18
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 589084 entries, 0 to 589083
Data columns (total 23 columns):
 #   Column            Non-Null Count   Dtype
---  ------            --------------   -----
 0   visitNumber       589084 non-null  int64
 1   visitId           589084 non-null  int64
 2   visitStartTime    589084 non-null  int64
 3   date              589084 non-null  int64
 4   visits            588831 non-null  float64
 5   hits              589084 non-null  int64
 6   pageviews         587566 non-null  float64
 7   timeOnSite        574995 non-null  float64
 8   bounces           13035 non-null   float64
 9   browser           589084 non-null  object
 10  browserVersion    589084 non-null  object
 11  browserSize       589084 non-null  object
 12  isMobile          589084 non-null  bool
 13  screenResolution  589084 non-null  object
 14  fullVisitorId     589084 non-null  float64
 15  channelGrouping   589084 non-null  object
 16  hitNumber         589084 non-null  int64
 17  time              589084 non-null  int64
 18  hour              589084 non-null  int64
 19  minute            589084 non-null  int64
 20  pagePath          589084 non-null  int64
 21  screenDepth       589084 non-null  int64
 22  sessionId         589084 non-null  float64
dtypes: bool(1), float64(6), int64(11), object(5)
memory usage: 99.4+ MB
None
```

1.1 5.1 Features

[2]:
```
# shows the names of the 22 features
print("Features: ")
for col in df.columns:
    print("- ", col)
```

```
Features:
-  visitNumber
-  visitId
-  visitStartTime
-  date
-  visits
-  hits
-  pageviews
-  timeOnSite
-  bounces
-  browser
-  browserVersion
-  browserSize
-  isMobile
-  screenResolution
-  fullVisitorId
-  channelGrouping
-  hitNumber
-  time
-  hour
-  minute
-  pagePath
-  screenDepth
-  sessionId
```

[3]:
```
# quick test to help figure out wether the attribute is hit, session or user␣
 ↪level.
# (if the value of an attribute is constant for all records belonging to the␣
 ↪same user, it is a user level attribute.)
# (if the value of an attribute is only constant for all records belonging to␣
 ↪the same session, it is a session level attribute.)

# shows the fullVisitorId and the associated count of records.
#print(df['fullVisitorId'].value_counts())
#print(df['sessionId'].value_counts())


# visitorX is the visitor with the most records present in the dataset, namely␣
 ↪639.
```

```
# sessionX is the session with the most records present in the dataset, namely␣
↪500.
visitorX = df[df['fullVisitorId'] == 7.785180e+18]
sessionX = df[df['sessionId'] == df['sessionId'].value_counts().index[0]]
#print(visitorX)
#print(sessionX)

#Check wether attribute stays constant for all records belonging to visitorX,␣
↪then it is a user level attribute.
#Tested for columns: browser, browserVersion, browserSize, isMobile,␣
↪screenResolution, fullVisitorId, visitNumber, visitId, visitStartTime, hits,␣
↪pageviews,  timeOnSite, bounces, channelGrouping, screenDepth
attribute = "channelGrouping"

if (visitorX[attribute] == visitorX.iloc[0][attribute]).all():
    print("User level attribute")
else:
    if (sessionX[attribute] == sessionX.iloc[0][attribute]).all():
        print("Session level attribute")
    else:
        print("Hit level attribute")
```

```
User level attribute
```

[4]:
```
# For session-level attributes we will work with a dataframe on session level.
df_session = df.groupby(df['sessionId']).aggregate('first')
#print(df_session)

# For user-level attributes we will work with a dataframe on user level.
df_user = df.groupby(df['fullVisitorId']).aggregate('last')
#df_user
```

[5]:
```
# 5.1.1 fullVisitorId
print("distribution records:")
print(df['fullVisitorId'].value_counts())
print(df['fullVisitorId'].value_counts(normalize = True))
print(df['fullVisitorId'].value_counts().describe())
#print(df['fullVisitorId'].value_counts().mode())
#print(sum(df['fullVisitorId'].value_counts() == 2))

print("distribution sessions:")
print(df_session['fullVisitorId'].value_counts())
print(df_session['fullVisitorId'].value_counts(normalize = True))
print(df_session['fullVisitorId'].value_counts().describe())
print(df_session['fullVisitorId'].value_counts().mode())
print(sum(df_session['fullVisitorId'].value_counts() == 1))
```

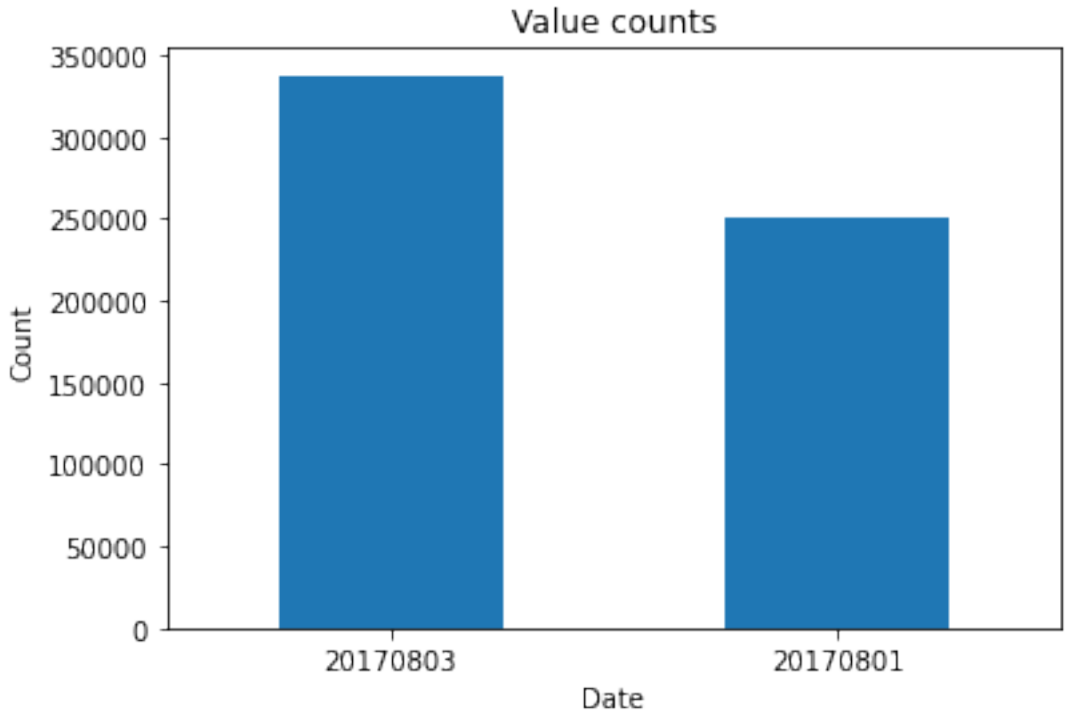
```
distribution records:
7.785180e+18    639
7.026930e+18    524
6.559620e+18    500
3.348390e+18    500
6.127210e+18    500
                …
2.383760e+18      2
6.198450e+18      2
7.501690e+16      2
7.661350e+18      2
8.106460e+18      2
Name: fullVisitorId, Length: 30475, dtype: int64
7.785180e+18    0.001085
7.026930e+18    0.000890
6.559620e+18    0.000849
3.348390e+18    0.000849
6.127210e+18    0.000849
                  …
2.383760e+18    0.000003
6.198450e+18    0.000003
7.501690e+16    0.000003
7.661350e+18    0.000003
8.106460e+18    0.000003
Name: fullVisitorId, Length: 30475, dtype: float64
count    30475.000000
mean        19.330074
std         32.289961
min          2.000000
25%          4.000000
50%          8.000000
75%         21.000000
max        639.000000
Name: fullVisitorId, dtype: float64
distribution sessions:
6.676460e+18    4
3.339700e+18    3
9.343160e+17    3
1.539900e+18    3
2.904900e+18    3
                ..
3.149540e+18    1
3.149340e+18    1
3.149210e+18    1
3.149090e+18    1
9.223290e+18    1
Name: fullVisitorId, Length: 30475, dtype: int64
6.676460e+18    0.000126
```

```
3.339700e+18    0.000095
9.343160e+17    0.000095
1.539900e+18    0.000095
2.904900e+18    0.000095
                  …
3.149540e+18    0.000032
3.149340e+18    0.000032
3.149210e+18    0.000032
3.149090e+18    0.000032
9.223290e+18    0.000032
Name: fullVisitorId, Length: 30475, dtype: float64
count    30475.000000
mean         1.041313
std          0.208831
min          1.000000
25%          1.000000
50%          1.000000
75%          1.000000
max          4.000000
Name: fullVisitorId, dtype: float64
0    1
Name: fullVisitorId, dtype: int64
29276
```

[6]:
```python
# 5.1.2 date
import matplotlib.pyplot as plt
print("distribution records:")
print(df['date'].value_counts())
print(df['date'].value_counts(normalize = True))

df['date'].value_counts().plot(kind="bar")
plt.title("Value counts")
plt.xlabel("Date")
plt.xticks(rotation=0)
plt.ylabel("Count")
plt.show()

print("distribution sessions:")
print(df_session['date'].value_counts())
print(df_session['date'].value_counts(normalize = True))

df_session['date'].value_counts().plot(kind="bar")
plt.title("Value counts")
plt.xlabel("Date")
plt.xticks(rotation=0)
plt.ylabel("Count")
plt.show()
```

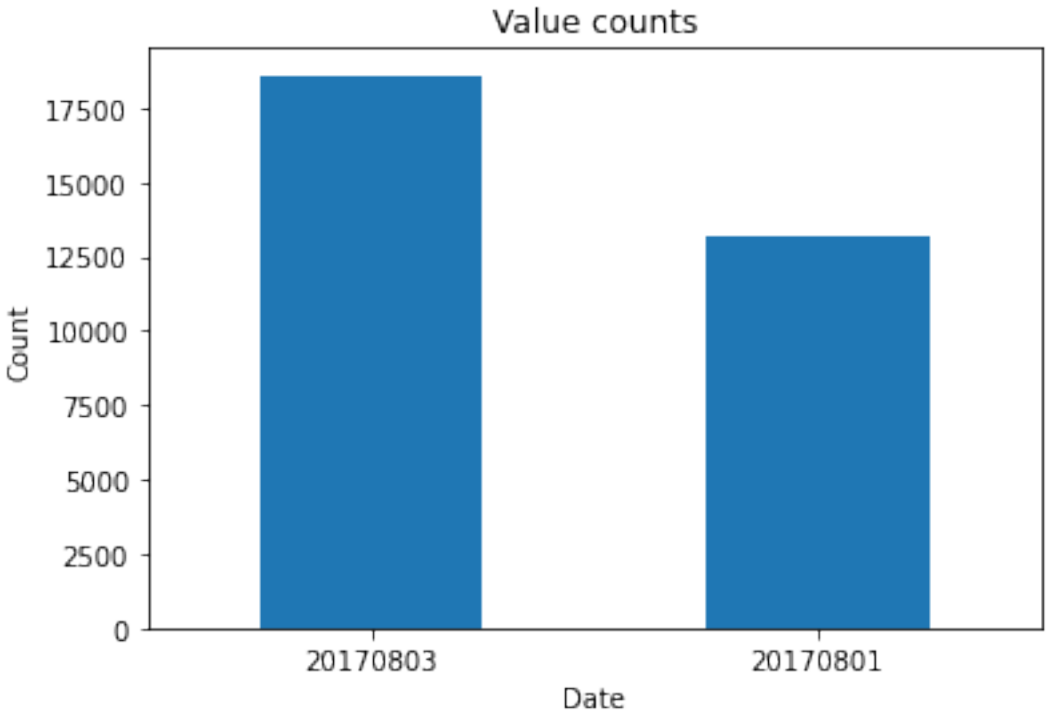
```
print("distribution users:")
print(df_user['date'].value_counts())
print(df_user['date'].value_counts(normalize = True))

df_user['date'].value_counts().plot(kind="bar")
plt.title("Value counts")
plt.xlabel("Date")
plt.xticks(rotation=0)
plt.ylabel("Count")
plt.show()
```

```
distribution records:
20170803    337642
20170801    251442
Name: date, dtype: int64
20170803    0.573164
20170801    0.426836
Name: date, dtype: float64
```

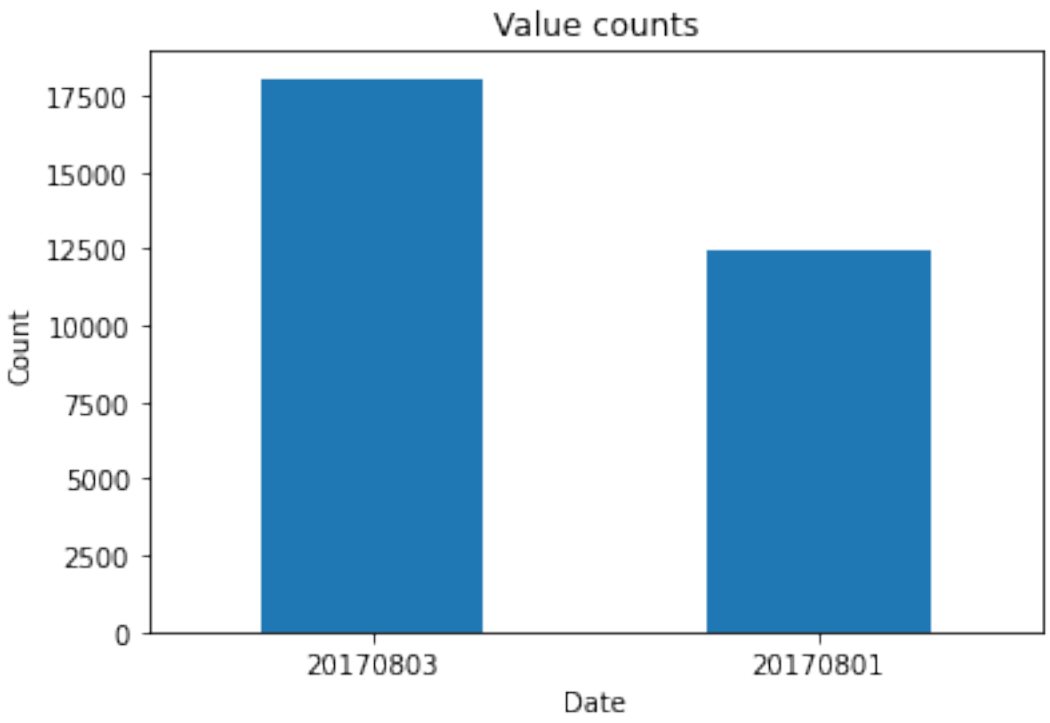



```
distribution sessions:
20170803    18590
20170801    13144
Name: date, dtype: int64
```

```
20170803    0.585807
20170801    0.414193
Name: date, dtype: float64
```




```
distribution users:
20170803    18058
20170801    12417
Name: date, dtype: int64
20170803    0.592551
20170801    0.407449
Name: date, dtype: float64
```

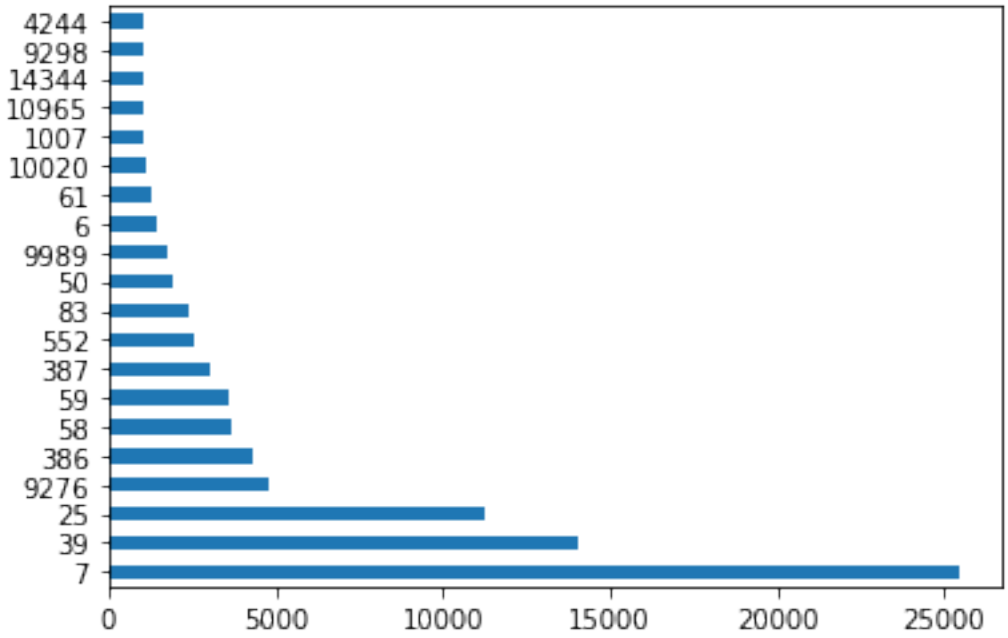


[7]:
```
# 5.1.3 pagePath
print("distribution records:")
print(df['pagePath'].value_counts())
print(df['pagePath'].value_counts(normalize = True))

df['pagePath'].value_counts()[:20].plot(kind='barh')
```

```
distribution records:
7         25448
39        14074
25        11244
9276       4794
386        4297
          …
36641         1
18021         1
18026         1
29618         1
46519         1
Name: pagePath, Length: 46520, dtype: int64
7         0.043199
39        0.023891
25        0.019087
9276      0.008138
```

```
386      0.007294
           …
36641    0.000002
18021    0.000002
18026    0.000002
29618    0.000002
46519    0.000002
Name: pagePath, Length: 46520, dtype: float64
```

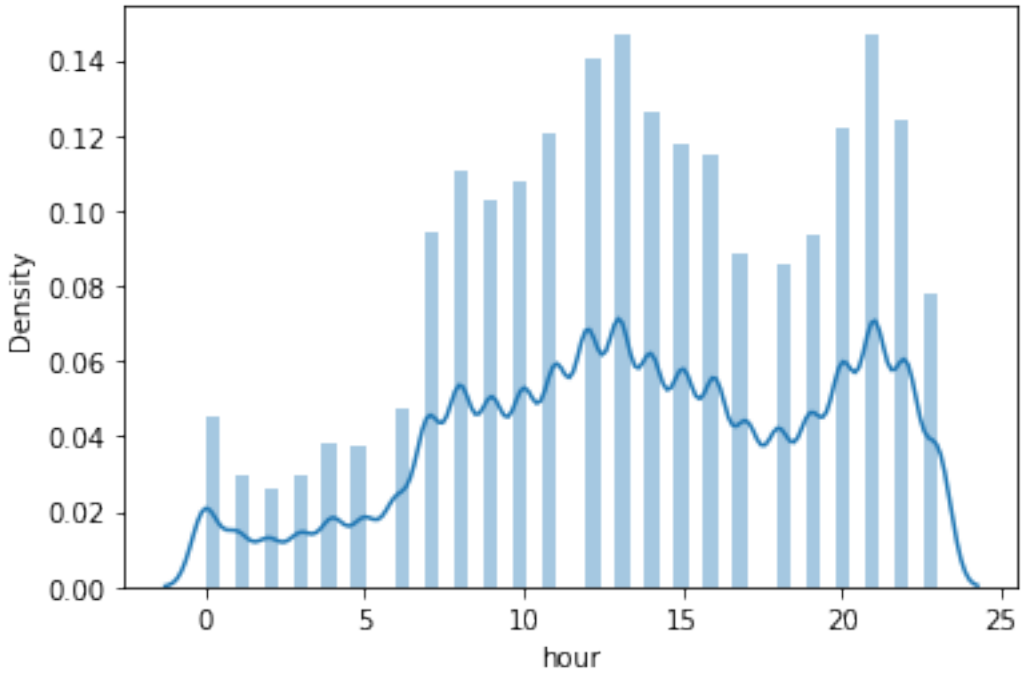
[7]: <AxesSubplot:>



[8]:
```
import seaborn as sns
# 5.1.4 hour
sns.distplot(df['hour'])
```

```
C:\Users\nvand\AppData\Local\Programs\Python\Python310\lib\site-
packages\seaborn\distributions.py:2619: FutureWarning: `distplot` is a
deprecated function and will be removed in a future version. Please adapt your
code to use either `displot` (a figure-level function with similar flexibility)
or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```

[8]: <AxesSubplot:xlabel='hour', ylabel='Density'>



[9]:
```
# 5.1.5 Other features
# visitNumber
print(df_session['visitNumber'].describe())
print(df_session['visitNumber'].value_counts())
```

```
count    31734.000000
mean        38.211760
std        124.644919
min          1.000000
25%          1.000000
50%          4.000000
75%         24.000000
max       7468.000000
Name: visitNumber, dtype: float64
1       9635
2       3188
3       1928
4       1309
5       1021
       ...
7468       1
2858       1
482        1
849        1
408        1
```

```
Name: visitNumber, Length: 751, dtype: int64
```

[10]:
```
# visitId
print(df['sessionId'].value_counts())
print(df['sessionId'].value_counts().describe())
#print(df['sessionId'].value_counts().mode())
#print(sum(df['sessionId'].value_counts() == 2))
```

```
6.127210e+18    500
7.026930e+18    500
3.348390e+18    500
4.655710e+18    500
4.353940e+18    500
                ...
5.271620e+18      2
2.837330e+18      2
3.091600e+18      2
7.017690e+18      2
8.106460e+18      2
Name: sessionId, Length: 31734, dtype: int64
count    31734.000000
mean        18.563181
std         31.093789
min          2.000000
25%          4.000000
50%          8.000000
75%         20.000000
max        500.000000
Name: sessionId, dtype: float64
```

[11]:
```
# hits
print(df_session['hits'].describe())
print(df_session['hits'].value_counts())
```

```
count    31734.000000
mean        18.541722
std         31.089795
min          2.000000
25%          4.000000
50%          8.000000
75%         20.000000
max        500.000000
Name: hits, dtype: float64
2      4224
3      3658
4      2451
5      2000
6      1482
```

```
...
349    1
218    1
208    1
394    1
472    1
Name: hits, Length: 292, dtype: int64
```

[12]:
```
# timeOnSite
print(df_session['timeOnSite'].describe())
```

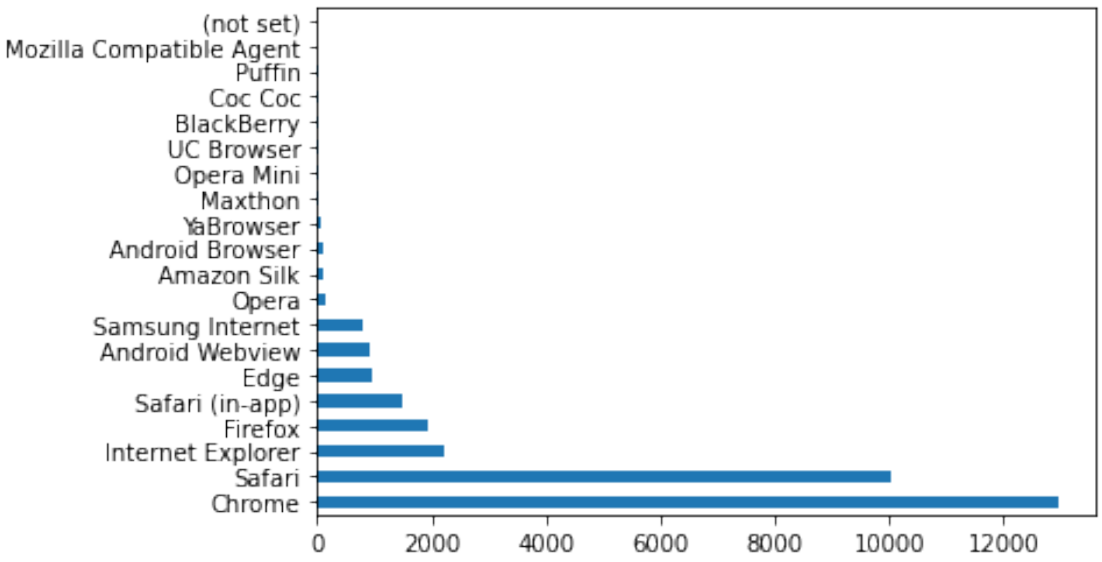
```
count    26999.000000
mean       360.215601
std        658.296859
min          1.000000
25%         33.000000
50%        112.000000
75%        371.000000
max      10663.000000
Name: timeOnSite, dtype: float64
```

[13]:
```
# browser
print("distribution sessions:")
print(df_session['browser'].value_counts())
print(df_session['browser'].value_counts(normalize = True))

df_session['browser'].value_counts()[:20].plot(kind='barh')

#df_session['browser'].value_counts().plot(kind="bar")
#plt.title("Value counts")
#plt.xlabel("browser")
#plt.xticks(rotation=0)
#plt.ylabel("Count")
#plt.show()
```

```
distribution sessions:
Chrome                     12969
Safari                     10025
Internet Explorer           2228
Firefox                     1939
Safari (in-app)             1476
Edge                         938
Android Webview              921
Samsung Internet             799
Opera                        154
Amazon Silk                   98
Android Browser               95
YaBrowser                     49
```

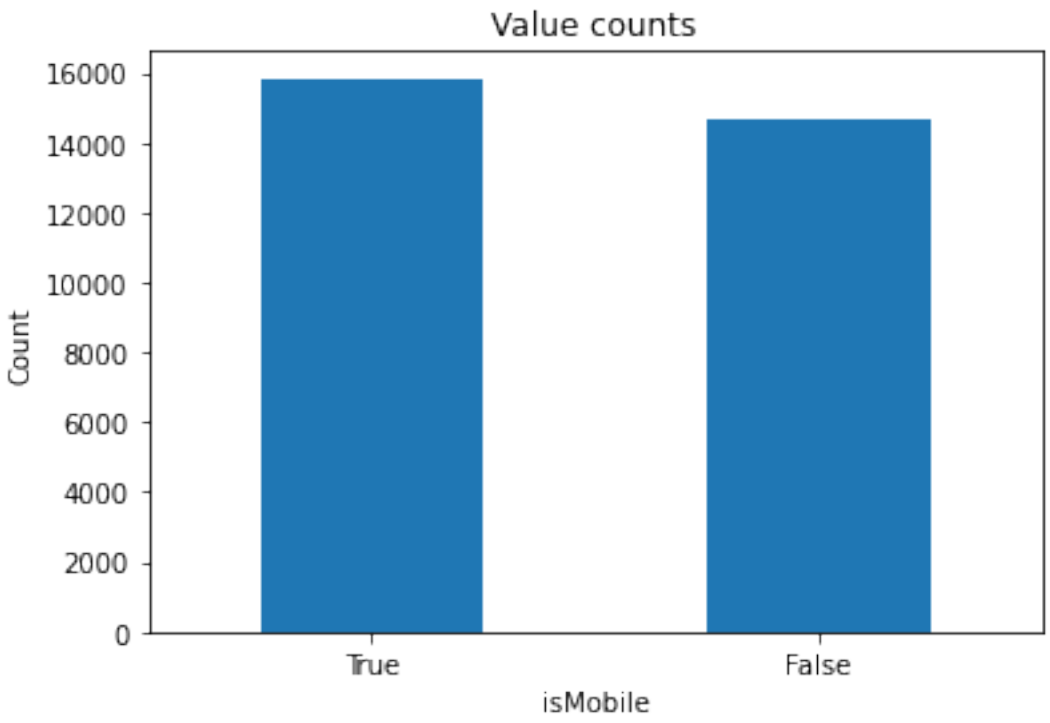
```
Maxthon                          13
Opera Mini                        8
UC Browser                        6
BlackBerry                        5
Coc Coc                           4
Puffin                            4
Mozilla Compatible Agent          1
(not set)                         1
MRCHROME                          1
Name: browser, dtype: int64
Chrome                       0.408678
Safari                       0.315907
Internet Explorer            0.070209
Firefox                      0.061102
Safari (in-app)              0.046512
Edge                         0.029558
Android Webview              0.029022
Samsung Internet             0.025178
Opera                        0.004853
Amazon Silk                  0.003088
Android Browser              0.002994
YaBrowser                    0.001544
Maxthon                      0.000410
Opera Mini                   0.000252
UC Browser                   0.000189
BlackBerry                   0.000158
Coc Coc                      0.000126
Puffin                       0.000126
Mozilla Compatible Agent     0.000032
(not set)                    0.000032
MRCHROME                     0.000032
Name: browser, dtype: float64
```

[13]: <AxesSubplot:>



[14]:
```
# isMobile
print("distribution users:")
print(df_user['isMobile'].value_counts())
print(df_user['isMobile'].value_counts(normalize = True))

df_user['isMobile'].value_counts().plot(kind="bar")
plt.title("Value counts")
plt.xlabel("isMobile")
plt.xticks(rotation=0)
plt.ylabel("Count")
plt.show()
```

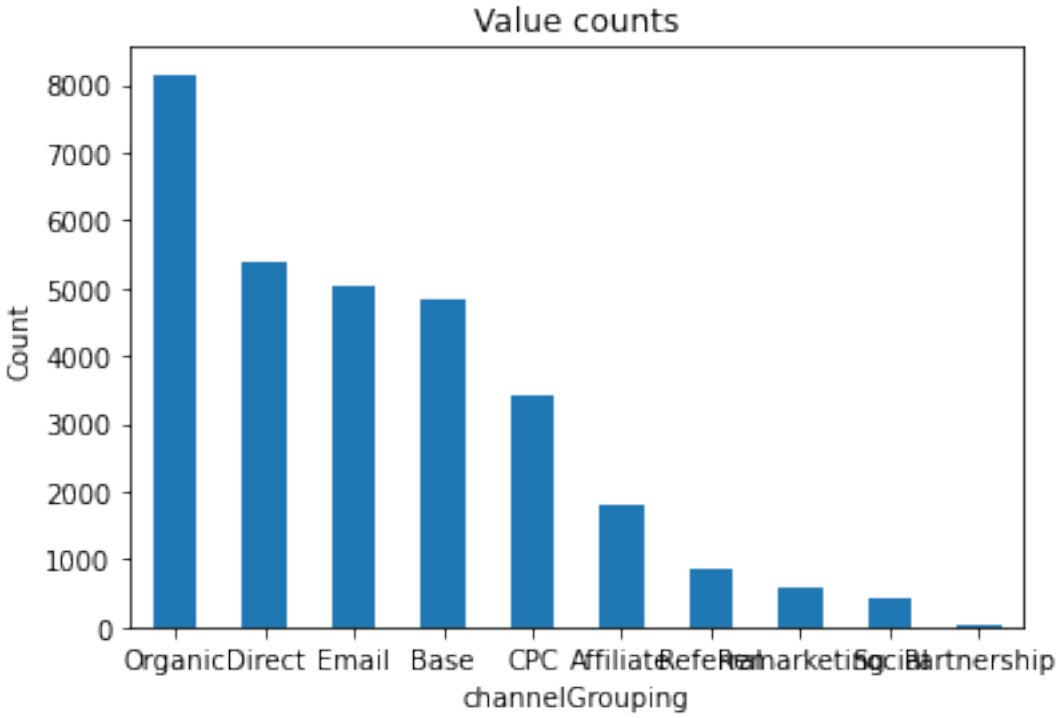
```
distribution users:
True     15837
False    14638
Name: isMobile, dtype: int64
True     0.519672
False    0.480328
Name: isMobile, dtype: float64
```



[15]:
```
# channelGrouping
print("distribution users:")
print(df_user['channelGrouping'].value_counts())
print(df_user['channelGrouping'].value_counts(normalize = True))

df_user['channelGrouping'].value_counts().plot(kind="bar")
plt.title("Value counts")
plt.xlabel("channelGrouping")
plt.xticks(rotation=0)
plt.ylabel("Count")
plt.show()
```

```
distribution users:
Organic        8150
Direct         5397
Email          5010
Base           4836
CPC            3409
Affiliate      1802
Referral        845
Remarketing     565
Social          428
Partnership      33
Name: channelGrouping, dtype: int64
```

```
Organic        0.267432
Direct         0.177096
Email          0.164397
Base           0.158687
CPC            0.111862
Affiliate      0.059130
Referral       0.027728
Remarketing    0.018540
Social         0.014044
Partnership    0.001083
Name: channelGrouping, dtype: float64
```



[16]:
```
# time
print(df['time'].describe())
```

```
count    5.890840e+05
mean     5.082300e+05
std      8.419655e+05
min      0.000000e+00
25%      3.756900e+04
50%      1.753960e+05
75%      5.936828e+05
max      1.066318e+07
Name: time, dtype: float64
```

[17]:
```
# not useful for this dataset

# screenDepth
print("distribution users:")
print(df_user['screenDepth'].value_counts())
print(df_user['screenDepth'].value_counts(normalize = True))

# visits
print(df_session['visits'].describe())

# pageviews
print(df_session['pageviews'].describe())
print(df_session['pageviews'].value_counts())

# bounces
print(df_session['bounces'].describe())

# hitNumber
print(df_session['hitNumber'].describe())
print(df_session['hitNumber'].value_counts())

# or not interesting for the further analysis

# browserVersion
# browserSize
# screenResolution
```

```
distribution users:
0    30475
Name: screenDepth, dtype: int64
0    1.0
Name: screenDepth, dtype: float64
count    31674.0
mean         1.0
std          0.0
min          1.0
25%          1.0
50%          1.0
75%          1.0
max          1.0
Name: visits, dtype: float64
count    31439.000000
mean         5.640256
std          8.528315
min          1.000000
25%          1.000000
50%          3.000000
```

```
75%          6.000000
max        208.000000
Name: pageviews, dtype: float64
1.0      10206
2.0       4779
3.0       3289
4.0       2313
5.0       1746
         …
121.0        1
133.0        1
89.0         1
107.0        1
126.0        1
Name: pageviews, Length: 110, dtype: int64
count    4419.0
mean        1.0
std         0.0
min         1.0
25%         1.0
50%         1.0
75%         1.0
max         1.0
Name: bounces, dtype: float64
count    31734.000000
mean         1.066049
std          2.654186
min          1.000000
25%          1.000000
50%          1.000000
75%          1.000000
max        272.000000
Name: hitNumber, dtype: float64
1      31675
2          5
6          4
9          4
10         3
14         3
34         2
45         2
4          2
11         2
12         2
29         1
24         1
114        1
38         1
```

```
93          1
54          1
15          1
225         1
17          1
20          1
21          1
23          1
37          1
52          1
62          1
156         1
3           1
13          1
16          1
28          1
51          1
58          1
97          1
272         1
44          1
43          1
8           1
57          1
90          1
41          1
Name: hitNumber, dtype: int64
```

2 7 Experimental Evaluation

2.1 7.1 Preprocessing

[18]:
```
print("total number of records in the dataset: ", len(df))
print("total number of pagePath values in the dataset: ", len(df['pagePath'].
 ↪unique()))
print("total number of fullVisitorId values in the dataset: ",␣
 ↪len(df['fullVisitorId'].unique()))

#df.info()
```

```
total number of records in the dataset:  589084
total number of pagePath values in the dataset:  46520
total number of fullVisitorId values in the dataset:  30475
```

2.1.1 Filter 1: Remove rare pagePath values

[19]:
```
import numpy as np

# calculate absolute frequency of each pagePath value.
afreq_pp = df['pagePath'].value_counts()
print(afreq_pp)

# calculate relative frequency of each pagePath value.
#print(len(df)) #589.084
rfreq_pp = df['pagePath'].value_counts(normalize=True)
print(rfreq_pp)

# select all pagePath values with a relative frequency lower than 0.0001.
rare_pp = np.where(rfreq_pp < 0.0001)
print(rare_pp[0])

# filter1: all records referring to a pagePath value with a relative frequency␣
 ↪< 0.0001 are removed.
subset1 = df[df.pagePath.isin(rare_pp[0]) == False]
#subset1

# total amount of pagePath values present in subset1 compared with in the␣
 ↪original dataset.
print("total amount of pagePath values present in subset1 compared with in the␣
 ↪original dataset:", len(subset1['pagePath'].unique()), len(df['pagePath'].
 ↪unique()))

# total amount of records present in subset1 compared with in the original␣
 ↪dataset.
print("total amount of records present in the dataset compared with in the␣
 ↪original dataset:", len(subset1), len(df))

# total amount of visitors present in subset1 compared with in the original␣
 ↪dataset.
print("total amount of visitors present in subset1 compared with in the␣
 ↪original dataset:", len(subset1['fullVisitorId'].unique()),␣
 ↪len(df['fullVisitorId'].unique()))
```

```
7         25448
39        14074
25        11244
9276       4794
386        4297
          …
36641         1
18021         1
```

```
18026        1
29618        1
46519        1
Name: pagePath, Length: 46520, dtype: int64
7        0.043199
39       0.023891
25       0.019087
9276     0.008138
386      0.007294
           …
36641    0.000002
18021    0.000002
18026    0.000002
29618    0.000002
46519    0.000002
Name: pagePath, Length: 46520, dtype: float64
[ 1264  1265  1266 … 46517 46518 46519]
total amount of pagePath values present in subset1 compared with in the original
dataset: 1264 46520
total amount of records present in the dataset compared with in the original
dataset: 109259 589084
total amount of visitors present in subset1 compared with in the original
dataset: 14087 30475
```

2.1.2 Filter 2: Remove all sessions that consist of only 1 record

[20]:
```
# calculate absolute frequency of each sessionId value.
afreq_si = subset1['sessionId'].value_counts()
#print(afreq_si)

# select all sessionId values with an absolute frequency equal to 1 and store␣
 ↪these in the list rare_si.
rare_si = afreq_si[afreq_si == 1].index.tolist()
#print(len(rare_si))

# subset2: remove all fullVisitorIds which only visit 1 webpage.
subset2 = subset1[subset1.sessionId.isin(rare_si)== False]
#subset2

# total amount of pagePath values present in subset2 compared with in subset1.
print("total amount of pagePath values present in subset2 compared with in␣
 ↪subset1:")
print(len(subset2['pagePath'].unique()), len(subset1['pagePath'].unique()))

# total amount of records present in subset2 compared with in subset1.
print("total amount of records present in subset2 compared with in subset1:")
print(len(subset2), len(subset1))
```

```
# total amount of visitors present in subset2 compared with in subset1.
print("total amount of visitors present in subset2 compared with in subset1:")
print(len(subset2['fullVisitorId'].unique()), len(subset1['fullVisitorId'].
 ↪unique()))
```

```
total amount of pagePath values present in subset2 compared with in subset1:
1264 1264
total amount of records present in subset2 compared with in subset1:
107653 109259
total amount of visitors present in subset2 compared with in subset1:
12554 14087
```

[21]:
```
subset2
```

[21]:

	visitNumber	visitId	visitStartTime	date	visits	hits	\
0	2	1501552553	1501552553	20170801	1.0	2	
1	2	1501552553	1501552553	20170801	1.0	2	
2	13	1501569067	1501569067	20170801	1.0	2	
3	13	1501569067	1501569067	20170801	1.0	2	
4	84	1501605097	1501605097	20170801	1.0	2	
…	…	…	…	…	…	…	
589072	4	1501753787	1501753787	20170803	1.0	3	
589073	4	1501753787	1501753787	20170803	1.0	3	
589081	175	1501749772	1501749772	20170803	1.0	3	
589082	175	1501749772	1501749772	20170803	1.0	3	
589083	175	1501749772	1501749772	20170803	1.0	3	

	pageviews	timeOnSite	bounces	browser	…	screenResolution	\
0	1.0	692.0	1.0	Chrome	…	1440x900	
1	1.0	692.0	1.0	Chrome	…	1440x900	
2	2.0	26.0	NaN	Opera	…	2560x1440	
3	2.0	26.0	NaN	Opera	…	2560x1440	
4	2.0	36.0	NaN	Firefox	…	1440x900	
…	…	…	…	…	…	…	
589072	2.0	28.0	NaN	Safari	…	1344x840	
589073	2.0	28.0	NaN	Safari	…	1344x840	
589081	1.0	8.0	NaN	Firefox	…	1920x1200	
589082	1.0	8.0	NaN	Firefox	…	1920x1200	
589083	1.0	8.0	NaN	Firefox	…	1920x1200	

	fullVisitorId	channelGrouping	hitNumber	time	hour	minute	\
0	8.106460e+18	Direct	1	0	2	55	
1	8.106460e+18	Direct	2	691661	3	7	
2	7.015010e+18	Organic	1	0	7	31	
3	7.015010e+18	Organic	2	25866	7	31	
4	6.623020e+18	Email	1	0	17	31	

```
…              …                …        …      …  …      …
589072  2.465450e+18            Email          1       0   10      49
589073  2.465450e+18            Email          2   25329   10      50
589081  1.783760e+18            Email          1       0    9      42
589082  1.783760e+18            Email          2    1919    9      42
589083  1.783760e+18            Email          3    8424    9      43

        pagePath  screenDepth     sessionId
0              0            0  8.106460e+18
1              0            0  8.106460e+18
2              1            0  7.015010e+18
3              2            0  7.015010e+18
4              3            0  6.623020e+18
…            …            …            …
589072       518            0  2.465450e+18
589073       518            0  2.465450e+18
589081        25            0  1.783760e+18
589082        25            0  1.783760e+18
589083        25            0  1.783760e+18

[107653 rows x 23 columns]
```

2.2 7.2 Web log to event log mapping

[22]:
```
# copying the df in df_eventlog on which manipulations will be performed.
df_eventlog = subset2
df_eventlog.head()

# merging the "hour" and "minute" attribute to 1 "timeStamp" attribute and add it to the df_eventlog.
df_eventlog['hour'] = df_eventlog['hour'].astype(str).str.zfill(2)
df_eventlog['minute'] = df_eventlog['minute'].astype(str).str.zfill(2)
df_eventlog['timeStamp'] = df_eventlog['date'].astype(str) + ' ' + df_eventlog['hour'] + ':' + df_eventlog['minute']


# export df_eventlog as csv file.
#df_eventlog.to_csv('df_eventlog.csv')
```

```
C:\Users\nvand\AppData\Local\Temp\ipykernel_28716\2888405801.py:6:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  df_eventlog['hour'] = df_eventlog['hour'].astype(str).str.zfill(2)
```

```
C:\Users\nvand\AppData\Local\Temp\ipykernel_28716\2888405801.py:7:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  df_eventlog['minute'] = df_eventlog['minute'].astype(str).str.zfill(2)
C:\Users\nvand\AppData\Local\Temp\ipykernel_28716\2888405801.py:8:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  df_eventlog['timeStamp'] = df_eventlog['date'].astype(str) + ' ' +
df_eventlog['hour'] + ':' + df_eventlog['minute']
```

[23]:
```
df_eventlog
```

[23]:

	visitNumber	visitId	visitStartTime	date	visits	hits	\
0	2	1501552553	1501552553	20170801	1.0	2	
1	2	1501552553	1501552553	20170801	1.0	2	
2	13	1501569067	1501569067	20170801	1.0	2	
3	13	1501569067	1501569067	20170801	1.0	2	
4	84	1501605097	1501605097	20170801	1.0	2	
…	…	…	…	…	…	…	
589072	4	1501753787	1501753787	20170803	1.0	3	
589073	4	1501753787	1501753787	20170803	1.0	3	
589081	175	1501749772	1501749772	20170803	1.0	3	
589082	175	1501749772	1501749772	20170803	1.0	3	
589083	175	1501749772	1501749772	20170803	1.0	3	

	pageviews	timeOnSite	bounces	browser	…	fullVisitorId	\
0	1.0	692.0	1.0	Chrome	…	8.106460e+18	
1	1.0	692.0	1.0	Chrome	…	8.106460e+18	
2	2.0	26.0	NaN	Opera	…	7.015010e+18	
3	2.0	26.0	NaN	Opera	…	7.015010e+18	
4	2.0	36.0	NaN	Firefox	…	6.623020e+18	
…	…	…	…	…	…	…	
589072	2.0	28.0	NaN	Safari	…	2.465450e+18	
589073	2.0	28.0	NaN	Safari	…	2.465450e+18	
589081	1.0	8.0	NaN	Firefox	…	1.783760e+18	
589082	1.0	8.0	NaN	Firefox	…	1.783760e+18	
589083	1.0	8.0	NaN	Firefox	…	1.783760e+18	

	channelGrouping	hitNumber	time	hour	minute	pagePath	screenDepth	\

```
0              Direct          1       0    02     55         0             0
1              Direct          2  691661    03     07         0             0
2             Organic          1       0    07     31         1             0
3             Organic          2   25866    07     31         2             0
4               Email          1       0    17     31         3             0
…                 …        …     …   …    …      …         …
589072          Email          1       0    10     49       518             0
589073          Email          2   25329    10     50       518             0
589081          Email          1       0    09     42        25             0
589082          Email          2    1919    09     42        25             0
589083          Email          3    8424    09     43        25             0

           sessionId       timeStamp
0       8.106460e+18  20170801 02:55
1       8.106460e+18  20170801 03:07
2       7.015010e+18  20170801 07:31
3       7.015010e+18  20170801 07:31
4       6.623020e+18  20170801 17:31
…                …                …
589072  2.465450e+18  20170803 10:49
589073  2.465450e+18  20170803 10:50
589081  1.783760e+18  20170803 09:42
589082  1.783760e+18  20170803 09:42
589083  1.783760e+18  20170803 09:43

[107653 rows x 24 columns]
```

2.3 7.3 Trace Clustering

2.3.1 7.3.1 Trace representation

[24]:
```
# creating feature matrix
feature_matrix = pd.crosstab(subset2.astype(str).fullVisitorId, subset2.
 ↪astype(str).pagePath)

# print(feature_matrix.index)
# print(feature_matrix.values)
# print(feature_matrix.shape)
# print(feature_matrix.sum(axis=1))
```

[25]:
```
# from sklearn.preprocessing import StandardScaler

# Scaling the data to keep the different attributes in same range.
# standard_fm = StandardScaler().fit_transform(feature_matrix)
# print(standard_fm)

# check wether standardization worked (mean = 0 and standard deviation is one).
```

```
# np.mean(standard_fm),np.std(standard_fm)
```

[26]:
```
# converting the normalized features into a tabular format
# feat_cols = ['pagepath '+ str(i) for i in range(standard_fm.shape[1])]
#print(feat_cols)

# df_standard_fm = pd.DataFrame(standard_fm,columns=feat_cols)
# df_standard_fm.head()
```

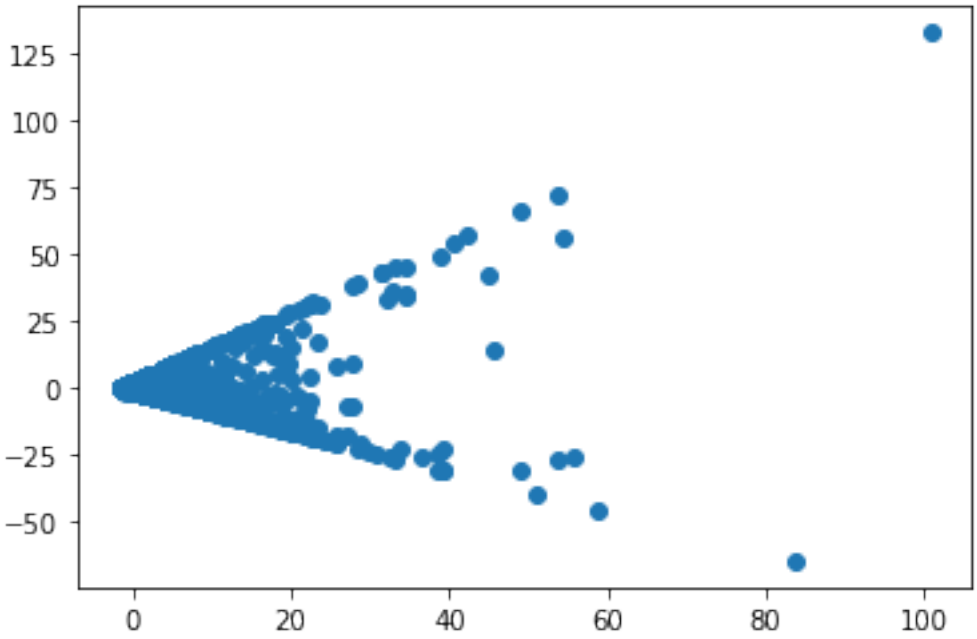
2.3.2 7.3.2 Representation learning

7.3.2.1 PCA

[27]:
```
# https://www.datacamp.com/tutorial/principal-component-analysis-in-python
from sklearn.decomposition import PCA

# pca with 2 components
pca = PCA(n_components=2)
pc = pca.fit_transform(feature_matrix)

# creating a df df_pc containing the principal component values for all 12.561␣
 ↪traces.
df_pc = pd.DataFrame(data = pc, columns = ['pc 1', 'pc 2'])
print(df_pc.tail())


# the amount of information or variance each principal component holds after␣
 ↪projecting the data to a lower dimensional subspace.
print('Explained variation per principal component: {}'.format(pca.
 ↪explained_variance_ratio_))
print(pca.explained_variance_ratio_.sum())

#cumulative explained variances
#print(pca.explained_variance_ratio_.cumsum())

plt.scatter(df_pc["pc 1"],df_pc["pc 2"])
plt.show
```

```
           pc 1      pc 2
12549 -1.434380 -0.066298
12550 -1.476979  0.031520
12551 -0.800084 -0.571434
12552 -1.476185  0.063358
12553 -1.277223 -0.251540
Explained variation per principal component: [0.22267412 0.20157856]
0.4242526832687594
```

[27]:
```
<function matplotlib.pyplot.show(close=None, block=None)>
```



[28]:
```
# pca with 10 components
pca10 = PCA(n_components=10)
pc10 = pca10.fit_transform(feature_matrix)

# creating a df df_pc10 containing the principal component values for all 12.
 ↪561 traces.
df_pc10 = pd.DataFrame(data = pc10)
df_pc10

# the amount of information or variance each principal component holds after␣
 ↪projecting the data to a lower dimensional subspace.
print('Explained variation per principal component: {}'.format(pca10.
 ↪explained_variance_ratio_))
pca10.explained_variance_ratio_.sum()
```

```
Explained variation per principal component: [0.22267412 0.20157856 0.09840811
0.04055463 0.02922279 0.02268426
 0.01850467 0.01296995 0.01145776 0.0113756 ]
```

[28]:
```
0.6694304650205815
```

[29]:
```
# pca with 100 components
pca100 = PCA(n_components=100)
pc100 = pca100.fit_transform(feature_matrix)
```

```
# creating a df df_pc100 containing the principal component values for all 12.
 ↪561 traces.
df_pc100 = pd.DataFrame(data = pc100)
df_pc100

# the amount of information or variance each principal component holds after␣
 ↪projecting the data to a lower dimensional subspace.
print('Explained variation per principal component: {}'.format(pca100.
 ↪explained_variance_ratio_))
pca100.explained_variance_ratio_.sum()
```

```
Explained variation per principal component: [0.22267412 0.20157856 0.09840811
0.04055463 0.02922279 0.02268426
 0.01850468 0.01296996 0.01145782 0.0113757  0.011091   0.00944374
 0.0080907  0.006629   0.00645997 0.00623067 0.00599096 0.00570083
 0.00561711 0.00526752 0.00511739 0.00509126 0.00476088 0.00465444
 0.00449646 0.00435401 0.00414742 0.00406931 0.00404608 0.00395004
 0.00389624 0.00376421 0.0037513  0.00350898 0.00345819 0.00345158
 0.00335246 0.00308578 0.00306869 0.00276487 0.00274186 0.00268963
 0.00266285 0.00257713 0.0025402  0.00240945 0.00235117 0.00233711
 0.00227303 0.00212512 0.0020851  0.00206244 0.00201095 0.00199107
 0.00195363 0.0019237  0.00181147 0.00180142 0.00177015 0.00168955
 0.00168612 0.00162979 0.0016223  0.00159756 0.00154802 0.00154008
 0.00151587 0.00144431 0.00138204 0.00137829 0.001376   0.00137327
 0.00132984 0.00131933 0.00130426 0.00125985 0.00125811 0.00122204
 0.00120055 0.0011921  0.00117294 0.00110607 0.00104607 0.0010382
 0.00101755 0.00100047 0.00099736 0.00098688 0.00095841 0.00095044
 0.0009375  0.00092832 0.00092277 0.00090889 0.00089741 0.00088918
 0.00087181 0.00084687 0.00081976 0.0007818 ]
```

[29]:
```
0.9098071736617885
```

[30]:
```
# selecting the best number of principal components that keeps 99% of the␣
 ↪variance in the original data.
# https://towardsdatascience.com/
 ↪how-to-select-the-best-number-of-principal-components-for-the-dataset-287e64b14c6d

pca99 = PCA(n_components=0.99)
pc99 = pca99.fit_transform(feature_matrix)
print(pca99.n_components_)

# creating a df df_pc99 containing the principal component values for all 12.
 ↪561 traces.
df_pc99 = pd.DataFrame(data = pc99)
df_pc99
```

```
# the amount of information or variance each principal component holds after␣
 ↪projecting the data to a lower dimensional subspace.
print('Explained variation per principal component: {}'.format(pca100.
 ↪explained_variance_ratio_))
pca99.explained_variance_ratio_.sum()
```

```
394
Explained variation per principal component: [0.22267412 0.20157856 0.09840811
0.04055463 0.02922279 0.02268426
 0.01850468 0.01296996 0.01145782 0.0113757  0.011091   0.00944374
 0.0080907  0.006629   0.00645997 0.00623067 0.00599096 0.00570083
 0.00561711 0.00526752 0.00511739 0.00509126 0.00476088 0.00465444
 0.00449646 0.00435401 0.00414742 0.00406931 0.00404608 0.00395004
 0.00389624 0.00376421 0.0037513  0.00350898 0.00345819 0.00345158
 0.00335246 0.00308578 0.00306869 0.00276487 0.00274186 0.00268963
 0.00266285 0.00257713 0.0025402  0.00240945 0.00235117 0.00233711
 0.00227303 0.00212512 0.0020851  0.00206244 0.00201095 0.00199107
 0.00195363 0.0019237  0.00181147 0.00180142 0.00177015 0.00168955
 0.00168612 0.00162979 0.0016223  0.00159756 0.00154802 0.00154008
 0.00151587 0.00144431 0.00138204 0.00137829 0.001376   0.00137327
 0.00132984 0.00131933 0.00130426 0.00125985 0.00125811 0.00122204
 0.00120055 0.0011921  0.00117294 0.00110607 0.00104607 0.0010382
 0.00101755 0.00100047 0.00099736 0.00098688 0.00095841 0.00095044
 0.0009375  0.00092832 0.00092277 0.00090889 0.00089741 0.00088918
 0.00087181 0.00084687 0.00081976 0.0007818 ]
```

[30]:
```
0.9900243053696626
```

[31]:
```
# # Finding important features with the help of PCA.
# dataset_pca = pd.DataFrame(abs(pca.components_), columns=feature_matrix.
 ↪columns, index=['PC_1', 'PC_2'])
# print(dataset_pca)
# print("\nMost important features:")
# print('As per PC 1:\n', (dataset_pca[dataset_pca > 0.01].iloc[0]).dropna())
# print('\n\nAs per PC 2:\n', (dataset_pca[dataset_pca > 0.01].iloc[1]).
 ↪dropna())
```

7.3.2.2 Isomap

[32]:
```
# from sklearn import manifold
# iso = manifold.Isomap(n_components=2)
# iso.fit(feature_matrix, 2)
```

[33]:
```
# manifold_2Da = iso.transform(feature_matrix)
# manifold_2D = pd.DataFrame(manifold_2Da, columns=['Component 1', 'Component␣
 ↪2'])
# manifold_2D.head()
```

[34]:
```
# plt.scatter(manifold_2D['Component 1'], manifold_2D['Component 2'])
# plt.show()
```

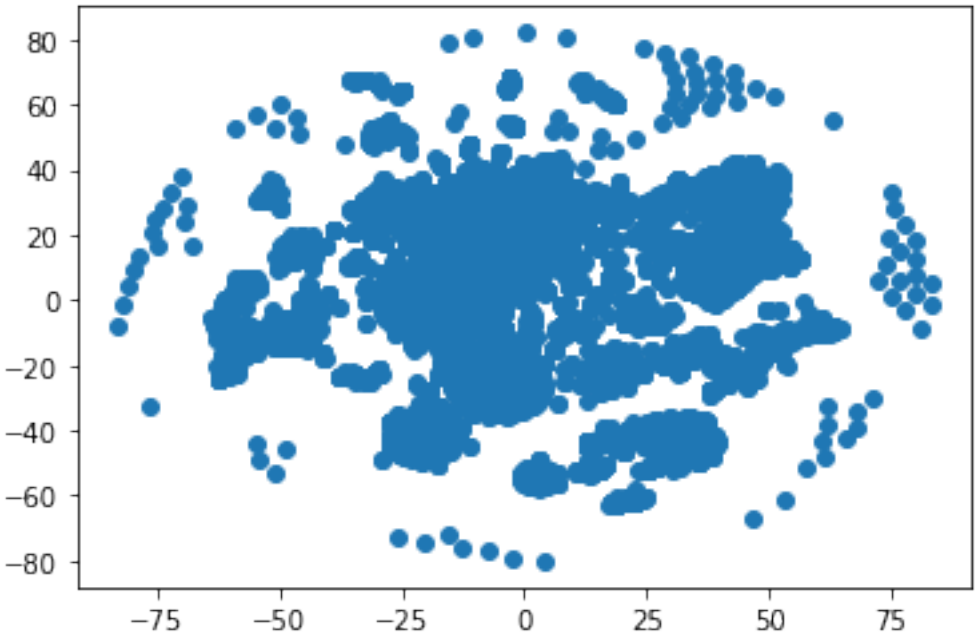
7.3.2.3 t-SNE

[35]:
```
# https://www.reneshbedre.com/blog/tsne.html

# run t-SNE
from sklearn.manifold import TSNE

# perplexity parameter can be changed based on the input datatset
# dataset with larger number of variables requires larger perplexity
# set this value between 5 and 50

tsne_em = TSNE(n_components=2, perplexity=50.0).fit_transform(feature_matrix)
```

```
C:\Users\nvand\AppData\Local\Programs\Python\Python310\lib\site-
packages\sklearn\manifold\_t_sne.py:800: FutureWarning: The default
initialization in TSNE will change from 'random' to 'pca' in 1.2.
  warnings.warn(
C:\Users\nvand\AppData\Local\Programs\Python\Python310\lib\site-
packages\sklearn\manifold\_t_sne.py:810: FutureWarning: The default learning
rate in TSNE will change from 200.0 to 'auto' in 1.2.
  warnings.warn(
```

[36]:
```
df_tsne_em = pd.DataFrame()
df_tsne_em["tsne_1"] = tsne_em[:,0].tolist()
df_tsne_em["tsne_2"] = tsne_em[:,1].tolist()


plt.scatter(data=df_tsne_em, x="tsne_1", y="tsne_2")
```

[36]:
```
<matplotlib.collections.PathCollection at 0x1b7220dec50>
```

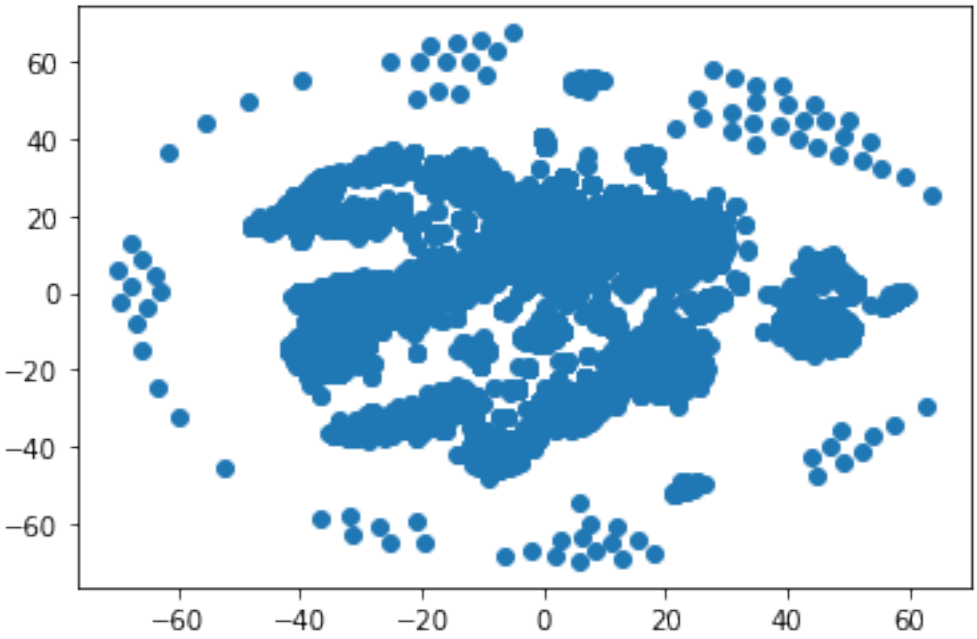


[37]:
```
tsne_em100 = TSNE(n_components=2, perplexity=100.0).
↪fit_transform(feature_matrix)
```

```
C:\Users\nvand\AppData\Local\Programs\Python\Python310\lib\site-
packages\sklearn\manifold\_t_sne.py:800: FutureWarning: The default
initialization in TSNE will change from 'random' to 'pca' in 1.2.
  warnings.warn(
C:\Users\nvand\AppData\Local\Programs\Python\Python310\lib\site-
packages\sklearn\manifold\_t_sne.py:810: FutureWarning: The default learning
rate in TSNE will change from 200.0 to 'auto' in 1.2.
  warnings.warn(
```

[38]:
```
df_tsne_em100 = pd.DataFrame()
df_tsne_em100["tsne_1"] = tsne_em100[:,0].tolist()
df_tsne_em100["tsne_2"] = tsne_em100[:,1].tolist()


plt.scatter(data=df_tsne_em100, x="tsne_1", y="tsne_2")
```

[38]:
```
<matplotlib.collections.PathCollection at 0x1b725bbb880>
```

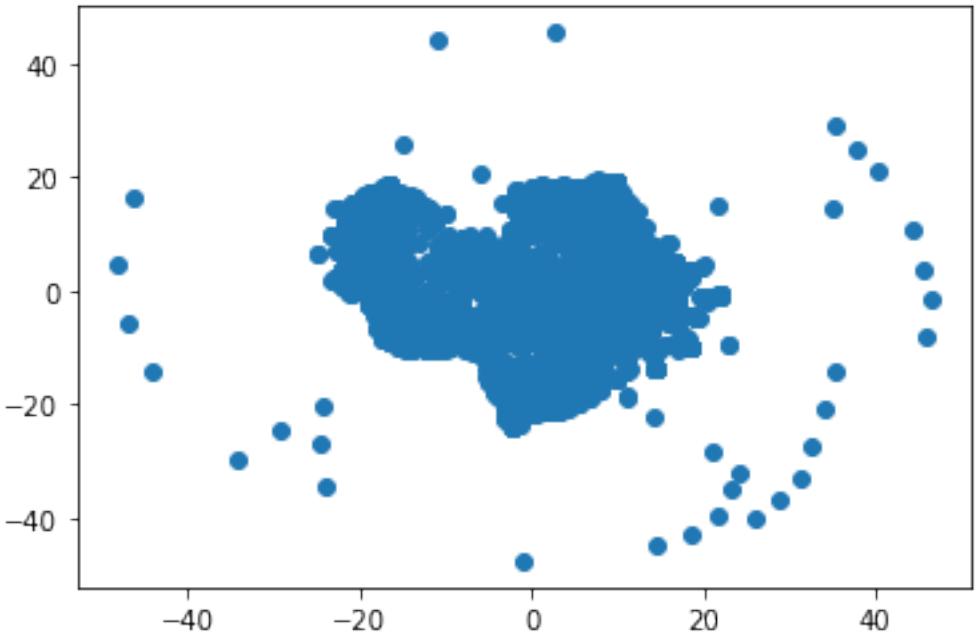


[39]:
```
tsne_em500 = TSNE(n_components=2, perplexity=500.0).
↪fit_transform(feature_matrix)
```

```
C:\Users\nvand\AppData\Local\Programs\Python\Python310\lib\site-
packages\sklearn\manifold\_t_sne.py:800: FutureWarning: The default
initialization in TSNE will change from 'random' to 'pca' in 1.2.
  warnings.warn(
C:\Users\nvand\AppData\Local\Programs\Python\Python310\lib\site-
packages\sklearn\manifold\_t_sne.py:810: FutureWarning: The default learning
rate in TSNE will change from 200.0 to 'auto' in 1.2.
  warnings.warn(
```

[40]:
```
df_tsne_em500 = pd.DataFrame()
df_tsne_em500["tsne_1"] = tsne_em500[:,0].tolist()
df_tsne_em500["tsne_2"] = tsne_em500[:,1].tolist()


plt.scatter(data=df_tsne_em500, x="tsne_1", y="tsne_2")
```

[40]:
```
<matplotlib.collections.PathCollection at 0x1b725bfbbe0>
```



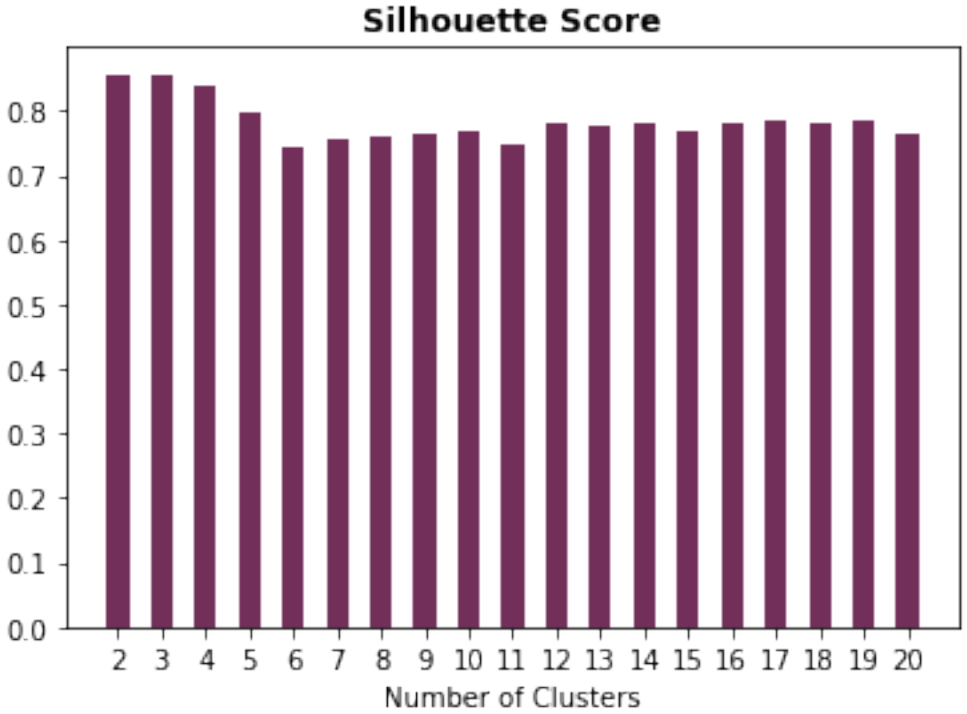
2.3.3 7.3.3 Clustering

7.3.3.1 K-means clustering

[41]:
```
# https://towardsdatascience.com/
 ↪machine-learning-algorithms-part-9-k-means-example-in-python-f2ad05ed5203
# https://medium.com/swlh/
 ↪k-means-clustering-on-high-dimensional-data-d2151e1a4240#:~:
 ↪text=KMeans%20is%20one%20of%20the,too%20much%20into%20mathematical%20details.

# determining the optimal number of clusters using the elbow method
# from sklearn.cluster import KMeans

# wcss = []
# K = range(2, 21)
# for k in K:
#     kmeans = KMeans(n_clusters=k, init='k-means++', max_iter=300, n_init=10,␣
 ↪random_state=0)
#     kmeans.fit(df_pc)
#     wcss.append(kmeans.inertia_)
# plt.plot(K, wcss, 'bx-')
# plt.title('Elbow Method')
# plt.xlabel('Number of clusters')
# plt.ylabel('WCSS')
# plt.show()
```

[42]:
```
# Determining optimal number of clusters using the silhouette score for dataset␣
↪'dataset'

dataset = df_pc

from sklearn.model_selection import ParameterGrid
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_samples, silhouette_score

# candidate values for our number of cluster
parameters = [2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19,␣
↪20]

# instantiating ParameterGrid, pass number of clusters as input
parameter_grid = ParameterGrid({'n_clusters': parameters})
best_score = -1
kmeans_model = KMeans()     # instantiating KMeans model
silhouette_scores = []
# evaluation based on silhouette_score
for p in parameter_grid:
    kmeans_model.set_params(**p)    # set current hyper parameter
    kmeans_model.fit(dataset)          # fit model on wine dataset, this will␣
↪find clusters based on parameter p
    ss = silhouette_score(dataset, kmeans_model.labels_)   # calculate␣
↪silhouette_score
    silhouette_scores += [ss]       # store all the scores
    print('Parameter:', p, 'Score', ss)
    # check p which has the best score
    if ss > best_score:
        best_score = ss
        best_grid = p
# plotting silhouette score
plt.bar(range(len(silhouette_scores)), list(silhouette_scores), align='center',␣
↪color='#722f59', width=0.5)
plt.xticks(range(len(silhouette_scores)), list(parameters))
plt.title('Silhouette Score', fontweight='bold')
plt.xlabel('Number of Clusters')
plt.show()

# for df_pc, K = 2, Silhouette Score = 0.8563948096312987.
# for df_pc10, K = 2, Silhouette Score = 0.7369992102662368.
# for df_pc100, K = 2, Silhouette Score = 0.6885837292032151.
# for df_pc99, K = 2, Silhouette Score = 0.6658501053962945.
# for df_tsne_em, K = 23, Silhouette Score = 0.5136529116814816.
# for df_tsne_em100, K = 25, Silhouette Score = 0.5409828439197543.
# for df_tsne_em500, K = 8, Silhouette Score = 0.5479944861545435.
```

```
Parameter: {'n_clusters': 2} Score 0.8566672101656615
Parameter: {'n_clusters': 3} Score 0.8534336508764172
Parameter: {'n_clusters': 4} Score 0.837160014502137
Parameter: {'n_clusters': 5} Score 0.7960220378339996
Parameter: {'n_clusters': 6} Score 0.7458103880383992
Parameter: {'n_clusters': 7} Score 0.7569513406768692
Parameter: {'n_clusters': 8} Score 0.7583696407605144
Parameter: {'n_clusters': 9} Score 0.7625789127995425
Parameter: {'n_clusters': 10} Score 0.7686873011284251
Parameter: {'n_clusters': 11} Score 0.745934549941781
Parameter: {'n_clusters': 12} Score 0.7814050297327375
Parameter: {'n_clusters': 13} Score 0.7777521466270835
Parameter: {'n_clusters': 14} Score 0.7823213768259614
Parameter: {'n_clusters': 15} Score 0.7704844536952422
Parameter: {'n_clusters': 16} Score 0.7819601610182801
Parameter: {'n_clusters': 17} Score 0.7839496127723029
Parameter: {'n_clusters': 18} Score 0.7829431320768562
Parameter: {'n_clusters': 19} Score 0.7846170221262991
Parameter: {'n_clusters': 20} Score 0.7646230038542957
```



[43]:
```
# # Determining optimal number of clusters using the Davies-Bouldin Index.

# results = {}
# for i in range(2,21):
```

```
#     kmeans = KMeans(n_clusters=i, random_state=30)
#     labels = kmeans.fit_predict(df_pc)
#     db_index = sklearn.metrics.davies_bouldin_score(df_pc, labels)
#     results.update({i: db_index})

# plt.plot(list(results.keys()), list(results.values()))
# plt.xlabel("Number of clusters")
# plt.ylabel("Davies-Boulding Index")
# plt.show()

# print(results)
# print(min(results.values())) # corresponds to 14.
```

K-Means Clustering K=k

[196]:
```
# fitting KMeans (for K=K and dataset=X)
K = 8
X = df_tsne_em500

kmeans = KMeans(n_clusters=K)
kmeans.fit(X)
labels = kmeans.fit_predict(X)

# print(kmeans.inertia_)

from collections import Counter
print(Counter(kmeans.labels_))

# Final locations of the centroid
# print(kmeans.cluster_centers_)
```

```
Counter({0: 2165, 4: 2152, 7: 2053, 2: 1681, 1: 1389, 3: 1224, 5: 1145, 6: 745})
```

[197]:
```
# Creating a dataframe with per fullVisitorId the corresponding cluster label.
df_labels = pd.DataFrame(kmeans.labels_)
df_labels.columns = ["label"]
df_labels.index = feature_matrix.index
print(df_labels)
```

```
                   label
fullVisitorId
1.00011e+18            3
1.00016e+18            2
1.00068e+18            7
1.00092e+18            1
1.00218e+18            7
…                      …
9.93636e+17            3
```

```
9.94661e+17             1
9.95298e+17             0
9.99493e+16             7
9957690000000000.0      1

[12554 rows x 1 columns]
```

[198]:
```
# Creating a dataframe with the fullVisitorIds with cluster label 0.
label0 = df_labels[df_labels['label'] == 0]
label0
```

[198]:
```
                label
fullVisitorId
1.00371e+18         0
1.01751e+18         0
1.01962e+17         0
1.02425e+18         0
1.02795e+17         0
…                   …
9.78597e+17         0
9.80774e+17         0
9.83326e+17         0
9.9344e+17          0
9.95298e+17         0

[2165 rows x 1 columns]
```

[47]:
```
# Creating a dataframe with the fullVisitorIds with cluster label 1.
label1=df_labels[df_labels['label'] == 1]
label1
```

[47]:
```
                label
fullVisitorId
1.0131e+18          1
1.02776e+18         1
1.07435e+17         1
1.07671e+18         1
1.11558e+18         1
…                   …
9.14516e+18         1
9.15066e+18         1
9.15981e+18         1
9.18126e+18         1
9.62326e+17         1

[496 rows x 1 columns]
```

[199]:
```
# Creating a list with all fullVisitorIds with cluster label 0.
label0visitors = np.float64(label0.index.values.tolist())
label0visitors
```

[199]:
```
array([1.00371e+18, 1.01751e+18, 1.01962e+17, …, 9.83326e+17,
       9.93440e+17, 9.95298e+17])
```

[49]:
```
# Creating a list with all fullVisitorIds with cluster label 1.
label1visitors = np.float64(label1.index.values.tolist())
label1visitors
```

[49]:
```
array([1.01310e+18, 1.02776e+18, 1.07435e+17, 1.07671e+18, 1.11558e+18,
       1.13842e+18, 1.14310e+18, 1.15161e+18, 1.15787e+18, 1.18840e+18,
       1.19361e+18, 1.20807e+18, 1.21919e+18, 1.22378e+18, 1.23113e+18,
       1.24785e+18, 1.25349e+18, 1.28936e+18, 1.29446e+18, 1.36357e+18,
       1.36366e+18, 1.37318e+18, 1.37946e+18, 1.38072e+18, 1.42988e+18,
       1.43201e+18, 1.43235e+18, 1.44251e+18, 1.45570e+18, 1.47778e+18,
       1.47824e+18, 1.49177e+18, 1.49585e+18, 1.50970e+18, 1.53737e+18,
       1.54246e+18, 1.54971e+18, 1.56325e+18, 1.60240e+18, 1.60747e+17,
       1.62832e+18, 1.63274e+18, 1.65773e+18, 1.66215e+18, 1.68322e+18,
       1.69385e+18, 1.76195e+18, 1.76757e+18, 1.78784e+18, 1.78903e+18,
       1.79026e+18, 1.79388e+18, 1.81943e+18, 1.83133e+18, 1.84968e+18,
       1.86209e+18, 1.87844e+18, 1.87871e+17, 1.91812e+18, 1.93814e+17,
       1.93951e+18, 1.98331e+18, 2.02842e+18, 2.03310e+18, 2.04363e+18,
       2.05913e+18, 2.08719e+18, 2.09089e+18, 2.09209e+16, 2.10696e+18,
       2.10735e+18, 2.10865e+18, 2.12673e+17, 2.17862e+18, 2.18261e+18,
       2.20060e+18, 2.24125e+18, 2.27771e+17, 2.29128e+18, 2.31281e+18,
       2.31511e+18, 2.33859e+18, 2.34292e+18, 2.35370e+18, 2.36001e+18,
       2.40206e+18, 2.41487e+18, 2.43565e+18, 2.43988e+18, 2.44836e+17,
       2.45586e+18, 2.48561e+18, 2.51062e+18, 2.52525e+18, 2.54187e+18,
       2.54468e+18, 2.54602e+18, 2.54608e+18, 2.56399e+17, 2.57893e+18,
       2.58709e+18, 2.60773e+18, 2.60837e+18, 2.61533e+18, 2.61699e+18,
       2.61798e+18, 2.62211e+18, 2.65297e+18, 2.65441e+18, 2.68466e+17,
       2.69001e+18, 2.71324e+18, 2.78409e+18, 2.78915e+18, 2.81927e+18,
       2.81975e+18, 2.83304e+18, 2.83805e+18, 2.86510e+18, 2.86850e+18,
       2.88624e+18, 2.88637e+17, 2.89540e+18, 2.94441e+18, 2.97196e+18,
       2.97258e+18, 3.01839e+18, 3.02041e+18, 3.02186e+18, 3.06161e+18,
       3.07897e+18, 3.09672e+18, 3.13548e+17, 3.13996e+18, 3.14958e+18,
       3.16897e+18, 3.17149e+18, 3.18181e+18, 3.19384e+18, 3.22407e+18,
       3.22479e+18, 3.25444e+18, 3.26512e+18, 3.27203e+18, 3.29360e+18,
       3.31119e+18, 3.39267e+18, 3.42348e+18, 3.43694e+18, 3.45657e+18,
       3.47676e+18, 3.49790e+18, 3.50014e+18, 3.51496e+18, 3.51633e+18,
       3.52708e+18, 3.53902e+18, 3.53936e+18, 3.55124e+18, 3.56000e+18,
       3.57889e+18, 3.60825e+17, 3.65806e+18, 3.68642e+18, 3.69381e+18,
       3.71843e+18, 3.72441e+17, 3.75119e+18, 3.79601e+18, 3.84898e+18,
       3.85182e+18, 3.86116e+17, 3.86579e+18, 3.87739e+18, 3.90080e+18,
       3.95880e+17, 3.95895e+17, 3.96312e+17, 3.96971e+18, 3.98309e+18,
```

3.99776e+17, 4.00460e+18, 4.02099e+18, 4.04591e+18, 4.04958e+18,
4.07727e+18, 4.07827e+18, 4.08110e+18, 4.09142e+16, 4.09531e+18,
4.11793e+17, 4.12024e+18, 4.13128e+18, 4.17010e+18, 4.17468e+18,
4.17777e+18, 4.20680e+18, 4.21614e+18, 4.25856e+18, 4.26911e+18,
4.33256e+18, 4.33258e+18, 4.33579e+18, 4.34805e+18, 4.36676e+18,
4.40304e+17, 4.42262e+18, 4.45393e+18, 4.47324e+18, 4.48206e+18,
4.48576e+18, 4.51302e+18, 4.52202e+18, 4.52472e+18, 4.53456e+18,
4.54840e+17, 4.55898e+18, 4.58397e+18, 4.66681e+18, 4.70030e+18,
4.72440e+18, 4.75794e+18, 4.77747e+18, 4.80991e+18, 4.82133e+18,
4.85960e+18, 4.86999e+18, 4.87203e+18, 4.88215e+18, 4.88348e+17,
4.88824e+17, 4.89057e+18, 4.89780e+18, 4.91007e+18, 4.93793e+18,
4.94193e+18, 4.94769e+18, 4.95934e+18, 4.99087e+17, 4.99876e+18,
5.06439e+18, 5.06634e+18, 5.06840e+18, 5.06998e+18, 5.08550e+18,
5.09409e+18, 5.09461e+18, 5.11333e+18, 5.11745e+18, 5.17627e+18,
5.18246e+18, 5.18281e+18, 5.20421e+18, 5.22641e+18, 5.22737e+18,
5.23320e+18, 5.23395e+17, 5.24449e+18, 5.25467e+18, 5.25967e+18,
5.30979e+18, 5.34932e+18, 5.35782e+18, 5.36240e+18, 5.37849e+18,
5.39791e+18, 5.43340e+18, 5.43443e+18, 5.47513e+18, 5.47759e+18,
5.48280e+18, 5.49269e+17, 5.50465e+18, 5.53820e+18, 5.55025e+18,
5.59667e+18, 5.63228e+18, 5.63302e+18, 5.63969e+18, 5.64057e+18,
5.64325e+18, 5.67082e+18, 5.68835e+18, 5.70477e+18, 5.71344e+18,
5.71903e+18, 5.72761e+18, 5.72966e+17, 5.74431e+18, 5.74958e+18,
5.78908e+18, 5.78987e+18, 5.79204e+18, 5.80639e+18, 5.81031e+18,
5.81096e+18, 5.82387e+18, 5.83789e+18, 5.84162e+18, 5.86537e+18,
5.87732e+18, 5.88008e+18, 5.91070e+17, 5.92186e+18, 5.95700e+18,
5.96894e+17, 5.97517e+18, 6.00902e+18, 6.01008e+18, 6.02881e+18,
6.06176e+18, 6.06556e+18, 6.10845e+18, 6.10983e+18, 6.11283e+17,
6.11715e+18, 6.15318e+18, 6.16066e+18, 6.20583e+18, 6.20600e+18,
6.21587e+18, 6.22265e+18, 6.22809e+18, 6.23241e+17, 6.23830e+18,
6.27451e+18, 6.27550e+18, 6.28343e+18, 6.30140e+18, 6.31659e+18,
6.34310e+18, 6.35730e+18, 6.36524e+18, 6.37617e+17, 6.40733e+17,
6.41926e+18, 6.42411e+17, 6.42923e+18, 6.43261e+18, 6.44255e+18,
6.47288e+18, 6.48081e+18, 6.55818e+18, 6.59383e+18, 6.61473e+18,
6.61761e+18, 6.62859e+18, 6.63815e+17, 6.65052e+18, 6.66309e+18,
6.67050e+18, 6.68221e+18, 6.71640e+18, 6.78266e+18, 6.82381e+18,
6.82791e+18, 6.83002e+18, 6.87853e+18, 6.95009e+18, 6.95862e+18,
6.96227e+17, 6.96449e+18, 6.96768e+18, 6.99063e+17, 7.04813e+18,
7.07632e+18, 7.08601e+18, 7.16222e+18, 7.18425e+18, 7.21222e+17,
7.23769e+18, 7.24725e+18, 7.25277e+18, 7.29166e+17, 7.29550e+18,
7.32464e+18, 7.34552e+18, 7.35876e+18, 7.38563e+18, 7.42551e+17,
7.42936e+18, 7.45694e+18, 7.47610e+18, 7.47799e+18, 7.48796e+18,
7.52393e+18, 7.53252e+18, 7.54058e+18, 7.56809e+18, 7.57715e+18,
7.60545e+18, 7.63731e+18, 7.64940e+18, 7.68361e+18, 7.68728e+18,
7.70187e+18, 7.70700e+18, 7.71110e+18, 7.71301e+18, 7.71321e+18,
7.74587e+18, 7.76982e+18, 7.79125e+17, 7.79623e+18, 7.81353e+18,
7.82842e+18, 7.83901e+18, 7.84818e+18, 7.85598e+18, 7.86701e+18,
7.87325e+18, 7.87444e+18, 7.91657e+18, 7.92023e+18, 7.92249e+18,

```
       7.94784e+18, 7.95216e+18, 7.95955e+18, 7.96221e+18, 7.97116e+18,
       7.97782e+18, 8.03327e+18, 8.04840e+18, 8.08780e+18, 8.08927e+18,
       8.09558e+18, 8.10268e+18, 8.11039e+18, 8.11405e+18, 8.12862e+18,
       8.13318e+17, 8.16229e+18, 8.18687e+18, 8.21762e+18, 8.22121e+18,
       8.22997e+18, 8.23063e+18, 8.29048e+18, 8.35077e+18, 8.35111e+18,
       8.38815e+18, 8.40459e+18, 8.44437e+18, 8.45094e+18, 8.45510e+18,
       8.49173e+18, 8.49584e+18, 8.50400e+18, 8.51114e+18, 8.52836e+18,
       8.54869e+18, 8.55430e+18, 8.56359e+18, 8.57361e+17, 8.59467e+18,
       8.60547e+18, 8.62896e+18, 8.65383e+18, 8.67020e+18, 8.68281e+18,
       8.71242e+18, 8.74676e+18, 8.74854e+18, 8.75345e+18, 8.76039e+18,
       8.78600e+18, 8.79345e+17, 8.79632e+18, 8.81724e+18, 8.83307e+18,
       8.83583e+17, 8.84283e+18, 8.85118e+18, 8.85676e+18, 8.86587e+17,
       8.86865e+17, 8.89192e+18, 8.89814e+18, 8.90999e+17, 8.95126e+18,
       8.95848e+18, 8.96367e+18, 8.97672e+18, 8.99720e+18, 9.02212e+18,
       9.02296e+17, 9.02747e+18, 9.03093e+18, 9.09970e+18, 9.10877e+18,
       9.10989e+18, 9.14516e+18, 9.15066e+18, 9.15981e+18, 9.18126e+18,
       9.62326e+17])
```

[200]:
```
# Creating a dataframe with all records belonging to cluster 0.
cluster0 = subset2[subset2['fullVisitorId'].isin(label0visitors)]
cluster0
```

[200]:

	visitNumber	visitId	visitStartTime	date	visits	hits	\
196	2	1501618052	1501618052	20170801	1.0	9	
197	2	1501618052	1501618052	20170801	1.0	9	
198	2	1501618052	1501618052	20170801	1.0	9	
199	2	1501618052	1501618052	20170801	1.0	9	
200	2	1501618052	1501618052	20170801	1.0	9	
…	…	…	…	…	…	…	
586518	12	1501739991	1501739991	20170803	1.0	8	
586519	12	1501739991	1501739991	20170803	1.0	8	
586520	12	1501739991	1501739991	20170803	1.0	8	
588216	7	1501753596	1501753596	20170803	1.0	2	
588217	7	1501753596	1501753596	20170803	1.0	2	

	pageviews	timeOnSite	bounces	browser	…	fullVisitorId	\
196	7.0	78.0	NaN	Safari (in-app)	…	8.061080e+18	
197	7.0	78.0	NaN	Safari (in-app)	…	8.061080e+18	
198	7.0	78.0	NaN	Safari (in-app)	…	8.061080e+18	
199	7.0	78.0	NaN	Safari (in-app)	…	8.061080e+18	
200	7.0	78.0	NaN	Safari (in-app)	…	8.061080e+18	
…	…	…	…	…	…	…	
586518	2.0	5.0	NaN	Safari	…	7.408570e+18	
586519	2.0	5.0	NaN	Safari	…	7.408570e+18	
586520	2.0	5.0	NaN	Safari	…	7.408570e+18	
588216	2.0	1554.0	NaN	Firefox	…	1.564760e+17	
588217	2.0	1554.0	NaN	Firefox	…	1.564760e+17	

```
       channelGrouping  hitNumber     time  hour minute  pagePath  \
196          Affiliate          1        0    21     07        38
197          Affiliate          2    12384    21     07        38
198          Affiliate          3    17282    21     07        39
199          Affiliate          4    24497    21     07        39
200          Affiliate          5    35947    21     08        82
…                  …        …       …    …    …       …
586518           Email          6     5343    06     59        39
586519           Email          7     5408    06     59        39
586520           Email          8    14639    07     00        39
588216            Base          1        0    10     46        39
588217            Base          2  1553759    11     12        39

        screenDepth     sessionId       timeStamp
196               0  8.061080e+18  20170801 21:07
197               0  8.061080e+18  20170801 21:07
198               0  8.061080e+18  20170801 21:07
199               0  8.061080e+18  20170801 21:07
200               0  8.061080e+18  20170801 21:08
…               …           …             …
586518            0  7.408570e+18  20170803 06:59
586519            0  7.408570e+18  20170803 06:59
586520            0  7.408570e+18  20170803 07:00
588216            0  1.564760e+17  20170803 10:46
588217            0  1.564760e+17  20170803 11:12

[36519 rows x 24 columns]
```

[51]:
```
# Creating a dataframe with all records belonging to cluster 1.
cluster1 = subset2[subset2['fullVisitorId'].isin(label1visitors)]
cluster1
```

[51]:
```
        visitNumber     visitId  visitStartTime      date  visits  hits  \
37                3  1501570206      1501570206  20170801     1.0     4
38                3  1501570206      1501570206  20170801     1.0     4
39                3  1501570206      1501570206  20170801     1.0     4
40                3  1501570206      1501570206  20170801     1.0     4
104               4  1501569996      1501569996  20170801     1.0     7
…               …           …               …         …     …   …
588541            2  1501778845      1501778845  20170803     1.0     2
588644           33  1501791756      1501791756  20170803     1.0     2
588645           33  1501791756      1501791756  20170803     1.0     2
588700            2  1501725694      1501725694  20170803     1.0     2
588701            2  1501725694      1501725694  20170803     1.0     2

        pageviews  timeOnSite  bounces  browser  … fullVisitorId  \
```

```
37            1.0       110.0      1.0   Chrome  …  8.868650e+17
38            1.0       110.0      1.0   Chrome  …  8.868650e+17
39            1.0       110.0      1.0   Chrome  …  8.868650e+17
40            1.0       110.0      1.0   Chrome  …  8.868650e+17
104           2.0        77.0      NaN   Chrome  …  1.939510e+18
…             …          …          …     …  …           …
588541        1.0         NaN      1.0   Chrome  …  2.108650e+18
588644        1.0         NaN      1.0  Firefox  …  1.790260e+18
588645        1.0         NaN      1.0  Firefox  …  1.790260e+18
588700        1.0         NaN      1.0   Chrome  …  8.958480e+18
588701        1.0         NaN      1.0   Chrome  …  8.958480e+18

       channelGrouping  hitNumber    time  hour minute  pagePath  screenDepth  \
37             Organic          1       0    07     50        24            0
38             Organic          2    1203    07     50        24            0
39             Organic          3  109837    07     51        25            0
40             Organic          4  185303    07     53        25            0
104              Email          1       0    07     46        52            0
…                …        …      …    …    …      …         …
588541            Base          2       1    17     47        25            0
588644           Email          1       0    21     22        25            0
588645           Email          2      13    21     22        25            0
588700       Affiliate          1       0    03     01        25            0
588701       Affiliate          2       1    03     01        25            0

           sessionId       timeStamp
37      8.868650e+17  20170801 07:50
38      8.868650e+17  20170801 07:50
39      8.868650e+17  20170801 07:51
40      8.868650e+17  20170801 07:53
104     1.939510e+18  20170801 07:46
…                …             …
588541  2.108650e+18  20170803 17:47
588644  1.790260e+18  20170803 21:22
588645  1.790260e+18  20170803 21:22
588700  8.958480e+18  20170803 03:01
588701  8.958480e+18  20170803 03:01

[2704 rows x 24 columns]
```

[201]:
```
# Exporting cluster 0.
cluster0.to_csv('cluster0_tsne500_k8.csv')
```

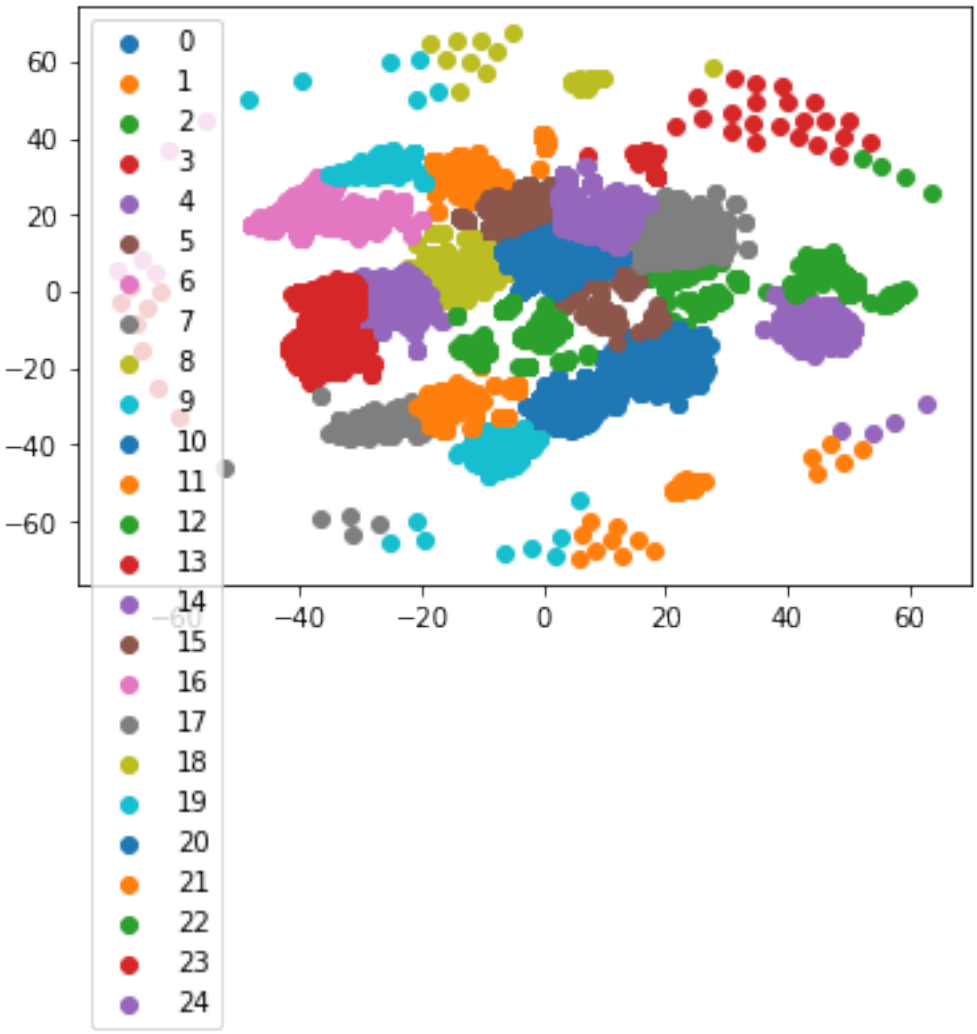
[53]:
```
# cluster0.tail()
```

[53]:
```
        visitNumber     visitId  visitStartTime      date  visits  hits  \
570830           45  1501726873      1501726873  20170803     1.0     6
```

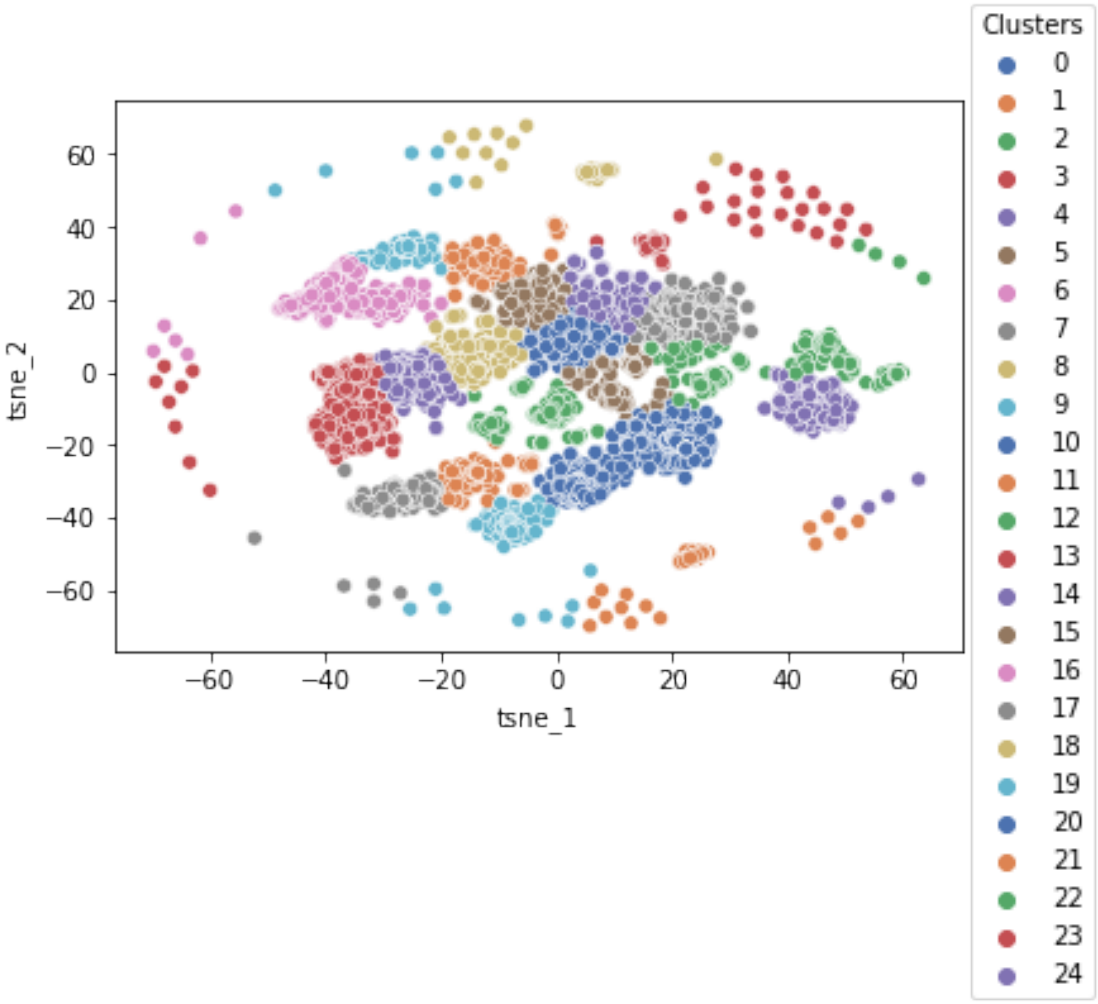
```
572045           73  1501797296      1501797296  20170803     1.0     7
572046           73  1501797296      1501797296  20170803     1.0     7
572047           73  1501797296      1501797296  20170803     1.0     7
572048           73  1501797296      1501797296  20170803     1.0     7

        pageviews  timeOnSite  bounces browser  … fullVisitorId  \
570830        5.0        79.0      NaN  Safari  …  7.822760e+18
572045        3.0        53.0      NaN  Chrome  …  3.154540e+18
572046        3.0        53.0      NaN  Chrome  …  3.154540e+18
572047        3.0        53.0      NaN  Chrome  …  3.154540e+18
572048        3.0        53.0      NaN  Chrome  …  3.154540e+18

       channelGrouping  hitNumber   time  hour minute  pagePath  screenDepth  \
570830           Email          2   6604    03     21         7            0
572045     Remarketing          1      0    22     54         7            0
572046     Remarketing          2   6792    22     55         7            0
572047     Remarketing          3   8981    22     55         7            0
572048     Remarketing          4  11359    22     55         7            0

           sessionId       timeStamp
570830  7.822760e+18  20170803 03:21
572045  3.154540e+18  20170803 22:54
572046  3.154540e+18  20170803 22:55
572047  3.154540e+18  20170803 22:55
572048  3.154540e+18  20170803 22:55

[5 rows x 24 columns]
```

[54]:
```
# Exporting cluster 1.
# cluster1.to_csv('cluster1.csv')
```

[55]:
```
#Visualizing clusters
u_labels = np.unique(kmeans.labels_)

for i in u_labels:
    df_filtered_label = X[labels == i]
#     plt.scatter(df_filtered_label["pc 1"] , df_filtered_label["pc 2"] , label␣
 ↪= i)
    plt.scatter(df_filtered_label["tsne_1"] , df_filtered_label["tsne_2"] ,␣
 ↪label = i)
plt.legend()
plt.show()
```



[56]:
```
#Visualizing clusters (second method)
# Dont forget to change x,y to pc 1, pc 2 or tsne_1, tsne_2
p = sns.scatterplot(data=X, x="tsne_1", y="tsne_2", hue=kmeans.labels_,␣
 ↪legend="full", palette="deep")
sns.move_legend(p, "upper right", bbox_to_anchor=(1.17, 1.2), title='Clusters')
plt.show()
```

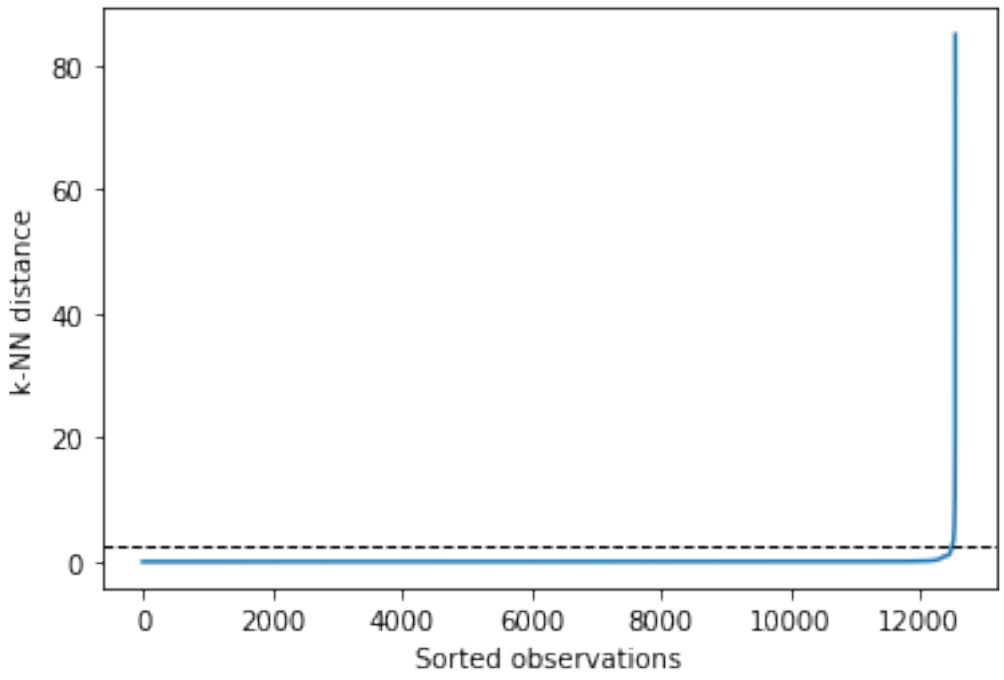


2.3.4 7.3.3.2 Density Based Spatial Clustering of Applications with Noise (DBSCAN)

[188]:
```
# https://www.reneshbedre.com/blog/dbscan-python.html

# DBSCAN on the following dataset
dataset = df_pc

epsilon = 1 # for pc
epsilon = 2 # for tsne500
epsilon = 2.5 # for tsne50, tsne100

minPts = dataset.shape[1]
```

[189]:
```
# Computing k-nearest neighbor (kNN) distances to determine the optimal
↪parameter.
```

```
from sklearn.neighbors import NearestNeighbors

# n_neighbors can be same as the minPts value.
# n_neighbors = 5 as kneighbors function returns distance of point to itself (i.
 ↪e. first column will be zeros)
nbrs = NearestNeighbors(n_neighbors=minPts + 1).fit(dataset)
# Find the k-neighbors of a point
neigh_dist, neigh_ind = nbrs.kneighbors(dataset)
# sort the neighbor distances (lengths to points) in ascending order
# axis = 0 represents sort along first axis i.e. sort along row
sort_neigh_dist = np.sort(neigh_dist, axis=0)

# plot the kNN distance plot
k_dist = sort_neigh_dist[:, minPts]
plt.plot(k_dist)
plt.axhline(y=2.5, linewidth=1, linestyle='dashed', color='k')
plt.ylabel("k-NN distance")
plt.xlabel("Sorted observations")
plt.show()
```



[190]:
```
# Compute DBSCAN clustering
from sklearn.cluster import DBSCAN
clusters = DBSCAN(eps=epsilon, min_samples=minPts).fit(dataset)
# get cluster labels
```

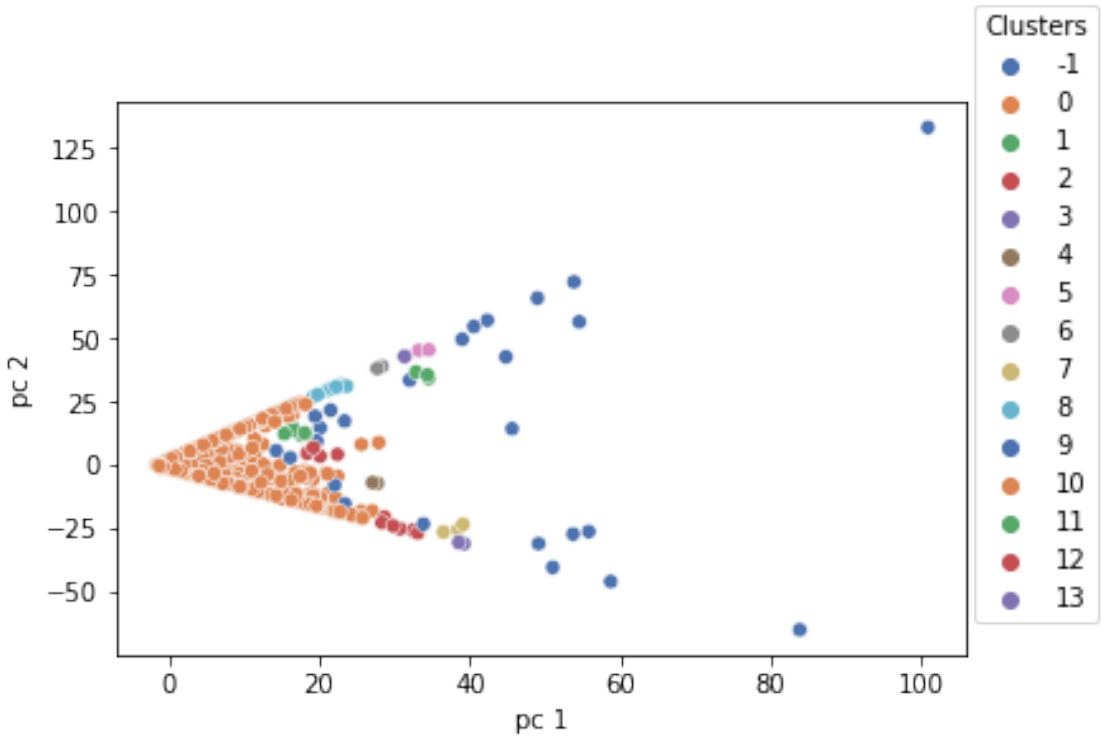
```
clusters.labels_


# check unique clusters
#set(clusters.labels_)
#{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21,␣
 ↪22, -1}
# -1 value represents noisy points could not assigned to any cluster
```

[190]:
```
array([0, 0, 0, …, 0, 0, 0], dtype=int64)
```

[191]:
```
from collections import Counter
Counter(clusters.labels_)
```

[191]:
```
Counter({0: 12484,
         1: 6,
         2: 6,
         -1: 24,
         3: 3,
         4: 2,
         5: 2,
         6: 2,
         7: 3,
         8: 9,
         9: 2,
         10: 2,
         11: 3,
         12: 4,
         13: 2})
```

[192]:
```
# Dont forget to change x,y to pc 1, pc 2 or tsne_1, tsne_2
# p = sns.scatterplot(data=dataset, x="tsne_1", y="tsne_2", hue=clusters.
 ↪labels_, legend="full", palette="deep")
p = sns.scatterplot(data=dataset, x="pc 1", y="pc 2", hue=clusters.labels_,␣
 ↪legend="full", palette="deep")
sns.move_legend(p, "upper right", bbox_to_anchor=(1.17, 1.2), title='Clusters')
plt.show()
```



2.4 7.3.4 Cluster Evaluation

2.4.1 Davies-Bouldin Index

[186]:
```
from sklearn.metrics import davies_bouldin_score
db_index = davies_bouldin_score(df_pc99, clusters.labels_)
print(db_index)

# pca2, dbscan: 1.5747449086142744
# pca10, dbscan: 1.4072284717615533
# pca100, dbscan: 3.0508102138926687
# pca99, dbscan: 3.8904035902213994
# tsne50, dbscan: 3.2500363391579787
# tsne100, dbscan: 2.8097855293065144
# tsne500, dbscan: 4.528563523243719
```

```
3.8904035902213994
```

2.4.2 Silhouette Score

[187]:
```
silhouette_score(df_pc99, clusters.labels_)

# pca2, dbscan: 0.8336750710559367
```

```
# pca10, dbscan: 0.6969762685620593
# pca100, dbscan: 0.23830295925779962
# pca99, dbscan: 0.34077493718556556
# tsne50, dbscan: -0.0980181313325531
# tsne100, dbscan: -0.2803119726664
# tsne500, dbscan: -0.339226608726153
```

[187]: 0.34077493718556556

2.4.3 Adjusted Rand Index

[ ]:
```
# #ARI between (PCA) k means K=3 and DBSCAN
# kmeans3labels=kmeans3.labels_
# dbscanlabels=clusters.labels_
# sklearn.metrics.adjusted_rand_score(kmeans3labels, dbscanlabels)
# #0.44051622524979533
```