**User Journey Analysis in Python Project**

**Preprocessing the Data**

In this project, you are provided with data containing the user journeys of people who bought our product. You need to create Python programs to analyze the sequence of visited pages with the objective of improving the front page flow and identifying which pages are important.

But before analyzing this data, you must first clean it and prepare it for the next step.

For this part of the process, you must create a Python program with three functions to help you transform the data into a more analysis-ready state and then export this new data to a CSV.

To begin, inspect the CSV itself and see if there is any need to clean the data. After inspection, you should notice that some user journey strings have multiple duplicate pages, one after another. While the Homepage reference in journeys like Homepage-Pricing-Homepage might be helpful for the analysis, the repeating reference in Homepage-Homepage-Homepage-Pricing is not.

The first function you need to create removes sequences of repeating pages. It should leave just a single entity in the place of the sequence. But it should only apply where the duplicate page is replicated sequentially. So, it should do nothing in the first example (Homepage-Pricing-Homepage) while replacing the second (Homepage-Homepage-Homepage-Pricing) with Homepage-Pricing. This operation should be done for each row of data.

The function details are as follows:

* Example name: remove\_page\_duplicates
* Input parameters:
  + data – the dataframe containing all the data
  + target\_column – the name of the column containing the user journey strings (default is 'user\_journey')
* Output: It should return a new dataframe with the cleaned-up journey strings. It should not modify the original dataframe.

Next, look at the structure of the data. Currently, there is a row for every session of the user. But when considering a user’s journey, we’re interested in the page sequences instead of the specific sessions. To prepare the data for the analysis, you'll need to group a single user's journey strings into one big string—which is what the second function will do.

Make the function as general as possible—grouping all the sessions will not suffice. What if we later decide that we want to consider just the first 10 sessions or the last 3? This is a component you need to add to this function, possibly achieved in many ways. Below, you can find one possible implementation.

The function details are as follows:

* Example name: group\_by
* Input parameters:
  + data – the dataframe containing all the data
  + group\_column – the name of the column which we want to group into a single record (default is 'user\_id')
  + target\_column – the name of the column containing the strings (default is 'user\_journey')
  + sessions – the number of sessions to group; if it’s the string 'All', consider all sessions (default is 'All')
  + count\_from – either 'first'; or 'last'; indicates what to group if the session parameter is an integer—e.g., if sessions is 10 and count\_from is 'last', the function should group only the last 10 sessions (the grouping order should still remain the same - from the earliest to latest session) (default is 'last')
* Output: The function should return a new dataframe that contains the grouped strings. It should not modify the supplied one.

The final function that remains removes unnecessary pages from the data. (Not all pages are essential in a user journey analysis.) Perhaps prompts like ‘log in’ should be removed. But this is not something we can hardcode into the preprocessing because it’s a decision that the data scientist can make and tinker with. That’s why we should create a function that can be called upon later if needed.

Note the details below:

* Example name: remove\_pages
* Input parameters:
  + data: the dataframe
  + pages – a list containing the strings of all the pages to be removed
  + target\_column – the name of the column containing the strings (default is 'user\_journey')
* Output: Return a new dataframe with the removed pages

Now that you’ve created all these functions, you can use them on the data to generate the CSV you’ll utilize in the next part of the project. At this point, you can use some default settings, such as grouping all sessions and not excluding any pages from the journey yet. Just make sure that you remove the duplicates only after you’ve used the other two functions.

**Analyzing the Data**

Given the preprocessed data, you can begin your analysis in a new notebook to keep matters clean. Now is the time to think what metrics we can generate to obtain valuable statistics about the behavior of purchasing customers. Please take your time and think of as many such metrics as possible. Meanwhile, consider the following list of metrics we’d like you to attempt to successfully complete the project:

* **Page count** is the most fundamental metric; it counts how many times each page can be found in all user journeys.
* **Page presence** is similar to ‘page count’ but counts each page only once if it exists in a journey; it shows how many times each page is part of a journey
* **Page destination** is a metric that shows the most frequent follow-ups after every page. It looks at every page and counts which pages follow next. If one is interested in what the users do after visiting page X, they can consult this metric.
* **Page sequences** look at what the most popular run of N pages is. I will consult this metric if I’m interested in the sequence of three (or any other number) pages that most often shows up. Count each sequence only once per journey.
* **Journey length** is a straightforward metric that considers the average length of a user journey in terms of pages.

You can create a function for each of these key metrics. Recall, however, that the data provided also had a subscription plan column. A vital part of the analysis is finding patterns and differences between buyers’ behavior of different plans. For example, compare the journey of monthly users versus the one of annual users. This is why it’s essential to incorporate a plan parameter in these functions, allowing a data scientist to obtain the metrics for all subscription plans or any specific one.

Completing all of this, you should’ve created the necessary code to make a complete analysis. You can check what all the metrics give and tinker with the preprocessing to see how the metrics change.