Metro Car Report

Ethan Golledge

17th of November 2023

1 Summary

Our analysis has examined our customer funnel in full, measured using the percent of previous and percent of top metrics. This report aims to pinpoint critical weaknesses in our service, highlighting areas where we are losing a significant portion of our customer base.

Our investigation metrics revolved around two perspectives: users and ride bookings. We found that only 50.8% of users progress from ride acceptance to ride completion and 64.4% of ride requests progress from a ride request to ride acceptance. We have drilled into metrics that could potentially explain these areas of weakness but based on our current data, we have concluded that we need to perform further research before making decisions on how to move forward.

Our conclusions and recommendations shed light on areas needing attention to address low conversion at stages involving ride requests, acceptances, and completions.

Stakeholder inquiries guided us to recognise IOS as our primary platform, which we suggest prioritising marketing efforts towards this platform. The age group of 35-44 emerged as our target demographic, contributing the highest revenue and comprising the largest user percentage. Furthermore, we've identified specific time frames—8am-9am and 4pm-7pm—as time periods for potential surge pricing implementation.

2 Context

In this report we aim to analyse MetroCar's customer funnel to pinpoint opportunities for improvement. We will look at specific areas of the app, leveraging a funnel chart to identify significant drop-offs or conversions at key customer interaction stages. Utilising data from January 2021 to April 2022 stored across seven Postgres tables.

Throughout this analysis, I'll be using the following tools:

- Beekeeper: for data exploration, processing and formatting.
- Python: for detailed exploration of specific areas.
- Tableau: for visualising and presenting the observations.

Our initial step involves listing the seven tables along with their respective fields, as these will serve as reference points throughout the report.

- **app_downloads**: contains information about app downloads
    - app_download_key: unique id of an app download
    - platform: ios, android or web
    - download_ts: download timestamp
- **signups**: contains information about new user signups
    - user_id: primary id for a user
    - session_id: id of app download
    - signup_ts: signup timestamp
    - age_range: the age range the user belongs to
- **ride_requests**: contains information about rides
    - ride_id: primary id for a ride
    - user_id: foreign key to user (requester)
    - driver_id: foreign key to driver
    - request_ts: ride request timestamp
    - accept_ts: driver accept timestamp
    - pickup_location: pickup coordinates
    - destination_location: destination coordinates
    - pickup_ts: pickup timestamp

    - dropoff_ts: dropoff timestamp
    - cancel_ts: ride cancel timestamp (accept, pickup and dropoff timestamps may be null)
- **transactions**: contains information about financial transactions based on completed rides:
    - ride_id: foreign key to ride
    - purchase_amount_usd: purchase amount in USD
    - charge_status: approved, cancelled
    - transaction_ts: transaction timestamp
- **reviews**: contains information about driver reviews once rides are completed
    - review_id: primary id of review
    - ride_id: foreign key to ride
    - driver_id: foreign key to driver
    - user_id: foreign key to user (requester)
    - rating: rating from 0 to 5
    - review: text response given by user/requester

To conduct a funnel analysis, we will track the sequential steps or stages a user undergoes. Our approach will follow the order of the tables listed above, progressing from the initial stage of downloads to the final stage of reviews.

3 Analysis in SQL

To understand our funnel, we will examine data at both user and ride levels, offering two distinct perspectives on our performance.

We will be counting the number of records in the users and rides columns, and it's important to note that both the user_id and ride_id exhibit high cardinality in each respective table. This indicates a substantial number of unique values. To perform a comprehensive funnel analysis, our primary requirement is the total count within each step that a user progresses through. Therefore, to manage our current dataset efficiently, we will need to aggregate the data. This approach allows us to work with a dataset that is less resource intensive.

When examining each stage of the funnel, distinct tables will be referenced. However, concerning rides, certain conditions need to be applied to ensure accurate identification of user_id and ride_ids. In particular, the stages of ride requested, ride accepted, and ride completed will involve distinct parameters obtained from the same table. These parameters encompass accept_ts, dropoff_ts, and cancel_ts, allowing us to navigate through the specific steps outlined in 6.1, this process will be repeated across other queries.

To assess our funnel, we will be using two primary metrics to compare the stages of the funnel.

- **Percent of Previous** - This metric calculates the percentage of users or events progressing from one stage of the funnel to the next.
  It determines the conversion rate by dividing the number of users or events in a particular stage by the number in the preceding stage. This metric will track user progression through each step of the funnel.

- **Percent of top** – This metric calculates the percentage of users or events that reach a specific stage of the funnel relative to the total number of users or events at the top of the funnel.
  It determines the conversion rate by dividing the number of users or events in a specific stage by the total number of users or events at the top of the funnel.

3.1 User-level

Initially, we'll examine our funnel at a user level granularity. This analysis will include every table, both 'app_downloads' and 'signups' are at user level granularity but the subsequent tables will contain duplicate records for the same 'user_id,' reflecting multiple contributions made by users who are actively engaged with the app. To navigate these circumstances, we will be using COUNT(DISTINCT()) to ensure that we are only counting unique user records.

We will need to execute separate queries for each table to count the unique user rows. Using the UNION operator, we'll merge SELECT statements. We need to ensure that each SELECT statement retrieves the same number of columns with aligned column aliases, especially for the count of users. While the counting procedure remains consistent, the columns being counted vary across tables. App_downloads will require us to count the app_download_key, but the remaining tables will count user_id.

We will query all seven tables to retrieve the funnel_name, step, unique user count, 'percent of previous,' and 'percent of top.'
Calculations for the latter two columns will utilise the LAG() function and the provided formulas above to derive the percentage values. It was essential to ensure that our numerator produced a float value, allowing interpretation as a percentage. Additionally, the denominator needed to refer to either the previous or initial unique user count. See results from this query in Figure 1 below.

funnel_name	user_count	perc_previous	perc_top
app_downloads	23608	(NULL)	(NULL)
signups	17623	0.74648424262961...	(NULL)
ride_requested	12406	0.70396640753560...	(NULL)
ride_accepted	12278	0.98968241173625...	(NULL)
ride_completed	6233	0.50765597002769...	(NULL)
approved_transaction	6233	1	(NULL)
reviewed	4348	0.69757741055671...	0.18417485598102337

Figure 1: Result set from query [1]

3.2 Ride-level

Next, we will examine our funnel at the ride level granularity. This analysis won't include all seven tables, as 'app_downloads' and 'signups' have null count values in this context. Rides commence with the funnel step 'ride_requests,' allowing us to focus on the number of rides at each stage of the funnel. Regarding ride identification, we have two options: utilising unique ride IDs or user IDs, the latter being non-unique within the tables we are utilising. For this query, I will use the ride_id and ensure accuracy by applying COUNT(DISTINCT()).

We will be following the same logic as before, so we will be making use of the UNION operator.

To maintain continuity, we will incorporate all tables for the step and funnel_name. However, our query will involve five tables, starting from 'ride_requests,' to retrieve the step, funnel_name, unique ride count, 'percent of previous,' and 'percent of top.'

Calculations for the latter two columns will utilise the LAG() function and the provided formulas above to derive the percentage values. It was essential to ensure that our numerator produced a float value, allowing interpretation as a percentage. Additionally, the denominator needed to refer to either the previous or initial unique user count.

See results from this query in Figure 2 below.

funnel_name	ride_count	perc_previous	perc_top
app_downloads	(NULL)	(NULL)	(NULL)
signups	(NULL)	(NULL)	(NULL)
ride_requested	385477	(NULL)	(NULL)
ride_accepted	248379	0.6443419451744202	(NULL)
ride_completed	223652	0.9004464950740602	(NULL)
approved_transaction	212628	0.9507091374099047	(NULL)
reviewed	156211	0.734668058769306	0.4052407796055277

Figure 2:Result set from query [2]

3.3 Summary of users and ride funnel

Having the metrics displayed for our funnel, we can now identify significant drop-offs. The top-ranking records for both 'percent of previous' and 'percent of top' are as follows:

Percent of Previous:

- Users Funnel: For each ride accepted, 50.1% progress to having the ride completed.
- Rides Funnel: For each ride requested, 64.4% progress to having the ride accepted.

Percent of Top:

- Users Funnel: 18.4% of users who downloaded the app proceed to writing a review.
- Rides Funnel: For each ride requested, 40.5% lead to writing a review.

These figures are preliminary; eventually, I'll present these totals and metrics visually in Tableau. This approach will eliminate any null values for 'percent of top,' and for the rides funnel, 'app_downloads' and 'signups' will be excluded as they don't offer significant insight.

For the moment, we will address stakeholder inquiries and explore potential elements to include in a more extensive query in the next section of the report. This query will involve grouping data points, enabling us to filter and generate more detailed visuals. These visuals aim to better grasp stakeholders' concerns and provide a deeper understanding.

3.4 Crafting our Plans for the Next Query

Stakeholders have raised multiple business inquiries that we aim to explore to provide recommendations derived from our observations. To achieve this, we'll develop a set of Tableau dashboards concentrating on the associated subjects, elaborated upon in the following sections.

1. What steps of the funnel should we research and improve? Are there any specific drop-off points preventing users from completing their first ride?

   In Tableau we will create a funnel chart offering the choice between ride-level or user-level granularity. Both perspectives provide valuable insights into customer behaviour, allowing users to select total rides or users at each step of the funnel. Inspired by our preliminary SQL query, each side of the visual will depict either 'percent of previous' or 'percent of top.' By visualising these metrics, we can promptly identify specific drop-offs within the funnel.

2. Metrocar currently supports 3 different platforms: ios, android, and web. To recommend where to focus our marketing budget for the upcoming year, what insights can we make based on the platform?

   To identify which platform a user is using, we will refer to the 'signups' table as it contains this specific information. Given that this is the sole table with this data, we will need to execute the relevant join to ensure that each stage of the funnel incorporates this information. Subsequently, we will perform a 'group by' operation based on ‘platform’, allowing us to count the number of users or rides within each platform.

3. What age groups perform best at each stage of our funnel? Which age group(s) likely contain our target customers?

   To identify the age group of users, we will once again refer to the 'signups' table as it holds this information. Similar to the process for platform identification, we will apply the necessary joins to integrate this data into each stage of the funnel. Following this, we will execute a 'group by' operation based on 'age_range,' enabling us to count the number of users or rides within each specific age group.

4. Surge pricing is the practice of increasing the price of goods or services when there is the greatest demand for them. If we want to adopt a price-surging strategy, what does the distribution of ride requests look like throughout the day?

   For this inquiry, it's fitting to segment our data based on timestamps. While there's an array of timestamps available, we must decide on the most effective method to extract them for segmentation purposes. Apart from the reviews table, every table contains a timestamp. However, we need to group by a single timestamp to ensure accurate counts for each stage of the funnel.

   Consequently, we'll pursue two separate approaches. Initially, we'll utilise 'download_ts' to construct the primary part of the funnel. Given that the 'downloads' table is at the top of the funnel, every user will have a 'download_ts'. This approach won't be ideal for specific ride-related columns. For instance, when examining ride-related timestamps, such as the time of day a ride is requested, utilising 'download_ts' will retrieve information linked to the 'download_ts'. Therefore, we'll conduct a separate query using 'request_ts' from the 'ride_requests' table. This will enable us to visualise the distribution of ride requests throughout the day. I will segment the data through grouping by the extracted year, month, and hour. Further details explaining why I will exclude the day will be provided below.

5. What part of our funnel has the lowest conversion rate? What can we do to improve this part of the funnel?

This query will be addressed through our funnel chart, coupled with an array of metrics to delve deeper into the factors influencing the lowest conversion rates.

These stakeholder inquiries will be guiding our next set of queries, I will be addressing the questions in further detail later in the report. Now that we have the foundations of how we will move on with creating a new dataset, we can think in further detail about how we are going to be format our data in a way that will be compatible with tableau.

3.5 Users query [3]

There are a number of attributes we want to add to our funnel and we want to address the cardinality of our new query.

1. Funnel cardinality = 7
2. Platform cardinality = 3 - IOS, Android, Web
3. Age Group cardinality = 6 - 18-24, 25-34, 35-44, 45-55, Not Specified, Unknown
4. Hour cardinality = 24
5. Month cardinality = 12
6. Year cardinality = 2

Including additional attributes in the query directly escalates the dataset's row count, contradicting our initial objective of minimising unique values. Such an approach often triggers data explosion, marked by an exponential surge in unique values within the dataset, leading to a substantial overall size increase.

To avoid this, we will restrict our query to group data based on specific attributes like platform, age group, hour, month, and year. Although this means excluding grouping by the date, the attributes we have decided on are pivotal for effectively addressing stakeholder inquiries. Additionally, we can execute aggregations on other columns to maintain a meaningful dataset without compromising the size of our dataset.

In the queries provided below, I've structured a series of left joins that correspond to the stages of the funnel, so that with each Common Table Expression (CTE), only data from the left will be present for counting users or rides. Due to variations in primary keys among tables, I have formed connections based on these keys between each table, detailed in the appendix. Each CTE contains the essential data necessary for the specific stage it represents. At the end stage of the query, I have used a UNION operator to combine each CTE so that we can extract the dataset into a CSV file.

Here are some functions and techniques used:

- EXTRACT() function: Used to transform existing timestamps into a numerical format, allowing for year, month and hour columns to be extracted and also calculation of various metrics based on time differences.
- CAST() function: Ensures that the UNION statements have the same number of columns in each join. Specifying data types helps execute the query, particularly in cases where certain funnel steps have no relevance.
- SUM(CASE WHEN()): Employed to count the number of cancellations by dividing them with the number of requests, yielding the proportion of cancellations to requests.
- COALESCE() function: Used to replace null values with 'Not specified' specifically in the age range column, providing a more informative representation of missing values.

This query will compile data concerning the six attributes listed above. Its design is relatively straightforward as no additional data needed to be incorporated into my funnel chart.

3.6 Rides query [4]

This query includes a more extensive range of columns. The stages such as 'ride_requested,' 'ride_accepted,' and 'ride_completed' exhibit notably lower conversions. To address this, I've retrieved additional information with the aim of constructing a comprehensive dataset. This expanded dataset will enable me to create detailed data visualisations and effectively address questions 1 and 5.

Additional metrics included:

- The average time between a ride request and acceptance.
- The average time between a ride request and pickup.
- The average time between a ride request and cancellation.
- The average time between ride acceptance and pickup.
- The average time between ride acceptance and cancellation.
- The average ride duration.
- Proportion of cancellations.
- The total sum of purchase amounts.
- The average rating.

These metrics made use of AVG(), EXTRACT(MINUTE()) AND SUM(). To see specifically how I have done this, please refer the appendix [4].

4 Python

4.1 Detailed Exploration of the Reviews table

Whilst constructing the rides query [4], I noticed the presence of the review column containing users' feedback, potentially pointing us to flaws in our service. Due to the text format of this data, its direct inclusion in SQL query would result information loss. Therefore, I opted to perform a detailed analysis using Python and Pandas library to dive into this column.

The objective was to construct a basic supervised learning script capable of categorising reviews into three attributes: positive, satisfactory, and negative sentiments. To begin, it was essential to extract the necessary data from relevant tables. Primarily, the 'reviews' table was required, supplemented by additional ride request details. For this, I performed a left join operation between 'reviews' and 'ride_requests' resulting in a query which extracts columns such as user_id, year, month, and hour, rating and review.

However, the 'reviews' table contained a vast amount of data, with 156,211 datapoints, posing a challenge for analysis within Beekeeper. To address this, a random sampling approach was adopted using the 'ORDER BY random()' clause to ensure a representative dataset spanning from January 2021 to April 2022. This sampling yielded 49,999 rows of data, subsequently extracted into a CSV file for further analysis in our Jupyter Notebook environment. [5]

```
# imports
import pandas as pd
from collections import Counter
```

```
# transferring the csv into a pandas dataframe, check for a successful transfer
review_df = pd.read_csv('metrocar_review.csv')
review_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 49999 entries, 0 to 49998
Data columns (total 6 columns):
 #   Column   Non-Null Count  Dtype 
---  ------   --------------  ----- 
 0   user_id  49999 non-null  int64 
 1   year     49999 non-null  int64 
 2   month    49999 non-null  int64 
 3   hour     49999 non-null  int64 
 4   rating   49999 non-null  int64 
 5   review   49999 non-null  object
dtypes: int64(5), object(1)
memory usage: 2.3+ MB
```

Figure 3: Transferring the CSV into a pandas data frame

The review dataset we obtained contains no null values, and the column datatypes align perfectly with the requirements for subsequent analysis in Python. Although our primary focus will be on the 'review' column, the additional columns will serve as way to join datasets in Tableau.

Anticipating the variety of reviews present in this column, I identified and categorised five distinct review types: App, Driver, Ride, Price, and Wait Time. To facilitate targeted analysis, I compiled lists starting with keywords corresponding to each review type. The rationale behind this approach was to efficiently detect and categorise reviews accurately based on their content.

For instance, consider a review like "this app was amazing, but the car had a horrid smell." In this case, it was essential to correctly classify this review as positive for the app aspect and negative for the ride aspect. To achieve this, I developed lists comprising keywords associated with each review type and their corresponding sentiments. This meticulous process aimed to accurately attribute each review to its relevant category.

Admittedly, distinguishing between these categories was challenging due to potential overlaps in word usage across various contexts. However, the subsequent section of the notebook is dedicated to counting the occurrences of each keyword, enabling a more precise categorisation of reviews.

```
for aspect in aspects:
    if aspect == app_keywords:
        for sentiment, sentiment_keywords in app_sentiments.items():
            column_name = f'app_{sentiment}'
            review_df[column_name] = review_df['review'].str.contains(sentiment_keywords, case=False, regex=True)

            review_df[column_name] = review_df[column_name].astype(int)

    elif aspect == driver_keywords:
        for sentiment, sentiment_keywords in driver_sentiments.items():
            column_name = f'driver_{sentiment}'
            review_df[column_name] = review_df['review'].str.contains(sentiment_keywords, case=False, regex=True)

            review_df[column_name] = review_df[column_name].astype(int)

    elif aspect == ride_keywords:
        for sentiment, sentiment_keywords in ride_sentiments.items():
            column_name = f'ride_{sentiment}'
            review_df[column_name] = review_df['review'].str.contains(sentiment_keywords, case=False, regex=True)

            review_df[column_name] = review_df[column_name].astype(int)

    elif aspect == price_keywords:
        for sentiment, sentiment_keywords in price_sentiments.items():
            column_name = f'price_{sentiment}'
            review_df[column_name] = review_df['review'].str.contains(sentiment_keywords, case=False, regex=True)

            review_df[column_name] = review_df[column_name].astype(int)


    elif aspect == wait_time_keywords:
        for sentiment, sentiment_keywords in wait_time_sentiments.items():
            column_name = f'wait_time_{sentiment}'
            review_df[column_name] = review_df['review'].str.contains(sentiment_keywords, case=False, regex=True)

            review_df[column_name] = review_df[column_name].astype(int)
```

Figure 4:Analysis of customer reviews

The code snippet above encompasses a loop structure designed to identify the specific aspect under consideration. Utilising conditional statements (if/elif), this code navigates to subsequent loops once a statement is validated. The secondary loop aims to determine the type of attribute present, generating a designated column for each review category. By categorising keywords and their respective sentiments as 0 or 1, this process enables the summation of total occurrences for each keyword and attribute. The aim of these new keyword and sentiment columns was to sum up the total in Tableau, with 1 being equal to True and 0 equal to False.

This methodology was adopted to efficiently discern and tally the frequency of individual attribute mentions within the reviews.

The outcome of this process was the generation of 15 columns. While this could be useful for analysis, our primary aim was to associate each keyword with its respective attribute: 'positive', 'satisfactory', or 'negative'. We structured the data in this format to ensure compatibility with Tableau, as it recognises and processes data standardised tabular format. The end objective of this script is to create visualisations for individual keywords, representing the percentage of the total in a clear and interpretable way.

```
def map_values(row):
    if row['app_positive'] == 1:
        return 'positive', 1
    elif row['app_satisfactory'] == 1:
        return 'satisfactory', 1
    elif row['app_negative'] == 1:
        return 'negative', 1
    else:
        return '', 0

def map_driver_values(row):
    if row['driver_positive'] == 1:
        return 'positive', 1
    elif row['driver_satisfactory'] == 1:
        return 'satisfactory', 1
    elif row['driver_negative'] == 1:
        return 'negative', 1
    else:
        return '', 0

def map_ride_values(row):
    if row['ride_positive'] == 1:
        return 'positive', 1
    elif row['ride_satisfactory'] == 1:
        return 'satisfactory', 1
    elif row['ride_negative'] == 1:
        return 'negative', 1
    else:
        return '', 0

def map_price_values(row):
    if row['price_positive'] == 1:
        return 'positive', 1
    elif row['price_satisfactory'] == 1:
        return 'satisfactory', 1
    elif row['price_negative'] == 1:
        return 'negative', 1
    else:
        return '', 0

def map_wait_time_values(row):
    if row['wait_time_positive'] == 1:
        return 'positive', 1
    elif row['wait_time_satisfactory'] == 1:
        return 'satisfactory', 1
    elif row['wait_time_negative'] == 1:
        return 'negative', 1
    else:
        return '', 0


# apply each function to create a keyword columns and a value column
review_df['app'], review_df['app_val'] = zip(*review_df.apply(map_values, axis=1))
review_df['driver'], review_df['driver_val'] = zip(*review_df.apply(map_driver_values, axis=1))
review_df['ride'], review_df['ride_val'] = zip(*review_df.apply(map_ride_values, axis=1))
review_df['price'], review_df['price_val'] = zip(*review_df.apply(map_price_values, axis=1))
review_df['wait_time'], review_df['wait_time_val'] = zip(*review_df.apply(map_wait_time_values, axis=1))
```

Figure 5: Standardising into tabular format

Above are five functions designed to categorise each keyword and its attribute, resulting in 10 columns: one for each keyword and its corresponding attribute, with a values column with values of 0 or 1.

Following the functions, my objective was to determine the frequency of occurrences for each keyword and their sentiments. This step would provide insight into the nature of reviews received for each of the five categories.

```python
# Looking at the first five rows determines if the code works but now we will check the total count for each column

aspects = ['app', 'driver', 'ride', 'price', 'wait_time']

for aspect in aspects:
    print(f'{aspect.capitalize()}:')
    aspect_columns = [f'{aspect}_{sentiment}' for sentiment in app_sentiments.keys()]
    for column in aspect_columns:
        print(f'{column}: {review_df[column].value_counts()}')
    print()
```

Figure 6: Counting each category of customer reviews

The results from categorisation of customer reviews is shown below.

```
App:
app_positive: app_positive
0    45699
1     4300
Name: count, dtype: int64
app_satisfactory: app_satisfactory
0    42048
1     7951
Name: count, dtype: int64
app_negative: app_negative
0    47528
1     2471
```

```
Driver:
driver_positive: driver_positive
0    33570
1    16429
Name: count, dtype: int64
driver_satisfactory:  driver_satisf
actory
0    48219
1     1780
Name: count, dtype: int64
driver_negative: driver_negative
0    32459
```

```
Wait_time:
wait_time_positive:  wait_time_posi
tiv
0    48666
1     1333
Name: count, dtype: int64
wait_time_satisfactory:  wait_time_
satisfactory
0    49535
1      464
Name: count, dtype: int64
wait_time_negative:  wait_time_nega
```

```
Price:
price_positive: price_positive
0    47468
1     2531
Name: count, dtype: int64
price_satisfactory:  price_satisfac
tory
0    46424
1     3575
Name: count, dtype: int64
price_negative: price_negative
0    48586
```

```
Ride:
ride_positive: ride_positive
1    29431
0    20568
Name: count, dtype: int64
ride_satisfactory:  ride_satisfacto
ry
0    42294
1     7705
Name: count, dtype: int64
ride_negative: ride_negative
0    36393
```

Figure 7: Customer review results

Up to this point, progress has been good. However, it's also crucial for us to understand the content and frequency of the words mentioned in the reviews. While the ideal approach would have been to export this data to Tableau, allowing us to apply filters and observe word mentions throughout the day, for simplicity, I intended to work with numeric data separately. I wanted to avoid data explosion as best as I could, and we can aggregate values to lower the size of the dataframe to achieve this.

This hard coded data will accompany a separate visualisation, offering insight into the most frequent occurrences throughout our history.

```
# Alright, we have the total count of each keyword and their aspects
# But what are people mentioning the most?
# The function counts the most occuring words in each keyword aspect
# It then prints the string of the most occuring word

def count_words_in_reviews(dataFrame, sentiment_keywords, sentiment_name):
    word_counts = Counter()
    for review in dataFrame['review']:
        for word in sentiment_keywords.split('|'):
            count = review.lower().count(word)
            word_counts[word] += count

    print(f'TOP 5 occurrences in {sentiment_name}:')
    for word, count in word_counts.most_common(5):
        print(f'{word}: {count}')

aspects = {
    'App': {
        'Keywords': app_keywords,
        'Sentiments': app_sentiments,
    },
    'Driver': {
        'Keywords': driver_keywords,
        'Sentiments': driver_sentiments,
    },
    'Ride': {
        'Keywords': ride_keywords,
        'Sentiments': ride_sentiments,
    },
    'Price': {
        'Keywords': price_keywords,
        'Sentiments': price_sentiments,
    },
    'Wait Time': {
        'Keywords': wait_time_keywords,
        'Sentiments': wait_time_sentiments,
    }
}
```

```
try:
    for aspect, data in aspects.items():
        print(f'{aspect}:')
        for sentiment, keywords in data['Sentiments'].items():
            count_words_in_reviews(review_df, keywords, sentiment)
        print()

except Exception as e:
    print(f"""An Error occured: {e}""")
```

Figure 8: Examining the keywords from customer reviews

Here is a function and subsequent loop that examines the keywords and identifies the top 5 occurrences of words mentioned. As the loop encounters similar sentiments, it increments the count of each word and prints the results so we can understand the prevalent topics discussed by customers.

The results from examining the keywords from customer reviews are shown below.

```
App:
TOP 5 occurrences in positive:
convenience: 1764
easy: 1308
unmatched: 1262
go-to: 1228
user_friendly: 0
TOP 5 occurrences in satisfactory:
glitches: 2363
decent: 1813
could have been more precise: 1308
was not accurate: 1235
could be more accurate: 1232
TOP 5 occurrences in negative:
crashed repeatedly: 1460
constant glitches: 536
incorrect arrival time: 475
incorrect arrival times: 475
```

```
Driver:
TOP 5 occurrences in positive:
professional: 6508
friendly: 5488
polite: 3798
courteous: 1806
punctual: 1282
TOP 5 occurrences in satisfactory:
could have been better: 1293
communication problems: 487
TOP 5 occurrences in negative:
rude: 3515
unprofessional: 3490
unfamiliar: 2795
poor: 2435
horrible: 1534
```

```
Ride:
TOP 5 occurrences in positive:
comfortable: 7242
reliable: 5195
good: 3902
smooth: 3814
clean: 3513
TOP 5 occurrences in satisfactory:
longer: 5108
fine: 1800
would be helpful: 1293
TOP 5 occurrences in negative:
uncomfortable: 3453
terrible: 2941
unsafe: 1433
broke down: 1415
wrong: 1415
```

```
Price:
TOP 5 occurrences in positive:
quickly: 1273
saves: 1258
affordable: 1258
TOP 5 occurrences in satisfactory:
increased: 1804
higher than expected: 1771
TOP 5 occurrences in negative:
overcharged: 1413
```

```
Wait Time:
TOP 5 occurrences in positive:
promptly: 1333
TOP 5 occurrences in satisfactory:
preferred: 464
TOP 5 occurrences in negative:
late: 2941
last minute: 2002
stranded: 1999
wrong: 1415
incorrect: 475
```

Figure 9: Customer reviews keyword results

Now that we've gathered all the necessary information, it's time to transform this new dataframe into the final format before proceeding. Initially, we will remove the columns that represent each keyword and sentiment, essentially the original 15 columns, as they've been formatted into a standardised structure.

```
columns_to_drop = [
    'app_positive', 'app_satisfactory', 'app_negative',
    'driver_positive', 'driver_satisfactory', 'driver_negative',
    'price_positive', 'price_satisfactory', 'price_negative',
    'ride_positive', 'ride_satisfactory', 'ride_negative',
    'wait_time_positive', 'wait_time_satisfactory', 'wait_time_negative'
]

review_df.drop(columns=columns_to_drop, inplace=True)
```

Figure 10: Formatting the dataframe

Subsequently, we will execute a grouping operation based on year, month, hour, app, driver, ride, price, and wait_time. This approach serves two purposes: first, it facilitates the data join onto the data from the ride requests query whilst in Tableau, and secondly, it allows us to acquire a comprehensive count for each keyword and attribute so that we can visualise these findings to stakeholders.

```
# Finally we need to perform a group by so we can link these new columns to our funnel data
# I decided the best way to do this would be to group by the time components as we do not have a common key
# We're also going to be grouping by each keyword to reduce the number of rows
# This format will be useful for tableau visualisations

grouped_df = review_df.groupby(['year','month', 'hour', 'app', 'driver', 'ride', 'price', 'wait_time']).agg({
    'app_val': 'sum',
    'driver_val': 'sum',
    'ride_val': 'sum',
    'price_val': 'sum',
    'wait_time_val': 'sum'
}).reset_index()

# Print the resulting DataFrame
grouped_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 6415 entries, 0 to 6414
Data columns (total 13 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   year           6415 non-null   int64
 1   month          6415 non-null   int64
 2   hour           6415 non-null   int64
 3   app            6415 non-null   object
 4   driver         6415 non-null   object
 5   ride           6415 non-null   object
 6   price          6415 non-null   object
 7   wait_time      6415 non-null   object
 8   app_val        6415 non-null   int64
 9   driver_val     6415 non-null   int64
 10  ride_val       6415 non-null   int64
 11  price_val      6415 non-null   int64
 12  wait_time_val  6415 non-null   int64
dtypes: int64(8), object(5)
memory usage: 651.7+ KB
```

Figure 11: Grouping the dataframe

Lastly, we will export the new dataframe into a csv format to later use in Tableau.

```
review_df.to_csv('output_FINAL_review.csv', index=False)
```

Figure 12: Exporting the dataframe into CSV

5 Tableau Visualisations

ethan golledge metrocar story | Tableau Public – Link to Tableau Story

My aim for presenting the results of findings is to dive deep into the stages of the funnel, aspects of user behaviour, aspects of ride behaviour and tie both together with aspects of the reviews. This section of the report will be broken up into four parts, explaining the logic behind each dashboard and an explanation of containing visuals.

5.1 Funnel Dashboard

The primary focus of this dashboard is primarily centered around a funnel chart that represents every stage of the funnel, focusing on the drop-offs for both users and rides across different stages.

In our initial query, we computed metrics such as 'percent of previous' and 'percent of top.' Given that a funnel chart inherently consists of two sides, I intended to divide each side of the funnel into 'percent of previous' and 'percent of top' segments.

The main visual in the dashboard is an area funnel chart as it allows for quicker interpretation compared to other visuals.

To offer flexibility and enable users to explore either the users' funnel or the rides' funnel, I integrated a parameter called 'toggle.' [1] This Boolean parameter allows toggling between users (when True) and rides (when False). Employing this parameter will result in configuring the dashboard.

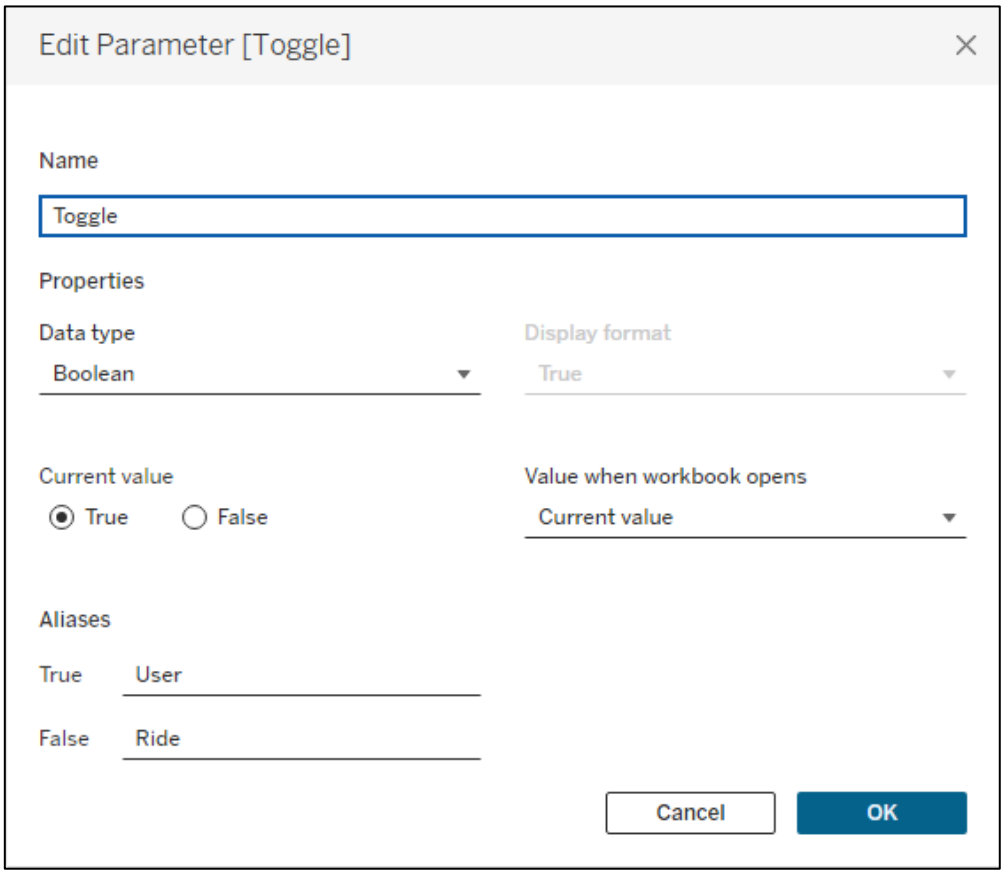


Figure 13: Creating a parameter

I arranged the funnel names based on their respective funnel steps, using the average.

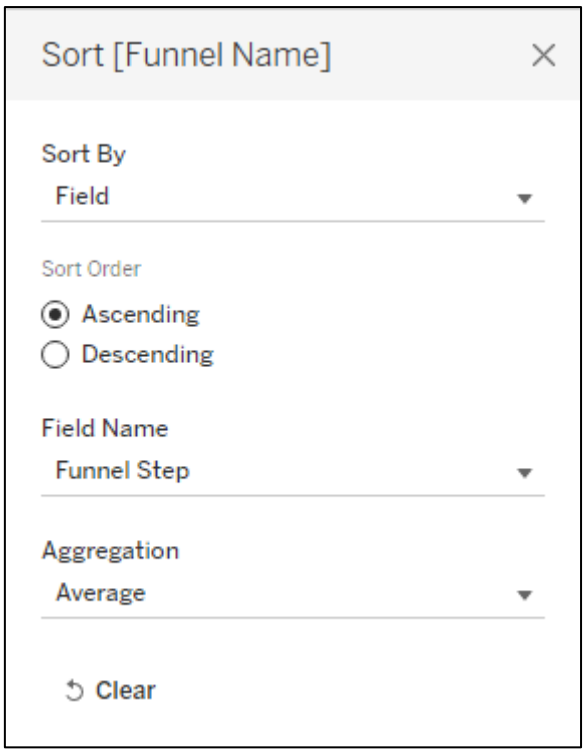


Figure 14: Sorting by funnel step

I devised a calculated field to guarantee that Tableau appropriately aggregated the values, whether it was 'ride count' or 'user count', based on the parameter.

Figure 15: Aggregating values in Tableau based on the parameter

For each stage of the funnel, we need to create a calculated field that will use the above calculation as inspiration, these are the values we will be adding to the chart.

Figure 16: Creating calculated fields for each stage of the funnel

The stages within the funnel differ between users and rides. Specifically, in the case of rides, the 'downloads' and 'signups' stages should display null values when the toggle is set to rides. This absence in the visualisation communicates to the observer that rides aren't considered for the initial two stages of the funnel.

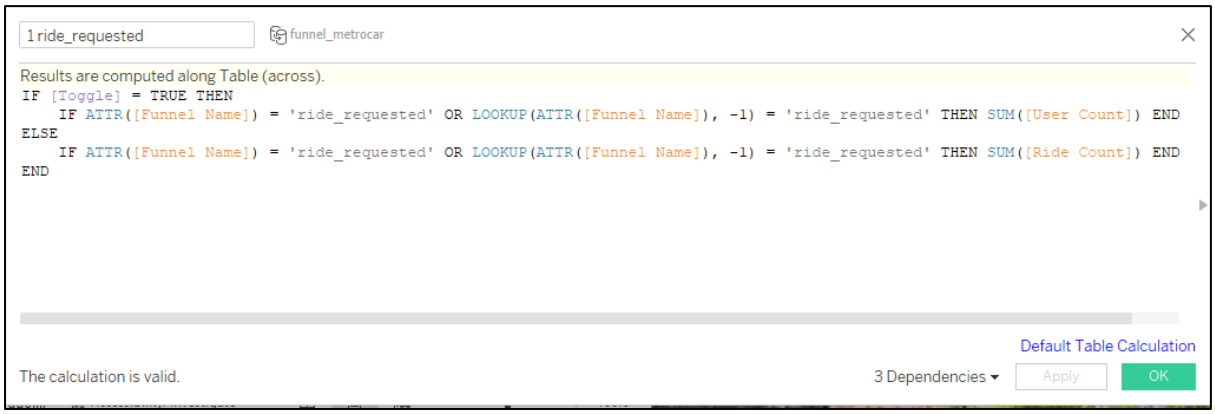


Figure 17: Creating calculated fields for each stage of the funnel [2]

This calculated field ensures that the chart area, either for user or ride, aligns accurately with each stage of the funnel, using the count for user or ride, resulting in two sides for the chart. I've created separate calculated fields for all seven stages within the funnel, adjusting the string value in accordance with the funnel name. This calculation ensures that the count is correctly assigned, enabling the area of the chart and the label to reflect the count accurately.

5.1.1 Added Features

As the visual representation of this chart is considerable in size, I aimed to segregate the values and metrics for a clearer focus on each perspective they offer. I envisioned crafting a figurative 'arm' for both sides of the funnel. This required the creation of a line chart that is wide in size, distinguished by a grey colour choice. Implementing a dual-axis, I planned to overlay a circular chart to represent each stage of the funnel, housing the respective metric values. Below I will explain the metrics for each side of the arm.

5.1.2 Percent of Previous

To achieve this, we will set up a table calculation by utilising the 'percent from' option and showcasing the percentage of the preceding value. This calculation is relatively straightforward since both the user and ride counts consistently possess a previous value for computation, irrespective of the toggle setting. Consequently, we'll generate a 'percent of previous' value for every stage within the funnel.

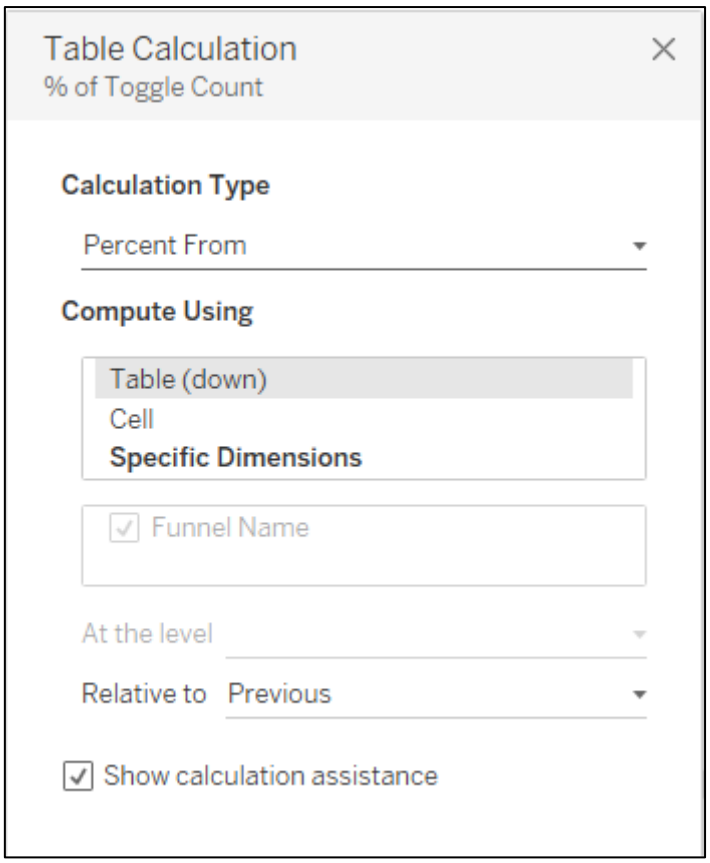
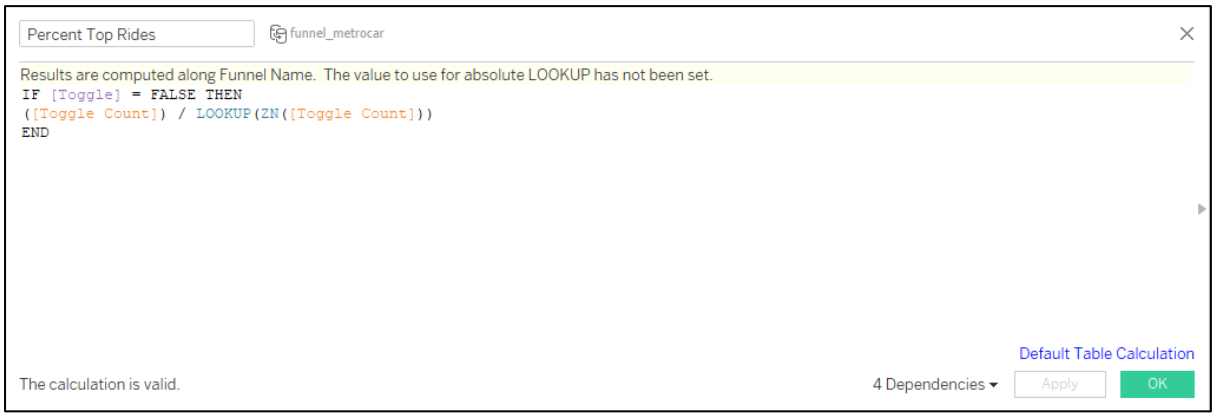


Figure 18: Percent of previous

5.1.3 Percent of Top

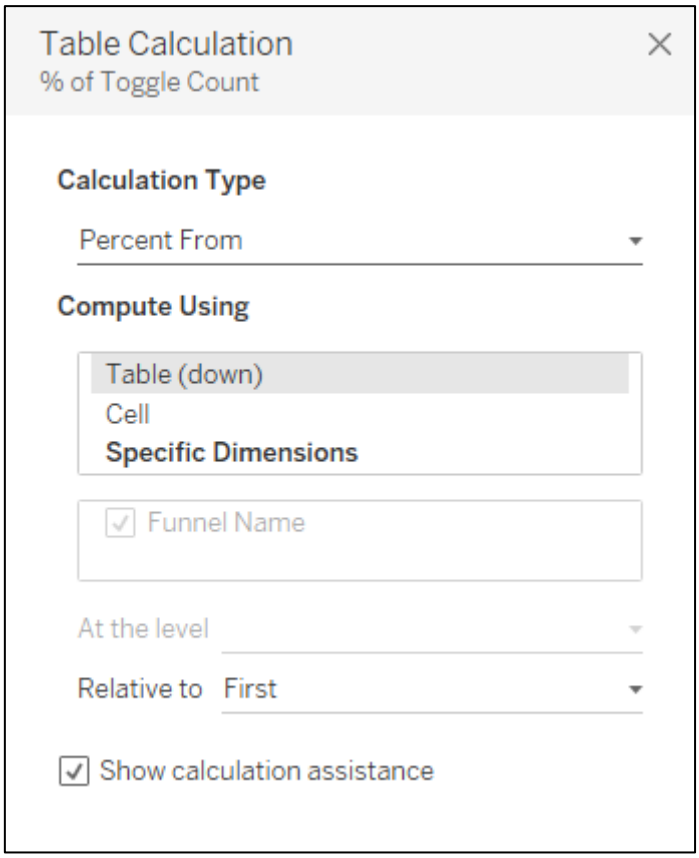
Switching gears to the opposite side of the chart, let's delve into 'percent of top.' This metric presents a challenge since users and rides do not share a common initial value. Setting both to 'relative' and 'first' leads to null values when toggling between the two. To resolve this, I'll establish a calculated field, allowing Tableau to discern that the initial value varies based on the toggle setting.



Percent of top rides

With the assistance of the 'lookup' function, we manually calculated the percentage to ensure accurate display of the 'percent of top' metric. However, a limitation arises when the toggle is set to 'rides,' resulting in a display of 100.0% instead of null.

On the other hand, for 'users,' we employ the 'toggle count' and implement a 'percent of top' calculation. Since the values are null for 'rides,' dragging 'Percent Top' onto labels allows the sheet to function as anticipated.



Percent of top users

5.1.4 Toggle Button

Rather than a conventional dropdown, I aimed to implement a button to toggle between 'users' and 'rides.' Since Tableau lacks this functionality, I designed two images using Figma—one for 'users' and another for 'rides.' By importing these images into Tableau shapes, I crafted a custom toggle button for seamless switching between the two datasets.

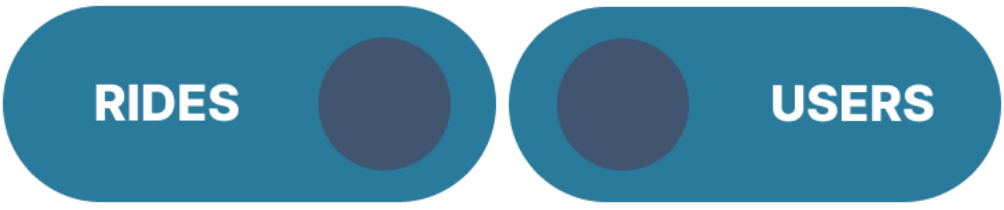


Figure 19: Creating custom toggle buttons

We introduced the parameter into the sheet. When the parameter is set to 'True,' we assigned the toggle button Boolean column to shapes and linked it with the 'users' toggle image. Conversely, when the parameter is set to 'False,' we linked it with the 'rides' toggle image. This dynamic setup allows the shape to change based on the toggle's selection.

5.1.5 Filter

The primary goal of this dashboard was to ensure interactivity. Utilising the 'month/year' columns, which were obtained from the initial query grouping data by the download date

spanning from January 2021 to December 2021, I configured the dashboard to offer dynamic insights. By placing the 'month' field onto rows and adjusting the column set to 1, clicking on individual bars within the dashboard dynamically alters the displayed values corresponding to each stage of the funnel.

Selecting the dropdown menu within the dashboard and choosing 'use as filter' yields the desired outcome.

5.1.6 User/Ride Funnel Dashboard Result

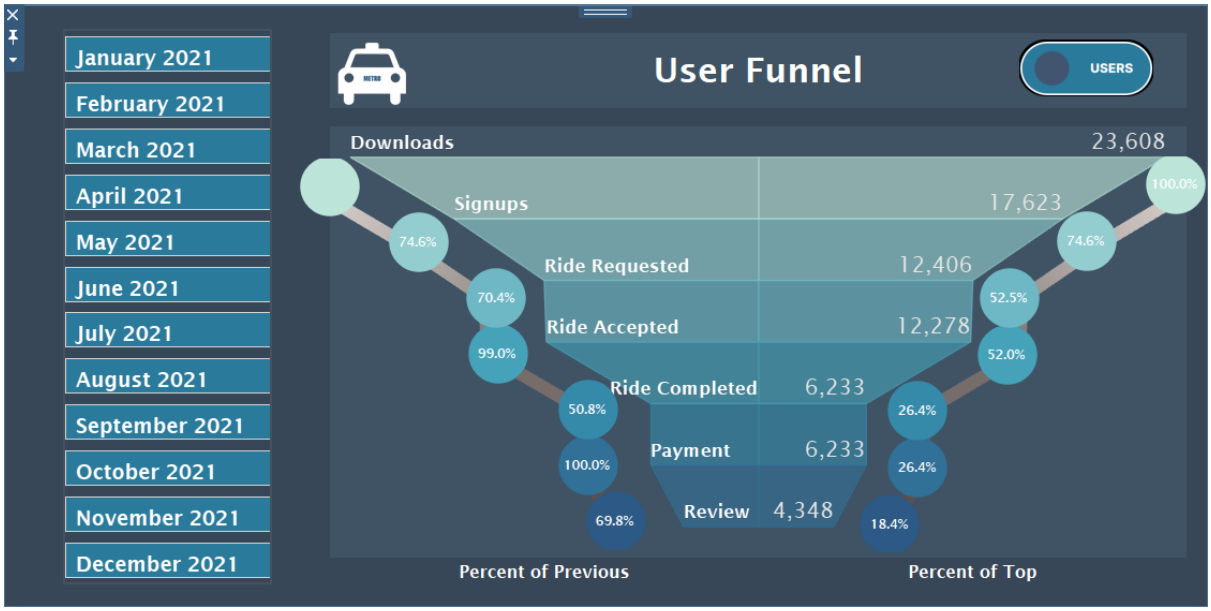


Figure 20: User and ride funnel dashboard

5.2 User Insights Dashboard

This dashboard was designed to respond to stakeholder queries about platform and age range data. It will feature visual representations displaying the funnel for category, monthly count of downloads and revenue distribution segmented by platforms and age groups. The primary focus is to create an interactive interface that allows users to seamlessly switch between exploring information based on age range or platform preferences. The sole focus of these visuals will be looking at user attributes.

Initial step: Create a new parameter to enable dynamic switching between platform and age range. This will be presented in the dashboard as a drop down, allowing the observer to look at either category.

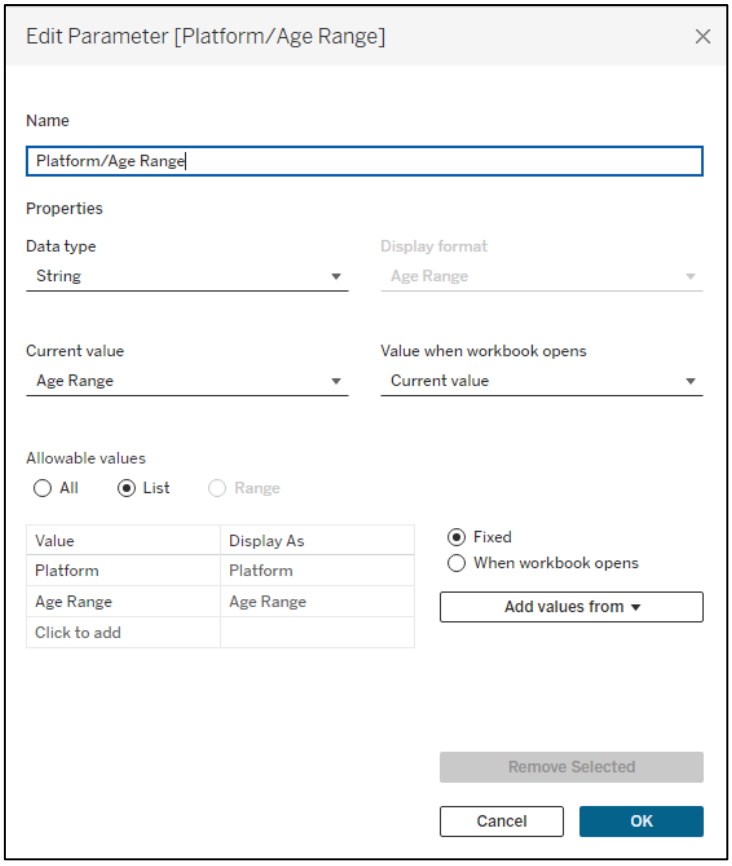


Figure 21: Creating parameter for platform/age range

Next, we need a calculated field that uses CASE WHEN, so that the right columns are being displayed based on the parameter.

Figure 22: Calculated field to aggregate the correct column

5.2.1 Funnel Visual by Platform or Age Group

This calculated field, combined with the user count summed across columns and the funnel name arranged in rows (sorted by the funnel step as mentioned earlier), results in individual funnel visuals for each category. The objective behind this visualisation was simplicity, so I solely included a table calculation for the percent of previous and kept distinct subset categories separate for easier interpretation.

5.2.2 Number of Downloads

For this visualisation, I introduced a new category called "month_year" to facilitate a more granular view of each month within the year 2021.

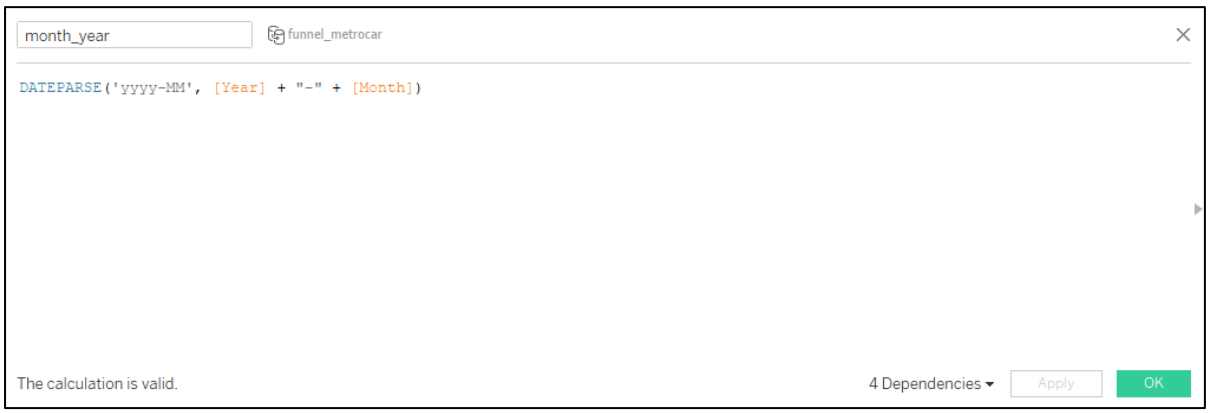


Figure 23: Creating a new column month_year

Combined with this, I crafted a bar chart and included the funnel name in the filters column. The objective was to focus solely on the total number of downloads per month. To ensure this, the filters for funnel name were set to include only the "downloads" category, thereby disregarding the total user count across each funnel step.

This visualisation won't incorporate filters for platform/age range due to the anticipated volume of bars for month/year and the subset categories in platform/age range. Instead, observers can refer to the funnel chart to assess the number of downloads per platform/age range.

5.2.3 Revenue

The sheet is based on our rides query, containing relevant purchase amount data. The columns represent the sum of the purchase amount, depicting the total revenue per group. Rows are organised by platform/age range. By adding the platform/age range parameter, the visualised category dynamically changes. The colour represents platform/age range, displaying a label indicating the subset category and the respective total revenue generated.

5.2.4 User Insights Dashboard Result

Figure 24: User insights dashboard

5.3 Ride Insights Dashboard

This dashboard was constructed with the stakeholders' queries in focus, primarily exploring the potential for surge pricing and investigating metrics possibly contributing to lower conversion rates. Flexibility was integrated into the dashboard to emphasise our performance trend over time and to encapsulate user behaviour for informed decision-making.

5.3.1 Rides by Hour

To encompass each stage of the funnel, I've designed a line chart that illustrates the counts for requests, acceptances, completions, payments, and cancellations. The aim was to identify potential trends and pinpoint where customer loss occurs, particularly pertinent when examining the possibility of surge pricing.

To generate separate lines for each stage, I crafted a calculated field.

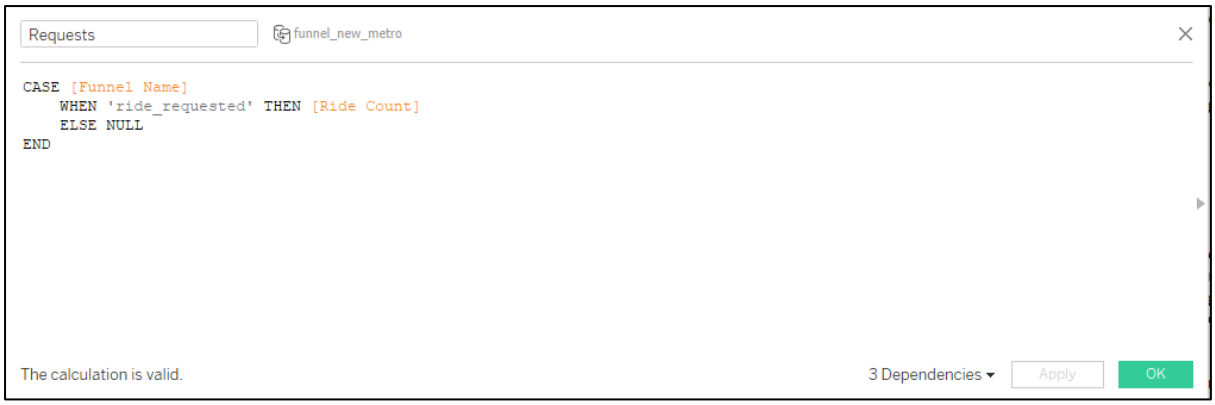


Figure 25: Calculated field for separate values in the line chart

For every other stage of the funnel, I repeated the process. However, when considering cancellations, I subtracted the number of completed rides from the total ride requests. This distinction was key as it provides a direct insight into the count of cancellations, offering a clearer view of potential considerations around price surging. Without this line, we would have to deduce the difference between ride requests and ride completions, which could be less direct.

Then, I placed the values SUM(measures) onto the sheet and associated each measure with its respective colour scheme. Adding mark labels set to display minimum and maximum values provided clarity to the viewer, allowing them to discern the peak times more effectively.

5.3.2 Rides by month, year

I aimed to explore the overall performance trend of our app since its inception. Using a similar approach to the previous visualisations, I focused on the month and year aspects. The distinction here is that I incorporated the month_year column into columns and configured the mark labels to display all values, ensuring there is no overlapping for improved readability. The calculations used are consistent with those utilised in previous analyses.

5.3.3 Metrics included as text

I considered plotting different extracted metrics against each other. For instance, when analysing metrics like wait times between requesting and cancelling, representing these times individually might provide a clearer understanding. My objective was to offer detailed insights without overwhelming viewers, focusing specifically on ride-level granularity to understand the duration individuals spend at each stage.

To sum up, my dataset included a variety of metrics, including counts, timings, and revenue sums. By extracting data for each stage of the funnel in my query, I aimed to format these metrics into a visually clear and easily understandable format in the suitable aggregation.

- Count of approved transactions
- Count of rides requested
- Count of rides accepted
- Count of rides completed
- Count of cancellations
- The average time between a ride request and acceptance.
- The average time between a ride request and pickup.
- The average time between a ride request and cancellation.
- The average time between ride acceptance and pickup.
- The average time between ride acceptance and cancellation.
- The average ride duration.
- The total sum of revenue

The dashboard contains an abundance of information. To enhance user experience, I incorporated interactivity, allowing viewers to focus on a specific month if desired. This dynamic feature alters all visuals except for the month-year representation. The aim was to offer flexibility and a minimalist approach, facilitating a comprehensive understanding of user behaviour in finer detail.

5.3.4 Rides Insights Dashboard Results

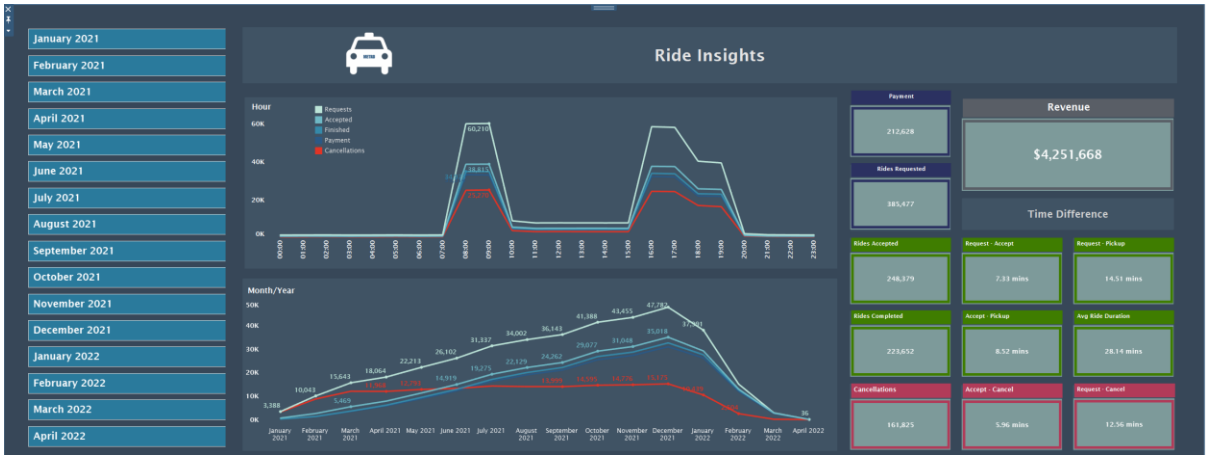


Figure 26: Ride insights dashboard

5.4 Review Insights Dashboard

The final dashboard utilised the Python-acquired data, aiming to bridge the preceding dashboard's analytics with user sentiments. My goal wasn't explicitly tied to addressing stakeholder inquiries; rather, I sought patterns between user reviews specific queries.

5.4.1 Reviews Percentages

I structured pie charts for visualisation, employing a technique involving duplicating '0' in rows and selecting the pie chart option. To divide the visual, I employed two distinct pies: the smaller one represented the average rating, while the larger pie displayed each review category, visualising positive, satisfactory, and negative via colour. I incorporated a table calculation for percent of total to delineate the overall percentages across the positive, satisfactory, and negative categories.

This procedure was repeated for the app, driver, ride, price, and wait time.

5.4.2 Percent of reviews

Following that, I aimed to represent the prevalent mentions by users to add context to the earlier pie charts. Each bar illustrates the percentage of the total count, allowing viewers to identify the category with the highest frequency among the five classifications.

To accomplish this, I developed a calculated field that divided each review value by the sum of all values, effectively creating a calculated field denoting the percentage of the total count for each review category.

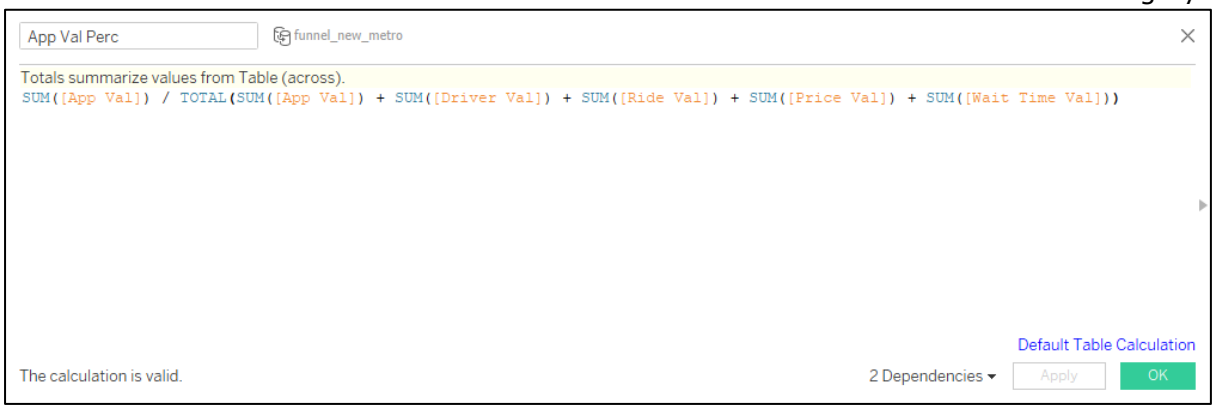


Figure 27: Creating a calculated fields for reviews

Subsequently, I included the measure values on the sheet, displaying the measure names and values in the marks label.

5.4.3 Reviews Keywords

Regarding the Reviews Keywords section, I opted for a straightforward approach by creating text boxes for each review type - positive, satisfactory, and negative. These details were extracted from the Jupyter notebook and listed the top three keywords accordingly.

5.4.4 Reviews Insights Dashboard Results

Figure 28: Review insights dashboard

6 Conclusion

Which steps of the funnel should we research and improve? Which areas have the lowest conversion rates?

By looking at the funnel chart, it is evident that the significant drop occurs between users having their ride accepted and completing the ride. We observe that 50.8% of users progress from ride acceptance to ride completion, whereas the previous stage sees a 99.0% progression rate.

If we switch the toggle button and focus on rides, 64.4% of ride requests progress to having their ride accepted. Whereas the next stage sees a 90.0% progression onto rides completed.

These are alarming figures; I am going to be performing a series of analyses primarily from the rides and reviews dashboards in the hopes that we are pointed in the right direction as to why we are seeing such significant drop offs. We will be using metrics from the rides and reviews dashboard to address low conversion rates in both users and rides, with some suggestions onto better understanding this topic.

Wait time analysis:

The time between ride request and acceptance averages at 7.33 minutes. However, users experience a longer wait time of 8.52 minutes between ride acceptance and pick-up. On average, users wait approximately 14.51 minutes from requesting a ride to being picked up.

Further investigation into user perceptions of this wait time could be beneficial.

At the moment, the time between ride request and cancellation is 12.56 minutes, with a shorter duration of 5.96 minutes between acceptance and cancellation.

- It would be valuable to gather more data to understand customer sentiments regarding wait times. To address the lower conversion rates between ride requests, acceptances, and completions, we need to ascertain whether customers are cancelling due to extended wait times. One approach could be implementing a dropdown menu with quick review options such as 'wait time too long', 'incorrect pickup time', 'wrong location', etc. This approach can provide insights into the reasons for cancellations.
  This would address why we lose 49.2% of users when requesting a ride to having it accepted.
- Research app functionality to optimise processes and reduce waiting periods between ride request and acceptance and also ride acceptance and pickup, ideally having request to pick-up below the 12.56-minute mark.
- We should also consider mining additional data to understand the topic of requests to acceptances. It would be useful to know the proportion of drivers to users at given times as only 58% of rides make it from requesting a ride to completing.
  It is crucial to understand whether this is because there are not enough drivers, whether the wait time is too long or if we need additional data on how functional our app is when assigning a driver to a ride.
- With this data, we could move forward with either updating the app to ensure quicker pickup times or having incentives for drivers to be out in higher numbers at peak times, potentially improving conversion rates in these sections of the funnel.

Further analysis on wait time using Reviews insights:

- About 8.9% of reviews refer to the wait time,
- The top word for wait time was late, mentioned about 5.9% of the time across all reviews.

Analysis techniques for the Data Analysts team:

- Consider different approaches to measure perception and time, such as visualising cancellation proportions against wait times or using median time differences, could offer additional insights. This effort warrants dedicated time and resources for comprehensive analysis and optimisation.

Unapproved transactions

It is worth noting that 95.1% of completed rides end in a successful transaction. However, approximately 5% of these completed rides don't result in an approved payment, raising concerns about potential revenue loss. We need to delve deeper to understand whether these unapproved transactions are permanently lost revenue or if there's a chance to recover these payments later. As of now, our available resources don't provide us with a clear understanding of the status of these transactions.

Reviews Analysis:

The top mentioned keyword in reviews was ride, 40.8%.

- 70.4% were positive.
- 17.3% were negative.
- 12.32% were satisfactory.
- 3.23 rating.
- The most mentioned word was comfortable, present in 14.5% of all reviews.

The second most mentioned keyword in reviews was driver, 30.5%.

- 52.5% were positive.
- 41.8% were negative.
- 5.7% were satisfactory.
- 2.70 rating.
- The most mentioned word was professional, present in 13% of all reviews.

The third most mentioned keyword was app, 12.6%.

- 51.6% were satisfactory.
- 33.4% were positive.
- 15% were negative.
- 3.58 rating.
- The most mentioned word was glitches, 4.7%.

As the top two review keywords were mainly positive. I will stick to the app reviews.

We recommend resolving the reported issues concerning the app's functionality. User feedback mention glitches, inaccuracies in GPS, and usability concerns are critical to our service's reputation and user experience. Addressing these issues is key for improving customer trust, satisfaction, and potentially enhancing conversion rates. Devoting resources to understand and rectify these app-related issues will play a vital role in ensuring a more reliable and satisfactory service for our users, potentially improving conversion rates between all stages of the funnel.

Downloads to first request

We experience a considerable drop-off of 47.5% in customer conversion rates between downloads, signups, and ride requests. While some level of drop-off is expected in a typical funnel, this significant loss warrants further investigation. Allocating time and resources to analyse the specific reasons behind this substantial drop could be beneficial in understanding any service-related issues contributing to user loss in these stages.

Historical Trend:

From January 2021 to December 2021, there was a steady increase in ride requests, acceptances, and completions, while cancellations remained relatively constant from March 2021 to December 2021. However, a noticeable decline has occurred across all stages of the funnel from December 2021 to April 2022. It is urgent that we understand this decline in full, this may include information that is limited to the scope of what we can gather.

We have also noticed that there are no downloads from December 2021 to April 2022, gathered from the user insights dashboard, specifically the downloads bar chart. We have no records of any downloads, contributing to this decline in 2022.

- Conduct user surveys or feedback sessions to understand their tolerance for wait times.
- Research external reasons

Platform analysis

Marketing budget

Currently, the dominant platform among our users is IOS, accounting for 60.52% of our user base (14290 users). IOS users show the most favourable progression through the funnel stages. Generating $2,586,629 in revenue, IOS surpasses other platforms, making it a key focus for our budget allocation.

We did not see any of the platforms performing significantly better in each stage of the funnel, only by marginal differences in the percent of previous.

Age Group Analysis

Target audience

Currently with the data that we have, the 35-45 age group have the highest number of signups out of the users who specify their age. This age group has 5181 signups and progresses through the stages of the funnel well. When we look at revenue, this age group surpasses every other group, including those who do not specify their age, coming in at $1,270,533. This is our target audience and we should be catering to targeting people in this group as they progress through the stages of the funnel very effectively.

It's important to highlight that the second-highest revenue-generating age group is the category of users who do not specify their age. Despite not providing age information, this group contributes significantly to our revenue, closely following the 35-45 age group. This raises a potential issue or choice where users may opt not to share their age, affecting our ability to effectively analyse and understand a considerable portion of our revenue and user base.

We did not see any of the age groups performing significantly better in each stage of the funnel, for the most part there were marginal differences apart from reviews where percent of previous varied up to 4%.

Surge pricing analysis

- Between 8 AM and 9 AM, there's a peak with 60,210 ride requests at 9am, indicating a high demand for rides during this hour.
- During these hours we have a peak of 25,270 cancellations.
- Another notable peak occurs between 4 PM and 5 PM, with 58,527 ride requests.
- During these hours we have a peak of 24,526 cancellations.
- A third peak appears between 5 PM and 7 PM, with 40,372 ride requests recorded at 6 PM.
- During these hours we have a peak of 17,109 cancellations.

These peak hours represent periods of high demand for rides. Implementing a price-surging strategy during these times would be advisable. Despite a slight drop-off in ride requests later in the evening, the numbers remain substantial, suggesting an opportunity to leverage increased pricing during these hours.

Suggestions:

- It is advisable to prioritise resolving the issues related to low conversion rates between ride requests, acceptances, and completions before considering the implementation of price surging. I've highlighted the number of cancellations to underscore the impact on conversion rates. By improving these conversion rates within the funnel stages, we can subsequently consider introducing price surging strategies.

Summary of recommendations:

- Research external factors and customer perception to understand the decline in rides.
- Implement a quick review system for users to express reasons for ride cancellations, focusing on wait times and pickup concerns, aiding in understanding the 49.9% loss from ride requests to acceptances.
- Optimise app functionality to minimise wait times between ride request, acceptance, and pickup, aiming for a request-to-pickup duration below 12.56 minutes.
- Analyse driver-user ratios to comprehend why only 58% of rides transition from request to completion, potentially involving driver incentives or app updates for quicker pickups.
- Resolve reported app-related issues like glitches, GPS inaccuracies, and usability concerns to enhance user trust and satisfaction, potentially improving conversion rates.
- Conduct user surveys or feedback sessions to gauge tolerance levels for wait times and gather insights for service improvements.

7 Appendix

7.1 Users Funnel Query

[1]

```
WITH
  user_funnel AS (
    SELECT
      1 AS step,
      'app_downloads' AS funnel_name,
      COUNT(DISTINCT app_download_key)
    FROM
      app_downloads
    UNION
    SELECT
      2 AS step,
      'signups' AS funnel_name,
      COUNT(DISTINCT user_id)

  FROM
      signups
    UNION
    SELECT
      3 AS step,
      'ride_requested' AS funnel_name,
      COUNT(DISTINCT user_id)
    FROM
      ride_requests
```

```
UNION
SELECT
  4 AS step,
  'ride_accepted' AS funnel_name,
  COUNT(DISTINCT user_id)
FROM
  ride_requests
WHERE
  accept_ts IS NOT NULL
UNION
SELECT
  5 AS step,
  'ride_completed',
  COUNT(DISTINCT user_id)
FROM ride_requests
WHERE dropoff_ts IS NOT NULL
UNION
SELECT
  6 AS step,
  'approved_transaction' AS funnel_name,
  COUNT(DISTINCT r.user_id)
FROM
  transactions t
LEFT JOIN ride_requests r
ON r.ride_id = t.ride_id
WHERE
  charge_status = 'Approved'
```

```
    UNION
    SELECT
      7 AS step,
      'reviewed' AS funnel_name,
      COUNT(DISTINCT re.user_id)
    FROM reviews re



    )
SELECT
  funnel_name,
  COUNT AS user_count,
  COUNT::FLOAT / LAG(COUNT) OVER (
        ORDER BY step
    ) AS perc_previous,
  COUNT::FLOAT / LAG(COUNT, 6) OVER (
        ORDER BY step
    ) AS perc_top
FROM
  user_funnel
ORDER BY
  step;
```

7.2 Rides Funnel Query

[2]

```
WITH
  ride_funnel AS (
    SELECT
```

```
  1 AS step,
  'app_downloads' AS funnel_name,
  CAST(NULL AS BIGINT) AS count
FROM
  app_downloads
UNION
SELECT
  2 AS step,
  'signups' AS funnel_name,
  CAST(NULL AS BIGINT) AS count
FROM
  signups
UNION
SELECT
  3 AS step,
  'ride_requested' AS funnel_name,
  COUNT(DISTINCT ride_id) AS count
FROM
  ride_requests
UNION
SELECT
  4 AS step,
  'ride_accepted' AS funnel_name,
  COUNT(DISTINCT ride_id) AS count
FROM
  ride_requests
WHERE
```

```
    accept_ts IS NOT NULL
  UNION
  SELECT
    5 AS step,
    'ride_completed' AS funnel_name,
    COUNT(DISTINCT ride_id) AS count
  FROM ride_requests
  WHERE
    dropoff_ts IS NOT NULL
  UNION
  SELECT
    6 AS step,
    'approved_transaction' AS funnel_name,
  COUNT(DISTINCT ride_id) AS count
  FROM
    transactions
  WHERE
    charge_status = 'Approved'
  UNION
  SELECT
    7 AS step,
    'reviewed' AS funnel_name,
    COUNT(DISTINCT ride_id) AS count
  FROM reviews

  )
SELECT
```

```
  funnel_name,
  COUNT AS ride_count,
  COUNT::FLOAT / LAG(COUNT) OVER (
      ORDER BY step
    ) AS perc_previous,
  COUNT::FLOAT / LAG(COUNT, 4) OVER (
      ORDER BY step
    ) AS perc_top

FROM
  ride_funnel
ORDER BY
  step;
```

7.3 User Query for Tableau

[3]

```
WITH
  user_downloads AS (
    SELECT
      1 AS funnel_step,
      'downloads' AS funnel_name,
      a.platform,
      s.age_range,
      EXTRACT(
        YEAR
        FROM
          download_ts
      ) AS year,
```

```
  EXTRACT(
    MONTH
    FROM
      download_ts
  ) AS month,
  EXTRACT(
    HOUR
    FROM
      download_ts
  ) AS hour,
  COUNT(DISTINCT app_download_key) AS user_count,
  CAST(NULL AS numeric) AS ride_count
FROM
  app_downloads a
  LEFT JOIN signups s ON a.app_download_key = s.session_id
GROUP BY
  platform,
  age_range,
  year,
  month,
  hour
),
user_signups AS (
SELECT
  2 AS funnel_step,
  'signups' AS funnel_name,
  platform,
```

```
  s.age_range,
  EXTRACT(
    YEAR
    FROM
      download_ts
  ) AS year,
  EXTRACT(
    MONTH
    FROM
      download_ts
  ) AS month,
  EXTRACT(
    HOUR
    FROM
      download_ts
  ) AS hour,
  COUNT(DISTINCT user_id) AS user_count,
  CAST(NULL AS numeric) AS ride_count
FROM
  signups s
  LEFT JOIN app_downloads a ON s.session_id = a.app_download_key
GROUP BY
  a.platform,
  s.age_range,
  year,
  month,
  hour
```

```
),
user_ride_requests AS (
SELECT
  3 AS funnel_step,
  'ride_requested' AS funnel_name,
  a.platform,
  s.age_range,
  EXTRACT(
    YEAR
    FROM
      download_ts
  ) AS year,
  EXTRACT(
    MONTH
    FROM
      download_ts
  ) AS month,
  EXTRACT(
    HOUR
    FROM
      download_ts
  ) AS hour,
  COUNT(DISTINCT r.user_id) AS user_count,
  COUNT(r.user_id) AS ride_count
FROM
  ride_requests r
  LEFT JOIN signups s ON r.user_id = s.user_id
```

```
  LEFT JOIN app_downloads a ON s.session_id = a.app_download_key
GROUP BY
  a.platform,
  s.age_range,
  year,
  month,
  hour
),
user_ride_accepted AS (
SELECT
  4 AS funnel_step,
  'ride_accepted' AS funnel_name,
  a.platform,
  s.age_range,
  EXTRACT(
    YEAR
    FROM
      download_ts
  ) AS year,
  EXTRACT(
    MONTH
    FROM
      download_ts
  ) AS month,
  EXTRACT(
    HOUR
    FROM
```

```
    download_ts
  ) AS hour,
  COUNT(DISTINCT r.user_id) AS user_count,
  COUNT(r.user_id) AS ride_count
FROM
  ride_requests r
  LEFT JOIN signups s ON r.user_id = s.user_id
  LEFT JOIN app_downloads a ON s.session_id = a.app_download_key
WHERE
  accept_ts IS NOT NULL
GROUP BY
  a.platform,
  s.age_range,
  year,
  month,
  hour
  ),
user_ride_completed AS (
SELECT
  5 AS funnel_step,
  'ride_completed' AS funnel_name,
  a.platform,
  s.age_range,
  EXTRACT(
    YEAR
    FROM
      download_ts
```

```
  ) AS year,
  EXTRACT(
    MONTH
    FROM
      download_ts
  ) AS month,
  EXTRACT(
    HOUR
    FROM
      download_ts
  ) AS hour,
  COUNT(DISTINCT r.user_id) AS user_count,
  COUNT(r.user_id) AS ride_count
FROM
  ride_requests r
  LEFT JOIN transactions t ON r.ride_id = t.ride_id
  LEFT JOIN signups s ON r.user_id = s.user_id
  LEFT JOIN app_downloads a ON s.session_id = a.app_download_key
WHERE
  cancel_ts IS NULL
GROUP BY
  a.platform,
  s.age_range,
  year,
  month,
  hour
),
```

```
user_payment AS (
SELECT
  6 AS funnel_step,
  'payment' AS funnel_name,
  a.platform,
  s.age_range,
  EXTRACT(
    YEAR
    FROM
      download_ts
  ) AS year,
  EXTRACT(
    MONTH
    FROM
      download_ts
  ) AS month,
  EXTRACT(
    HOUR
    FROM
      download_ts
  ) AS hour,
  COUNT(DISTINCT r.user_id) AS user_count,
  COUNT(r.user_id) AS ride_count
FROM
  transactions t
  LEFT JOIN ride_requests r ON t.ride_id = r.ride_id
  LEFT JOIN signups s ON r.user_id = s.user_id
```

```
  LEFT JOIN app_downloads a ON s.session_id = a.app_download_key
WHERE
  t.charge_status = 'Approved'
GROUP BY
  a.platform,
  s.age_range,
  year,
  month,
  hour
),
user_review AS (
SELECT
  7 AS funnel_step,
  'review' AS funnel_name,
  a.platform,
  s.age_range,
  EXTRACT(
    YEAR
    FROM
      download_ts
  ) AS year,
  EXTRACT(
    MONTH
    FROM
      download_ts
  ) AS month,
  EXTRACT(
```

```
      HOUR
      FROM
        download_ts
    ) AS hour,
    COUNT(DISTINCT re.user_id) AS user_count,
    COUNT(re.user_id) AS ride_count
  FROM
    reviews re
    LEFT JOIN ride_requests r ON re.ride_id = r.ride_id
    LEFT JOIN signups s ON re.user_id = s.user_id
    LEFT JOIN app_downloads a ON s.session_id = a.app_download_key
  GROUP BY
    a.platform,
    s.age_range,
    year,
    month,
    hour
)
SELECT
  *
FROM
  user_downloads
UNION
SELECT
  *
FROM
  user_signups
```

```
UNION
SELECT
  *
FROM
  user_ride_requests
UNION
SELECT
  *
FROM
  user_ride_accepted
UNION
SELECT
  *
FROM
  user_ride_completed
UNION
SELECT
  *
FROM
  user_payment
UNION
SELECT
  *
FROM
  user_review
ORDER BY
  funnel_step;
```

7.4 Rides Query for Tableau

[4]

```
WITH
  rides_requested AS (
    SELECT
      3 AS funnel_step,
      'ride_requested' AS funnel_name,
      COUNT(DISTINCT r.ride_id) AS ride_count,
      a.platform,
      COALESCE(s.age_range, 'Not specified') AS age_range,
      EXTRACT(
        YEAR
        FROM
          r.request_ts
      ) AS year,
      EXTRACT(
        MONTH
        FROM
          r.request_ts
      ) AS month,
      EXTRACT(
        HOUR
        FROM
          r.request_ts
      ) AS hour,
      CAST(NULL AS numeric) AS avg_time_requested_pickup,
      CAST(NULL AS numeric) AS avg_time_requested_accepted,
```

```
  CAST(NULL AS numeric) AS avg_time_accepted_pickup,
  ROUND(
    AVG(
      EXTRACT(
        MINUTE
        FROM
          (r.cancel_ts - r.request_ts)
      )
    ),
    2
  ) AS avg_time_ride_requested_cancelled,
  CAST(NULL AS numeric) AS avg_time_accepted_cancelled,
  (SUM(
    CASE
      WHEN r.cancel_ts IS NOT NULL THEN 1
      ELSE 0
    END
  ))::float / NULLIF(COUNT(r.request_ts), 0) AS proportion_of_cancellations,
  CAST(NULL AS numeric) AS avg_ride_duration,
  CAST(NULL AS numeric) AS sum_purchase_amount,
  CAST(NULL AS numeric) AS avg_rating
FROM
  ride_requests r
  LEFT JOIN signups s ON r.user_id = s.user_id
  LEFT JOIN app_downloads a ON s.session_id = a.app_download_key
GROUP BY
  a.platform,
```

```
    s.age_range,
    year,
    month,
    hour
),
rides_accepted AS (
  SELECT
    4 AS funnel_step,
    'ride_accepted' AS funnel_name,
    COUNT(DISTINCT r.ride_id) AS ride_count,
    a.platform,
    COALESCE(s.age_range, 'Not specified') AS age_range,
    EXTRACT(
      YEAR
      FROM
        r.request_ts
    ) AS year,
    EXTRACT(
      MONTH
      FROM
        r.request_ts
    ) AS month,
    EXTRACT(
      HOUR
      FROM
        r.request_ts
    ) AS hour,
```

```
CAST(NULL AS numeric) AS avg_time_requested_pickup,
AVG(
  EXTRACT(
    MINUTE
    FROM
      (r.accept_ts - r.request_ts)
  )
) AS avg_time_requested_accepted,
AVG(
  EXTRACT(
    MINUTE
    FROM
      (r.pickup_ts - r.accept_ts)
  )
) AS avg_time_accepted_pickup,
CAST(NULL AS numeric) AS avg_time_ride_requested_cancelled,
AVG(
  EXTRACT(
    MINUTE
    FROM
      (r.cancel_ts - r.accept_ts)
  )
) AS avg_time_accepted_cancelled,
CAST(NULL AS float) AS proportion_of_cancellations,
CAST(NULL AS numeric) AS avg_ride_duration,
CAST(NULL AS numeric) AS sum_purchase_amount,
CAST(NULL AS numeric) AS avg_rating
```

```
  FROM
    ride_requests r
    LEFT JOIN signups s ON r.user_id = s.user_id
    LEFT JOIN app_downloads a ON s.session_id = a.app_download_key
  WHERE
    r.accept_ts IS NOT NULL
  GROUP BY
    a.platform,
    s.age_range,
    year,
    month,
    hour
),
rides_completed AS (
  SELECT
    5 AS funnel_step,
    'ride_completed' AS funnel_name,
    COUNT(DISTINCT r.ride_id) AS ride_count,
    a.platform,
    COALESCE(s.age_range, 'Not specified') AS age_range,
    EXTRACT(
      YEAR
      FROM
        r.request_ts
    ) AS year,
    EXTRACT(
      MONTH
```

```
  FROM
    r.request_ts
) AS month,
EXTRACT(
  HOUR
  FROM
    r.request_ts
) AS hour,
AVG(
  EXTRACT(
    MINUTE
    FROM
      (r.pickup_ts - r.request_ts)
  )
) AS avg_time_requested_pickup,
CAST(NULL AS numeric) AS avg_time_requested_accepted,
CAST(NULL AS numeric) AS avg_time_accepted_pickup,
CAST(NULL AS numeric) AS avg_time_ride_requested_cancelled,
CAST(NULL AS numeric) AS avg_time_accepted_cancelled,
CAST(NULL AS float) AS proportion_of_cancellations,
AVG(
  EXTRACT(
    MINUTE
    FROM
      (r.dropoff_ts - r.pickup_ts)
  )
) AS avg_ride_duration,
```

```
    CAST(NULL AS numeric) AS sum_purchase_amount,
    CAST(NULL AS numeric) AS avg_rating
  FROM
    ride_requests r
    LEFT JOIN signups s ON r.user_id = s.user_id
    LEFT JOIN app_downloads a ON s.session_id = a.app_download_key
  WHERE
    r.accept_ts IS NOT NULL
    AND r.dropoff_ts IS NOT NULL
  GROUP BY
    a.platform,
    s.age_range,
    year,
    month,
    hour
),
rides_paid AS (
  SELECT
    6 AS funnel_step,
    'payment' AS funnel_name,
    COUNT(DISTINCT r.ride_id) AS ride_count,
    a.platform,
    COALESCE(s.age_range, 'Not specified') AS age_range,
    EXTRACT(
      YEAR
      FROM
        r.request_ts
```

```
  ) AS year,
  EXTRACT(
    MONTH
    FROM
      r.request_ts
  ) AS month,
  EXTRACT(
    HOUR
    FROM
      r.request_ts
  ) AS hour,
  CAST(NULL AS numeric) AS avg_time_requested_pickup,
  CAST(NULL AS numeric) AS avg_time_requested_accepted,
  CAST(NULL AS numeric) AS avg_time_requested_accepted_pickup,
  CAST(NULL AS numeric) AS avg_time_ride_requested_cancelled,
  CAST(NULL AS numeric) AS avg_time_accepted_cancelled,
  CAST(NULL AS float) AS proportion_of_cancellations,
  CAST(NULL AS numeric) AS avg_ride_duration,
  SUM(t.purchase_amount_usd) AS sum_purchase_amount,
  CAST(NULL AS numeric) AS avg_rating
FROM
  ride_requests r
  LEFT JOIN signups s ON r.user_id = s.user_id
  LEFT JOIN app_downloads a ON s.session_id = a.app_download_key
  LEFT JOIN transactions t ON r.ride_id = t.ride_id
WHERE
  r.accept_ts IS NOT NULL
```

```
    AND r.dropoff_ts IS NOT NULL
    AND t.charge_status = 'Approved'
  GROUP BY
    a.platform,
    s.age_range,
    year,
    month,
    hour
),
rides_reviewed AS (
  SELECT
    7 AS funnel_step,
    'ride reviewed' AS funnel_name,
    COUNT(DISTINCT r.ride_id) AS ride_count,
    a.platform,
    COALESCE(s.age_range, 'Not specified') AS age_range,
    EXTRACT(
      YEAR
      FROM
        r.request_ts
    ) AS year,
    EXTRACT(
      MONTH
      FROM
        r.request_ts
    ) AS month,
    EXTRACT(
```

```
    HOUR
    FROM
      r.request_ts
  ) AS hour,
  CAST(NULL AS numeric) AS avg_time_requested_pickup,
  CAST(NULL AS numeric) AS avg_time_requested_accepted,
  CAST(NULL AS numeric) AS avg_time_accepted_pickup,
  CAST(NULL AS numeric) AS avg_time_ride_requested_cancelled,
  CAST(NULL AS numeric) AS avg_time_accepted_cancelled,
  CAST(NULL AS float) AS proportion_of_cancellations,
  CAST(NULL AS numeric) AS avg_ride_duration,
  CAST(NULL AS numeric) AS sum_purchase_amount,
  AVG(re.rating) AS avg_rating
FROM
  ride_requests r
  LEFT JOIN signups s ON r.user_id = s.user_id
  LEFT JOIN app_downloads a ON s.session_id = a.app_download_key
  LEFT JOIN transactions t ON r.ride_id = t.ride_id
  LEFT JOIN reviews re ON t.ride_id = re.ride_id
WHERE
  r.accept_ts IS NOT NULL
  AND r.dropoff_ts IS NOT NULL
  AND t.charge_status = 'Approved'
GROUP BY
  a.platform,
  s.age_range,
  year,
```

```
    month,
    hour
)
SELECT
  *
FROM
  rides_requested
UNION
SELECT
  *
FROM
  rides_accepted
UNION
SELECT
  *
FROM
  rides_completed
UNION
SELECT
  *
FROM
  rides_paid
UNION
SELECT
  *
FROM
  rides_reviewed
```

7.5 Reviews Query

```
WITH user_reviews AS (


  SELECT
    re.user_id,
    DATE_PART('year', request_ts) AS year,
    DATE_PART('month', request_ts) AS month,
    DATE_PART('hour', request_ts) AS hour,
    re.rating,
    re.review
  FROM
    reviews re
    LEFT JOIN ride_requests r ON re.ride_id = r.ride_id
)
SELECT
  user_id,
  year,
  month,
  hour,
  rating,
  review
FROM user_reviews
ORDER BY
  random()
LIMIT 49999;
```