NOVA IMS Information Management School

MGI

Mestrado em Gestão de Informação
Master Program in Information Management

ATRIO - ATTRIBUTION MODEL ORCHESTRATOR

Author: Bernardo Duarte Siré de Magalhães Mexia Alves

Project Report, as partial requirement to obtain the Information Management Master's Degree with specialization in Information Systems and Technologies Management.

Supervision: Prof. Roberto Henriques

Prof. André Melo

NOVA Information Management School
Instituto Superior de Estatística e Gestão de Informação
Universidade Nova de Lisboa

NOVA Information Management School
Instituto Superior de Estatística e Gestão de Informação
Universidade Nova de Lisboa

ATRIO – ATTRIBUTION MODEL ORCHESTRATOR

by

Bernardo Duarte Siré de Magalhães Mexia Alves

Project Report, as partial requirement to obtain the Information Management Master's Degree with specialization in Information Systems and Technologies Management.

Supervisor: Prof. Roberto Henriques

Co-Supervisor: Prof. André Melo

July 2021

ACKNOWLEDGEMENTS

GroupM provided the challenge, the data, and the means to make this project. Thanks to José Pedro Dias Pinheiro for the trust, to António Duarte for many years of mentorship and to Professor André Melo for the guidance and encouragement to enroll into this Master Course and develop a project to make Attribution Modelling accessible.

Nova IMS provided the academic foundation to deliver this project within the highest standards. Thanks to Professor Roberto Henriques for accepting the supervision of this project and for the valuable lessons on academic research.

Last but not least, families are always part of these projects. Even if children or spouses know little about the subjects we are writing about, they do listen to the ramblings, bear the occasional frustrations, share the eureka moments and most important, they waive their share of the family time to make this possible. Thank you.

ABSTRACT

In Digital Advertising, Attribution Modelling is used to assess the contribution of media touchpoints to the campaign outcome, by analyzing each person's sequence of contacts and interactions with these touchpoints, designated as the Consumer Journey. The ability to acquire, model and analyze campaign data to derive meaningful insights, usually involves proprietary tools, provided by campaign delivery platforms. ATRIO is proposed as an open-sourced framework for Attribution Modelling, orchestrating the data pipeline through transformation, integration, and delivery, to provide Attribution Modelling capabilities for digital media agencies with proprietary data, who need control over the Attribution Modeling process. From a tabular dataset, ATRIO can produce simple heuristics such as last-click analysis, but also data-driven attribution models, based on Shapley's Game Theory and Markov Chains. As opposed to the black-boxed tools offered by campaign delivery platforms, which are focused in their media channels performance, ATRIO empowers digital media agencies to customize and apply different Attribution Models for each campaign, providing an agnostic, open-source based, holistic and multi-channel analysis.

KEYWORDS

Attribution Modelling; Digital Advertising; Analytics; Insights; Shapley; Markov; Jupyter; Python; Plotly

CONTENTS

1. INTRODUCTION

1.1. BACKGROUND

Attribution is the process of identifying a set of user actions across media touch points which contribute in some manner to a desired outcome, and then assigning value to each of these events (Dalessandro, Stitelman, Perlich, & Provost, 2012). A good attribution model helps campaign analysts to make the right choices to distribute advertising budgets across a mix of different media, channels, supports and formats. This optimization creates long-term efficiencies yielding improved results without budget increases, when compared to less-informed approaches.

Digital media agencies often rely on the Attribution Modelling tools provided by the campaign delivery platforms where their advertising budgets are allocated. While these platforms allow some degree of integration to ingest data from other media channels, and even accept some degree of parameterization, they don't offer an agnostic approach where analysts can have visibility over the algorithms that define the outcome of the model. In other words, their mechanisms are not publicly available and therefore, irreproducible (Dalessandro, et al., 2012). In many cases though, this simplified approach is enough, because it abstracts the complexity of the underlying algorithms, reducing the available mix of choices and mitigating the risk of choosing an incorrect set of options, from statistical-significance and model-fitness points of view. However, by limiting the model choices and their available parameters, the outcome may bias results in favor of specific media channels, leading to non-optimal channel-mix choices.

Along this report, two different techniques for data-driven attribution models will be developed, one based on Lloyd Shapley's Game Theory and the other on Markov Chains. Simple heuristics such as Last-click analysis will also be presented. A Cloud Platform environment setup will be configured to host the solution, demonstrating how it can be productized by any organization who possesses their own datasets, statistical skills, and IT resources to implement their own solution. The final product will be capable of ingesting data and shape it into a model that can be consumed by the mentioned algorithms, to deliver the respective results. This will allow analysts to compare algorithm results and derive their conclusions to obtain an optimal channel mix for a given dataset.

1.2. PROBLEM IDENTIFICATION

The 19th century tycoon John Wanamaker allegedly claimed that half the money he spent on advertising was wasted. The trouble was knowing which half. Even business operations in general face similar dilemmas and this could explain why Waste Analysis is such an important field of study in Business Process Management. Hence, the question is how to achieve better conversion results with the same budget, by reducing waste and investing on the right channel mix.

In the era of digital advertising, instruments such as tags and cookies provide valuable data to calculate the effectiveness of media channels, so it could be argued that Wanamaker's dilemma is close to being resolved. But what about the dynamic interaction of all the touchpoints from these channels? (Xu, et al., 2011) What if the order in which they are allocated is altered? (SHAO & LI, 2011) Is it possible to estimate the carryover rates to optimize frequency efficiencies and reduce waste? (Li & Kannan, 2014) How does a given sequence of media touchpoints influence the conversion rate? (Anderl, et al., 2014)

As pointed out by (Anderl, et al., 2014) advertisers use a mix of online marketing channels, which include social media, paid search and display marketing. However, without the results provided by an Attribution Model, they don't know to which degree each channel is contributing to the overall result of the campaign. In the previous section, we mentioned data-driven algorithms and simple heuristics as the two main approaches for attribution analysis. Simple heuristics consider first-click or last-click as the "winning events" to determine which channel led to a conversion. This was the original approach used by the industry, named "last-touch attribution" (LTA) (SHAO & LI, 2011).

In an effort to compensate the limitations of LTA, the concept of "multi-touch attribution" emerged, proposing more than one touchpoint can have its share of the credit, based on how much it contributed to the outcome of a consumer's action (SHAO & LI, 2011).

At this point, having two techniques, LTM and MTA, the capability of comparing the results between them for the same dataset, is one of ATRIO's key proposals. We will discuss the characteristics of these techniques and summarize the opinion from the authors we reference, in next chapter's literature review, but we will not rank them or suggest an optimal model over the other. That could be the subject for a thesis dissertation. This paper is the foundation of a project to solve a specific problem: to provide a framework for an advertising agency to analyze the datasets obtained from their digital campaign logs, by offering an application capable of ingesting such data, transform it and display the results from applying the proposed techniques, in the form of tables and rich visualizations.

To use ATRIO, agencies or analysts need to provide their own dataset containing the campaign logs. Without data with the required granularity and quality, it's not possible to use the approach we propose. Chapter 3 will provide the necessary detail about the data content and format that is needed to produce a digital marketing attribution model.

1.3. OBJECTIVES

Considering the questions from the previous section, the main objective of this project is the creation of an application capable of ingesting the digital campaign results data and shape it into a data model where the attribution algorithms can be applied, generating tables and interactive visualizations enabling media analysts to derive insights about their campaigns.

This project proposes to process digital campaign data, and apply Attribution Modeling techniques in a consistent, reproducible, and open format, based on modern open-source languages and frameworks. Attribution models analyze each consumer journey across several media touchpoints, from the first contact with the campaign to the last event, which could yield conversions, i.e. actions such as an order or a request for information.

As we'll explain in the next sections, the main driver for this project is a business need from GroupM – the leading media investment and strategy firm in the industry – to improve their in-house Attribution Modelling practice. Throughout this document, we will demonstrate how ATRIO can add significant value to the current Attribution Modeling practice at GroupM by modernizing their framework and introducing a new algorithm.

1.4. BUSINESS RELEVANCE

The capability to analyze campaign data is crucial for digital media agencies. This capability drives the generation of actionable insights for campaign stewardship, maximizing the conversion rate for a given budget. However, these insights can only become actionable if they are delivered in a consistent, structured, and reproducible framework.

Traditional media campaigns combine a set of structured processes, starting with the setting of KPIs during the planning phase, and then, they monitor the pace of the campaign until the budget is fully allocated. In these cases, agencies compare these baseline KPIs against the final performance to assess the campaign success. While some adjustments can be made, the course of the campaign is relatively static. However, in digital media, the growing competition and the technological capability has pushed the envelope towards extreme agility. The ability to rapidly pivot between different creativities, target groups and digital channels, has rendered traditional media planning obsolete. To implement this flexible approach, agencies must have good quality data, IT and analytic capabilities.

The analysis of the log data generated from the execution of digital campaigns can deliver actionable insights. Again, the challenges to capture and model the right input data may differ. There are situations where little or no data is logged at all; others where a highly structured platform provides standard views of advertising entities and metrics, requiring only a few data mapping operations. In other cases, agencies might have unstructured data containing the information, while lacking a schema to read the data, which requires data science skills to shape it into a meaningful data structure. Extreme Big Data scenarios are now common, where data arrives in a stream requiring a dedicated architecture to derive structured data from that stream.

This data can be shaped to represent a Consumer Journey, where an individual is tracked along several media touchpoints and eventually ends that journey converting into a defined business goal, such as a request for information or a purchase. ATRIO will apply different algorithms to analyze this consumer journey, so the results can be compared side by side. There are challenges in the comparison of results from different algorithms for the same campaign since each of these relies on different mathematical principles. However, the purpose of ATRIO is to present the outputs of these algorithms to campaign analysts, so they can compare results and make informed decisions.

ATRIO will compute two data-driven attribution models in parallel, based on the Game Theory algorithms from Lloyd Shapley (Shapley & Aumann, 1968) and based on Markov chain directed graphs (Markov, 1907). Additionally, the simple heuristic models proposed by Google (Google Inc, 2020) such as first-click or last-click analysis will also be computed. To visualize data, among other visualizations, ATRIO will mostly use Sankey Charts to analyze the data flows between the start and the end of a consumer journey, mapping the passage through the recorded touchpoints.

ATRIO aims to be an open Attribution Modelling Framework for media agencies who want to perform their own analysis of campaign logs, obtained from tags, cookies, or other logging forms. The framework will deliver a side-by-side comparison of different algorithms' results, including the customization of their model parameters, instead of relying on the black-box solutions offered by campaign delivery platforms.

1.5. REQUIREMENTS AND APPROACH

The key requirement for an agency to use ATRIO will be having enough data with an adequate level of granularity, a solid definition of the baseline KPIs to be measured and the ability of mapping that data into the model. GroupM has in-house capabilities to create attribution models from their proprietary datasets. However, their existing framework which was internally developed is limited and off the knowledge scope of current staff, and since its code is no longer maintained, it cannot be improved with new additions.

Therefore, the proposed framework, ATRIO, needs to be built using today's coding languages, data transformation and analysis libraries. By choosing a technology stack within the breadth of today's typical analyst skillset, we allow **margin for future improvements**. Therefore, while the legacy framework is a combination of executable files, SQL databases and a web-based front-end that is no longer maintained, ATRIO will be entirely written in Python and implemented in Jupyter, meaning all the code can be kept in the same project. We can then satisfy the requirements for **maintainability, reproducibility, versioning, and continuous integration & development**.

The big difference between ATRIO and the legacy GroupM attribution framework is be the **capability of incorporating new algorithms**, which can be coded as new functions by other developers. For now, the **key differentiator will be the Markov Chain**-based attribution model, which is not available in the current framework. Given the academic context of this project – achievement of a master's degree of the author, by delivering a valuable project for a firm – there are three conditions established by GroupM for this development:

- GroupM owns the source data, the source code, and the results of the data analysis
- The advertiser and the campaign name in the dataset will remain anonymous
- Results should remain unpublished for as long as possible, within the norms of Nova IMS

Jupyter Notebooks will be used for this project. The main reasons for this choice are:

- Along with R, Python has become one of the most popular languages for data processing and analysis
- The language is open-sourced and there are many libraries and frameworks that are freely available such as NumPy, Pandas, Matplotlib, iPython and Plotly.py, among others
- There are open-sourced tools and Integrated Development Environments, such as Visual Studio Code with add-ons such as the Python add-on published by Microsoft, that ease the development of Python code and can run and debug Jupyter Notebooks
- All code is Python and all package dependencies can be saved for version control, reproducibility, and portability, since all of them are open-sourced

The platform chosen to host and render the Jupyter notebook was Heroku. This platform-as-a-service has a free starter tier and then offers a competitive use-based pricing model. Moreover, it provides an API to enable automation, which opens the possibility for integration and agent-based executions. However, Jupyter Notebooks can run on many other hosting environments, so the choice of this hosting platform is not limitative nor compromises the output. Given this, **ATRIO is considered portable**.

2. LITERATURE REVIEW

There are a few studies dedicated to Attribution Modeling, which are available publicly or by request. During the research for this project, a total of 87 documents were obtained and reviewed. These included published academic papers, books, essays, and technical documentation about existing Attribution Modeling solutions. Out of these 87 documents, a subset of 11 documents were classified as “Core” and another subset of 32 documents classified as “Shortlist”, were also singled out. This process left out 44 documents from the original 87, which were archived an “Others” subset. The first criteria to drive the Core-Shortlist-Others classification was the relevance for this project, which relegated about half of the initial 87 to the “Others” bucket. The second criteria were based on the amount of references from the remaining half, which consistently pointed towards a group of 11 papers, marked as “core”.

To summarize, some references to the “shortlist” papers can be made along this report, but the focus will be on the “core” list of 11 papers, which consistently referenced by papers on the shortlist. Both “core” and “shortlist” document folders have been made available to project stakeholders - academic and business - by sharing the access to their Mendeley repository.

2.1. CORE REFERENCE PAPERS ANALYSIS

All 11 papers from the Core list appear in the References section of this report. These papers are part of the relevant half - 43 documents - and are consistently referred by the other 32 papers. Therefore, a deeper analysis of these 11 papers will follow, setting the base for the theoretical approach of this project.

We’ll start our review with Media Exposure through the Funnel (Abhishek, et al., s.d.) where authors raise the key question of attribution: *which ads get credit for a conversion and how much credit does each of these ads get?* They address the problem of Attribution by developing a **Hidden Markov Model (HMM)**, mapping consumer behavior through the concept of a conversion funnel[1] and applying their model to a dataset based on a campaign for a new car launch. Their observation is that different ad formats, i.e. display ads vs. search ads influence the decision process. According to their study, Display ads tend to have a trigger role to shift consumers from a disengaged state to an interactive state. However, search ads seem to have a pronounced effect across all funnel stages. Still, the interaction with display ads appears to increase the conversion probability. This study sustains their claims by implementing the HMM, stating this technique provides different insights from the traditional approaches, allowing them to conclude that display ads do have effects at early stages of the conversion process.

References are made to (SHAO & LI, 2011) and (Dalessandro, et al., 2012), suggesting these authors use simple statistical models to address the issue of attribution, whereas (Li & Kannan, 2014) and (Anderl, et al., 2014) already incorporate models of consumer behavior as a step forward. The authors claim that, **compared to Last Touch Attribution scheme (LTA)** – where all the credit for the conversion is assigned to the last interaction (click or impression) – the importance of Display ads at early stages

[1] The conversion funnel is a model of a consumer's search and purchase process that is commonly used by marketers (Kotler and Armstrong, 2011). Using this metaphor, advertising efforts can be aimed at "upper funnel", "middle funnel", or "lower funnel" potential customers. (Jansen, B. J. and Schuster, S., 2011)

of a campaign tends to be undervalued. Hence, their methodology gives a relatively higher credit to display ads and a lower one to search ads, arguing against a common misconception that display ads are not effective in the conversion process. As a result, authors claim three contributions: first, the introduction of temporal dynamics across the whole conversion process; second, a new attribution technique based on HMM that builds upon existing techniques; third, a comprehensive description of which ad formats may better influence consumers, depending on the stage of deliberation.

In their graph-based framework for online attribution modeling (Anderl, et al., 2014), authors introduce a novel practice, based on **Markovian graph-based data mining** techniques, that extends an original approach used for search-engine marketing. They analyze each customer journey as a first-order Markov graph, using a property they call "removal effect" to determine the individual contribution of each channel.

This paper provides summary table describing many of the attributes we observe on the other works in this section, namely: **objectivity; robustness; predictive accuracy; interpretability; versatility; algorithmic efficiency**. They conclude with the objective to develop a framework that satisfies all these criteria. Using a real dataset from an online media campaign, they analyze bot converters and non-converter journeys. A small remark is made to the fact that we still rely on cookies to retrieve this data, with all the potential inaccuracies that stem from cookie deletion or shared devices, to name a few.

Quoting the authors, *"Markov chains are probabilistic models that can represent dependencies between sequences of observations of a random variable"*. Hence, their approach is to represent the customer journeys as chains in directed Markov graphs, with the advantage of performing more efficient calculations, since the size of the graph depends on the number of states, rather than the number of journeys in the dataset. Hence, in their model, each state corresponds to a specific media channel. Additionally, three special states are added: **start**, as the first touchpoint; **conversion**, representing a successful journey; **null**, for non-converters.

The transition probability corresponds to the likelihood that a contact with a channel, let's say, "Display", is followed by a contact with "Search" channel. Because authors suggest that clickstreams should not be regarded as pure Markovian, they introduce alternative higher orders to in which the present depends on *k* previous observations. However, considering computational capability, this research has not gone over 4^{th}-order level, since the increase of levels yields exponential results.

Finally, authors explain how they apply their "removal effect" as a mechanism to fit the Markovian model into the attribution analysis. This process aims to reflect the changes in the conversion rates to estimate the outcome if a given channel, or channel sequence, is removed from the chain. In this case, the removal effect is calculated using a multiplication matrix and enables to measure the contribution of each individual channel.

After the application of their framework, authors compare the results of a third-order Markov to the common attribution heuristics we already mentioned (first-click, last-click…) and with the most common data-driven method, Shapley value. Besides the differences in the contribution weights assigned to different media channels by these different methods, authors conclude that their approach yields a significative advantage over all others, in the sense that it can be used prospectively, for example in real-time bidding exchanges. They consider both simple heuristics and Shapley value to be purely retrospective, thus having limited practical applicability. They argue that, by using four real-

world datasets they were able to extend prior studies from (Abhishek, et al., n.d.) (Li & Kannan, 2014) (Xu, et al., 2011).

The study Beyond the last touch: Attribution in online advertising (Berman, 2015) begins with a broad approach of digital campaign analysis. This breadth includes the CPM (cost per *millare*) analysis where a campaign is analyzed in terms of the cost of each batch of 1000 impressions. This method is good enough for most situations where no other information exists to distinguish the individual contribution of specific media channels and a baseline is defined from the overall CPM. In other words, this is also known as an effort-based compensation method. The alternative is the CPA (Cost per Action) analysis where a performance-based method is derived from the actions – some of these, resulting in conversions – that outcome from each channel. The problem of a simplistic CPA analysis is assessing how much of the result derives from the free-riding effects of prior campaign touchpoints, and how much of that result is already leveraged by an existing consumer awareness baseline.

Based on this premise, authors suggest that in the absence of a good Attribution Model, a CPM compensation is more beneficial to advertisers than a CPA that is biased by free-riding and baseline effects. Their main argument is, the Attribution Process creates a contest between publishers and as a result, this can benefit the advertiser when no baseline exists. Therefore, in these cases, even a Last-Touch attribution model can bring benefits when no other means are available. However, when baseline effects exist, the authors claim the Last-Touch method is inaccurate and suggest that a method based on the Shapley value is more suitable. Since ads interact in a non-trivial manner to influence consumers, advertisers can't easily assess the "piggyback" effects of specific ads over the others. Another problem is an asymmetry of information, since publishers have more data on past consumer actions regarding their interactions between websites and search engines or social media platforms, leading to choices that benefit publishers rather than advertisers.

In a cooperative game, each coalition of players is assigned to the contribution they generate. This led the authors to suggest the use of a **cooperative game theory based on the Shapley value** to allocate value among players in such cooperative game, that can potentially allocate the commission among publishers with four requirements: Efficiency, Symmetry, Pay-to-play and Marginality. However, since the use of Shapley value as-is, returns equal probability to the order of appearance of publishers, its use will be flawed in cases when not every order of arrival is possible. This led authors to develop a modified Shapley value calculation to deal with these issues.

The aim of the paper Attribution Models and the cooperative game theory (Cano-Berlanga, et al., 2011) is to provide a **compatible framework to the Google approach**, while in addition, defining a way to transfer conversions into a **cooperative game based on the Shapley value**. Authors claim the Shapley value selects an efficient allocation and results in a univalued distribution. The authors consider that Shapley's theory is somewhat stable, since no player or group of players could improve on it. Two key features are well explained, to support this assumption:

The Marginal Contribution feature considers that the Shapley value measures the individual contribution of each player to each coalition, so the players are awarded for their contribution in each case.

The Temporal Sequence feature recognizes the ordering in which players join the coalition is an issue. Hence, each player's marginal contribution is computed taking all possible orderings into account.

As mentioned by (Berman, 2015), but now with more detail, the Shapley value must jointly satisfy four properties:

Efficiency: all gains or costs are distributed among players.

Symmetry: Two players are considered substitutes when they make equal contributions to the game and should therefore receive the same amount.

Dummy player: If the marginal contribution of a player is zero, no additional payment should be given to him, but in terms of the game he must still receive an allocation equal to his individual worth.

Additivity: The sum of the player's allocation for each individual game is equal to his allocation for the sum of those games.

Analyzing the paper Causally motivated attribution for online advertising (Dalessandro, et al., 2012) we understand they position attribution as a causal estimation problem and two approximation methods are proposed as alternatives for the cases where a full causal estimation cannot be done. The proposed alternatives derive from their causal approach and **incorporate prior work based on cooperative game theory**. Last-touch attribution is considered as the default rule-based methodology while multi-touch attribution is considered by the authors as a technique that is still lacking proper standardization. Stemming from this assumption they propose a set of properties to define what is considered a "good" attribution system that stands on a solid causal framework.

For the authors, a good attribution system should align the incentives of the advertiser with the incentives of the channels that will display their ads. This means balancing between the advertiser perspective of driving as many conversions for the lowest cost with the channel perspective of receiving as much payment for the converted events while minimizing running costs. However, the optimal strategy for a given channel does not focus on influencing conversions but solely on earning full credit for them. The authors propose three principles to reach a standard that aligns both advertiser and channel incentives:

Fairness: to reward a channel for his ability to influence a conversion.

Data-driven: meaning data is treated and conversions are captured along the campaign and both advertiser and channel have access to the same data.

Interpretability: based on statistical merit, both parties accept the methodology and the results and understand the components of the system. The authors argue that (SHAO & LI, 2011) present a coefficient-based method for MTA that is difficult to interpret intuitively, as an example.

The attribution method proposed satisfies fairness by using counterfactual analysis based on the question *"what is the effect of an ad in user conversion?"*. This counterfactual framework shall be based on parameters that represent several levels of ad exposure and their estimation can be achieved by controlled experimentation or observational data methods. Three assumptions are stated for this to be feasible: treatment precedes the outcome; any attribute that affect both ad treatment and conversion outcome must be accounted for; every user has a non-zero probability of receiving and ad treatment.

On their empirical approach, the authors explored normal **maximum-likelihood logistic regression**, **elastic-net regularized logistic regression,** and **smoothed empirical probability estimation**. Authors admit that "truth" of an attribution analysis is a quantity that may never be known. In this sense, there is no benchmark to evaluate what is "better", in quantitative terms. To compare their methodology with the "last-touch" standard, they synthesized a campaign data set, controlling the factors that influence attribution allocations.

Their conclusion is that while they developed a **game-based theoretic approach** that standardizes the process and delivers practicality, its **adherence to real-life conditions is limited**. In their own words, *"we make simplifying assumptions about the data in order to fit attribution into a game-theoretic framework"*. However, this exercise is still valid because it provides insights about how an ideal framework for attribution should be set up and to which degree, we can expect to draw realistic conclusions from these models.

In the paper Do display ads influence search? (Kireyev, et al., 2013) authors present a **Multivariate Time series model** to investigate the interaction between paid search and display ads. As seen in other papers, authors also defend that Display ads contribute to an increase of search conversion. Heuristic models are mentioned as a poor instrument that does not reflect the true value of each impression's contribution and propose a data-driven approach. This paper sustains the theory that display ads influence consumers at the top or middle of the purchase funnel, while search ads can be more effective at the bottom of the funnel.

The presented methodology uses **Persistence Modelling** techniques to extend multivariate time-series to account for the spill-over effects resulting from advertising recall accumulated along the campaign. To assess the sensitivity of the impulse response analysis to the modelling assumptions, a **vector-auto-regressive model** was used. The **VAR** analysis revealed the marketing metrics should consider not only attribution but also the dynamic effects of marketing. It's also worth noting that this analysis revealed search ads don't affect display applications but display impressions do influence both search and display applications. While this influence is positive, since search often accounts as a driver for conversion, the author recommends a careful consideration about the interaction and dynamic effects between display and search advertising.

The authors of Attributing conversions in a multichannel online marketing environment (Li & Kannan, 2014) propose a model to measure how consumers consider online channels, analyzing the visits through these channels over time and the purchase results to estimate carryovers and spillovers, to assess conversion credit to specific channels. They start by justifying why the simple heuristic models, such as last-click, first-click, uniform, and weighted methods do not consider the timing and sequence of earlier impressions, thus ignoring the carryover effects. They also reveal an interesting fact that media channels are often planned by different teams and systems resulting in incompatible data.

A three-level measurement is proposed to estimate carryover and spillover effects: impressions prior to the visit of the website; at the stage of visiting; at the state of purchase. Considering each individual consumer journey, they account for: heterogeneity across customer's consideration channels; carryover and spillover of prior marketing initiatives; subsequent conversions. They estimate the carryover and spillover effects using a **Monte-Carlo simulation** where they perform 5000 iterations using a **Markov chain**. In a posterior analysis this paper also refers the use of **Shapley** value to calculate

the total contribution of each channel, as others have done. This framework is quite interesting in the sense that it combines both techniques, although at different stages of the analysis.

Their most significant conclusion is that the impact of paid search is not as high as suggested by the last-click model, and that much of this impact could be recaptured by organic search, possibly driven by the carryover effects of display ad contribution.

The investigation published under Purchase conversions and attribution modelling in online advertising (Nisar & Yeung, 2015) is based on the **comparison between last-click attribution and cooperative game theory**. Authors propose a computation of contributing factors using the Shapley regressed approach, decomposing the purchase funnel by regressed sources. Their hypothesis is that display advertising acts as a conversion driver and generates more conversions under the last-click model than it does according to Shapley value estimation. They base their argument in the fact that while organic search accounts for most of the impressions that precede a conversion, that organic search results from display advertising. Authors base their statement on a two-sample t-test and challenge other predictive models such as (Li & Kannan, 2014) as being more focused on predictive accuracy, while paying less attention to the stability issue of the variable contribution estimate.

The reason behind different results obtained applying Shapley value is, that this method accounts for the marginal contributions of all possible contributing channels, thus diluting part of the contribution of display advertising. An interesting fact about this paper, if we compare it with the others we analyzed so far, is that it does not claim that the last-click heuristic model is less valid or less accurate than a data-driven approach such as Shapley's game theory. By applying an additional set of t-tests based on the last click model they arrive at a similar rank of contributing channels for conversion than they obtain with Shapley value, although the weights of these contributions are different.

Data-driven Multi-touch attribution models (SHAO & LI, 2011) is the most referred paper throughout the bibliography we curated to support ATRIO. It starts with a brief explanation about the distinction between attribution Modelling and Marketing Mix Modelling, which is limited to the temporal analysis of marketing channels.

The authors of this paper begin by outlining why the last-click model is insufficient and why multi-touch attribution analysis can provide a deeper analysis over the whole interactions between touchpoints, which the last-click model ignores. Before diving into their proposal for a Multi-Touch Attribution model (MTA), they establish what are the characteristics that make up a good MTA model. The first, is **accuracy** to classify every consumer journey as positive or negative i.e. converters vs. non-converters. Second, they define the capability of providing a stable estimation of individual variables such as each media channel contribution. **Stable and reproducible** are two attributes they outline for any attribution framework. Third, the black-box tools that have been mentioned in the introduction of this report, are referred as difficult to interpret and not stable.

Hence, a bivariate metric is proposed to estimate the variability of the metric on one side, and on the other, the accuracy of the converter vs. non-converter classification. Then, a **bagged logistic regression model** is applied, which according to their findings, is comparable to common logistic regression in terms of accuracy, but more stable when it comes to compare individual contributions of each channel.

Authors then explain the possible approaches for their choice, by considering vector machines and neural networks as complex, black-boxed and not always stable nor easy to interpret and derive consistent data from. Then, they turn to a combination of logistic regression with the "bagging" concept derived from random forests where decision tree modes are stacked to increase performance. As a result, they obtain two of the most important attributes of the model: **ease of interpretation** and a **reproducible estimation result**. In addition to the bagged logistic regression model they also apply a combination of first and second-order probabilities resulting on a simpler model.

Their overall conclusion, comparing LTA with MTA is that while results don't differ that much for search clicks, email clicks and social clicks, the effectiveness of display (as we've seen on previous literature) yields significant differences. In this case, LTA undervalues display ads, since in most cases these touchpoints are far behind in time on the conversion funnel. They also suggest future work on their MTA model improving additional heuristics: better selection of predictive dimensions to reduce noise and improve predictability; control and possibly reduction of cardinality in the chosen dimensions; better variable encoding to increase reproducibility and interpretability of the model.

Then, a novel approach is proposed by the authors of Path to purchase – mutually exciting point process (Xu, et al., 2011), to study the dynamic interactions among advertisement clicks themselves. The idea is that while these clicks may not precede an immediate conversion, they stimulate further clicks which eventually result in the desired outcome. They develop a **stochastic model founded on a Bayesian hierarchical framework**, where individual random effects are incorporated to account for consumer heterogeneity, considering advertisement clicks and purchases as dependent random events in time. They acknowledge building upon (Li & Kannan, 2014) and (Abhishek, et al., n.d.) work, while claiming to enrich it with their novel approach.

As previous studies have shown, they find that display advertising has a relatively low effect on direct purchase conversion, but it's still an important indirect driver that stimulates visits through other ad formats. They also dwell on the fact that simple heuristics such as last click, undervalue the precedent touchpoints since they don't account for the so-called mutually exciting effects between all the touchpoints in the conversion funnel. They suggest that a valid model should account for the multivariate nature of non-purchase activities if we aim to study the conversion effects of combined channels in a holistic perspective.

Authors also introduce the relativeness of decay[2] depending on the elapsed time between impressions and incorporate these effects into their model. Based on their model and its Bayesian results, authors claim to achieve a solid conversion probability estimation for different types of ads. They claim as well to introduce the first model that successfully applies **Bayesian inference using Markov Chain Monte-Carlo (MCMC)**, achieving good fitness between a complex hierarchical model with random effects in correlated stochastic processes.

[2] Decay is commonly referred in advertising as the natural effect by which the awareness (a.k.a. ad-recall) of an advertising message tends to fade in the consumer's mind as time passes. The carryover and spillover effects are related to this topic as well; the first one concerns the accumulation of brand awareness across a series of ad impressions until a point where additional impressions add little or no value or can even be detrimental to awareness; the second, implies part of this awareness is perceived by the consumer as originating from other related products or brands, who often "piggyback" upon other's advertising effort. (based on author's 20-year experience in the advertising & media business)

Tests were based on a dataset from a vendor of consumer electronics and include data from online media campaigns with conversion results. Channels were classified on two categories: search, display, and "others". As a preliminary step, a quick descriptive analysis of the dataset was performed, to reveal the most common strings of ad click and purchase sequences. This a very simple but quite interesting approach, not seen on previous papers, that allows a quick identification of the most common patterns and helps to set the direction of the complex approach that will follow.

The main algorithm starts by simulating a homogeneous Poisson process with a high intensity to assess the mutually exciting point processes between impressions, using a Monte-Carlo approach. Several iterations are then run, with progressively lowered intensities. After the simulations are complete, a constant intensity of the mutually exciting process is derived and is used as a baseline. This baseline allows for a second iteration where another simulation is performed, allowing to generate time intervals from an exponential distribution. By applying this algorithm to repeatedly simulate the point processes in their model, an average conversion probability is obtained. The authors also consider their model to be predictive of each individual behavior, based on historical data. They claim the capability to predict behavior for the fourth campaign month, after applying the algorithm to the first three campaign months. This is a very important feature, since it means it could be applicable during the campaign and not just as an ad-hoc analysis.

Just as several papers we've discussed so far, Multi-touch attribution in online advertising with survival theory (Zhang, et al., 2015) refers the limitations of last-touch attribution (LTA) and the need for multi-touch attribution (MTA) models to allocate a fair distribution of conversion credit to the events along the funnel. Still, as authors point out, many data-driven MTA models suffer from two common problems: they ignore the presentation biases introduced by advertisement placement and they focus only in the attribution modelling without providing a predictive framework. Hence, an **Additive Hazard model** is proposed, that considers the differences between impact strength among different channels, but also, the variations of their time-decaying speed. This model then named as Additive Hazard (AH) thereafter.

The AH model is a combination of two functions: **survival function** and the **hazard function**. The survival function returns the probability that the time at which a conversion occurs is later than a specified time *t*. The hazard function yields the instantaneous rate of occurrence of a given event. Authors propose the AH model for user conversions in online advertising to model the impact of a given ad on a conversion by determining the strength of this influence and its time decaying property. There seem to be some similarities with the approach from the previous paper, namely on the relevance of calculating the decay of each impression's impact.

To prove the results of their framework, the authors performed a benchmark against other models, namely: last touch; logistic regression; causal and simple probabilistic where they re-use the work from (Dalessandro, et al., 2012) for their conversion prediction. Using a real dataset, they compared the accuracy to make predictions for conversion. According to their results, their method outperformed the others. For future work, authors propose the same approach as (SHAO & LI, 2011) in the sense of limiting dimensionality and cardinality, since in the current version, the model is fitted at the individual channel level. They also refer the possibility to analyze mutual exciting effects, as we've seen on the previous paper by (Xu, et al., 2011).

2.2. CORE REFERENCE SUMMARY

After the extensive analysis of the papers considered as "core", here is the summary table:

Table 1 is presented to outline similarities and differences between papers:

Author	Main Model	Main contribution / keywords
Abhishek	Hidden Markov Model	Temporal dynamics
Anderl	Markovian Graph	Prospective / predictive use
Berman	Cooperative game theory	Improved Shapley calculation
Cano-Berlanga	Cooperative game theory	Efficiency, Symmetry, Additivity
Dalessandro	Shapley value	Standardization of game-based approach
Kireyev	Vector Auto-Regressive	Marketing perspective, beyond advertising
Li & Kannan	Markov chain + Shapley	Combines Markov & Shapley
Nisar & Yeung	Shapley value	Extensive comparison last click / Shapley
Shao & Li	Bagged Logistic Regression	Ease of interpretation and reproducibility
Xu	Bayesian Hierarchical Framework	Dynamic interaction between touchpoints
Zhang	Survival Theory	Prediction accuracy

From the previous section, we can observe a clear majority of models based on Shapley Value or Markov Chains. While (Li & Kannan, 2014) actually combine both, they do it at different stages of the process. ATRIO's approach is to provide separate outputs for both models, given the relevance both reveal if we analyze the available literature. Without preference for one or the other, we believe there is value in providing a framework that can calculate two data-driven attribution models in parallel, based on two different algorithms. Additionally, since the heuristic models such as first-click or last-click easy to implement and considering that some of the literature indicates these are still valid under certain contexts, ATRIO will deliver these as well.

While a side-by side comparison of all outputs is delivered, this project does not include a benchmark between these. It will be the analyst's role to draw the conclusions about which model has the best fit, which will largely depend on the source data. In the last section of Chapter 5, we'll demonstrate how ATRIO can support this analysis and check the adherence of the output results to this literature.

To summarize some of the conclusions of the core literature review, there seems to be a consensus towards the idea that **display advertising has an important, but indirect contribution for conversion**. This is also the reason why **simple heuristics still provide valuable insights for attribution models** since they deliver the basic statistics that are easily observable and sustainable for most datasets, where "search" has an unmistakable relation with conversion. Moreover, many of the papers analyzed, refer a pragmatic approach from analysts and decision makers who need solid statistics they can understand, and consistently apply and reproduce. This means that the high optimization opportunities that data-driven models yield, often come at a cost, lacking consistency, reproducibility, and a sound comprehension of the processes that produce a specific output.

Considering these conclusions, ATRIO is proposed as “agnostic”, in the sense that it’s the analyst’s role to find the correct balance between simple heuristics and complex data-driven algorithms, case by case. This balance implies there is truth and validity in simple heuristics, while there’s room for optimization and improvement in the analysis of the marginal contributions from each channel or in the timing sequence these touchpoints impact consumers, which only data-driven methods can assess.

3. PROJECT METHODOLOGY

In this project, we will process a dataset provided by GroupM Portugal, containing data from a digital advertising campaign. Our objective is to provide a framework capable of delivering the attribution modeling features we described on the previous chapters, by processing the dataset and producing an output that is fit for analytics. This paper describes the identification of a business problem, relies on the relevant literature review to address Attribution Modelling, in order to solve that problem, provides the technical implementation of Attribution Modelling algorithms through code, and delivers the results of ATRIO's application to the dataset provided by GroupM. The sections of this chapter are meant to explain the methodology we followed, which can be divided in four steps:

1. A description of the dataset we received from GroupM
2. Techniques for loading and refreshing data
3. Data structure requirements and data preparation process
4. The algorithms we'll use to process the data, to deliver a rich output for analysis

3.1. DATASET DESCRIPTION

The dataset we received contains all impressions, clicks and conversions from an online media campaign that ran between October and November 2017.

Table 2 shows all the columns of the dataset:

Column Name	Type	Category	Cardinality	Example Value	Description
ID	Integer	ID	n/a	22576	
UserID	Integer	ID	n/a	2222392006655	
ConversionID	Integer	ID	n/a	508218723740	
ConversionDate	Datetime	Time	n/a	2017-10-10 02:03:03.445	
AdvertiserName	String	Attribute	1	Advertiser	
ConversionTagID	Integer	ID	n/a	1093727	
Tag	String	Attribute	13	CT_Advertiser_Comprar_Passo_2	
QueryString	String	Attribute	n/a	cn=as&ActivityID=1093727	
WinningEventType	String	Attribute	2	click	impression, click
WinningEventDate	Datetime	Time	n/a	2017-10-10 01:59:26.939	
WinningDisplaySiteName.WinningSearchEngineName	String	Attribute	3	Xaxis PT	Facebook PT, Xaxis PT, YouTube PT
WinningPlacementName.WinningSearchKeywordName	String	Attribute	6	Billboard	Billboard, pre-roll, promoted post…
WinningMediaBuyChannel	String	Attribute	3	Unclassified	Social, Unclassified, video
EventType	String	Attribute	3	click	Click, impression, site visit
EventDate	Datetime	Time	n/a	2017-10-10 01:59:26.939	
EntityID	Integer	ID	n/a	48444189	
ConversionTimeLag	Time	Time	n/a	00:00:03:36	
WinnerEventTimeLag	Time	Time	n/a	00:00:00:00	
SectionName	String	Attribute	4	Broad Reach Premium XL	Promoted post, TrueView, broad reach
CampaignID	Integer	ID	n/a	827647	
CampaignName	String	Attribute	1	Campaign theme 1 - Out17	

DisplaySiteID.SearchEngineID	Integer	ID	n/a	48851	
Site	String	Attribute	3	Xaxis PT	Facebook, Xaxis, YouTube
DisplayAdName.SearchAdName	String	Attribute	5	Billboard - Campaign theme 1 - Out17_Polite	
MediaBuyChannel	String	Attribute	3	Unclassified	Social, Unclassified, video
Brand	String	Attribute	1	None	
SiteVisitEventDate	Datetime	Time	n/a	2017-10-10 14:56:07.731	
SiteVisit	String	Attribute	3	SEARCH	Other, search, social
Referral	String	Attribute	37	www.google.pt	
DiaEvent	Date	Time	60	2017-10-10	
HEvent	Integer	Time	24	1	
MEvent	Integer	Time	60	59	
SEvent	Decimal	Time	60	26.939	
DHMSEvent	Datetime	Time	n/a	2017-10-10 01:59	
DiaConversion	Date	Time	60	2017-10-10	
HConversion	Integer	Time	24	2	
MConversion	Integer	Time	60	3	
SConversion	Decimal	Time	60	03.445	
DHMSConversion	Datetime	Time	n/a	2017-10-10 02:03	

These are the main characteristics of the dataset:

- 50.236 records: each record is a touchpoint for a specific individual
- 10.627 unique individuals
- 20.212 conversion journeys

Here is the description of the fields – which we formatted as **bold** in the previous list – that we will import into our Dataframe:

ConversionID: defines a unique identifier for each conversion journey. The dataset we are working with, only includes journeys that resulted in a conversion. Other datasets may also have journeys that do not end in a conversion. This *ConversionID* is normally obtained from cookies or other tracking mechanisms and is assumed to reference a single journey performed by the same person. Unfortunately, the accuracy of this reference is not absolute, since many factors such as device sharing, cookie deletion, or anti-tracking software will bias this variable. **For this dataset**, we work under the assumption that each *ConversionID* represents one journey and each record with the same *ConversionID* represents a touchpoint in that journey, or a conversion if it's the last one of the sequence, since this dataset only contains converting journeys.

EventDate: is a timestamp representing the date and time when the event – impression, click or site visit – took place. Each event can then be sequenced within a given journey, defining its beginning, its end, and all touchpoints in between. In our dataset, 41% of all the journeys only have one event, which means, that single event represents the beginning and end of the journey simultaneously. We can't discard the possibility of bias created by the factors we described in the previous paragraph, but we have no means to sort out which of these journeys are really made of a single event, and which of

these records are the result of fragmented journeys across multiple conversion IDs. Therefore, the combination of *ConversionID* and *EventDate* constitutes the primary key of our dataset.

MediaBuyChannel: Defines the media channel type for an event unless it's a site visit. Therefore, the typical values for this field are "Video", "Social", "Billboard" or "Search". Some of these channel nomenclatures tend to be correlated to the big players in the digital advertising industry. For example, while Google is not the only search engine in the web, a "search" event has a high probability of stemming from Google. Likewise, there are other social networks besides Facebook, but they are still by far, the largest source for "social" impressions. Billboards, or banners, are more fragmented across site networks, so while there is some aggregation of these networks through platforms like Xaxis, we could have hundreds of sites, affiliated to these networks, serving banners or video. The latter has a big connotation with YouTube, as a dominant platform.

SiteVisit: In our dataset, the *SiteVisit* field only displays values when an event represents a site visit. In these cases, the value of the *SiteVisit* field will be the channel of the touchpoint that preceded the visit. We only observed three values for this field: "OTHER", "SOCIAL" and "SEARCH". We understood the data platform where this information was recorded did not provide more detail about the site visit source. Thus, we had to assume the preceding touchpoint was the last recorded source, in the absence of a structured record and a timestamp we could work with. Moreover, we are working with a dataset containing only journeys which ended in conversion. As we'll observe later, in the transition matrix supporting the Markov values, 54% of site visit events will transition to a conversion state. However, looking at the sources of conversions, only 28% stem from a site visit, with the remaining share spread across video (53%), social (11%) and billboard (8%).

3.2. DATA REFRESH OPTIONS

The dataset GroupM provided for this project, can be used for an ad-hoc analysis of an executed campaign. This is why our dataset does not need to be refreshed at regular intervals. However, for future usage with running campaigns, data should be updated at regular intervals, daily or even hourly. Depending on format and size, we have different ways to update data efficiently.

Full reprocess is the simplest way when the size of the dataset and its foreseeable size until the end of campaign is reasonably small. This means the dataset can be fully reprocessed without significant time or compute expense. This is a practice that simplifies the development and ensures data integrity, without needing a synchronization mechanism. Regardless of how the source data is delivered, the process will fetch all the data available. **This is the method we use in this project**.

It's a common scenario to have daily source files with a backward correction **rolling window**, where the source systems for campaign logging deliver flat files on a periodic basis. This periodicity is usually daily, but for large datasets it can be hourly. Hence, files are made available on a file share and will have a timestamp indicating the period they refer to. The rolling window is used because there are frequent delays in data updates from publishers, which implies making retroactive updates to files from previous periods. The process would need to look for files on the source system that have been updated since the last integration process took place. This would naturally flag the past day or hours but could also bring up files from older periods. To make this type of synchronization possible, the integration method should append extra fields to the dataset records, to reference the source file and

its last update. With this information, it's possible to check which set of records needs to be deleted from the current dataset before they get inserted back from the updated file.

Data can also be delivered through APIs or real-time publish-subscription methods. In these cases, we can't rely on time-stamped files to build our reference key. Hence, a composite key needs to be created on the dataset, combining the event id with user id and a timestamp. This composite key will index the dataset and enable the integration process to check if the record is receiving is new or is an update. If it's an update, the previous record is deleted before the new one is inserted. When dealing with very large datasets, this method is more efficient than the previous ones, since less input-output is performed, but it's harder to implement and prone to error. API-based source systems usually publish their rolling-window timelines, so advertisers can know how far back they can expect to reach out for updates. As an example, Facebook's update rolling window has a maximum timespan of 28 days, after which, no more updates are expected to be published.

3.3. DATA PREPARATION

We start from the tabular dataset provided to us where each record will represent an event. Each event will be composed of several attributes, separated by a delimiting character, in equal number for each record. The first row represents a header, where the names of the columns, or fields, are separated by the delimiting character. In most cases, this is the common data shape received from source systems, where parameters can be provided to set the delimiting character or to specify quotes to enclose strings. At the time of this writing and considering the shape of the dataset we received from GroupM – a comma separated text file – we can easily import the content of this file into a Pandas Dataframe. For other datasets we may work with in the future – for example graph-based data written in object notation – an adapter package will be required to import this format into a Dataframe.

The data source of an attribution model needs a to have minimum set of attributes, or **essential fields**, in order to work, so we'll need to map specific fields of our dataset that correlate to these attributes:

- CONVERSION_ID: a unique identifier for each user that will enable us to single out every journey for each person
- EVENT_DATE_TIME: a timestamp with date and time precision to the second, that will enable us to construct a timeline connecting every event from start to conversion
- EVENT_TYPE: this is usually one of three possibilities. An Impression or view, a click, and a visit. Depending on the model, the nature of these events can be more detailed.
- CONVERSION_FLAG: usually marks specific events as conversions, signaling the end of the journey. Depending on the source system, they can assume a null value indicating no conversion has occurred yet. For our dataset, this field does not exist as all journeys are considered as converters.
- CHANNEL: the media channel where the event took place, usually represented as Display, Paid Search, Social etc.

If available, other attributes could be used to enrich an attribution model and expand the possible findings from a richer dataset:

- CREATIVE: usually refers to the name or description of the advertising creative item. Other fields may also relate to this one expanding details about the creative item, such as size, message, format, target group etc.
- PLACEMENT: usually describes the type of placement within the advertising channel. In the case of a display ad, the placement on the webpage often yields different results in visibility, interaction, or awareness.
- REFERRAL: the site where a visit originated. As previously noted, it's often a rich source of information that can also be reshaped into categories, depending on the site type.

As explained in the Dataset description section, we will import the fields *ConversionID*, *EventDate*, *MediaBuyChannel* and *SiteVisit* from GroupM's dataset. These are the **essential fields**, that correlate to the attributes we just described.

After the field mapping, we need to **verify if the data has the required integrity**. This includes consistency verifications between the required data types and those of each field we mapped from our dataset, including verifications for:

- Missing values: for primary key fields, this process will discard the records with missing values, as we can't infer keys from these; for non-key fields, records can either be discarded or missing values can be replaced by hardcoded values in the respective columns, if the other attributes of the record are still valuable for analysis
- Data format: some fields require intermediate steps to transform data types into the expected format to be computed as such. Typical examples are date and time fields that require parsing and decimal values where separators need to be encoded on a different locale
- Attribution rules: this process singles-out orphaned records where a typical consumer journey cannot be inferred. This is a common problem on these datasets and one of the most frequent causes are browser settings for deleting cookies at the end of a session, which will orphan events and often generates new a new ID for the same individual

Hence, in this data preparation phase, where we were working with the dataset as we received it, we performed these steps to shape our dataset, so it would conform with the previously explained rules:

- Selection and mapping of the required fields: *ConversionID*, *EventDate*, *MediaBuyChannel*, *SiteVisit*
- Date parsing of the field *EventDate* to use date and time as a sequencing and sorting method
- Sorting (ascending) by *ConversionID* and *EventDate* to enable sequential operations along the vertical axis
- Deletion of duplicated rows found while analyzing the data. The original assumption was these should not exist, but 6 duplicate rows were found on our 50k row dataset. The criteria for identifying and deleting these rows was *ConversionID* and *Event date*. More detail can be found in *Code Excerpt* 5, in the annexes chapter.

3.4. DATA PROCESSING AND OUTPUT

ATRIO will deliver results using tables and rich visualizations embedded on a Jupyter Notebook. One of the most compelling reasons to choose Plotly.py for visualization, besides being open-sourced, is the rich set of data visualization components it offers and the interactivity these components provide.

Interactivity means the user can interact with the visualizations and perform filters, groupings, and drilldowns, but also, the capability of setting parameters dynamically, using Jupyter Widgets. This type of functionality is comparable to the Shiny framework for the R language.

A summary table with the results from all heuristics and algorithms will be presented, along with an interactive Sankey Chart to show the path combination of the consumer journeys. Additionally, ATRIO will also show the distribution of consumer journey by number of touchpoints, which provides an immediate insight about the pattern of consumer journeys.

Attribution model calculations will then apply probabilistic approaches to evaluate the contribution of each channel, based on the cooperative game theory (Shapley & Aumann, 1968), Markov chains (Markov, 1907) and simple heuristics (Google Inc, 2020). The interface will also allow the configuration of threshold parameters and attribution model types. Hence, the output of these calculations will consist of a set of tables and interactive charts, that will allow analysts to compare the different approaches.

To deliver the **Simple Heuristics**, ATRIO will apply the models Google presents in their document "About the default Multi-channel Funnels Attribution Models" (Google Inc, 2020) :

- **Last Interaction**: the last touchpoint will receive all the credit for the conversion
- **First Interaction**: the first touchpoint will receive all the credit for the conversion
- **Linear**: each touchpoint in the conversion path will have equal credit for the conversion
- **Time Decay**: the touchpoints closest in time to the conversion get most of the credit
- **Position Based**: 40% credit is assigned to each the first and last touchpoints, and the remaining 20% is distributed evenly among the middle ones

To implement the Cooperative Game Theory, ATRIO will compute the credit to allocate to each channel, based on the **Shapley value**. This approach has the advantage of accounting for the marginal contributions of each channel. From the channel rank towards conversion perspective, it will be possible to compare the differences between the results of this method and the simple heuristics.

ATRIO will also deliver an analysis based on Markov Chains. The expected result is the probability of users transitioning from a certain channel to another along the path to conversion. Within the same principles as the Shapley value, but under a different algorithm, ATRIO will be able to compute the marginal contribution for conversion from each channel, defined as **Markov value**.

After performing the calculation for heuristics and algorithms, ATRIO will present a table where the results of channel contributions towards conversion can be compared. For each section – heuristics, Shapley, Markov – ATRIO will deliver rich visualizations for analysis. It will be possible to interact with filter parameters in real time, to observe the effects of different filter combinations on the results. For this, ATRIO will present three interactive controls to set the journey sequence order, size, and filter option for journeys with more touchpoints than the selected size.

Chapter 4 will provide the technical details of the Shapley and Markov algorithms implementation, and Chapter 5 will present a use-case of how these algorithms' results can be interpreted, from an end-user perspective.

4. PROJECT IMPLEMENTATION

This chapter will cover the full implementation of ATRIO from a technical perspective. As previously explained, we'll be using Jupyter Notebooks to perform the implementation of our framework, so we'll use technical language and a logical approach to explain how the code is implemented and deployed. This chapter is oriented for those who will need to deploy ATRIO, or as a reference to interpret the code implementation.

The "End-user perspective" chapter will follow, with a business-oriented perspective towards end-users, so they can make the most of ATRIO. There, we will describe ATRIO's interface, parameters, and outputs, providing an analytic context to query the dataset and interpret the results.

ATRIO is coded into a Jupyter Notebook, written in Python. The main advantage of Jupyter is the ability to present data transformation and analysis processes in a rich format, including interactive visualizations. The Code Excerpts we'll reference, were extracted from the source notebook and published in the Annexes chapter. Since the notebook code is commented, the details about the logical implementation of each cell's functional purpose are explained in the comments. Comments will not be used in statements where the language implementation, the variable naming and assignment are self-explanatory and easily interpretable. According to Python syntax, comments are preceded by a hashtag. The syntax highlight system we use is Visual Studio Code's 2019 Light Theme, so Python comments are highlighted in green. Colors[3] represent functions, variable names, variable values, and other syntactic elements.

4.1. LIBRARIES AND GLOBAL VARIABLES

To start our implementation, we'll need three libraries: Pandas, NumPy and Plotly. Let's briefly explain what these libraries do, and then discuss the reasons to choose Voilà as a web rendering framework and Heroku's platform, to deploy our application and make it available to any user through a web browser.

Pandas and NumPy are the most common choices to perform the data transformations and data analysis for this type of project. They package most of the functionality that would otherwise have to be written from scratch. As we'll explain later, the cornerstone of our data transformation is the Dataframe object from the Pandas library.

We used other libraries besides Pandas and NumPy, to ease our work for handling data structures such as dictionaries, collections, and string manipulations. These libraries provide good shortcuts for iterating, manipulating, parsing, and converting these data structures and they can be found *Code Excerpt* 1.

Our choice to plot the results was the open source version of Plotly. The main reasons behind this choice, were good compatibility with Jupyter and Voilà, and the wide variety of charts and customization options, namely the Sankey Diagram, an essential visualization for the consumer journey. Matplotlib is also a common choice for these projects, but we found some limitations for the visualizations we needed, namely the Sankey Diagram. In *Code Excerpt* 1, we start by importing all the

[3] https://docs.microsoft.com/en-us/visualstudio/python/editing-python-code-in-visual-studio?view=vs-2019#code-coloring

libraries we will use in our code and then declare the four global variables we'll need. There is a coding pattern consensus in the Python community, to avoid the use of global variables. After some attempts to modify and reload dataframes invoked outside the context of their cell, it became obvious the workaround would be cumbersome. Therefore, we declared the main dataframe *df* containing our dataset, and the transformed dataframe versions to hold the transformed data for our algorithms, as global variables. By sharing these dataframes as global variables, we can react dynamically to the parameter inputs from the user interface widgets and invoke these objects in the code functions maximizing re-usability and avoiding repetition. These are the only variables we needed to declare as global. All others, as we'll see through the following sections and chapters, are declared within the scope of their cell, such as Plotly Graph Objects, Jupyter Widgets, and custom functions, but become accessible for reading and writing from other notebook cells, after we instantiate them.

4.2. USER INTERFACE CREATION

ATRIO's interface is composed by two Jupyter Widgets Tab objects inside one another. Here is the hierarchy of tabs and objects:

```
Home Tab
  Dataset load controls
  Dataset head sample and reference format
Analysis Tab
  Interactive Widgets for algorithm parameter settings
  Journeys Tab
    Summary table, consumer journey diagram and touchpoint frequency plot
  Heuristic Tab
    Heuristic analysis descriptive table
  Shapley Tab
    Shapley Value results and Coalitions Table
  Markov Tab
    Markov Chain results, Transition Matrix and Diagram
```

In *Code Excerpt* 2, we declared all the widgets and plot elements with their layout and format options, starting with the inner elements of the hierarchy, and progressing to the outer elements acting as containers. The use of Jupyter Widgets vertical and horizontal boxes proved quite useful to align the elements in the correct order. These provide an abstraction to CSS (cascading style sheets) used to format HTML, which is often challenging. In situations where the default format was not suitable, we were able to override it with a layout dictionary. For example, the default spacing between Plotly objects was too large by default, so we created a reusable layout dictionary for all Plotly charts and tables, as shown in *Code Excerpt* 3.

4.3. THE CHOICE OF THE VOILÀ FRAMEWORK

In our research to implement a framework to render our interactive notebook we considered several options:

1. Using Jupyter server as-is it would have been the easiest way, with the drawback of always rendering the Jupyter environment, exposing the whole cell sequence, including the code.
2. Implementing Streamlit was the original option as it provides a rich environment to develop Python-based data analytics and dashboarding solutions. After a few promising tests, we discovered some drawbacks, mostly related to the complexity of managing callbacks between

a client-server architecture. For a more demanding solution, this would have been a good choice and their model is quite similar to other solutions such as Shiny and Dash.

3. Our final choice was the Voilà framework, for simplicity. Since all the code is based in Jupyter, Voilà is the simplest way to serve Jupyter notebooks. While it started as a separate project, it is now part of the Jupyter package as a native component. All the complexity of callback code is abstracted, and it easily renders Jupyter Widgets and Plotly charts and tables. The second reason for choosing Voilà was the ease to deploy it into a public web environment. For our project, we chose the free tier of Heroku, as the content can be set private and the deployment is made with simple GIT commands. All the complexity of how the application is packaged into a container is completely abstracted and requires only minor configurations as we'll explain later. Voilà can also be deployed to Binder, although the repository seems to be public, given the nature of the project and also to Google App Engine, where we did not find a free tier, but should be a suitable environment for a production solution.

4.4. IMPORT SOURCE DATA INTO THE PANDAS DATAFRAME

A Pandas Dataframe[4] is a 2-dimensional labeled data structure with columns of potentially different types. In cell # 3, we'll import the columns we need from our csv file and set the parameters to define the column separator, main type, Boolean for interpreting missing values, and the variables containing our columns and which of those are to be parsed as dates. *Code Excerpt* 4 shows how to perform the data load. In a single method call, we're able to load the content of the .csv file into the Dataframe by defining the file name, the columns we want to extract from the file and which of these we can safely parse into dates.

In Section 3.1 we described all the columns available in our dataset, as provided by the campaign logs, we obtained from GroupM. In that section, we also discussed which would be the essential fields we need to create an attribution model. Therefore, by declaring only the columns we need the import process will ignore the remaining columns when loading the dataset, avoiding unnecessary data.

4.5. TRANSFORM DATA: EVENT SEQUENCE, SIZE, SOURCES, TARGETS, CLASSIFICATION

To calculate simple heuristics or the more advanced algorithms we'll describe in the next chapters, we'll need to transform, clean, and enrich our data. The following sections present all the operations required to derive our first metrics.

The first operation is to create two new columns: *EventSequence* and *SequenceSize*. These columns are based on the sorted sequence of *ConversionID* and *EventDate*. We'll start by sorting our dataframe by these two columns, to obtain a list of records to infer groups of conversions where every step of the journey is sorted from first to last event, by its timestamp. *EventSequence* enumerates each event in ordinal sequence and *SequenceSize* represents the total number of events for each conversion, so the value of the last event in the sequence is equal to the sequence size. As other variables we'll add, note that *SequenceSize* is denormalized, which would be sub-optimal in a transactional system. However, for analytic purposes, we can afford this redundancy as it facilitates the row-based and index-based offset calculations we'll need to perform in the next operations, shown in *Code Excerpt* 5.

[4] https://pandas.pydata.org/pandas-docs/stable/user_guide/dsintro.html#dataframe

The result of this transformation is represented in Table 3

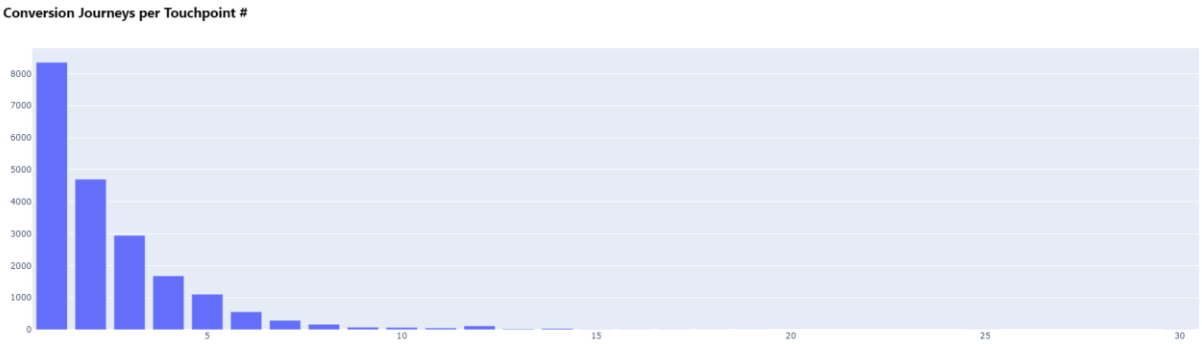
index	ConversionID	EventDate	[…]	EventSequence	SequenceSize
0	508218723740	2017-10-10 01:59:26.939	[…]	1	1
1	508218723741	2017-10-10 01:59:26.939	[…]	1	1
[…]	[…]	[…]	[…]	[…]	[…]
7	508264379526	2017-10-10 14:55:26.827	[…]	1	2
8	508264379526	2017-10-10 14:56:07.731	[…]	2	2
9	508264379527	2017-10-10 14:55:26.827	[…]	1	2
10	508264379527	2017-10-10 14:56:07.731	[…]	2	2
11	508273478197	2017-10-10 10:09:48.230	[…]	1	5
12	508273478197	2017-10-10 10:10:16.584	[…]	2	5
13	508273478197	2017-10-10 10:21:43.235	[…]	3	5
14	508273478197	2017-10-10 10:25:18.770	[…]	4	5
15	508273478197	2017-10-10 10:29:07.998	[…]	5	5
[…]	[…]	[…]	[…]	[…]	[…]
50227	514715334593	2017-10-11 22:29:52.274	[…]	1	3
50228	514715334593	2017-11-08 23:46:32.964	[…]	2	3
50229	514715334593	2017-11-08 23:47:38.871	[…]	3	3

During the discovery phase, we identified variables where we could reduce their cardinality by re-classifying them, without losing relevant detail. As a result, we now have a more readable output. Moreover, to get a better perspective over the consumer journey, we transformed our dataset to highlight the source and the target for each event by creating a target column as a copy of the source, shifted 1 row up. Since this needed to be well documented in the code comments, we can interpret the problem and the solution from *Code excerpt* 6 and its comments.

Now that we have source and target at the row level, we're able to perform the necessary calculations for our new columns: *AggregatedSource*, *AggregatedTarget*, *AggregatedSequence*. These columns will be used by all the algorithms and heuristics going forward, so we need to understand why we need them and how to calculate them.

As the number of touchpoints in a consumer journey can vary, ATRIO is capable of aggregating the number of touchpoints, depending on the maximum number of touchpoints the user selects in the interface. This variable is initialized by calculating the maximum number of touchpoints in a consumer journey, for a given campaign. In our current dataset, this number is 30. However, looking at the frequency distribution for number of touchpoints per journey we quickly realize only 3 journeys have 30 touchpoints, which is hardly representative, in a total of 20.212 journeys. Looking at the distribution chart in Figure 1 presented next, we can observe a long-tail distribution where 2/3 journeys have less than 3 touchpoints:

Figure 1 shows the journeys up to 5 touchpoints account for 93% of the total.



Considering this, ATRIO lets the end-user choose how to aggregate the dataset, based on the maximum number of touchpoints to be considered. Along with this parameter, the user can also choose to discard the journeys which have more than n touchpoints or to include them, but discarding the touchpoints > n. We will discuss the usage of these parameters in the next chapter but, we'll briefly mention the user can also choose to invert the sequence of touchpoints, for the cases where "recency" criteria need to be evaluated. With these parameters in mind we can analyze the criteria to calculate the aggregated variables.

AggregatedSource will equal the value of the *Source* column when the touchpoint # is not higher than the max_sequence parameter. The remaining records are filled with empty string.

AggregatedTarget will equal the value of the *Target* column when the touchpoint # is not higher than the max_sequence parameter. Additionally, if the last touchpoint of the sequence is within the max_sequence parameter its value will be hardcoded as "CONVERSION". The remaining records are filled with empty string. For datasets containing "non-converters" this rule will need to be adapted.

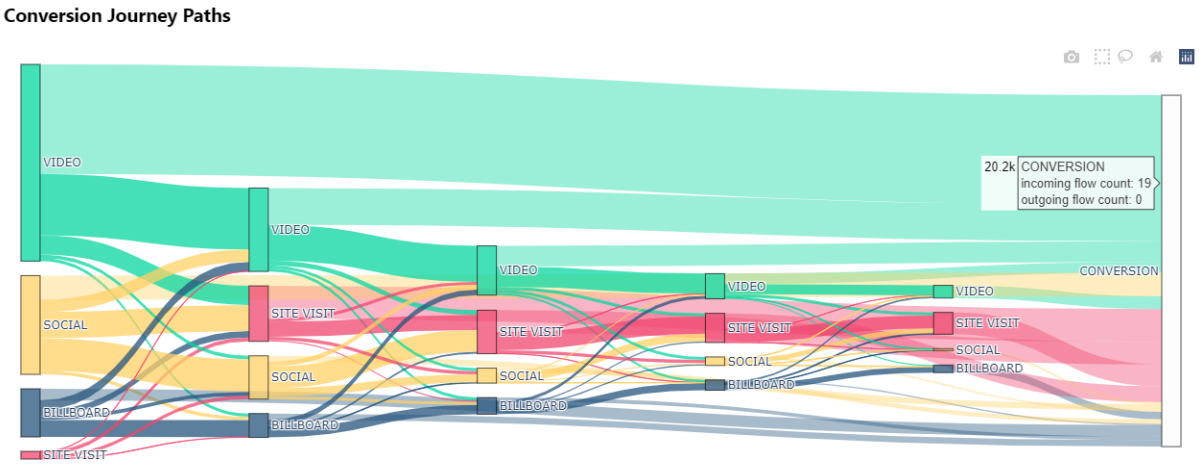
AggregatedSequence will equal the value of the *EventSequence* column when the touchpoint # is not higher than the max_sequence parameter. The remaining records are filled with empty string. *Code Excerpt* 7 demonstrates how to calculate these variables with the *first-to-last* analysis option.

Table 4 shows a snapshot for a sequence of 7 touchpoints where max_sequence equals 5:

EventSequence	*SequenceSize*	Source	Target	AggregatedSource	AggregatedTarget	AggregatedSequence
6	6	VIDEO	CONVERSI…			
1	7	BILLBOARD	BILLBOARD	BILLBOARD	BILLBOARD	1
2	7	BILLBOARD	VIDEO	BILLBOARD	VIDEO	2
3	7	VIDEO	BILLBOARD	VIDEO	BILLBOARD	3
4	7	BILLBOARD	BILLBOARD	BILLBOARD	BILLBOARD	4
5	7	BILLBOARD	VIDEO	BILLBOARD	CONVERSION	5
6	7	VIDEO	VIDEO			
7	7	VIDEO	CONVERSI…			
1	3	VIDEO	VIDEO	VIDEO	VIDEO	1
2	3	VIDEO	VIDEO	VIDEO	VIDEO	2
3	3	VIDEO	CONVERSI…	VIDEO	CONVERSION	3
1	1	VIDEO	CONVERSI…	VIDEO	CONVERSION	1

4.6. VISUALIZE JOURNEY PATHS WITH SANKEY CHARTS

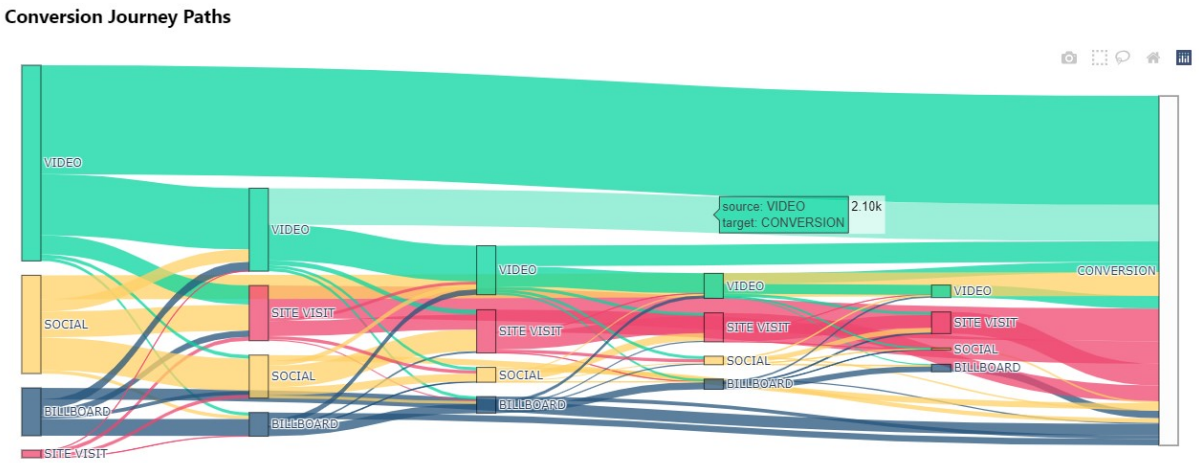
The Sankey chart in Figure 2 is the most illustrative visualization of the conversion journey for Attribution Models:



In this example, the user set the maximum number of touchpoints to 5 and kept the partial journeys. This means, we don't show touchpoints ranked over #5 but we account for all the conversions and assume the fifth touchpoint as the last touchpoint for journeys over 5 touchpoints. This is a common technique in Attribution model analysis, to reduce clutter from the visualizations by filtering outliers. Depending on the perspective, analysts can argue about the importance of first touchpoints over last ones. This is why ATRIO has the option to reverse the rank of touchpoints from last to first to show a last-to-first perspective.

The Sankey chart can also reveal how many paths are incoming to a particular node and how many stem from that node. Looking at the previously shown Figure 2, we observe the "Conversion" node results from 19 path combinations flowing in.

Figure 3 reveals 2100 conversions stem from path combinations where the first and second touchpoints were Video displays, by hovering the mouse over an edge between two nodes.



To create this Sankey diagram using Plotly[5], we'll implement a function to copy the data we need from the main Dataframe containing our dataset, into a temporary Dataframe to apply the required transformations. These transformations are needed to create the data structures for nodes and links.

Nodes correspond to touchpoints in the journey and are shown in blue on the chart. Their data structure is a Python dictionary composed of several keys to define the properties of the nodes. The most important key is the "label", which can hold a Python list, NumPy array, or Pandas series of numbers, strings, or datetimes.

Links provide the necessary information to connect the nodes and are shown in gray on the chart. They are also structured as a Python dictionary with keys defining their properties. We will only outline the three fundamental keys for a link to connect nodes: "source", "target", and "value". The structure of these keys is the same as the "label", thus Python list or equivalent.

"source" takes a list of integer values to reference the index of the node inside the "label" key where the flow is coming from.

"target" takes a list of integer values to reference the index of the node inside the "label" key where the flow is going to.

"value" takes a list of numeric values to set the flow volume value, so in our case, this will represent the count of records for the corresponding source and target.

We will start our Sankey chart function by creating a temporary dataset grouping the aggregated columns we created earlier, and applying the transformations as commented in *Code Excerpt* 8, creating the data structures we need to build our Sankey chart. Let's look at the content of dataframes *df_labels*, *df_paths* and the *labels* array after the last transformation has been done, considering a maximum of 5 touchpoints in the journey filter:

```
# df_labels for first 5 touchpoints
    SourceLabelIndex         Source
0                  0   BILLBOARD_00
1                  1   BILLBOARD_01
2                  2   BILLBOARD_02
3                  3   BILLBOARD_03
4                  4   BILLBOARD_04
5                  5     CONVERSION
6                  6  SITE VISIT_00
7                  7  SITE VISIT_01
8                  8  SITE VISIT_02
9                  9  SITE VISIT_03
10                10  SITE VISIT_04
11                11      SOCIAL_00
12                12      SOCIAL_01
13                13      SOCIAL_02
14                14      SOCIAL_03
15                15      SOCIAL_04
16                16       VIDEO_00
17                17       VIDEO_01
18                18       VIDEO_02
19                19       VIDEO_03
```

[5] Visit https://plotly.com/python/sankey-diagram/ for more information. Quoting their page: "Sankey diagrams visualize the contributions to a flow by defining source to represent the source node, target for the target node, value to set the flow volume, and label that shows the node name."

```
20               20       VIDEO_04

# labels array for first 5 touchpoints
['BILLBOARD', 'BILLBOARD', 'BILLBOARD', 'BILLBOARD', 'BILLBOARD',
 'CONVERSION', 'SITE VISIT', 'SITE VISIT', 'SITE VISIT',
 'SITE VISIT', 'SITE VISIT', 'SOCIAL', 'SOCIAL', 'SOCIAL', 'SOCIAL',
 'SOCIAL', 'VIDEO', 'VIDEO', 'VIDEO', 'VIDEO', 'VIDEO']

# df_paths for first 5 touchpoints
          Source        Target EventSequence  counts  SourceLabelIndex  TargetLabelIndex
0   BILLBOARD_00  BILLBOARD_01             1     975                 0                 1
1  SITE VISIT_00  BILLBOARD_01             1      17                 6                 1
2      SOCIAL_00  BILLBOARD_01             1     200                11                 1
3       VIDEO_00  BILLBOARD_01             1     181                16                 1
4   BILLBOARD_00    CONVERSION             1     664                 0                 5
..           ...           ...           ...     ...               ...               ...
77      VIDEO_03     SOCIAL_04             4      47                19                15
78  BILLBOARD_03      VIDEO_04             4     101                 3                20
79 SITE VISIT_03      VIDEO_04             4      15                 9                20
80     SOCIAL_03      VIDEO_04             4      36                14                20
81      VIDEO_03      VIDEO_04             4     575                19                20
```

With all the data we need, we can now populate the Sankey diagram as shown in *Code Excerpt* 9. In this step, we'll create *FirstInteraction*, *LastInteraction*, *LinearInteraction*, *PositionBased* and *TimeDecayBased* metrics. All our new columns are documented in *Code Excerpt* 10. First, we'll need to create a temporary Dataframe to transform the data we need. Again, this will be copied from a view of the main Dataframe. We're taking the records from the main Dataframe where aggregated source and sequence columns are not empty and selecting these columns and the sequence size. After our new *heuristic* Dataframe is created, we can create the heuristic columns as shown *Code Excerpt* 12, *TimeDecayBased* metrics can be subject to several approaches. In this case, to simulate decay, we used the linear equation $y = mx + b$ where $m = 2$ and $b = 0$. Events are distributed evenly along the (x) axis, so each one takes the corresponding (y) value as contribution. In a future release, Atrio could incorporate other models to calculate decay over time, allowing an analyst to implement a function with more fitness to model how memory fades over time. In such model, the event timestamp should also be considered as an additional parameter since the linear model we currently implemented is assuming an equal event distribution over time.

The new heuristic columns are represented in Table 5

index	[…]	FirstInteraction	LastInteraction	LinearInteraction	PositionBased	TimeDecayBased
0	[…]	1,00	1,00	1,00	1,00	1,00
1	[…]	1,00	1,00	1,00	1,00	1,00
[…]	[…]	[…]	[…]	[…]	[…]	[…]
7	[…]	1,00	0,00	0,50	0,50	0,33
8	[…]	0,00	1,00	0,50	0,50	0,67
9	[…]	1,00	0,00	0,50	0,50	0,33
10	[…]	0,00	1,00	0,50	0,50	0,67
11	[…]	1,00	0,00	0,20	0,40	0,07
12	[…]	0,00	0,00	0,20	0,07	0,13
13	[…]	0,00	0,00	0,20	0,07	0,20
14	[…]	0,00	0,00	0,20	0,07	0,27
15	[…]	0,00	1,00	0,20	0,40	0,33
[…]	[…]	[…]	[…]	[…]	[…]	[…]
50227	[…]	1,00	0,00	0,33	0,40	0,17
50228	[…]	0,00	0,00	0,33	0,20	0,33
50229	[…]	0,00	1,00	0,33	0,40	0,50

The final step, is to calculate and append totals to our new Dataframe and load its data into a Plotly Table, as described *Code Excerpt* 13. We needed to transpose the result before loading into the Plotly Table, as a convenience method to match the expected data shape. Additionally, some workaround customization was applied to ensure a correct table height and width, by matching the width to the Sankey chart, as the Table does not autofit in the same automated way as the chart.

4.7. MULTI-TOUCH ATTRIBUTION

In this section, we'll show the implementation of the two algorithms we discussed for multi-touch attribution: Shapley Game Theory and Markov Chains. After a long research of several coding approaches, we based our algorithm in the Python library built by Igor Korostil, named MTA, hosted in GitHub repository https://github.com/eeghor/mta (Korostil, 2020).

The reason to adapt code from an existing library instead of coding all the implementation of these algorithms from zero, has practical reasons. It's not the purpose of this project to develop an original implementation of these algorithms. The purpose of ATRIO is to provide a framework where these algorithms and other functionalities can be packaged into a single application and delivered to GroupM, as a tool to solve an existing business problem.

We hereby disclose that part of the code presented in this chapter is not original and was adapted from the mentioned repository. The original code is licensed under MIT, as available in https://pypi.org/project/mta/, so we are free to use, modify and distribute.

However, while we could have imported the MTA library as a reference in our code, we opted to understand the core logic behind the calculation of the Shapley and Markov algorithms and reuse it with our own modifications where needed. This means, ATRIO does not distribute a modified version of the library as such, but reuses part of their logic inside its functions.

4.7.1. Conversion sets for Shapley values

As mentioned in section 3.5.2, ATRIO will compute the credit to allocate to each channel, based on the Shapley value. We'll take the same approach from the previous chapter and walk through the code, to explain how the Shapley values are computed and how this calculation attributes a specific credit to each channel. This credit is the result of the channel's intrinsic contribution plus the marginal contribution to coalitions with the other channels, from each consumer journey.

The first step is to compute the conversion sets. Actually, the implementation is performed on a common function to both Shapley and Markov algorithms, that we are showing in *Code Excerpt* 14, with a small difference we'll explain. This would be the result of this function for Shapley, when applied with the same parameters as the previous examples:

```
ConversionID
508218723740             [BILLBOARD]
508218723741             [BILLBOARD]
508240210132                [SOCIAL]
508240210133                [SOCIAL]
508240210134                [SOCIAL]
                    ...
514704439686    [SITE VISIT, SOCIAL]
514704439687    [SITE VISIT, SOCIAL]
514715334591     [SITE VISIT, VIDEO]
```

```
514715334592     [SITE VISIT, VIDEO]
514715334593     [SITE VISIT, VIDEO]
Length: 20212
```

To calculate Shapley values, we want our function to return unique channel values per list item. We'll see how this is different for Markov chains and why. Next, let's see the result when we group the previous list with a count:

```
channels
['BILLBOARD', 'SITE VISIT', 'SOCIAL', 'VIDEO']      48
['BILLBOARD', 'SITE VISIT', 'SOCIAL']              197
['BILLBOARD', 'SITE VISIT', 'VIDEO']               123
['BILLBOARD', 'SITE VISIT']                        473
['BILLBOARD', 'SOCIAL', 'VIDEO']                   204
['BILLBOARD', 'SOCIAL']                            199
['BILLBOARD', 'VIDEO']                             979
['BILLBOARD']                                     1231
['SITE VISIT', 'SOCIAL', 'VIDEO']                  373
['SITE VISIT', 'SOCIAL']                          3100
['SITE VISIT', 'VIDEO']                           1467
['SITE VISIT']                                       3
['SOCIAL', 'VIDEO']                                775
['SOCIAL']                                        1765
['VIDEO']                                         9275
```

4.7.2. Channels and Coalition Values

The conversion sets list is all the input we need to calculate the Shapley value for each channel. To do this, *Code Excerpt* 14 will take this list and create three fundamental objects from it:

shapley_channels: list of unique channels

coalition_values: dictionary of values for each coalition

shapley_values: dictionary of values for each channel, based on Shapley values for each coalition

Calculating the coalition values is more involving and requires the use of two auxiliary functions as described in *Code Excerpt* 15. By iterating through all the subset combinations that we can obtain from the *shapley_channels* list we can then parse all these combinations though our conversions list. This how we obtain the coalition values. At this point, *shapley_values* is still an empty dictionary, but we can already observe the content of *shapley_channels* and *coalition_values*:

```
# shapley channels
['BILLBOARD', 'SITE VISIT', 'SOCIAL', 'VIDEO']

# shapley coalition values
['BILLBOARD']                                     1231
['SITE VISIT']                                       3
['BILLBOARD', 'SITE VISIT']                       1707
['SOCIAL']                                        1765
['BILLBOARD', 'SOCIAL']                           3195
['SITE VISIT', 'SOCIAL']                          4868
['BILLBOARD', 'SITE VISIT', 'SOCIAL']             6968
['VIDEO']                                         9275
['BILLBOARD', 'VIDEO']                           11485
['SITE VISIT', 'VIDEO']                          10745
```

```
['BILLBOARD', 'SITE VISIT', 'VIDEO']            13551
['SOCIAL', 'VIDEO']                             11815
['BILLBOARD', 'SOCIAL', 'VIDEO']                14428
['SITE VISIT', 'SOCIAL', 'VIDEO']               16758
['BILLBOARD', 'SITE VISIT', 'SOCIAL', 'VIDEO']  20212
```

4.7.3. Calculating Shapley values

Using the data from *shapley_channels* and *coalition_values*, we can proceed to the calculation of Shapley values. *Code Excerpt* 16 shows where the Shapley values for each channel are calculated, by iterating each coalition through each channel. This code returns the Shapley value for each channel, by taking the weighted average of the channel's contribution in every coalition. The calculation accounts for the marginal contributions and also the channel's individual contribution for the final result:

```
# shapley values
{'BILLBOARD': 2243,
'SITE VISIT': 2766,
'SOCIAL': 4072,
'VIDEO': 11130}
```

Just as in the previous example for rendering the Plotly Table with the values from the heuristics calculation we took the same approach to calculate a "Total" row and merge it on the final result. As the code is quite similar and has already been explained we won't reproduce it here. Besides the Shapley values for each channel, we have also added a table to display the number of journeys for each coalition which provides an insightful view over the frequency of the channel coalitions.

4.7.4. Grouped conversion sets for Markov values

We'll now go through all the steps involved in the Markov Values calculation. We'll end by showing the contribution of each channel, after applying the Markov algorithm, but also, describing the importance of the Transition Matrix as an essential component of Markov Chains.

At the beginning of this section, we explained how to compute the conversion sets for Shapley and Markov algorithms and mentioned a small difference in the output, depending on the calling algorithm. For the Shapley calculation we applied a function to return unique channel values per list item. To calculate our conversion sets for Markov, we won't do that, because we don't want to aggregate the result for repeated channels and we also won't apply any sorting function, so we can later map all the transitions between the channel touchpoints, preserving their organic sequence, for each journey. This is a sample of the result of the conversion sets calculation for Markov:

```
ConversionID
508218723740                       [BILLBOARD]
508218723741                       [BILLBOARD]
508240210132                          [SOCIAL]
508240210133                          [SOCIAL]
508240210134                          [SOCIAL]
                                ...
514704439686       [SOCIAL, SOCIAL, SITE VISIT]
514704439687       [SOCIAL, SOCIAL, SITE VISIT]
514715334591                [VIDEO, SITE VISIT]
514715334592                [VIDEO, SITE VISIT]
```

```
514715334593    [VIDEO, SITE VISIT, SITE VISIT]
Length: 20212
```

As we can observe in the last records, the sequence is maintained in the original order. If we group this list, as we did for Shapley's calculation, we'll get a much bigger list than Shapley's, as the source combinations have much higher cardinality. Here is a sample of the Markov conversion groups:

```
                                                  path  total_journeys
0    ['BILLBOARD', 'BILLBOARD', 'BILLBOARD', 'BILLB...             152
1    ['BILLBOARD', 'BILLBOARD', 'BILLBOARD', 'BILLB...               3
2    ['BILLBOARD', 'BILLBOARD', 'BILLBOARD', 'BILLB...               2
3    ['BILLBOARD', 'BILLBOARD', 'BILLBOARD', 'BILLB...              28
4    ['BILLBOARD', 'BILLBOARD', 'BILLBOARD', 'BILLB...              52
..                                                 ...             ...
335      ['VIDEO', 'VIDEO', 'VIDEO', 'VIDEO', 'VIDEO']             329
336               ['VIDEO', 'VIDEO', 'VIDEO', 'VIDEO']             280
337                        ['VIDEO', 'VIDEO', 'VIDEO']             720
338                                 ['VIDEO', 'VIDEO']            1645
339                                          ['VIDEO']            6301

[340 rows x 2 columns]
```

In the previous section, we obtained 15 conversion groups for Shapley. Markov's conversion groups return 340 records, as a result of aggregating data with more cardinality.

4.7.5. Transition Matrix calculation

Along with the conversion groups, the transition matrix is the other key element to calculate the Markov value for each channel. This matrix shows the percentages of inter-channel transitions. In other words: looking at the consumer journeys, we are calculating the ratio of transitions from each channel to the other channels, including the channel itself. This last part is called a self-transition, where the sequence of two touchpoints in the consumer journey happens within the same channel. In fact, there are many examples in the dataset where we observe repeated instances of self-transition in a single journey. In practice, this means that if we had three channels, Channel A would have a probability of p1 to transition to itself, p2 to transition to Channel B, p3 to transition to Channel C and p4 to end the journey. As there are no other options for transition, the sum of these probabilities for each channel, represents 100% of all the channel's transitions.

Figure 4 illustrates the previous explanation, with an example of ATRIO's output

Channel Transitions

source	BILLBOARD	SITE VISIT	SOCIAL	VIDEO	(end)	TOTAL
(start)	13.7%	2.2%	28.1%	56.0%	0.0%	100.0%
BILLBOARD	37.0%	9.8%	5.8%	18.9%	28.5%	100.0%
SITE VISIT	1.1%	31.0%	6.4%	2.3%	59.1%	100.0%
SOCIAL	4.5%	36.2%	25.2%	11.4%	22.8%	100.0%
VIDEO	2.6%	8.2%	2.8%	34.8%	51.6%	100.0%

In the next sub-section, we'll see why the transition matrix is needed to calculate the Markov values, but now we'll focus on how to create this matrix. To do this, we need to create two auxiliary tables first: *pair_counts* and *pair_totals*.

The *pair_counts* object will be a Python dictionary where we store the counts of each pair of channels, considering all the channel transition combinations of our dataset. By "pair of channels" we are implying a transition, so the order of the pair will determine the transition direction. The pair key (A, B) will hold the counts of transitions from channel A to channel B, while the pair key (B, A) will hold the counts of transitions from channel B to channel A. To do this:

1. We created a utility function to calculate the pairs from the conversion sets, which returns the sequential pairs of a consumer journey in the same sequence. For example, given the list [a, c, b, b, a, c, c, b] the result will be [[a, c], [c, b], [b, b], [b, a], [a, c], [c, c], [c, b]].
2. We iterate over each conversion set and calculate the pairs for each set. Still inside the parent iteration, we iterate over each calculated pair and populate our *pair_counts* dictionary with a unique key for each pair and set the corresponding count we read from the current set. Alternatively, the code increments the pair value with the current count if the pair count key was already created on a previous iteration.

Here is the resulting *pair_counts* dictionary:

```
# markov pair_counts
{('(start)', 'BILLBOARD'): 2769, ('BILLBOARD', 'BILLBOARD'): 2264, ('BILLBOARD', '(null)'
): 0, ('BILLBOARD', '(end)'): 1748, ('BILLBOARD', 'SITE VISIT'): 602, ('SITE VISIT', '(nu
ll)'): 0, ('SITE VISIT', '(end)'): 5361, ('BILLBOARD', 'SOCIAL'): 355, ('SOCIAL', '(null)
'): 0, ('SOCIAL', '(end)'): 2192, ('BILLBOARD', 'VIDEO'): 1158, ('VIDEO', '(null)'): 0, (
'VIDEO', '(end)'): 10911, ('SITE VISIT', 'SITE VISIT'): 2815, ('SITE VISIT', 'VIDEO'): 21
2, ('VIDEO', 'BILLBOARD'): 556, ('VIDEO', 'SITE VISIT'): 1724, ('VIDEO', 'VIDEO'): 7344,
('SOCIAL', 'BILLBOARD'): 435, ('SOCIAL', 'VIDEO'): 1095, ('VIDEO', 'SOCIAL'): 593
('SITE VISIT', 'BILLBOARD'): 103, ('SITE VISIT', 'SOCIAL'): 578, ('SOCIAL', 'SITE VISIT')
: 3486, ('SOCIAL', 'SOCIAL'): 2424, ('(start)', 'SITE VISIT'): 442, ('(start)', 'SOCIAL')
: 5682, ('(start)', 'VIDEO'): 11319}
```

If we sum all entries beginning with *(start)* we'll get the total number of journeys, which is 20.212.

The *pair_totals* object is also a dictionary where we summarize the counts of each channel, taking the left side of the pair as aggregating key:

```
# Pair totals
{'(start)': 20212, 'BILLBOARD': 6127, 'SITE VISIT': 9069, 'SOCIAL': 9632, 'VIDEO': 21128}
```

Now that we have our auxiliary tables, we can easily create the Transition Matrix, by dividing the counts from each pair by the total count of the channel. Here is the result:

```
# Transition Matrix result
{('(start)', 'BILLBOARD'): 0.13, ('BILLBOARD', 'BILLBOARD'): 0.36, ('BILLBOARD', '(null)'
): 0.0, ('BILLBOARD', '(end)'): 0.28, ('BILLBOARD', 'SITE VISIT'): 0.09, ('SITE VISIT', '
(null)'): 0.0, ('SITE VISIT', '(end)'): 0.59, ('BILLBOARD', 'SOCIAL'): 0.05, ('SOCIAL', '
(null)'): 0.0, ('SOCIAL', '(end)'): 0.22, ('BILLBOARD', 'VIDEO'): 0.18, ('VIDEO', '(null)
'): 0.0, ('VIDEO', '(end)'): 0.51, ('SITE VISIT', 'SITE VISIT'): 0.31, ('SITE VISIT', 'VI
DEO'): 0.02, ('VIDEO', 'BILLBOARD'): 0.02, ('VIDEO', 'SITE VISIT'): 0.08, ('VIDEO', 'VIDE
O'): 0.34, ('SOCIAL', 'BILLBOARD'): 0.04, ('SOCIAL', 'VIDEO'): 0.11, ('VIDEO', 'SOCIAL'):
 0.02, ('SITE VISIT', 'BILLBOARD'): 0.01, ('SITE VISIT', 'SOCIAL'): 0.06, ('SOCIAL', 'SIT
E VISIT'): 0.36, ('SOCIAL', 'SOCIAL'): 0.25, ('(start)', 'SITE VISIT'): 0.02, ('(start)',
 'SOCIAL'): 0.28, ('(start)', 'VIDEO'): 0.56}
```

These are the values we showed in Figure 4, where each channel sums 100% across all its possible transitions.

4.7.6. Calculating Markov values

For our final calculation, we also created an auxiliary function, described in *Code Excerpt* 17, because its logic is invoked twice along the code. This function takes three parameters: the original Markov conversion sets we created earlier, the transition matrix, and a string parameter to define the channel to be dropped. The return value is a decimal number representing a probability.

We will first use this function in a single invocation with the *drop* parameter set to empty string, meaning we use all channels to calculate a total probability for reference. Then, we'll iterate over each channel in the list, and invoke this function on each iteration passing the channel name in the *drop* parameter. This will give us the partial probabilities for the remaining group of channels, creating the removal effect discussed in the literature review. Finally, we'll take the subtraction of this probability from the total probability and divide it by the total probability, to get the result for the channel. We'll invoke this function as shown in *Code Excerpt* 18, and calculate the Markov value for each channel, obtaining these results:

```
{'BILLBOARD': 0.16,'SITE VISIT': 0.24,'SOCIAL': 0.30,'VIDEO': 0.67}
```

The last step is to normalize these results and return a Python dictionary with the Markov values and the data needed for the transition outputs, to be used in the calling function for rendering output. This normalization is implemented in *Code Excerpt 19*.

In addition to the transition matrix, we came up with the idea to develop a Transition Graph. The concept behind this visualization is to illustrate how the transitions between channels are flowing and what is the actual count of these transitions. One of the most striking evidences from this visualization, is the amount of self-transition flows. This perspective was not so obvious when we looked at the Conversion Journey Paths visualization, which aims to show the conversion paths from start to end.

The perspective from the Transition Graph is different because it provides a visual graph of the channel transitions in the Markov Chain. While this kind of visualization would normally be rendered with a Graph chart, Plotly's Graph version did not have all the needed features to represent the dimensionality of the flows – edges, in graph terminology – between the channel labels, which would be the nodes in a graph. We reused Plotly's Sankey chart to simulate a graph, because it can show the true volume of the transition flows through the thickness of the edges and their direction, by using the same color of the source node. Please refer to *Code Excerpt* 20 for implementation details.

We acknowledge *Code Excerpt* 20 needs more abstraction to work well with other datasets, but still, we believe it's an important addition to this project, and worth improving in future versions. The remaining code is similar to what we already presented for rendering the output, in previous chapters, so we won't discuss it. The output is composed by two tables with totals – one for the Markov values by channel and the Transition Matrix we showed in Figure 4 – and the Transition Graph.

Figure 5 – Markov table

Markov Attribution by Channel

Source	Markov
BILLBOARD	2368
SITE VISIT	3523
SOCIAL	4520
VIDEO	9801
TOTAL	20212

Figure 6 – Transitions derived from pair counts with node properties displayed

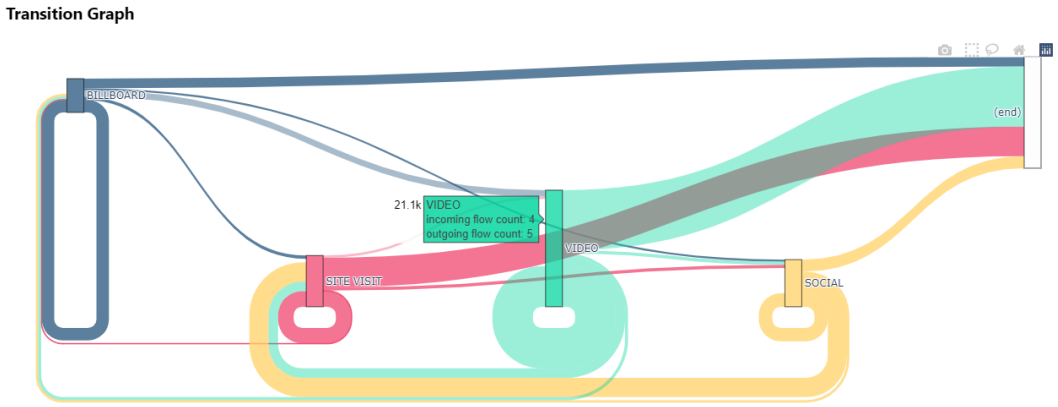
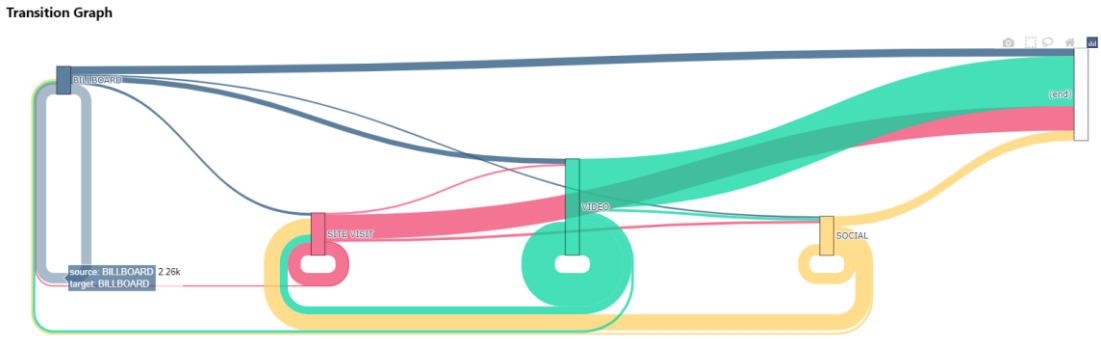


Figure 7 – Transitions with edge properties of a self-transition displayed



4.8. DEVELOPMENT AND PRODUCTION DEPLOYMENT

This last section will show how to run ATRIO in a development environment as a Jupyter Notebook, and how to deploy it to a containerized environment, where end-users can access the interface using a web application.

We'll assume a clean start with a Windows 10 machine, where no Python installation exists.

1. Download and run the Python installer from https://www.python.org/downloads/. At the time of this writing, the recommended version is 3.9.6.

2. We recommend following the “Customize Installation” option in the wizard and check all the “Advanced options”, including “Install for all users”. This will also change the installation path to the machine’s “Program Files” directory, rather than an application folder in the user space.
3. After Python is installed, choose a root folder to create a Python virtual environment and store the needed files for ATRIO.

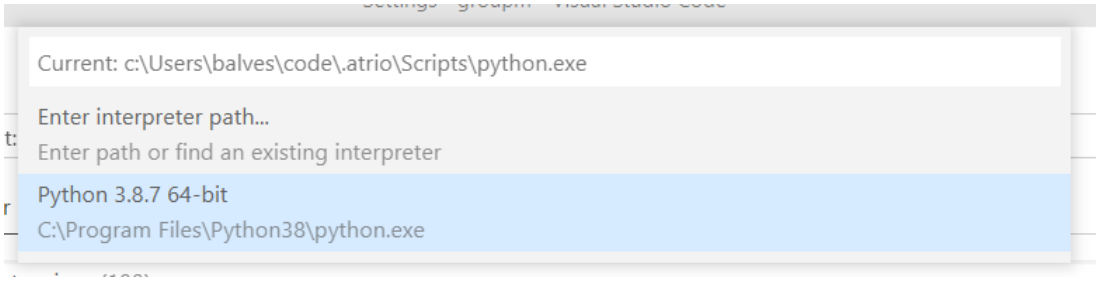
What is a Python virtual environment? It’s a location where we install all the packages which are not native to the Python distribution we are using. Depending on the user’s preference, we can install Python from the native distribution – as we are doing for ATRIO – or from one of the many distributions available. Anaconda, for example, is an alternative distribution which already includes many packages, libraries, and utilities for data science. When we create a virtual environment, we are creating a separate location for the additional packages we want to use. We can install additional packages to our native environment, but it’s a good practice to create virtual environments to host the sets of libraries separately. This is also a way to perform tests for different package versions, avoid package dependency conflicts and redistributing a well-defined set of libraries, to work on top of a standard installation, as a way to containerize applications.

Going back to the instructions, we’ll continue from the folder we chose to run our application and use a command line terminal for the next steps. We’ll be using *rem* as it’s the normal notation for “remarks” in Windows .bat files but note *rem* lines cannot be entered into the command prompt as shown in *Code Excerpt* 21. This sequence of library installations will bring their required dependency libraries. This is why we didn’t need to install Jupyter nor NumPy – because they are dependencies for libraries in our list – although we’ll be using them in our code. After these steps, upload the ATRIO files referred in the Annexes chapter to the current directory. This will be the working directory for our environment. From this point forward, we can enter “*jupyter notebook*” into the command prompt and we’ll get a Jupyter environment where we can test our notebook, “*Atrio.ipynb*”. We can also render ATRIO as a web application from the command prompt by typing “*voila Atrio.ipynb”* which will launch a web server rendering ATRIO for an end-user without showing the underlying code.

For our development, we used Visual Studio Code, a free and open sourced Interactive Development Environment maintained by Microsoft. Additionally, we installed the Python, Jupyter, Gather and Pylance extensions, because they improve the development experience with code linting and debugging. Moreover, VS Code has a native interface to use .Git version control. As we’ll see in the deployment section, we’ll use simple .Git commands to upload our code to a web-hosted production environment. Again, the use of VS Code is not an essential requirement but, in our opinion, it brings the advantages we mentioned at no cost.

As a final remark for running in development mode, either in a Jupyter environment or using VS Code’s Jupyter implementation, we’ll need to use the Python 3 Kernel provided by the .atrio virtual environment. VS Code automatically prompts the user to select the interpreter path which supports this Kernel. If the list of interpreters does not contain the .atrio virtual environment, all we need to do is navigate to the folder where it’s installed and select the python.exe file as shown, to make it part of the list for next sessions:

Figure 8 – select interpreter location



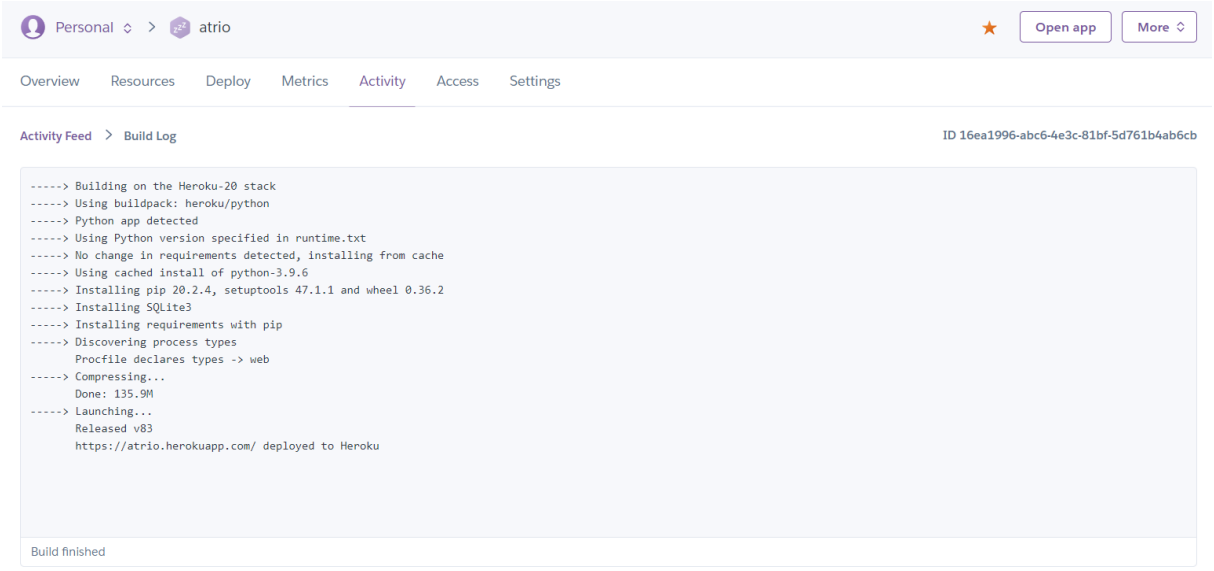
Jupyter Notebooks rendered with Voilà can be deployed to a web server, hosted on-premises or on the cloud. However, unless that infrastructure already exists for another purpose or specific security or performance conditions apply, we can avoid the need to manage infrastructure. The alternatives are provided by platform-as-a-service options available in cloud providers or as part of an existing infrastructure. The three possibilities presented in the Voilà reference for deploy are Heroku, Binder and Google App engine, although any other cloud provider such as Azure or AWS offers PaaS solutions capable of handling the deployment of Jupyter Notebooks and Voilà web applications.

We chose Heroku, because it has a free tier available and the deployment remains private, only exposing the web interface. Another reason was the simplicity of the deployment process, through .Git commands. For a corporate solution, Heroku works as well since their commercial offering's containers can be scaled to meet higher compute demands. The first step is to create an account, either on the free tier, for personal or academic purpose, or in the commercial tier, normally used for corporate solutions. After creating the account, their web interface provides the wizard to create an application. This application will host the code we'll upload using git and will provide a Linux container with the chosen runtime environment. In our case it's Python, but Heroku has other runtimes available for Node, Ruby, Java, and Scala, among others.

The Linux container, or cluster of containers in Heroku is called a Dyno. The free tier Dyno we are using has limitations both in performance and available hours of compute per month but is sufficient for this demonstration. To deploy our Python application to Heroku we'll need to download their command line interface and setup the connection to our Heroku account, choose the application to deploy and define the reference to our local .Git repository. This is a one-off step, and after this is done, we will rarely need to use the CLI again, unless we want to connect or build a different application, and all we need to redeploy our applications is a "git push" command. We won't detail the instructions, as they can be found here: https://devcenter.heroku.com/articles/getting-started-with-python, so we'll focus on the necessary files and commands to define the deploy parameters. These are well explained here: https://voila.readthedocs.io/en/stable/deploy.html#deployment-on-heroku

The files we need to place on the same .Git repository of our Notebook, for Heroku deployment are detailed in the Annexes chapter. Here is a print screen of our Atrio app while processing a new deployment after we committed our changes to our local branch and pushed them to Heroku master branch with .Git:

Figure 9 – Heroku build log after deployment



5. END-USER PERSPECTIVE

To demonstrate the deployment process of ATRIO, as part of the requirements for an end-to-end implementation, and to provide end-users a live ATRIO application they can use to follow these guidelines, **we deployed ATRIO to a web hosting platform. The deployed application is available at https://atrio.herokuapp.com/**. This application is exposed to the public web, so we hat to set a simple mechanism to prevent unauthorized users to run ATRIO with the internal dataset, because it's property of GroupM. To access the internal dataset, users need to insert this passcode, which is only provided in this document: **NovaIMS2021**

Inserting this passcode in the required field of the home page, triggers the application execution with the internal dataset. Alternatively, any user without the passcode can test ATRIO by uploading their own dataset. The required data structure is illustrated with a header and two example rows, so the user can prepare the data with the same structure, before uploading it. This is also a demonstration for the capability of ATRIO to ingest, process and render other datasets.

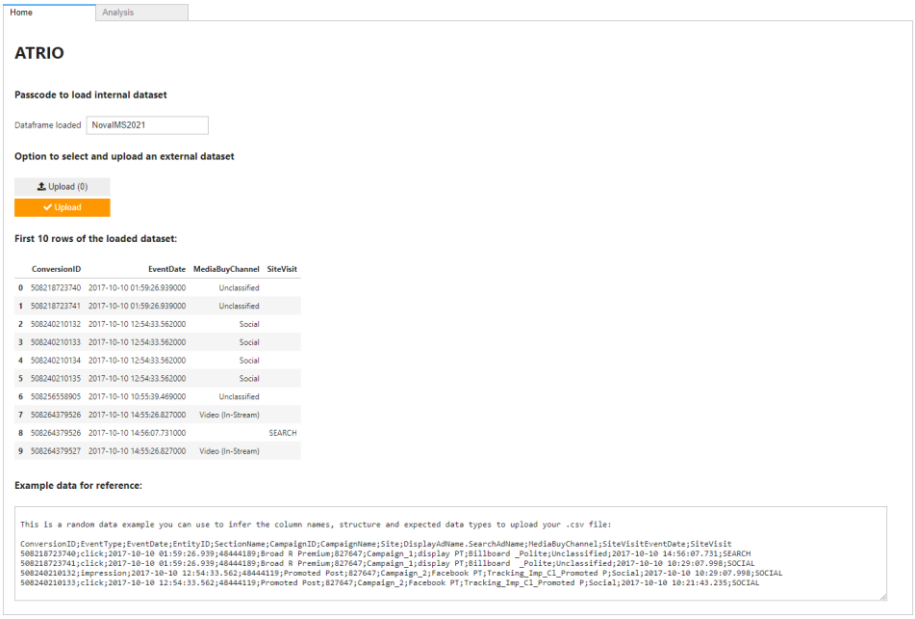
To test ATRIO without the capacity limitations of Heroku's free tier, we recommend a local installation of the files listed in the Annexes and a review of the previous chapter for a local deployment. Assuming the user's PC will have average resources, the use-case we'll present in section 5.3 can be followed without noticing any delays to calculate the visualizations after changing parameters on the interface.

5.1. LOAD INTERNAL DATASET OR UPLOAD EXTERNAL DATASET

To load the internal dataset type **NovaIMS2021** into the "Insert passcode" input. The input label will change to "Passcode ok - Loading df". It can take a few seconds in the demo web application, but it will render the visualizations in 2-3 seconds in a regular PC. When the dataset is loaded and the visualizations are rendered, the input label will display "Dataframe loaded" and the interface will automatically switch to the "Analysis" tab.

Alternatively, the user can select a .csv file from their PC using the Upload(0) button, selecting the file, an clicking "Open". The button will display Upload(1), meaning one file is marked for upload. Then, clicking the orange button below will actually trigger the upload and the input label will show "Loading external dataframe". After the process and render is completed, ATRIO moves to the "Analysis" tab.

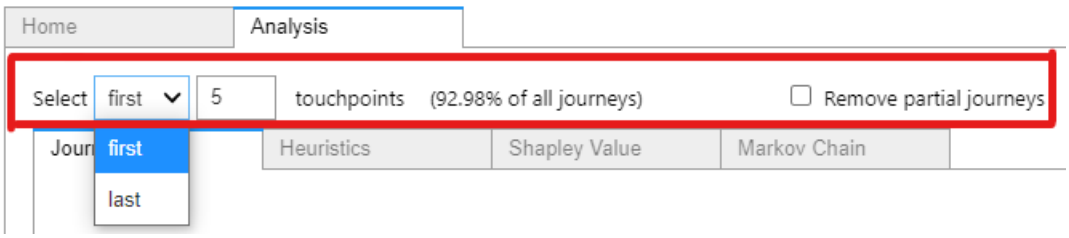
Figure 10 – Home Tab



5.2. USING THE PARAMETER CONTROLS

ATRIO has three parameters for the user to control how to filter the dataset:

Figure 11 – ATRIO parameters



The first control is a dropdown menu with two options, “first” and “last”, and it controls the sequence of touchpoints to be presented in the charts and analyzed in the tables. “first” is the default option and it means touchpoints are sequenced from first to last, for each journey, by chronological order. The details of this calculation were explained in Chapter 4. The user will decide if the last touchpoints are more relevant to the conversion, and in that case the option should be “last”, so the touchpoints would be presented in reverse sequence.

The second option is presented with a bounded integer text box, which has been limited during the dataset calculation to allow a range between 1 and the maximum number of touchpoints per conversion. For the internal dataset, we have three journeys with 30 touchpoints, so this defines the maximum boundary. As we can observe in the touchpoint frequency chart, this is hardly representative, so the analyst can choose to limit the number of touchpoints to count. For the current dataset, 5 touchpoints yields a good compromise between the Sankey chart readability and the representativeness of journeys up to 5 touchpoints in the dataset.

So, what happens to the journeys with more touchpoints than the filter number, are they discarded? That’s the option we can control with the last parameter “Remove partial journeys”. If we check this

box, ATRIO will discard all journeys with more touchpoints than the number we set on the previous parameter. Per default, this box is not checked. In this case, ATRIO keeps all the journeys but considers the n^{th} touchpoint – where n equals the integer filter for maximum touchpoints – as the last touchpoint for the journeys with more than n touchpoints.

Between the touchpoint filter and the partial journeys textbox, there is a dynamic label that changes every time we alter these parameters. This label indicates the percentage of journeys we are covering if we filter more or less touchpoints. If we decide to "Remove the partial journeys" this indicator will always be 100%. The same happens if we set the journey filter to the maximum number of touchpoints, obviously. While we can see this implicit proportion in the touchpoint frequency chart, we can use the filter in incremental steps to understand the accumulated frequency. For long-tail distributions as these, we can get most cases within the first frequencies.

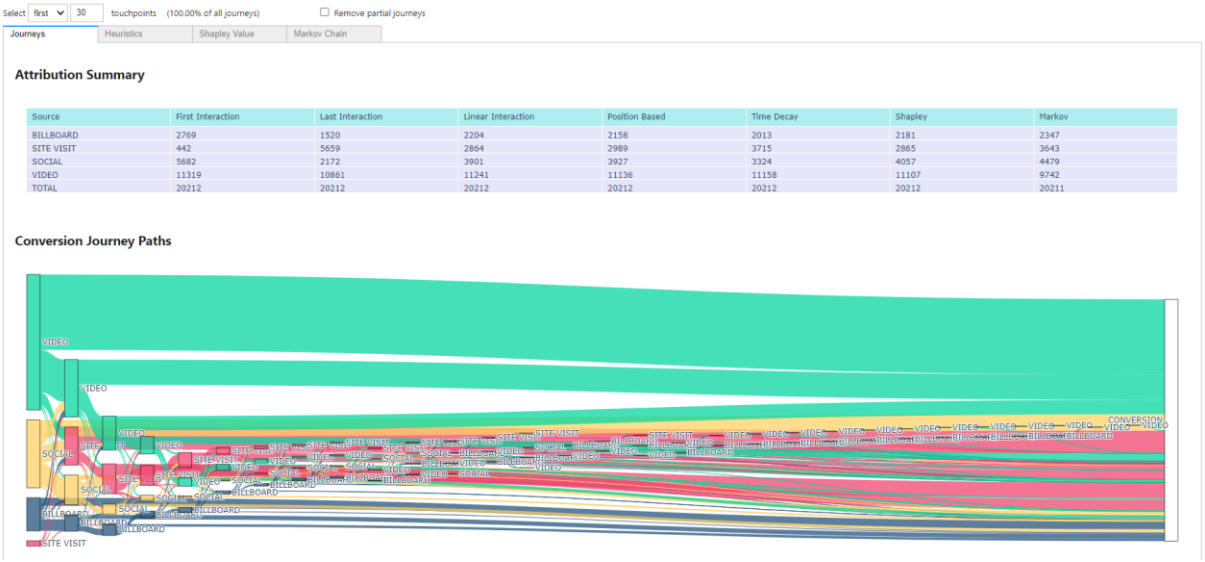
Each time any of these widgets is changed, the entire dataset is calculated again an all the visualizations are updated, in all the tabs inside the analysis pane, regardless if we are currently viewing Journeys, Heuristics, Shapley Value or Markov Chain. In slower machines, we may need to wait a few seconds to ensure all visualizations and tables were refreshed [6].

5.3. DATA ANALYSIS USE-CASE

We'll now demonstrate how to use ATRIO with our dataset, to understand how it works and what it can deliver, by going through the whole application from a business-driven perspective. We will explore how a balanced choice of filter parameters can deliver representative visualizations and lead to better conclusions.

With the internal dataset loaded, we will start by selecting all touchpoints in the consumer journey and observe the result:

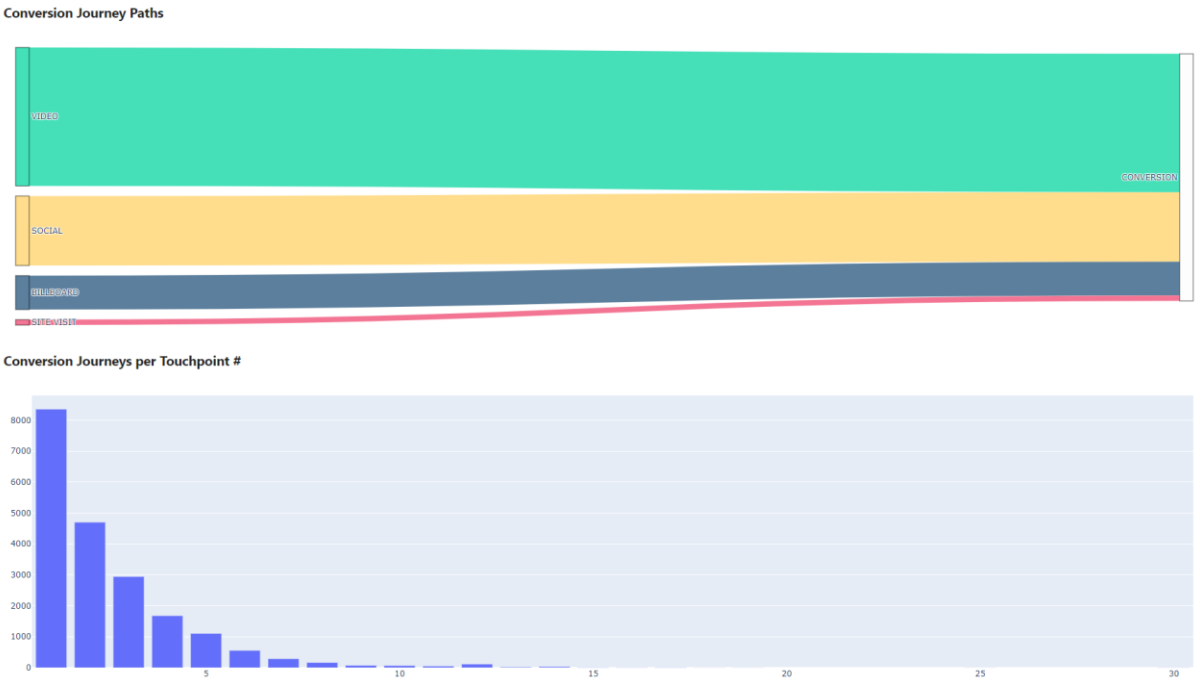
Figure 12 – all touchpoints selected



[6] This is an aspect for improvement in future versions, but it may require a finer control of asynchronous function callbacks to let the user know the calculation is still in progress or is complete. Running in development mode in a Jupyter environment we have a visual cue from the "Interrupt" button indicator of the Kernel

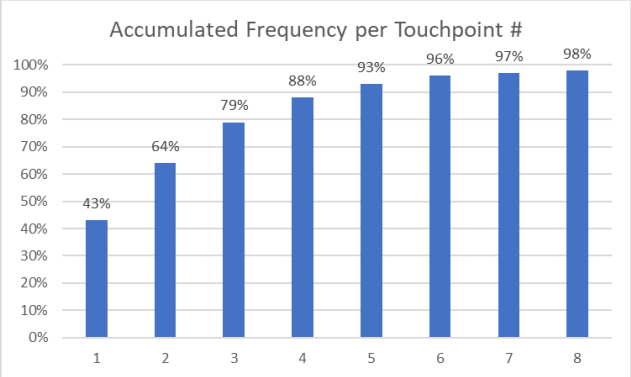
From the "Attribution Summary" table perspective, this could be the most representative view of our dataset, because we consider all the touchpoints from all journeys ant the values of the all calculations – simple heuristics and algorithms – is based on all the records. However, while the "Conversion Journey Paths" diagram depicts the reality, it's hard to interpret it, and it's not truly representative. Why? Because most journeys don't have 30 touchpoints, which is the maximum we observed in our dataset. In fact, most of them have less than 3 touchpoints. So, what if we set our filter to 1 touchpoint?

Figure 13 – Filter set to 1 touchpoint



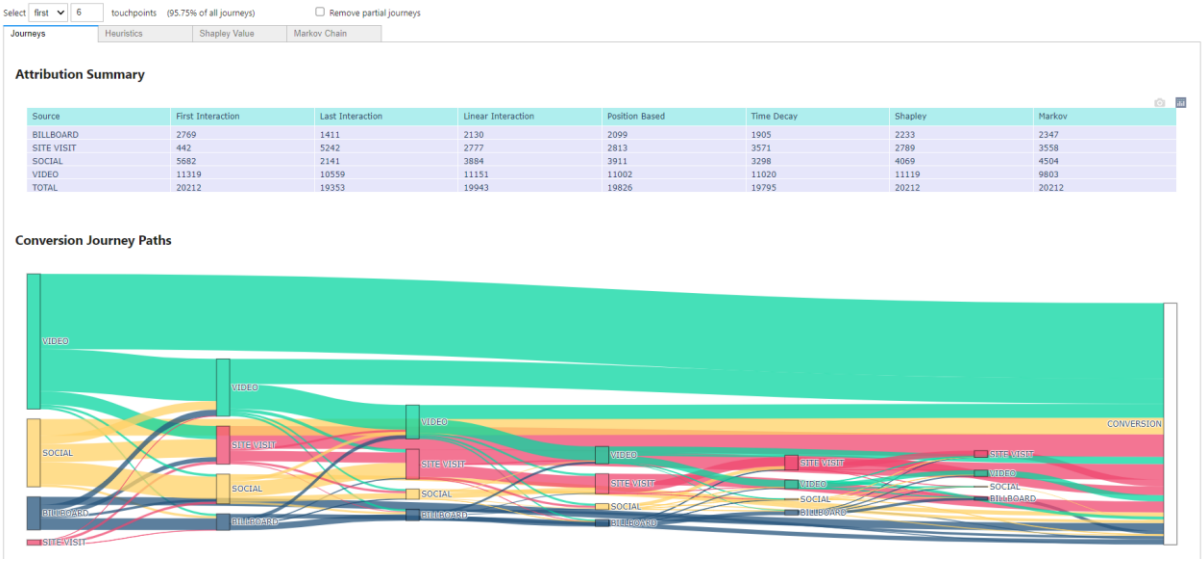
Looking at the "Conversion Journeys per Touchpoint #", we may be led to believe that single-touchpoint journeys are representative, since they are the most frequent, representing 43% of our dataset. However, the extreme of this long-tail distribution is not representative. First, because we know many of these single-touchpoint journeys are the result of bias effects mentioned earlier, originated by cookie deletion and tracking errors. Second, even if no bias existed, 43% is not representative. So, where is the right balance? If we observe the next figure – not delivered in ATRIO's output – we can observe the accumulated frequencies of the previous chart until the 8th touchpoint:

Figure 14 – Accumulated frequency



We observe how the accumulated frequency progresses and starts to flatten around the 6th touchpoint, over 95%. Empirically, this can be a good indicator of a balanced filter choice to represent our dataset. If we look at our "Conversion Journey Paths" diagram, we see a pattern where "Site Visit" has the lowest frequency in the first touchpoint, and gradually moves up the rank to become the most frequent channel from touchpoint #5 onward:

Figure 15 – Conversion journey paths for 6 touchpoints



Does it mean "Site Visit" will be the most frequent channel overall? No, because the frequency we mentioned is observed for touchpoints #5 onward, which are the less representative in absolute numbers. The absolute ratio of conversions stemming from "Site Visit" is 27% and "Video" is still worth 54% [7].

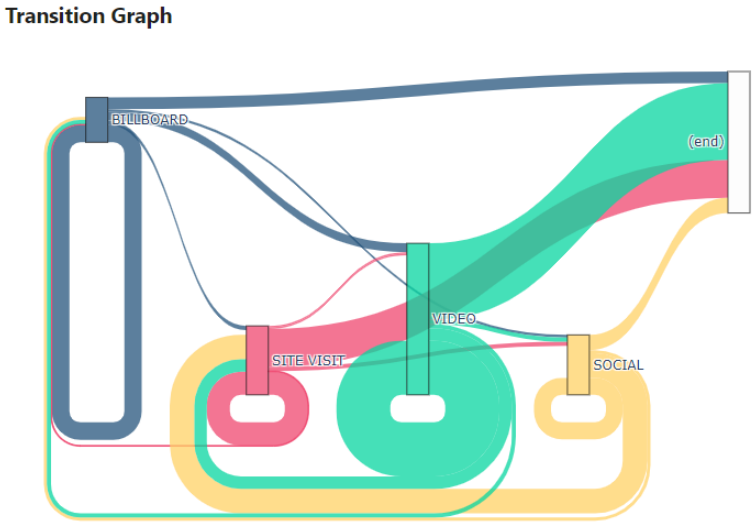
What does this say about the "Conversion Journey Paths" diagram? We can visualize how channels appear and shift positions along the consumer journey, we can observe how the frequency of longer consumer journeys diminishes along touchpoints, and we get a notion of source and destination transition pattern changes, between shallow and deep journeys.

Moving along the tabs on the "Analysis" pane, we will now move from the "Journeys" tab directly to "Markov chain". We are skipping "Heuristics" and "Shapley Value" because the implementation has already been explained in Chapter 4 and there are no visualizations other than simple tables. We will review their values at the end of this chapter, while analyzing the "Attribution Summary" table.

In the "Markov Chain" tab we will find the Markov values per Channel in the "Markov Attribution by Channel" table. The Markov algorithm implementation has been explained in Chapter 4, along with the "Channel Transitions" table construction. We will focus on the "Transitions Graph", which derives from the information contained in the "Channel Transitions" table, to understand how we can interpret it and how it complements the "Conversion Journey Paths" diagram. A first glance at this graph without proper context raises some questions we'll try to answer.

[7] We will see how to obtain this values from the Markov Transition Graph in the next page

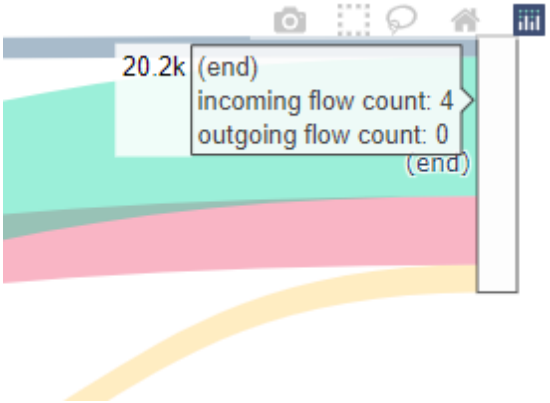
Figure 16 – Transition Graph with 6 touchpoint filter



When we briefly explained the construction of this chart in Chapter 4, we mentioned it represents the number of transitions between channels. This view is a different from the "Conversion Journey Paths" diagram because we show an aggregated view of all the transitions between channels, instead of the full path. so how does this view complement the previous one?

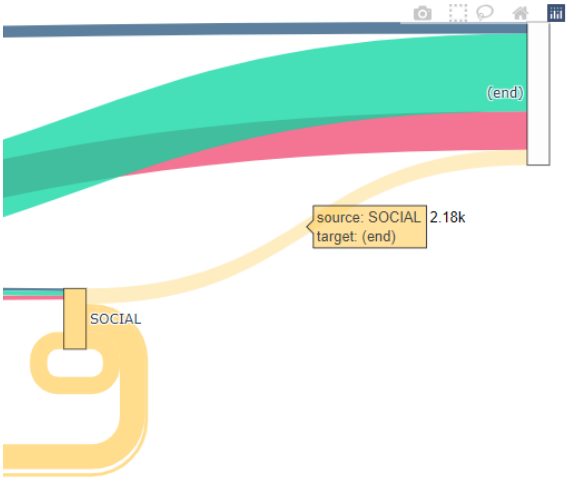
First, it provides absolute numbers for each channel. By hovering over the nodes, we can see a legend with the node transition count values, incoming and outgoing flow counts:

Figure 17 – Transition Node detail



If we hover over an edge, we will get the count of transitions between two channels, or, between a channel and the end of the journey. We can also observe self-transition flows between channels. These represent the counts where two sequential touchpoints of a journey are in the same channel. If we look at the "Channel Transitions" table diagonal, we can see the percentages of these self-transitions. For our dataset, "Billboard" is the channel with the highest percentage of self-transitions (38,9%). The variance of self-transition percentages across channels is not significant though, with "Social" in the lower end (25,1%). Without other supporting evidences we can only assume a simple probability cause, related to the number of channels

Figure 18 – Transition Edge detail

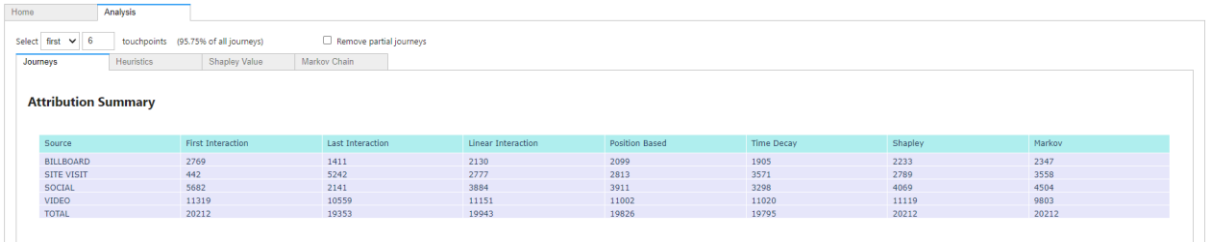


In this case, we can observe 2.180 conversions cane directly from “Social” channel. This value is represented as a percentage in the “Channel Transitions” table, if we intersect the row where “Source” value is “Social” with the column “(end)”

The most significant information we can draw from this diagram are the indirect contributions. In the literature review, we found several references to this effect, where a given channel is not particularly performant as a direct converter but contributes indirectly to other channels that convert well. For our dataset, the most evident relation we can observe in Figure 19 is the contribution of “Social” to “Site Visit”. While “Social” has less direct conversions than “Site Visit”, the majority of “Site Visit” events are originated from “Social”. While this relationship is also visible in the “Conversion Journey Paths” diagram, it becomes evident here, because data is aggregated by channel.

To conclude this use case, we’ll take a look at the “Attribution Summary” table:

Figure 19 – Summary for 6 touchpoints filter



Source	First Interaction	Last Interaction	Linear Interaction	Position Based	Time Decay	Shapley	Markov
BILLBOARD	2769	1411	2130	2099	1905	2233	2347
SITE VISIT	442	5242	2777	2813	3571	2789	3558
SOCIAL	5682	2141	3884	3911	3298	4069	4504
VIDEO	11319	10559	11151	11002	11020	11119	9803
TOTAL	20212	19353	19943	19826	19795	20212	20212

Looking at the values for the “Social” channel, we observe the Markov algorithm attributes more value to this channel than all others, except “First Interaction”. This is possibly an effect of the indirect contribution we observed on the “Transitions Graph”. We can also observe that the Markov value for “Site Visit” is significantly higher than Shapley’s, and trade-off between the two algorithms is observed in the “Video” channel where the Shapley value is higher. We estimate this trade-off might happen because the proportion of transitions from “Video” to other channels is lower than the equivalent from other channels. This means, the “Video” channel transitions are mostly towards conversion and to itself, which will impact its score with the Markov algorithm.

6. CONCLUSION

ATRIO was designed to respond to a business need from GroupM and was produced in the context of an academic project. This document – a project report based on standard academic research – and the source code file we produced will allow GroupM to deploy and use this tool. Like in most projects, the expectations we had were high, as we imagined all the possibilities we could deliver. As we reach the delivery moment, we're satisfied for having solved the practical problems we set out to tackle, but we know the end product is different than what we imagined.

What did we achieve? We were able to provide an application to replace a legacy one, which could no longer be maintained, and we introduced a novel approach to GroupM, by adding the Markov chain algorithm, on top of their existing Attribution Analysis practice. We believe the perspective provided by the channel transitions analysis and the improved consumer journey diagram can bring more insights about their datasets than was previously possible. The result is delivered using an open source collection of modern languages and practices. It is reproducible and can easily be deployed.

What could have we achieved more? ATRIO needs more work to be productized. While the key algorithms are in place, we fell short of the "framework" concept. The code needs more abstraction and testing and ATRIO could provide an API to integrate it with the campaign operational processes, to deliver real-time insights. In its current form, it works well for ad-hoc analysis but needs improvement for the automation of real-time data ingestion. We remain available to cooperate with GroupM in further developments.

7. REFERENCES

Abhishek, V., Fader, P. S. & Hosanagar, K., n.d. *Media Exposure through the Funnel: A Model of Multi-Stage Attribution,* s.l.: s.n.

Anderl, E. et al., 2014. *A graph-based framework for online attribution modeling,* s.l.: s.n.

Berman, R., 2015. *Beyond the Last Touch: Attribution in Online Advertising,* s.l.: s.n.

Cano-Berlanga, S., Gimenez-Gomez, J. M. & Vilella, C., 2011. Attribution models and the Cooperative Game Theory. pp. 1 - 29.

Dalessandro, B., Stitelman, O., Perlich, C. & Provost, F., 2012. *Causally motivated attribution for online advertising.* s.l., s.n.

Google Inc, 2020. *About the default MCF attribution models.* [Online].

Kireyev, P., Pauwels, K. & Gupta, S., 2013. *Do Display Ads Influence Search? Attribution and Dynamics in Online Advertising,* s.l.: s.n.

Korostil, I., 2020. *MTA,* s.l.: s.n.

Li, H. & Kannan, P. K., 2014. Attributing conversions in a multichannel online marketing environment: An empirical model and a field experiment. *Journal of Marketing Research,* 51(1), pp. 40-56.

Markov, A. A., 1907. Extension of the Limit Theorems of Probability Theory to a Sum of Variables Connected in a Chain. *The Notes of the Imperial Academy of Sciences of St. Petersburg VIII Series, Physio-Mathematical College, XXII/9*.

Nisar, T. & Yeung, M., 2015. Purchase Conversions and Attribution Modeling in Online Advertising: An Empirical Investigation. *44th EMAC Annual Conference - Collaboration in Research, Leuven, BE, 24 - 27 May 2015,* p. 8.

SHAO & LI, 2011. Data-driven Multi-touch Attribution Models. *Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining.,* p. 1416.

Shapley, L. S. & Aumann, R. J., 1968. *Values of non-atomic games.* Santa Monica: Rand Corp.

Xu, L. et al., 2011. *Path to Purchase: A Mutually Exciting Point Process Model for Online Advertising and Conversion * Path to Purchase: A Mutually Exciting Point Process Model for Online Advertising and Conversion,* s.l.: s.n.

Zhang, Y., Wei, Y. & Ren, J., 2015. *Multi-touch Attribution in Online Advertising with Survival Theory.* s.l., Institute of Electrical and Electronics Engineers Inc., pp. 687-696.

8. ANNEXES AND CODE EXCERPTS

The following files are delivered along with this document, in a zipped file:

- A Jupyter Notebook with the full, commented source code: *Atrio.ipynb*
- A text file with the dataset provided by GroupM: *Dataset_Anon.csv*
- A text file with requirements to deploy to Heroku: *requirements.txt*
- A text file with runtime information to deploy to Heroku: *runtime.txt*
- A text file with commands to deploy to Heroku (no extension): *Procfile*

The last three files are optional and only needed for deployment to the Heroku platform.

The dataset is anonymized but contains real data from a digital campaign and is property of GroupM.

The Jupyter Notebook and this document constitute the work performed as the ATRIO framework.

8.1. CODE EXCERPTS

Code excerpt 1

```
# 1 - IMPORT LIBRARIES AND DECLARE GLOBAL DATAFRAMES
import pandas as pd
import numpy as np
import plotly.graph_objects as go
import ipywidgets as widgets
from collections import defaultdict
from functools import reduce
from operator import mul
from itertools import tee
import sys
from io import StringIO
global df, heuristic, shapley, markov
df = pd.DataFrame()
heuristic = pd.DataFrame()
shapley = pd.DataFrame()
markov = pd.DataFrame()
```

Code excerpt 2

```
# CREATE USER INTERFACE
header_dict = dict(values = list(), fill_color = 'paleturquoise', align = 'left')
cells_dict = dict(values = [], fill_color = 'lavender', align = 'left')
layout_dict = dict(autosize = True, margin = dict(l = 20, r = 20, b = 20, t = 20))
passcode = widgets.Text(value = '', description = 'Insert passcode',
style = {'description_width': 'initial'})
uploader = widgets.FileUpload(accept = '.csv', multiple = False)
button_upload = widgets.Button(description = 'Upload', disabled = False, button_style = '
warning', tooltip = 'Click to Upload', icon = 'check')
df_show_head = widgets.Output()
```

Code excerpt 3

```
layout_dict = dict(autosize = True, margin = dict(l = 20, r = 20, b = 20, t = 20))
```

Code excerpt 4

```
        df = pd.read_csv(
            'Dataset_Anon.csv',
            sep = ';',
            dtype = object,
            na_filter = False,
            usecols = ['ConversionID', 'EventDate', 'MediaBuyChannel', 'SiteVisit'],
            parse_dates = ['EventDate','SiteVisitEventDate']
        )
```

Code excerpt 5

```
# Create 'EventSequence' and 'SequenceSize' columns
# Sort our dataframe by ConversionID and Event date, to enable sequential
# operations along the vertical axis
    df = df.sort_values(
        by = ['ConversionID', 'EventDate'],
        ascending = [True, True]
    )
# Remove Duplicated rows found while analyzing data
# Assumption was these should not exist, but 6 duplicate rows were found on
# our 50k row dataset
    df = df.drop_duplicates(subset=['ConversionID', 'EventDate'], keep="first")
# Create a Rank based on the events timestamps for each conversion and add as
# a new 'EventSequence' column to our Dataframe
    df['EventSequence'] = df.groupby('ConversionID')['EventDate'].rank(
        method = 'first',
        ascending = True
    ).astype('uint8')

# Create a count aggregation df based on group by conversion and event timestamp and
# merge it back to original df on ConversionID creating a new 'SequenceSize' column
    df = pd.merge(
        df, df.groupby('ConversionID').agg(
            SequenceSize = pd.NamedAgg(
                column = 'EventDate',
                aggfunc = pd.Series.nunique
            )
        ),
        on = 'ConversionID'
    )
```

Code excerpt 6

```
# Create Real Source and Target calculations
# Create 'Source' column. We'll take the MediaBuyChannel field value as default for
# Impression or Click events and SiteVisit field value for site visit events
    df['Source'] = np.select(
```

```
        [df.SiteVisit != ''],
        ['SITE VISIT'],
        default = df.MediaBuyChannel.str.upper()
    )
# Rename and simplify source channels
# For this dataset, 'UNCLASSIFIED' pertains to 'BILLBOARD'
# and we can merge 'VIDEO (IN-STREAM)' into 'VIDEO'
    df['Source'] = np.select(
        [df.Source == 'UNCLASSIFIED', df.Source == 'VIDEO (IN-STREAM)'],
        ['BILLBOARD', 'VIDEO'],
        default = df.Source
    )
# Create 'Target' column. Since the dataframe is sorted, we can create a copy of
# the Source column and shift this new column by 1 row up.
# This operation enables a side-by-side comparison of source and target on the
# same row, since the target now equals the value of the Source column on the next row.
# We will use this transformation to create path combinations for the Sankey diagram
    df['Target'] = df.Source.shift(-1)
# The previous operation has a problem, as the last 'Target' event overflows to
# the next conversion sequence, which is unrelated to the previous one.
# This is resolved by hardcoding the value 'CONVERSION' into the last event of
# every sequence, to overwrite the incorrect values.
# Note we are working with a dataset where all event sequences result in a conversion.
# If we had non-converters, another flag value would need to be written for those cases.
    df['Target'] = np.select(
        [df.EventSequence == df.SequenceSize],
        ['CONVERSION'],
        default = df.Target
    )
```

Code excerpt 7

```
df['AggregatedSource'] = np.select(
    [(df.EventSequence <= max_sequence.value)],
    [df['Source']],
    default=''
)
df['AggregatedTarget'] = np.select(
    [(df.EventSequence < max_sequence.value), (df.EventSequence == max_sequence.value)],
    [df['Target'], 'CONVERSION'],
    default=''
)
df['AggregatedSequence'] = np.select(
    [(df.EventSequence <= max_sequence.value)],
    [df.EventSequence],
    default=''
)
```

Code excerpt 8

```
sdf = df.loc[df.SequenceSize<=journey_filter()]
touchpoint_label.value = 'touchpoint{}'.format('s' if max_sequence.value > 1 else '')
# Create 'df_paths' dataframe by grouping 'Source', 'Target' and 'EventSequence' from
# 'df' and adding a 'counts' column aggregation and rename the columns.
df_paths = sdf[
    (sdf['AggregatedSource'] != '') & (sdf['AggregatedSequence'] != '')
].groupby(
    ['AggregatedSource', 'AggregatedTarget', 'AggregatedSequence']
).size().reset_index(name = 'counts').rename(
    columns={'AggregatedSource': 'Source',
```

```
        'AggregatedTarget': 'Target',
        'AggregatedSequence': 'EventSequence'}
)
# We overwrite the 'Source' column values by appending the EventSequence with the '_00'
# format to the original value. This will be a composite key to lookup the labels in
# the conversion flow, having source and event sequence combination as the primary key.
df_paths['Source'] = (df_paths.Source + '_'
+ (df_paths.EventSequence.astype('int') - 1).astype('str').str.rjust(2, '0'))
# The previous logic is applied to the 'Target' column, except for the rows where the
# value is 'CONVERSION', since these are always the last event on the sequence
df_paths['Target'] = np.select(
    [df_paths.Target == 'CONVERSION'],
    ['CONVERSION'],
    default = df_paths.Target + '_'
    + df_paths.EventSequence.astype('str').str.rjust(2, '0')
)
# Append 'Target' and 'Source' labels into a single list to create chart series
labels = np.append(df_paths.Source.unique(), df_paths.Target.unique())
# Remove duplicate values from the list and sort it
labels = np.unique(labels)
labels.sort()
# Create a new dataframe from the list of labels, in order to merge them into
# new columns on 'df_paths' dataframe
df_labels = pd.DataFrame(labels)
# Keep the current index and transform it into a new column 'index' we'll need
# to use on the Sankey chart variables
df_labels.reset_index(inplace=True)
# 'index' is renamed to 'SourceLabelIndex' and column 0, which is actually
# the source, is renamed as such
df_labels = df_labels.rename(columns={'index': 'SourceLabelIndex', 0 : 'Source'})
# Since both dataframes have a 'source' column, it's used by default as a join to
# merge both, resulting in a new 'df_paths' dataset, now including 'SourceLabelIndex'
df_paths = df_paths.merge(df_labels)
# 'SourceLabelIndex' is now 'TargetLabelIndex' and 'Source' is now 'target'
df_labels = df_labels.rename(
    columns={'SourceLabelIndex': 'TargetLabelIndex', 'Source' : 'Target'}
)
# Since both dataframes have a 'target' column, it's used by default as a join to
# merge both, resulting in a new df_paths, now including 'TargetLabelIndex'
df_paths = df_paths.merge(df_labels)
# Now that we have copied our suffixed labels into a dataframe, to enable their match
# on lookup, we can remove the suffix from the original label array elements, for a
# clean output on the chart. We will also assign the sources, targets and values
# needed for the Sankey chart.
labels = np.array([item[0] for item in np.char.rsplit(
        labels.astype(str), sep='_', maxsplit=1
)])
```

Code excerpt 9

```
# Declare custom color dictionary
colors = {"BILLBOARD": "rgba(38, 84, 124, 0.75)",
"SITE VISIT": "rgba(239, 71, 111, 0.75)",
"SOCIAL": "rgba(255, 209, 102, 0.75)",
"VIDEO": "rgba(6, 214, 160, 0.75)",
"CONVERSION": "rgba(252, 252, 252, 0.75)"}
# Declare and populate node colors by iterating each
# label and looking up the corresponding color in the dictionary
n_colors = []
for _ in labels:
    n_colors.append(colors[_])
# Declare and populate link colors by iterating each source
# path and looking up the corresponding color in the dictionary
```

```
l_colors = []
for _ in df_paths.Source:
    # Take only the source name by removing the postion suffix
    l_colors.append(colors[_.split("_")[0]])
with sankey_chart.batch_update():
    sankey_chart.data[0].node.label = labels
    sankey_chart.data[0].node.color = n_colors
    sankey_chart.data[0].link.source = df_paths.SourceLabelIndex
    sankey_chart.data[0].link.target = df_paths.TargetLabelIndex
    sankey_chart.data[0].link.value = df_paths.counts
    sankey_chart.data[0].link.color = l_colors
```

Code excerpt 10

```
heuristic = df.loc[
    (df.SequenceSize<=journey_filter()) &
    (df['AggregatedSource'] != '') &
    (df['AggregatedSequence'] != '')][[
        'AggregatedSource', 'AggregatedSequence', 'SequenceSize']].astype(
        {'AggregatedSequence': 'int32', 'SequenceSize': 'int32'})
```

Code excerpt 11

```
def journey_filter():
    journey_filter = max_sequence.max
    if filter_journeys.value == True:
        journey_filter = max_sequence.value
    return journey_filter
```

Code excerpt 12

```
# Create first and last Interaction columns where credit goes to each of these:
heuristic['First Interaction'] = np.select([heuristic.AggregatedSequence == 1], [1])
heuristic['Last Interaction'] = np.select([heuristic.AggregatedSequence
 == heuristic.SequenceSize], [1])
# Create 'Linear Interaction' column: every touchpoint will receive equal
# credit for the conversion
heuristic['Linear Interaction'] = 1 / heuristic.SequenceSize
# Create 'Position Based' column: first and last touchpoints will receive 40%
# credit each, for the conversion. Remaining touchpoints divide 20% credit among them.
# If we only have 2 touchpoints, then each one gets 50% credit.
# If only one touchpoint exists, it gets 100% credit.
heuristic['Position Based'] = np.select(
    [heuristic.SequenceSize == 1, heuristic.SequenceSize == 2, heuristic.SequenceSize
     >= 3], [1, 0.5, np.select(
        [(heuristic.AggregatedSequence == 1) | (heuristic.AggregatedSequence
         == heuristic.SequenceSize)], [0.4], 0.2 / (heuristic.SequenceSize - 2)
    )]
)
# Create 'Time Decay' column: linear function where m (slope) = 2.
# It's applied to all touchpoints in descending order starting from the last one.
# This provides higher credit to the last events and lower credit to the first ones,
# assuming a time decay effect exists, on their contributions to conversion
heuristic['Time Decay'] = (2 * heuristic.AggregatedSequence / heuristic.SequenceSize
 / (heuristic.SequenceSize + 1))
```

Code excerpt 13

```
# Rename source column and create a Total to present in the Plotly Table
heuristic = heuristic.rename(columns={'AggregatedSource': 'Source'})
numcols = ['First Interaction', 'Last Interaction','Linear Interaction',
'Position Based','Time Decay']
heuristic = heuristic.groupby('Source')[numcols].sum().reset_index()
total = heuristic.apply(np.sum)
total['Source'] = 'TOTAL'
# Append the total values and transpose the result to load into Plotly Table
heuristic = heuristic.append(pd.DataFrame(total.values, index=total.keys()).T,
ignore_index=True)
heuristic[numcols] = heuristic[numcols].astype(int).round()
with heuristic_table.batch_update():
    heuristic_table.data[0].header.values = list(heuristic.columns)
    heuristic_table.data[0].cells.values = [heuristic[col] for col in heuristic.columns]
    heuristic_table.layout = dict(
        height = heuristic.shape[0] * 20 + 86, width = sankey_chart.layout.width
    )
```

Code excerpt 14

```
# CONVERSION SETS - COMMON TO SHAPLEY & MARKOV
# For Shapley: apply set() function to reduce duplicated touchpoints to unique sources
# For Markov: keep all touchpoints even if a path has repeated occurrences of the same
# source and we won't sort them, to account for the original transitions
# This is why we differentiate the end result with the applySet parameter,
# depending on the caller algorithm
def conversion_sets(applySet=True):
    max_journey_size = journey_filter()
    df_conv_paths = (df.loc[df.SequenceSize<=max_journey_size]
        [(df['AggregatedSource'] != '') & (df['AggregatedSequence'] != '')].pivot(
            index=['ConversionID'],
            columns='AggregatedSequence',
            values='AggregatedSource'
        ).reindex(columns=list(map(str,range(1,max_journey_size+1))))
    )
    if applySet:
        return df_conv_paths[list(map(str,range(1,max_journey_size+1)))].apply(
            lambda row: sorted(set(filter(lambda v: v==v, row.values))), axis=1
        )
    else:
        return df_conv_paths[list(map(str,range(1,max_journey_size+1)))].apply(
            lambda row: list(filter(lambda v: v==v, row.values)), axis=1
        )
```

Code excerpt 15

```
def subsets(item_list):
    # This utility function returns all subset combinations from a list
    output = [[]]
    for item in item_list:
        output.extend([current + [item] for current in output])
    output.pop(0)
    return output

def value_of_A(A,C_values):
    # This function returns the number of conversions for each coalition subset
    subsets_of_A = subsets(A)
```

```
    worth_of_A=0
    for subset in subsets_of_A:
        if str(subset) in C_values:
            worth_of_A += C_values[str(subset)]
    return worth_of_A

shapley_channels = []
# Clean the list syntax characters and append each channel to the list
for s in conv_group.index:
        for t in s.strip('][').replace("'","").split(', '):
            shapley_channels.append(t)
# Return the channels in a unique, sorted list
shapley_channels = list(sorted(set(shapley_channels)))
# Create a Python dictionary form the grouped conversion sets list
conversion_values = conv_group.to_dict()
coalition_values = {}
# Calculate coalition values by combining all subsets from the Shapley Channels
for A in subsets(shapley_channels):
    coalition_values[str(A)] = value_of_A(A,conversion_values)
# Record channel count in n variable for reuse
n=len(shapley_channels)
shapley_values = defaultdict(float)
```

Code excerpt 16

```
shapley_values = defaultdict(float)
# Calculate Shapley value for each channel by iterating channel list
for channel in shapley_channels:
    # Iterate over each coalition
    for A in coalition_values.keys():
        # Clean the list syntax characters of the coalition
        # and return an array of channels
        AA = A.strip('][').replace("'","").split(", ")
        # Run this code for coalitions in the parent iteration
        # that don't have the current channel
        if channel not in AA:
            # Record  number of channels of current coalition array
            cardinal_A=len(AA)
            # Copy coalition array to temporary variable
            A_with_channel = AA
            # Append the "missing" channel to the coalition copy array
            A_with_channel.append(channel)
            # Reassign the coalition copy array variable as a list,
            # applying list syntax
            A_with_channel= "['" + "', '".join(sorted(A_with_channel)) + "']"
            # Calculate and add the marginal contribution for the
            # coalition to the channel value
            shapley_values[channel] += (
                (coalition_values[A_with_channel]-coalition_values[A])*(
                    np.math.factorial(cardinal_A)
                    *np.math.factorial(n-cardinal_A-1)
                    /np.math.factorial(n)
                )
            )
    # Add the individual channel contribution to the channel value and normalize result
    shapley_values[channel] += coalition_values["['" + channel + "']"]/n
```

Code excerpt 17

```
def prob_convert(mk_data, trans_mat, drop=None):
```

```
    # This utility function calculates conversion probability and
    # can receive a parameter to omit conversion groups containing
    # a specific channel, to account for removal effects.
    # The inputs are the conversion sets and the transition matrix.
    # The first step is to apply the channel removal (drop)
    # and return the result into a temporary variable
    _d = mk_data[
        mk_data['path'].apply(
            lambda x: drop not in x.strip('][').replace("'","").split(', ')
        ) & (mk_data['total_journeys'] > 0)
    ]
    # Instantiate the probability. It will get incremented through
    # the following iterations over the conversion groups, by
    # looking up the partial probabilities from the transition matrix
    p = 0
    for row in _d.itertuples():
        # Temporary list to record the probabilities found for the
        # current conversion path
        pr_this_path = []
        # Loop over the pairs for the current conversion path and
        # append the corresponding value from the transition matrix
        # to our temporary list
        for t in pairs(
            ['(start)']
            + row.path.strip('][').replace("'","").split(', ')
            + ['(end)']
        ):
            pr_this_path.append(trans_mat.get(t, 0))
        # Increment the total probability value by adding the multiplication
        # of the partial probabilies recorded for the current path
        p += reduce(mul, pr_this_path)
    return p
```

Code excerpt 18

```
markov = defaultdict(float)
# Calculate total conversion probability (without dropping channels)
p_conv = prob_convert(mk_data=markov_data, trans_mat=tr, drop='')
# Iterate channels to calculate probabilities with removal effects
for c in markov_channels:
    # Calculate the probability, applying channel removal effect and
    # subtract the result value from the total probability, to obtain
    # the actual channel contribution. Then, divide that result by the
    # total probability, to obtain the relative contribution of the channel.
    markov[c] = (
        ( p_conv - prob_convert(mk_data=markov_data, trans_mat=tr, drop=c) ) / p_conv
    )
```

Code excerpt 19

```
sum_all_values = sum(markov.values())
for _ in markov:
    markov[_] = np.around(markov[_]/sum_all_values*conv_sets.count())
return {'Values': markov, 'Transitions': tr}
```

Code excerpt 20

```
n_labels = ["BILLBOARD", "SITE VISIT", "SOCIAL", "VIDEO", "(end)"]
n_colors = ["rgba(38, 84, 124, 0.75)", "rgba(239, 71, 111, 0.75)",
"rgba(255, 209, 102, 0.75)", "rgba(6, 214, 160, 0.75)",
"rgba(252, 252, 252, 0.75)"]
# Source and target values are the node indexes from the n_labels list
l_sources = [0,0,0,0,0,1,1,1,1,1,2,2,2,1,2,3,3,3,3,3]
l_targets = [0,1,2,3,4,0,1,2,3,4,0,1,2,3,4,0,1,2,3,4]
l_values = []
l_colors = []
# Get the Pair Counts from the result dictionary
t_pairs = markovResult.get('PairCounts')
# Loop through the 'maximum' number of combinations for the dataset
for _ in np.arange(20):
    # Define the key to look for, looking up the lables through
    # the indexes from sources and targets (length is 20)
    t_key = n_labels[l_sources[_]], n_labels[l_targets[_]]
    # Look for the channel tuple key in the Pair Counts dictionary
    # and append the result to our values or zero if key is not found
    if t_key in t_pairs:
        l_values.append(t_pairs[t_key])
    else:
        l_values.append(0)
    l_colors.append(n_colors[l_sources[_]])
```

Code excerpt 21

```
rem - create the virtual environment named .atrio
C:\Users\user\code>py -3.9 -m venv .atrio
rem - assuming a GroupM folder with the ATRIO code files exists, change directory
C:\Users\user\code>cd atrio/groupm
rem - run the batch file created automatically on our virtual environment to activate
rem - Interactive Python in our terminal
C:\Users\user\code\atrio\groupm>c:/Users/user/code/.atrio/Scripts/activate.bat
rem - the (.atrio) prefix signals we're now in Interactive Python through our .atrio Virtual
Environment
rem - let's use pip to install the wheel library and update pip
(.atrio) C:\Users\user\code\atrio\groupm>pip install wheel
(.atrio) C:\Users\user\code\atrio\groupm>c:\users\user\code\.atrio\scripts\python.exe -m pip
install --upgrade pip
rem - the next steps install pandas, plotly, ipywidgets and voila
(.atrio) C:\Users\user\code\atrio\groupm>pip install pandas
(.atrio) C:\Users\user\code\atrio\groupm>pip install plotly
(.atrio) C:\Users\user\code\atrio\groupm>pip install ipywidgets
(.atrio) C:\Users\user\code\atrio\groupm>pip install voila
```