# Waze Project

**Course 2 - Get Started with Python**

Welcome to the Waze Project!

Your Waze data analytics team is still in the early stages of their user churn project. Previously, you were asked to complete a project proposal by your supervisor, May Santner. You have received notice that your project proposal has been approved and that your team has been given access to Waze's user data. To get clear insights, the user data must be inspected and prepared for the upcoming process of exploratory data analysis (EDA).

A Python notebook has been prepared to guide you through this project. Answer the questions and create an executive summary for the Waze data team.

# Course 2 End-of-course project: Inspect and analyze data

In this activity, you will examine data provided and prepare it for analysis. This activity will help ensure the information is,

1. Ready to answer questions and yield insights

2. Ready for visualizations

3. Ready for future hypothesis testing and statistical methods

**The purpose** of this project is to investigate and understand the data provided.

**The goal** is to use a dataframe contructed within Python, perform a cursory inspection of the provided dataset, and inform team members of your findings.

*This activity has three parts:*

**Part 1:** Understand the situation

- How can you best prepare to understand and organize the provided information?

**Part 2:** Understand the data

- Create a pandas dataframe for data learning, future exploratory data analysis (EDA), and statistical activities
- Compile summary information about the data to inform next steps

**Part 3:** Understand the variables

- Use insights from your examination of the summary data to guide deeper investigation into variables

Follow the instructions and answer the following questions to complete the activity. Then, you will complete an Executive Summary using the questions listed on the PACE Strategy Document.

Be sure to complete this activity before moving on. The next course item will provide you with a completed exemplar to compare to your own work.

# Identify data types and compile summary information

## Task 1. Understand the situation

- How can you best prepare to understand and organize the provided driver data?

*Begin by exploring your dataset and consider reviewing the Data Dictionary.*

## Task 2a. Imports and data loading

Start by importing the packages that you will need to load and explore the dataset. Make sure to use the following import statements:

- `import pandas as pd`
- `import numpy as np`

In [ ]:
```python
# Import packages for data manipulation
import pandas as pd
import numpy as np
```

Then, load the dataset into a dataframe. Creating a dataframe will help you conduct data manipulation, exploratory data analysis (EDA), and statistical activities.

**Note:** As shown in this cell, the dataset has been automatically loaded in for you. You do not need to download the .csv file, or provide more code, in order to access the dataset and proceed with this lab. Please continue with this activity by completing the following instructions.

```
In [2]: # Load dataset into dataframe
df = pd.read_csv('waze_dataset.csv')
```

## Task 2b. Summary information

View and inspect summary information about the dataframe by **coding the following:**

1. df.head(10)
2. df.info()

*Consider the following questions:*

1. When reviewing the `df.head()` output, are there any variables that have missing values?
2. When reviewing the `df.info()` output, what are the data types? How many rows and columns do you have?
3. Does the dataset have any missing values?

```
In [3]: ### YOUR CODE HERE ###
df.head(10)
```

Out[3]:

| | ID | label | sessions | drives | total_sessions | n_days_after_onboarding | total_navig |
|---|---|---|---|---|---|---|---|
| **0** | 0 | retained | 283 | 226 | 296.748273 | 2276 | |
| **1** | 1 | retained | 133 | 107 | 326.896596 | 1225 | |
| **2** | 2 | retained | 114 | 95 | 135.522926 | 2651 | |
| **3** | 3 | retained | 49 | 40 | 67.589221 | 15 | |
| **4** | 4 | retained | 84 | 68 | 168.247020 | 1562 | |
| **5** | 5 | retained | 113 | 103 | 279.544437 | 2637 | |
| **6** | 6 | retained | 3 | 2 | 236.725314 | 360 | |
| **7** | 7 | retained | 39 | 35 | 176.072845 | 2999 | |
| **8** | 8 | retained | 57 | 46 | 183.532018 | 424 | |
| **9** | 9 | churned | 84 | 68 | 244.802115 | 2997 | |

```
In [5]: ### YOUR CODE HERE ###
df.info()
df.describe()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 14999 entries, 0 to 14998
Data columns (total 13 columns):
 #   Column                   Non-Null Count  Dtype
---  ------                   --------------  -----
 0   ID                       14999 non-null  int64
 1   label                    14299 non-null  object
 2   sessions                 14999 non-null  int64
 3   drives                   14999 non-null  int64
 4   total_sessions           14999 non-null  float64
 5   n_days_after_onboarding  14999 non-null  int64
 6   total_navigations_fav1   14999 non-null  int64
 7   total_navigations_fav2   14999 non-null  int64
 8   driven_km_drives         14999 non-null  float64
 9   duration_minutes_drives  14999 non-null  float64
 10  activity_days            14999 non-null  int64
 11  driving_days             14999 non-null  int64
 12  device                   14999 non-null  object
dtypes: float64(3), int64(8), object(2)
memory usage: 1.5+ MB
```

Out[5]:

| | ID | sessions | drives | total_sessions | n_days_after_onbo |
|---|---|---|---|---|---|
| **count** | 14999.000000 | 14999.000000 | 14999.000000 | 14999.000000 | 14999.0 |
| **mean** | 7499.000000 | 80.633776 | 67.281152 | 189.964447 | 1749.8 |
| **std** | 4329.982679 | 80.699065 | 65.913872 | 136.405128 | 1008.5 |
| **min** | 0.000000 | 0.000000 | 0.000000 | 0.220211 | 4.0 |
| **25%** | 3749.500000 | 23.000000 | 20.000000 | 90.661156 | 878.0 |
| **50%** | 7499.000000 | 56.000000 | 48.000000 | 159.568115 | 1741.0 |
| **75%** | 11248.500000 | 112.000000 | 93.000000 | 254.192341 | 2623.5 |
| **max** | 14998.000000 | 743.000000 | 596.000000 | 1216.154633 | 3500.0 |

## Task 2c. Null values and summary statistics

Compare the summary statistics of the 700 rows that are missing labels with summary statistics of the rows that are not missing any values.

**Question:** Is there a discernible difference between the two populations?

In [7]:

```
# Isolate rows with null values
null_rows = df[df.isnull().any(axis=1)]
print(null_rows)

# Display summary stats of rows with null values
### YOUR CODE HERE ###
null_rows.describe()
```

```
          ID label  sessions  drives  total_sessions  n_days_after_onboardin
g  \
77        77   NaN        63      50      133.104155                      78
3
80        80   NaN       116      93      436.060183                     158
4
98        98   NaN        78      64      583.492789                     341
4
111      111   NaN       106     102      113.379056                     222
8
142      142   NaN        32      26      222.129310                      20
8
...      ...   ...       ...     ...             ...
...
14941  14941   NaN       191     160      485.328204                     128
7
14943  14943   NaN        48      38       96.797017                      55
5
14945  14945   NaN        34      29      134.416604                     164
3
14972  14972   NaN       220     181      256.212166                     171
8
14993  14993   NaN        67      57       97.570074                     113
1

       total_navigations_fav1  total_navigations_fav2  driven_km_drives  \
77                        201                       0       2649.015822
80                        283                      62       4183.409514
98                          0                       0       1811.140893
111                        14                       0       2817.481840
142                        55                      10       2459.816477
...                       ...                     ...               ...
14941                      25                       0       6468.181924
14943                       0                       6       8266.129497
14945                     268                       2       4554.007843
14972                     360                      23       5586.913459
14993                     207                     102       2267.052913

       duration_minutes_drives  activity_days  driving_days   device
77                 1517.209970             19            13   iPhone
80                 3121.889952             18            15   iPhone
98                  642.189122             12            11  Android
111                2011.724274             17            13  Android
142                 874.427617             11             7   iPhone
...                        ...            ...           ...      ...
14941              3466.104564             14            14   iPhone
14943              5902.351711             19            19   iPhone
14945              1579.211201             18            17  Android
14972              4104.440202             19            18   iPhone
14993               318.120634             27            26   iPhone

[700 rows x 13 columns]
```

Out[7]:

| | ID | sessions | drives | total_sessions | n_days_after_onboardi |
|---|---|---|---|---|---|
| **count** | 700.000000 | 700.000000 | 700.000000 | 700.000000 | 700.00000 |
| **mean** | 7405.584286 | 80.837143 | 67.798571 | 198.483348 | 1709.2957 |
| **std** | 4306.900234 | 79.987440 | 65.271926 | 140.561715 | 1005.3065 |
| **min** | 77.000000 | 0.000000 | 0.000000 | 5.582648 | 16.00000 |
| **25%** | 3744.500000 | 23.000000 | 20.000000 | 94.056340 | 869.00000 |
| **50%** | 7443.000000 | 56.000000 | 47.500000 | 177.255925 | 1650.50000 |
| **75%** | 11007.000000 | 112.250000 | 94.000000 | 266.058022 | 2508.75000 |
| **max** | 14993.000000 | 556.000000 | 445.000000 | 1076.879741 | 3498.00000 |

In [9]:

```python
# Isolate rows without null values
non_null_rows = df.dropna()

# Display summary stats of rows without null values
non_null_rows.describe()
```

Out[9]:

| | ID | sessions | drives | total_sessions | n_days_after_onbo |
|---|---|---|---|---|---|
| **count** | 14299.000000 | 14299.000000 | 14299.000000 | 14299.000000 | 14299.0 |
| **mean** | 7503.573117 | 80.623820 | 67.255822 | 189.547409 | 1751.8 |
| **std** | 4331.207621 | 80.736502 | 65.947295 | 136.189764 | 1008.6 |
| **min** | 0.000000 | 0.000000 | 0.000000 | 0.220211 | 4.0 |
| **25%** | 3749.500000 | 23.000000 | 20.000000 | 90.457733 | 878.5 |
| **50%** | 7504.000000 | 56.000000 | 48.000000 | 158.718571 | 1749.0 |
| **75%** | 11257.500000 | 111.000000 | 93.000000 | 253.540450 | 2627.5 |
| **max** | 14998.000000 | 743.000000 | 596.000000 | 1216.154633 | 3500.0 |

The Number of null values are nearly half of the nonnull values.

## Task 2d. Null values - device counts

Next, check the two populations with respect to the `device` variable.

**Question:** How many iPhone users had null values and how many Android users had null values?

In [16]:

```python
# Get count of null values by device
### YOUR CODE HERE ###
```

```
device_count = null_rows['device'].nunique()
```

700

Now, of the rows with null values, calculate the percentage with each device—Android and iPhone. You can do this directly with the `value_counts()` function.

In [19]:
```
# Get rows with null values
null_rows = df[df.isnull().any(axis=1)]
# Calculate % of iPhone nulls and Android nulls
total_iphone_rows = df[df['device'] == 'iPhone'].shape[0]
total_android_rows = df[df['device'] == 'Android'].shape[0]
iphone_null_percentage = (null_rows[null_rows['device'] == 'iPhone'].shape[0
android_null_percentage = (null_rows[null_rows['device'] == 'Android'].shape

print("Percentage of null values for iPhone:", iphone_null_percentage)
print("Percentage of null values for Android:", android_null_percentage)
```

```
Percentage of null values for iPhone: 4.621588089330024
Percentage of null values for Android: 4.749389900506852
```

How does this compare to the device ratio in the full dataset?

In [20]:
```
# Calculate % of iPhone users and Android users in full dataset
# Calculate the device ratio in the full dataset
total_devices = df.shape[0]
iphone_ratio = (df[df['device'] == 'iPhone'].shape[0] / total_devices) * 100
android_ratio = (df[df['device'] == 'Android'].shape[0] / total_devices) * 1

print("Percentage of null values for iPhone:", iphone_null_percentage)
print("Percentage of null values for Android:", android_null_percentage)
print("Device ratio in the full dataset:")
print("iPhone:", iphone_ratio)
print("Android:", android_ratio)
```

```
Percentage of null values for iPhone: 4.621588089330024
Percentage of null values for Android: 4.749389900506852
Device ratio in the full dataset:
iPhone: 64.48429895326355
Android: 35.515701046736446
```

The percentage of missing values by each device is consistent with their representation in the data overall.

There is nothing to suggest a non-random cause of the missing data.

Examine the counts and percentages of users who churned vs. those who were retained. How many of each group are represented in the data?

In [25]:
```
# Calculate counts of churned vs. retained
churned_count = df[df['label'] == 'churned'].shape[0]
retained_count = df[df['label'] == 'retained'].shape[0]
```

```python
print("Churned count:", churned_count)
print("Retained count:", retained_count)
```

```
Churned count: 2536
Retained count: 11763
```

This dataset contains 82% retained users and 18% churned users.

Next, compare the medians of each variable for churned and retained users. The reason for calculating the median and not the mean is that you don't want outliers to unduly affect the portrayal of a typical user. Notice, for example, that the maximum value in the `driven_km_drives` column is 21,183 km. That's more than half the circumference of the earth!

In [27]:

```python
# Calculate median values of all columns for churned and retained users
churned_df = df[df['label'] == 'churned']
retained_df = df[df['label'] == 'retained']

churned_median = churned_df.median()
retained_median = retained_df.median()

# Print the median values
print("Median values for churned users:")
print(churned_median)

print("\nMedian values for retained users:")
print(retained_median)
```

```
Median values for churned users:
ID                         7477.500000
sessions                     59.000000
drives                       50.000000
total_sessions              164.339042
n_days_after_onboarding    1321.000000
total_navigations_fav1       84.500000
total_navigations_fav2       11.000000
driven_km_drives           3652.655666
duration_minutes_drives    1607.183785
activity_days                 8.000000
driving_days                  6.000000
dtype: float64

Median values for retained users:
ID                         7509.000000
sessions                     56.000000
drives                       47.000000
total_sessions              157.586756
n_days_after_onboarding    1843.000000
total_navigations_fav1       68.000000
total_navigations_fav2        9.000000
driven_km_drives           3464.684614
duration_minutes_drives    1458.046141
activity_days                17.000000
driving_days                 14.000000
dtype: float64
```

This offers an interesting snapshot of the two groups, churned vs. retained:

Users who churned averaged ~3 more drives in the last month than retained users, but retained users used the app on over twice as many days as churned users in the same time period.

The median churned user drove ~200 more kilometers and 2.5 more hours during the last month than the median retained user.

It seems that churned users had more drives in fewer days, and their trips were farther and longer in duration. Perhaps this is suggestive of a user profile. Continue exploring!

Calculate the median kilometers per drive in the last month for both retained and churned users.

Begin by dividing the `driven_km_drives` column by the `drives` column. Then, group the results by churned/retained and calculate the median km/drive of each group.

In [28]:
```
# Add a column to df called `km_per_drive`

df['km_per_drive'] = df['driven_km_drives'] / df['drives']

# Group by `label`, calculate the median, and isolate for km per drive

median_km_per_drive_by_label = df.groupby('label')['km_per_drive'].median()

print(median_km_per_drive_by_label)
```

```
label
churned     74.109416
retained    75.014702
Name: km_per_drive, dtype: float64
```

The median retained user drove about one more kilometer per drive than the median churned user. How many kilometers per driving day was this?

To calculate this statistic, repeat the steps above using `driving_days` instead of `drives`.

In [29]:
```
# Add a column to df called `km_per_driving_day`
df['km_per_driving_day'] = df['driving_days'] / df['drives']

# Group by `label`, calculate the median, and isolate for km per driving day
km_per_driving_day = df.groupby('label')['km_per_driving_day'].median()

print(km_per_driving_day)
```

```
label
churned     0.100000
retained    0.246154
Name: km_per_driving_day, dtype: float64
```

Now, calculate the median number of drives per driving day for each group.

In [31]:
```
# Add a column to df called `drives_per_driving_day`
df['drives_per_driving_day'] = df['drives'] / df['driving_days']

# Group by `label`, calculate the median, and isolate for drives per driving

drives_per_driving_day = df.groupby('label')['drives_per_driving_day'].media

print(drives_per_driving_day)
```

```
label
churned     10.0000
retained     4.0625
Name: drives_per_driving_day, dtype: float64
```

The median user who churned drove 698 kilometers each day they drove last month, which is almost ~240% the per-drive-day distance of retained users. The median churned user had a similarly disproporionate number of drives per drive day compared to retained users.

It is clear from these figures that, regardless of whether a user churned or not, the users represented in this data are serious drivers! It would probably be safe to assume that this data does not represent typical drivers at large. Perhaps the data—and in particular the sample of churned users—contains a high proportion of long-haul truckers.

In consideration of how much these users drive, it would be worthwhile to recommend to Waze that they gather more data on these super-drivers. It's possible that the reason for their driving so much is also the reason why the Waze app does not meet their specific set of needs, which may differ from the needs of a more typical driver, such as a commuter.

Finally, examine whether there is an imbalance in how many users churned by device type.

Begin by getting the overall counts of each device type for each group, churned and retained.

In [32]:
```
# For each label, calculate the number of Android users and iPhone users
# For each label, calculate the number of Android users and iPhone users
android_users_by_label = df[df['device'] == 'Android'].groupby('label').size
iphone_users_by_label = df[df['device'] == 'iPhone'].groupby('label').size()
```

Now, within each group, churned and retained, calculate what percent was Android and what percent was iPhone.

In [33]:
```
# For each label, calculate the percentage of Android users and iPhone users
total_users_by_label = df.groupby('label').size()
android_percentage_by_label = (android_users_by_label / total_users_by_label
iphone_percentage_by_label = (iphone_users_by_label / total_users_by_label)
```

```python
print("\nAndroid users by label:")
print(android_users_by_label)

print("\niPhone users by label:")
print(iphone_users_by_label)

print("\nAndroid percentage by label:")
print(android_percentage_by_label)

print("\niPhone percentage by label:")
print(iphone_percentage_by_label)
```

```
Android users by label:
label
churned      891
retained    4183
dtype: int64

iPhone users by label:
label
churned     1645
retained    7580
dtype: int64

Android percentage by label:
label
churned     35.134069
retained    35.560656
dtype: float64

iPhone percentage by label:
label
churned     64.865931
retained    64.439344
dtype: float64
```

The ratio of iPhone users and Android users is consistent between the churned group and the retained group, and those ratios are both consistent with the ratio found in the overall dataset.