Analyzing Behavioral Differences in Satisfied vs. Dissatisfied Sessions

Understanding how user journeys differ between **satisfied** and **dissatisfied** sessions can reveal valuable behavioral patterns. Given 250k sessions (sequences of page hits length 5–150) with a satisfaction label (1 for satisfied, 0 for dissatisfied), there are several approaches to uncover distinguishing patterns. Below we outline **scalable methods** for quick insights as well as **slower, detailed analyses**, along with ideas for **visualizing** these differences. All methods avoid building a predictive model; instead, they focus on mining patterns from the existing sequences.

1. Transition Graphs & Markov Chain Analysis

One practical approach is to build **directed graphs of page transitions** for each user group (satisfied vs. dissatisfied). In these graphs, each node represents a page, and each directed edge represents a transition from one page to the next in the clickstream. By aggregating all sessions in each group into a graph (rather than one graph per session), you obtain a **state transition graph** for satisfied sessions and another for dissatisfied sessions. The edges can be weighted by frequency or probability of that transition in the group. This essentially corresponds to fitting a Markov Chain model to each segment of users, capturing how likely each navigation step is for that segment.

How to use these graphs: Once you have the transition graphs, compare them to spot differences. For example, an edge (Page A → Page B) might have a much higher weight in the dissatisfied graph than in the satisfied graph, indicating that dissatisfied users frequently follow that path (whereas satisfied users do not). Such differences may highlight *friction points* or *loops* that lead to dissatisfaction. You can quantify this by looking at the transition probability matrices for each Markov Chain and identifying which probabilities diverge significantly between the two groups. Key indicators to compare include:

- **Common vs. rare transitions:** Identify navigation steps that are common in one group but rare in the other.
- **Start and exit pages:** Determine if the distribution of **starting pages or ending pages** differs between satisfied and dissatisfied sessions. (For instance, perhaps a high fraction of dissatisfied sessions end on a support or error page.)
- **Looping or backtracking behavior:** Check if dissatisfied users are more likely to repeat certain pages or go back and forth (e.g. A → B → A), which might indicate confusion or

difficulty completing tasks.

Because constructing these graphs mostly involves counting occurrences of page transitions, this method is **highly scalable** for 250k sessions – essentially an $O(N)$ pass through the data to tally transitions. It will be much faster and more memory-efficient than exhaustive sequence mining. You can implement it with simple Python (e.g. using Pandas or collections to count transitions) or even leverage **Spark** for distributed counting if needed. The result is a first-order Markov Chain for each class, but you could also incorporate higher-order transitions if appropriate (e.g. including pairs of previous pages) at the cost of more complexity. Often a first-order (or second-order) chain is sufficient to capture the main behavioral flow.

For **visual interpretation**, the transition graph itself can be visualized. You might use a library like **NetworkX** or Graphviz to draw the graph, with edge thickness proportional to transition frequency. Another effective visualization is a **transition matrix heatmap**: create a matrix of probabilities for "From → To" transitions for each group, and display each as a heatmap (darker color = higher probability). For example, a heatmap of a transition matrix helps see which next-page transitions are most likely from each current page. By comparing the two heatmaps side by side (or subtracting one from the other), you can quickly spot which transitions are over-represented in the dissatisfied sessions.

2. Frequent Sequential Pattern Mining (PrefixSpan, SPADE, SPAM)

To capture longer patterns or more complex sequences of pages, you can turn to **sequential pattern mining** algorithms. These algorithms find subsequences that occur frequently in the dataset. In your case, you are especially interested in patterns that differentiate the two classes. There are a few strategies here:

- **Mine frequent patterns in each group separately:** Run a sequential pattern mining algorithm on just the dissatisfied sessions, and separately on the satisfied sessions. This will give you sets of frequent page sequences for each group (given some minimum support threshold). You can then compare the two sets of patterns to see which sequences appear for one group and not the other. For example, you might find that a sequence like [Home → Search → Product Page → **Help Page**] is frequent among dissatisfied users but either rare or non-existent among satisfied users, suggesting that reaching the help page could be a hallmark of frustration.

- **Find discriminative patterns directly:** Rather than mining both and comparing, some techniques aim to find **discriminative sequential patterns** (also called **contrast patterns** or **emerging patterns**) in a single step. These are subsequences whose occurrence frequency differs significantly between classes. In other words, they appear a lot in one class and far less in the other. Such patterns are very informative for characterizing the behavioral differences. For example, a pattern that appears in 15% of dissatisfied sessions but only 1% of satisfied sessions is a strong discriminative pattern for dissatisfaction. There are research algorithms for this (e.g. LEAP, DiffSeq, or **DDPMine** for discriminative pattern mining), though they may not be readily available in standard libraries. A simpler approach is to mine frequent sequences with a relatively low support threshold on the full data, then post-filter the results by computing support in each class and looking for large ratios or statistical significance differences (e.g. using a chi-square test for the 2x2 occurrence table of [pattern present/absent] vs [satisfied/dissatisfied]).

Common algorithms you can consider include **PrefixSpan**, **SPADE/cSPADE**, **SPAM**, and others:

- *PrefixSpan:* You already attempted PrefixSpan. It is a well-known algorithm for mining frequent sequential patterns, but it can be memory- and time-intensive on large datasets. Spark's MLlib has a PrefixSpan implementation that might be more scalable on a distributed cluster, but on a single machine with 250k sequences of up to length 150, it might indeed be too slow or cause memory issues.

- *SPADE (Sequential Pattern Discovery using Equivalence classes):* cSPADE is another algorithm that uses a depth-first search with a vertical database format to find frequent sequences. It can sometimes be faster than PrefixSpan depending on data characteristics. There are implementations in R and Java (e.g. in the SPMF library by Fournier-Viger), but fewer in pure Python. If you are open to using R or a Java library, you could try those; otherwise Python options include wrappers to SPMF.

- *SPAM (Sequential Pattern Mining using bitmaps):* The SPAM algorithm uses a bitmap representation to efficiently count occurrences of candidate sequences. There is a Python library called `sequence-mining` that implements SPAM in C++ for speed. Using SPAM might be beneficial as it was designed for efficiency. For instance, you could specify a minimum support (say 5% or 1% of sessions) and retrieve all sequences above that support. SPAM will output frequent sequences (with their support counts) quite quickly in many cases. You can then interpret those sequences and see which ones are class-specific.

Whichever algorithm you choose, **controlling the search space** is crucial for speed. You can do this by setting a relatively high **minimum support threshold** initially (to find only the most common patterns) and by possibly **limiting the max pattern length** to something reasonable (e.g. look for patterns up to length 3 or 4 first). Many algorithms will enumerate a huge number of patterns if the threshold is too low (especially given 250k sequences, even a 1% support means 2,500 sequences, which might allow a lot of rare patterns). Start with a higher threshold to get the top differences (you might even start with patterns that appear in, say, >10% of sessions in one group) then lower it gradually if needed. Also, consider **reducing the event space** by grouping similar pages into categories (as was done in some clickstream analyses) –

this makes patterns less sparse. For example, rather than treating every unique page URL as distinct, categorize pages by type (product page, category page, search results, help page, etc.), so that patterns emerge at a higher level.

After mining, compile a list of patterns with their support in satisfied vs. dissatisfied sessions. You could tabulate something like:

Pattern (sequence of pages)	**% of Dissatisfied sessions**	**% of Satisfied sessions**
A → B → C (example pattern)	12%	2%
X → Y (example pattern)	1%	9%
…	...	...

From such a list, you can pick out **differential patterns** easily. In the example above, Pattern A→B→C is much more common for dissatisfied users (perhaps indicating a problematic flow), whereas X→Y might be a **positive** pattern that mostly satisfied users follow (maybe a quick checkout path). These insights can then be investigated further (why is A→B→C leading to dissatisfaction? Is there a bug or UX issue on page C? etc.).

Performance note: Full sequential pattern mining can be slow on large data. If even the above optimizations don't work on the full 250k dataset, you might consider **sampling** (e.g. analyze 50k sessions first), or focus on **specific hypotheses** (like check frequencies of a limited set of patterns that you suspect). The **slower, exhaustive mining methods** will yield detailed patterns and potentially unexpected insights, but they may not scale easily. In contrast, bigram/trigram analysis (Markov chain) will scale fine but only captures short-range patterns. In practice, it's common to use the fast methods to narrow down candidates, then perhaps validate or explore a few longer sequences manually.

3. Statistical & Exploratory Analysis of Behavior Differences

Before diving into complex pattern mining, do some basic exploratory checks – these are fast and can guide your next steps:

- **Session Length and Click Depth:** Look at the distribution of session lengths (number of page hits) for satisfied vs. dissatisfied users. Are dissatisfied sessions typically shorter (e.g. users give up quickly) or longer (e.g. users struggling, spending more time without success)? A significant difference here might indicate different behavior – for example, if dissatisfied sessions tend to be longer, perhaps those users are looping around trying to find something.

- **Page Frequency Analysis:** Check which pages or actions are overrepresented in one group. For instance, maybe **“Help” or “Support” pages** occur in 20% of dissatisfied sessions but only 5% of satisfied sessions. Or an “Error” page appears mostly for the dissatisfied group. Conversely, a “Purchase Confirmation” page might appear mostly in satisfied sessions (assuming satisfaction might be tied to a successful purchase or task completion). Simply counting how often each page is visited in each group and comparing frequencies (via ratios or statistical tests) can pinpoint specific pages that correlate with dissatisfaction.

- **Position of events:** Similarly, look at *when* certain pages occur. Perhaps many dissatisfied sessions have a pattern where users go to the **FAQ page mid-session**, suggesting they got confused at that point. Or many satisfied sessions start with a certain landing page that seems effective. These kinds of observations might be gleaned by conditional probabilities (e.g., if on Page X, probability of eventually being satisfied vs not).

- **Sequential anomalies:** You could specifically test for known “frustration patterns”. For example, **repeated attempts** could be one: if a user visits the same page multiple times in a session (A → ... → A → ... → A), it might indicate they are stuck. You can search for such repeat patterns or high frequency of toggling between two pages (A ↔ B back and forth). If those are more prevalent in dissatisfied sessions, it’s a clear behavioral difference. Even without full sequential mining, you can detect simple repeats or back-and-forth patterns with tailored code (scanning each session for occurrences of A twice, or A then B then A, etc., and counting by group).

The above analyses are straightforward to do with Python (Pandas, NumPy) on a single machine and are **very scalable**. They don’t provide the full detail of longer sequences, but they can confirm or rule out certain hypotheses quickly.

4. Clustering and Session Profiling (Unsupervised)

Another angle is to use unsupervised learning to identify **common session profiles**, and then see how they relate to satisfaction. For example, you could perform a **clustering** of the sessions based on their sequences. One way to do this is to extract features from each sequence: they could be as simple as the frequency of each page in the session, or more complex like embedding the sequence into a vector space (using techniques like sequence2vec, or even treating page IDs like words and using a Doc2Vec model). Once you have a feature representation, cluster the sessions (k-means, hierarchical clustering, etc.) to find groups of similar behavior. You might discover clusters such as “quick checkout sessions”, “extensive product browsing sessions”, “search-dominated sessions”, etc. After clustering, check the **satisfaction rate in each cluster**. If you find a cluster where, say, 80% of the sessions are dissatisfied, then the pattern typical to that cluster is a likely culprit for dissatisfaction. This method can reveal patterns (in a broader sense) without predefining them. However, it requires more effort to set up (especially the sequence embedding part) and interpret, and may not be as directly actionable as explicit sequence patterns.

If clustering seems too complex, even doing a simpler segmentation could help. For instance, segment sessions by a simple criterion (like “contained a search action” vs “did not contain search”; or “went to help page or not”) and see how satisfaction differs. This is more of a hypothesis-driven approach but can validate what factors correlate with dissatisfaction.

5. Visualizing User Journey Differences

Finally, representing the differences visually will make them easier to interpret and communicate. Here are a few visualization ideas:

Example Sankey diagram illustrating customer journey flows in an e-commerce context. Each colored node is a page or state (e.g., Start, Product Page, Cart, Checkout, Exit), and the gray bands show the flow of users through these pages. The thickness of a band represents the volume of users taking that path. Such diagrams help identify where users go and where they drop off.

As shown above, **Sankey diagrams** are a great way to depict user flows from a starting point through various paths to outcomes. To compare satisfied vs. dissatisfied sessions, you could create two Sankey diagrams (one for each group) and place them side by side. This would let you visually contrast the common pathways. For example, you might see that in the satisfied Sankey, a large portion of users flow through “Product Page → Add to Cart → Checkout → Confirmation”, whereas in the dissatisfied Sankey, many users flow “Product Page → Reviews → Search → Product Page (different) → … → Exit”. The points at which the flows diverge or drop off are immediately apparent in a Sankey diagram. An alternative is to incorporate the outcome into a single Sankey: for instance, have a node at the end of each flow that is either “Satisfied” or “Dissatisfied”. This way, a single diagram can show common paths and how they split into the two outcomes. However, that can get a bit cluttered if the flows are complex; it might be simpler to do separate diagrams or highlight one outcome at a time.

State transition diagrams (graphs) are another option. They are similar to Sankey but more general – you draw nodes and directed edges for transitions. These are useful if you want to illustrate loops or complex branching clearly. You can annotate the edges with transition probabilities or frequencies from each group. Perhaps use color coding to indicate which transitions are **significantly higher for dissatisfied** (e.g. highlight in red) vs. those higher for satisfied (highlight in green). This overlay approach directly visualizes the **differential transitions** on one graph. If the graph is too dense (many pages), focus on the top N pages that cover, say, 80% of traffic, or a particular subset of interest (maybe the checkout flow).

As mentioned earlier, **heatmaps** are a compact visualization for transition matrices. You could create a grid of [From-page vs. To-page] with intensity representing probability. Creating one heatmap for each class and comparing them can show, for instance, that from Page “Search

Results”, satisfied users most often go to “Product Page”, while dissatisfied users often perform another search or exit (just as an example). Any cell that is bright in one heatmap and dim in the other is a notable difference.

Lastly, if you have access to specialized tools or libraries, consider using **process mining** techniques. Libraries like *PM4Py* in Python can take event logs (each session is a case, page hits are events) and derive a **process model** or **process map**. These often look like directed graphs with frequencies, similar to what we described manually. Process mining algorithms (like the inductive miner or heuristic miner) can automatically create a simplified model of the workflow and even highlight *variants*. You could potentially apply such tools to each group’s event log and compare the resulting models. The advantage is these tools are built to handle complex branching and can simplify less important paths for clarity.

Summary of recommendations: Start with the **transition graph/Markov analysis** and some basic frequency stats to get quick, scalable insights. This will likely pinpoint obvious differences (e.g. certain pages or transitions linked to dissatisfaction). Then, if needed, move to **sequential pattern mining** for more in-depth pattern discovery – using efficient algorithms (SPADE, SPAM, etc.) and focusing on **discriminative sequences** between the two groups. Keep an eye on runtime and perhaps use sampling or thresholds to manage the computation. Throughout, use **visualizations** like Sankey diagrams or state transition networks to interpret and present the findings. These approaches, used in combination, should help you identify the key behavioral differences between satisfied and dissatisfied users in your clickstream data, without requiring any predictive model building.

References

- Data Science Salon – *“Clickstream Data Mining With Markov Chain and cSPADE.”* (Illustrates using Markov chains for state transition probabilities and cSPADE for sequential patterns in clickstream analysis).

- Laurin Brechter (2023) – *“Customer Analytics: Pattern Mining on Clickstream Data in Python.”* Medium. (Demonstrates a workflow for mining frequent sequences using the SPAM algorithm with a support threshold, and discusses preprocessing steps to reduce sequence complexity).

- Research on discriminative sequential patterns – *“Discriminative sequential patterns are sub-sequences whose occurrences exhibit significant differences across data sets with different class labels.”* (Definition of patterns that differentiate classes, motivating the search for such patterns in satisfied vs dissatisfied sessions).

- Quantum Metric Blog – *“Sunburst vs. Sankey diagram: differences”* (Notes that Sankey diagrams provide a graphical representation of paths customers take to reach outcomes, whether desired (e.g. purchase) or undesired (e.g. exit due to frustration)).

- Data Science Salon – *State transition heatmap example* (Mentions using a heat map to represent transition probability matrices, useful for visualizing which transitions are most likely from each state).