Activity_Course 2 Waze project lab

September 28, 2025

1 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.

2 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.

3 Identify data types and compile summary information

4 PACE stages

Throughout these project notebooks, you'll see references to the problem-solving framework, PACE. The following notebook components are labeled with the respective PACE stages: Plan, Analyze, Construct, and Execute.

4.1 PACE: Plan

Consider the questions in your PACE Strategy Document and those below to craft your response:

4.1.1 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.*

1. **Understand the Business Context**: Clearly define the project's goal (churn prediction) and how churn is defined by Waze.
2. **Detailed Data Review**: Beyond basic `head()` and `info()`, thoroughly examine `describe()` output, check unique values for categorical columns, and understand variable relationships.
3. **Review and confirm the Data Dictionary**: Confirm the meaning, units, and potential range of values for every column. If anything is unclear or missing from the provided dictionary, that's when we'd add to it or seek clarification.

- What follow-along and self-review codebooks will help you perform this work?
  > 1. **Python Libraries**: Core libraries like Pandas (for data manipulation), NumPy (for numerical ops), and Matplotlib/Seaborn (for visualization).
    2. **Interactive Environments**: Use *Jupyter Notebooks* or *Google Colab*. They allow you to combine code, output, and explanations, making it easy to follow your own logic and review.
    3. **Clear Documentation**: Use *Markdown cells* within your notebooks to explain steps, observations, and decisions. This acts as your personal "codebook."

- What are some additional activities a resourceful learner would perform before starting to code? > * **Formulate Hypotheses** (optional): Brainstorm questions you want the data to answer (e.g., "Do users who drive fewer days churn more?").
  - **Sketch Out Analysis Plan**: Outline the high-level steps: data cleaning, initial exploration, feature engineering, modeling, and evaluation.
  - **Research Domain Knowledge** (optional): Learn more about typical Waze usage patterns, factors influencing app churn, and common metrics in mobile analytics.
  - **Consider Data Limitations**: Think about what questions the data *cannot* answer and what biases might exist.

4.2 PACE: Analyze

Consider the questions in your PACE Strategy Document to reflect on the Analyze stage.

4.2.1 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`

[1]:
```
# Import packages for data manipulation
import pandas as pd
import numpy as np

from IPython.display import display
```

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.

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

4.2.2 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:

Question 1: When reviewing the `df.head()` output, are there any variables that have missing values?

Question 2: When reviewing the `df.info()` output, what are the data types? How many rows and columns do you have?

Question 3: Does the dataset have any missing values?

[3]:
```
# Display and examine the first ten rows of the dataframe
df.head(10)
```

[3]:
```
   ID     label  sessions  drives  total_sessions  n_days_after_onboarding  \
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

   total_navigations_fav1  total_navigations_fav2  driven_km_drives  \
0                     208                       0       2628.845068
1                      19                      64      13715.920550
2                       0                       0       3059.148818
3                     322                       7        913.591123
4                     166                       5       3950.202008
5                       0                       0        901.238699
6                     185                      18       5249.172828
7                       0                       0       7892.052468
8                       0                      26       2651.709764
9                      72                       0       6043.460295

   duration_minutes_drives  activity_days  driving_days   device
0              1985.775061             28            19  Android
1              3160.472914             13            11   iPhone
2              1610.735904             14             8  Android
3               587.196542              7             3   iPhone
4              1219.555924             27            18  Android
5               439.101397             15            11   iPhone
6               726.577205             28            23   iPhone
7              2466.981741             22            20   iPhone
8              1594.342984             25            20  Android
9              2341.838528              7             3   iPhone
```

[4]:
```
df.info()
```

```
<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
```

[5]:

```
summary_df = pd.DataFrame({
    'Column Name': df.columns,
    'Non-Null Count': df.notnull().sum().values,
    'Null Count': df.isnull().sum().values,
    'Data Type' : df.dtypes.values,
    '% Missing' : ((df.isnull().sum().values / len(df)) * 100).round(2)
})
display(summary_df)
```

	Column Name	Non-Null Count	Null Count	Data Type	% Missing
0	ID	14999	0	int64	0.00
1	label	14299	700	object	4.67
2	sessions	14999	0	int64	0.00
3	drives	14999	0	int64	0.00
4	total_sessions	14999	0	float64	0.00
5	n_days_after_onboarding	14999	0	int64	0.00
6	total_navigations_fav1	14999	0	int64	0.00
7	total_navigations_fav2	14999	0	int64	0.00
8	driven_km_drives	14999	0	float64	0.00
9	duration_minutes_drives	14999	0	float64	0.00
10	activity_days	14999	0	int64	0.00
11	driving_days	14999	0	int64	0.00
12	device	14999	0	object	0.00

[6]:

```
# Get summary statistics
df.describe()
```

[6]:

```
                 ID      sessions        drives  total_sessions  \
count  14999.000000  14999.000000  14999.000000    14999.000000
mean    7499.000000     80.633776     67.281152      189.964447
std     4329.982679     80.699065     65.913872      136.405128
min        0.000000      0.000000      0.000000        0.220211
25%     3749.500000     23.000000     20.000000       90.661156
50%     7499.000000     56.000000     48.000000      159.568115
75%    11248.500000    112.000000     93.000000      254.192341
max    14998.000000    743.000000    596.000000     1216.154633

       n_days_after_onboarding  total_navigations_fav1  \
count             14999.000000            14999.000000
mean               1749.837789              121.605974
std                1008.513876              148.121544
min                   4.000000                0.000000
25%                 878.000000                9.000000
50%                1741.000000               71.000000
75%                2623.500000              178.000000
max                3500.000000             1236.000000

       total_navigations_fav2  driven_km_drives  duration_minutes_drives  \
count            14999.000000      14999.000000             14999.000000
mean                29.672512       4039.340921              1860.976012
std                 45.394651       2502.149334              1446.702288
min                  0.000000         60.441250                18.282082
25%                  0.000000       2212.600607               835.996260
50%                  9.000000       3493.858085              1478.249859
75%                 43.000000       5289.861262              2464.362632
max                415.000000      21183.401890             15851.727160

       activity_days  driving_days
count   14999.000000  14999.000000
mean       15.537102     12.179879
std         9.004655      7.824036
min         0.000000      0.000000
25%         8.000000      5.000000
50%        16.000000     12.000000
75%        23.000000     19.000000
max        31.000000     30.000000
```

RESPONSES TO QUESTIONS 1-3 HERE

Answer 1: > Based on a visual inspection of the `df.head()` output provided, there are no immediately apparent missing values. All cells in the first 10 rows appear to be populated.

Answer 2: > The dataset contains eight columns with a data type of `int64`, three columns with a data type of `float64`, and two columns with a data type of `object`. The dataset has **14,999 rows** (represents one unique user) and **13 columns.**

Answer 3: > Yes, the dataset has missing values. The `df.info()` output shows that the `label` column has only 14,299 non-null entries out of a total of 14,999 rows. This indicates there are **700 missing values** in the `label` column.

Initial Cleaning

- Replace columnn name `ID` with `id` for consistency
- Converting `label` and `device` to category:
  - Saves memory
  - Speeds up operations like grouping, filtering
  - Makes the meaning of the data clearer (they're not meant for math)
- Converting `device` to string

[7]:
```
waze_users_df = df.rename(columns={'ID': 'id'})
waze_users_df['label'] = waze_users_df['label'].astype('category')
waze_users_df['device'] = waze_users_df['device'].astype('category')
waze_users_df.info()
```

```
<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  category
 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  category
dtypes: category(2), float64(3), int64(8)
memory usage: 1.3 MB
```

4.2.3 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?

[8]:
```python
# Isolate rows with null values
null_label_waze_users_df = waze_users_df[waze_users_df['label'].isnull()]
# null_label_waze_users_df.head(10)
```

[9]:
```python
# Display summary stats of rows with null values
null_label_waze_users_df.describe()
```

[9]:
```
                 id    sessions      drives  total_sessions  \
count    700.000000  700.000000  700.000000      700.000000
mean    7405.584286   80.837143   67.798571      198.483348
std     4306.900234   79.987440   65.271926      140.561715
min       77.000000    0.000000    0.000000        5.582648
25%     3744.500000   23.000000   20.000000       94.056340
50%     7443.000000   56.000000   47.500000      177.255925
75%    11007.000000  112.250000   94.000000      266.058022
max    14993.000000  556.000000  445.000000     1076.879741

       n_days_after_onboarding  total_navigations_fav1  \
count               700.000000              700.000000
mean               1709.295714              118.717143
std                1005.306562              156.308140
min                  16.000000                0.000000
25%                 869.000000                4.000000
50%                1650.500000               62.500000
75%                2508.750000              169.250000
max                3498.000000             1096.000000

       total_navigations_fav2  driven_km_drives  duration_minutes_drives  \
count              700.000000        700.000000               700.000000
mean                30.371429       3935.967029              1795.123358
std                 46.306984       2443.107121              1419.242246
min                  0.000000        290.119811                66.588493
25%                  0.000000       2119.344818               779.009271
50%                 10.000000       3421.156721              1414.966279
75%                 43.000000       5166.097373              2443.955404
max                352.000000      15135.391280              9746.253023

       activity_days  driving_days
count     700.000000    700.000000
mean       15.382857     12.125714
std         8.772714      7.626373
min         0.000000      0.000000
25%         8.000000      6.000000
50%        15.000000     12.000000
75%        23.000000     18.000000
max        31.000000     30.000000
```

[10]:

```python
# Isolate rows without null values
non_null_label_waze_users_df = waze_users_df[~waze_users_df['label'].isnull()]
# non_null_label_waze_users_df.head(10)
```

[11]:

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

[11]:

	id	sessions	drives	total_sessions
count	14299.000000	14299.000000	14299.000000	14299.000000
mean	7503.573117	80.623820	67.255822	189.547409
std	4331.207621	80.736502	65.947295	136.189764
min	0.000000	0.000000	0.000000	0.220211
25%	3749.500000	23.000000	20.000000	90.457733
50%	7504.000000	56.000000	48.000000	158.718571
75%	11257.500000	111.000000	93.000000	253.540450
max	14998.000000	743.000000	596.000000	1216.154633

	n_days_after_onboarding	total_navigations_fav1
count	14299.000000	14299.000000
mean	1751.822505	121.747395
std	1008.663834	147.713428
min	4.000000	0.000000
25%	878.500000	10.000000
50%	1749.000000	71.000000
75%	2627.500000	178.000000
max	3500.000000	1236.000000

	total_navigations_fav2	driven_km_drives	duration_minutes_drives
count	14299.000000	14299.000000	14299.000000
mean	29.638296	4044.401535	1864.199794
std	45.350890	2504.977970	1448.005047
min	0.000000	60.441250	18.282082
25%	0.000000	2217.319909	840.181344
50%	9.000000	3496.545617	1479.394387
75%	43.000000	5299.972162	2466.928876
max	415.000000	21183.401890	15851.727160

	activity_days	driving_days
count	14299.000000	14299.000000
mean	15.544653	12.182530
std	9.016088	7.833835
min	0.000000	0.000000
25%	8.000000	5.000000
50%	16.000000	12.000000
75%	23.000000	19.000000
max	31.000000	30.000000

Answer: > There is no discernible difference in both the rows with null values in the label column and the rows with non-null values.The mean values are extremely close for both population, there are no significant differences for other variable metrics as well.
This finding suggests that the missing values in the label column may be missing at random, and are not due to any specific user behavior patterns.

4.2.4 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?

[12]:
```
# Get count of null values by device
null_label_waze_users_df['device'].value_counts()
```

[12]:
```
iPhone     447
Android    253
Name: device, dtype: int64
```

Answer: > Out of 700 rows with null `label` value, 447 are IPhone users and 253 are Android users.

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.

[13]:
```
# Calculate % of iPhone nulls and Android nulls
null_label_waze_users_df['device'].value_counts(normalize=True).round(4) * 100
```

[13]:
```
iPhone     63.86
Android    36.14
Name: device, dtype: float64
```

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

[14]:
```
# Calculate % of iPhone users and Android users in full dataset
waze_users_df['device'].value_counts(normalize=True).round(4) * 100
```

[14]:
```
iPhone     64.48
Android    35.52
Name: device, dtype: float64
```

Answer: > 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.

Question: How many of each group are represented in the data?

[15]:
```
# Calculate counts of churned vs. retained
print(waze_users_df['label'].value_counts())
print('-----------------')
print(waze_users_df['label'].value_counts(normalize=True).round(4) * 100)
```

```
retained    11763
churned      2536
Name: label, dtype: int64
-----------------
retained    82.26
churned     17.74
Name: label, dtype: float64
```

Answer: > 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!

[16]:
```
waze_users_label_group = waze_users_df.groupby(['label'])
```

[17]:
```
pd.set_option('display.max_columns', None)
display(waze_users_label_group.agg(['mean', 'median', 'min', 'max']))
```

```
                   id                          sessions                     \
                 mean  median min    max        mean median min  max
label
churned   7544.852918  7477.5   9  14997  87.238959   59.0   0  743
retained  7494.673553  7509.0   0  14998  79.197654   56.0   0  725

             drives                  total_sessions                         \
               mean median min  max            mean      median       min
label
churned   72.730678   50.0   0  596     196.893424  164.339042  1.362129
retained  66.075491   47.0   0  582     187.963672  157.586756  0.220211

                      n_days_after_onboarding                      \
                  max                        mean  median min   max
label
churned   1216.154633             1471.027603  1321.0   6  3496
retained  1117.893821             1812.359432  1843.0   4  3500

         total_navigations_fav1                  total_navigations_fav2  \
                           mean median min   max                   mean
label
churned              139.414826   84.5   0  1170              31.596609
retained             117.938451   68.0   0  1236              29.216101
```

```
                      driven_km_drives                                    \
         median min  max              mean       median         min
label
churned    11.0   0  396       4147.171864  3652.655666  178.232313
retained    9.0   0  415       4022.245150  3464.684614   60.441250

                   duration_minutes_drives                               \
                 max                    mean       median        min
label
churned  19214.47511             1975.459630  1607.183785  23.022685
retained 21183.40189             1840.213146  1458.046141  18.282082

                   activity_days                driving_days
                 max          mean median min max          mean median min max
label
churned  10040.56896      9.644716    8.0   0  31     7.218060    6.0   0  29
retained 15851.72716     16.816628   17.0   0  31    13.252827   14.0   0  30
```

[18]:

```python
# Calculate median values of all columns for churned and retained users
display(waze_users_label_group.median(numeric_only=True))
```

```
              id  sessions  drives  total_sessions  n_days_after_onboarding  \
label
churned   7477.5      59.0    50.0      164.339042                   1321.0
retained  7509.0      56.0    47.0      157.586756                   1843.0

         total_navigations_fav1  total_navigations_fav2  driven_km_drives  \
label
churned                    84.5                    11.0       3652.655666
retained                   68.0                     9.0       3464.684614

         duration_minutes_drives  activity_days  driving_days
label
churned              1607.183785            8.0           6.0
retained             1458.046141           17.0          14.0
```

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.

[19]:
```
# Add a column to df called `km_per_drive`
waze_users_df['km_per_drive'] = waze_users_df['driven_km_drives'] /␣
↪waze_users_df['drives']

# Group by `label`, calculate the median, and isolate for km per drive
waze_users_label_group = waze_users_df.groupby(['label'])
waze_users_label_group[['km_per_drive']].median(numeric_only=True)
```

[19]:
```
          km_per_drive
label
churned      74.109416
retained     75.014702
```

The median retained user drove about one more kilometer per drive than the median churned user.

Question: How many kilometers per driving day was this?

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

[20]:
```
# Add a column to df called `km_per_driving_day`
waze_users_df['km_per_driving_day'] = waze_users_df['driven_km_drives'] /␣
↪waze_users_df['driving_days']

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

[20]:
```
          km_per_driving_day
label
churned           697.541999
retained          289.549333
```

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

[21]:
```
# Add a column to df called `drives_per_driving_day`
waze_users_df['drives_per_driving_day'] = waze_users_df['drives'] /␣
↪waze_users_df['driving_days']

# Group by `label`, calculate the median, and isolate for drives per driving day
waze_users_label_group = waze_users_df.groupby(['label'])
waze_users_label_group[['drives_per_driving_day']].median(numeric_only=True)
```

[21]:
```
          drives_per_driving_day
label
churned                  10.0000
retained                  4.0625
```

Answer: > 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.

[22]:
```
# For each label, calculate the number of Android users and iPhone users
# waze_users_label_group['device'].value_counts()
waze_users_label_device_group = waze_users_df.groupby(['label', 'device'])
waze_users_label_device_group.size()
```

[22]:
```
label     device
churned   Android     891
          iPhone     1645
retained  Android    4183
          iPhone     7580
dtype: int64
```

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

[23]:
```
# For each label, calculate the percentage of Android users and iPhone users
waze_users_label_group['device'].value_counts(normalize=True).round(4) * 100
```

[23]:
```
label
churned   iPhone     64.87
          Android    35.13
retained  iPhone     64.44
          Android    35.56
Name: device, 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.

---

[24]:
```
# Import vizualization packages
import seaborn as sns
import matplotlib.pyplot as plt
```

```
from matplotlib.ticker import FuncFormatter
from matplotlib.ticker import PercentFormatter
from matplotlib.cbook import boxplot_stats
from IPython.display import display, HTML

# Formatter to display y-axis numbers with commas
formatter = FuncFormatter(lambda x, _: f'{int(x):,}')

# Set style
sns.set(style='whitegrid')
```

[25]:
```
# Helper function to safely add text if value is finite
def safe_text(ax, x, y, text, **kwargs):
    if np.isfinite(y):
        ax.text(x, y, f'{text}:\n{y:.2f}', va='center', fontsize=8, **kwargs)

def summarize_column_stats(source_df, columns_of_interest, bool_mask=None,␣
↪mask_label=None):
    # Apply mask if any
    masked_df = source_df
    if bool_mask is not None:
        masked_df = source_df[bool_mask]

    # Select only the columns of interest
    subset_df = masked_df[columns_of_interest]

    # Combine into one DataFrame
    summary_df = pd.DataFrame({
        'total': len(subset_df),                     # Total row count (same for␣
↪all columns)
        'non-null': subset_df.count(),               # Count non-null values
        'null': subset_df.isna().sum(),              # Count null values
        'min': subset_df.min(numeric_only=True),     # Minimum value
        'max': subset_df.max(numeric_only=True)      # Maximum value
    })

    # If mask label is provided, show it
    if mask_label:
        display(HTML(f"<span>Summary for condition: <u>{mask_label}</u></
↪span>"))
    print(summary_df)

#␣
↪------------------------------------------------------------------------------------
# NOT A GENERIC FUNCTION - SPECIFIC TO WAZE DATA
```

```
def summarize_waze_column_stats_multi(columns_of_interest, mask_column_name,␣
↪mask_column_label):
    summarize_column_stats(waze_users_df, columns_of_interest,
                           waze_users_df[mask_column_name] == 0,␣
↪f'{mask_column_label} that have zero values')
    summarize_column_stats(waze_users_df, columns_of_interest,
                           (waze_users_df[mask_column_name] == 0) &␣
↪(waze_users_df['label'] == 'churned'),
                           f'{mask_column_label} that have zero values for␣
↪Churned Users')
    summarize_column_stats(waze_users_df, columns_of_interest,
                           (waze_users_df[mask_column_name] == 0) &␣
↪(waze_users_df['label'] == 'retained'),
                           f'{mask_column_label} that have zero values for␣
↪Retained Users')
    summarize_column_stats(waze_users_df, columns_of_interest,
                           (waze_users_df[mask_column_name] == 0) &␣
↪(waze_users_df['label'].isna()),
                           f'{mask_column_label} that have zero values for NULL␣
↪User Labels')
#␣
↪-----------------------------------------------------------------------------------------

def plot_bar_by_group(source_df,
                      x_column_name,
                      x_column_label,
                      hue_column_name,
                      hue_column_label,
                      count_label):
    # Grouped data
    grouped_data = source_df.groupby([x_column_name, hue_column_name]).size().
↪reset_index(name='count')

    plt.figure(figsize=(10, 6))
    barplot = sns.barplot(data=grouped_data, x=x_column_name, y='count',␣
↪hue=hue_column_name)

    # Manually add text labels
    for bar in barplot.patches:
        height = bar.get_height()
        x = bar.get_x() + bar.get_width() / 2   # Center the text on the bar
        plt.text(x, height + 100, int(height), ha='center', va='bottom',␣
↪fontsize=9)

    plt.title(f'Distribution of {count_label} by {x_column_label} and␣
↪{hue_column_label}')
```

```
    plt.xlabel(f'{x_column_label}')
    plt.ylabel(f'{count_label}')
    plt.legend(title=hue_column_label)
    plt.tight_layout()
    plt.show()

def group_summary_stats(grouped_df, column_name):
    # Define the aggregation functions
    agg_funcs = {
                    column_name: [lambda x: round(x.mean(), 2),
                                 'median',
                                 lambda x: x.quantile(0.25),
                                 lambda x: x.quantile(0.75),
                                 lambda x: x.quantile(0.75) - x.quantile(0.25),
                                 'min',
                                 'max']
                }

    # Perform the aggregation
    stats_df = grouped_df.agg(agg_funcs)
    stats_df.columns = ['mean', 'median', 'Q1', 'Q3', 'IQR', 'min', 'max']

    # Reset index for clean output
    stats_df = stats_df.reset_index()

    pd.set_option('display.max_columns', None)
    return stats_df

def plot_box_by_group(source_df,
                      ax,
                      column_name,
                      column_label,
                      group_by_column_name,
                      group_by_column_label,
                      grouped_df=None,
                      show_legend=False):

    sns.boxplot(data=source_df, x=group_by_column_name, y=column_name, ax=ax)
    ax.yaxis.set_major_formatter(formatter)
    ax.set_title(f'Distribution of {column_label} by {group_by_column_label}')
    ax.set_xlabel(f'{group_by_column_label}')
    ax.set_ylabel(f'{column_label}')

    # Plot means
    means = grouped_df[column_name].mean()
    for i, mean in enumerate(means):
        if np.isfinite(mean):   # Safeguard against NaN, inf, -inf
```

```
            ax.scatter(i, mean, color='red', marker='D',
                       s=40, zorder=5, label='Mean' if i == 0 else "")
            ax.hlines(mean, xmin=i - 0.4, xmax=i + 0.4,
                      colors='red', linestyles='dashed', linewidth=1)
            ax.text(i, mean, f'{mean:.2f}',
                    ha='center', va='bottom', fontsize=8)

    # Calculate boxplot stats
    stats = []
    for i, (name, group) in enumerate(grouped_df):
        # stats = boxplot_stats(group[column_name])[0]   # Get stats for this␣
↪group

        # Remove NaN values from the column_name
        column_data = group[column_name].dropna()

        # Replace positive/negative infinity with very large/small placeholders
        # so they can still be included in the stats calculation
        temp_data = column_data.replace(np.inf, 1e12).replace(-np.inf, -1e12)

        # Calculate boxplot statistics (min, Q1, median, Q3, max, etc.)
        stats = boxplot_stats(temp_data, whis=1.5)[0]

        # Min and Max (whiskers)
        safe_text(ax, i - 0.2, stats['whislo'], 'Lower Whisker', ha='right')
        safe_text(ax, i - 0.2, stats['whishi'], 'Upper Whisker', ha='right')

        # Q1 and Q3
        safe_text(ax, i + 0.2, stats['q1'], 'Q1', ha='left')
        safe_text(ax, i + 0.2, stats['q3'], 'Q3', ha='left')

    # Show legend only if there are labeled plot elements.
    # If no handles are found, it means all relevant values were NaN or␣
↪infinite,
    # so we print an info message instead of showing an empty legend.
    if show_legend:
        handles, labels = ax.get_legend_handles_labels()
        if handles:
            ax.legend()
        else:
            print(f'[INFO] Legend not shown for "{column_label}" because all␣
↪relevant values were NaN or infinite.')

def plot_box_with_stats_by_group(source_df,
                                 column_name,
                                 column_label,
                                 group_by_column_name,
```

```
                                  group_by_column_label,
                                  grouped_df=None,
                                  show_legend=False):
    fig, ax = plt.subplots(figsize=(10, 6))

    if grouped_df is None:
        grouped_df = source_df.groupby([group_by_column_name])

    # ********** Plot box chart **********
    plot_box_by_group(source_df,
                      ax,
                      column_name,
                      column_label,
                      group_by_column_name,
                      group_by_column_label,
                      grouped_df,
                      show_legend)
    plt.tight_layout()
    plt.show()

    # ********** Show statistical summary **********
    print(f'{column_label} per {group_by_column_label}')
    display(group_summary_stats(grouped_df, column_name))

def plot_histogram_all_and_by_group(source_df,
                                     x_column_name,
                                     x_column_label,
                                     group_by_column_name,
                                     group_by_column_label,
                                     count_label,
                                     grouped_df=None,
                                     as_percentage=False,
                                     include_overall=True):
    if include_overall:
        # ********** Plot histogram for overall data **********
        # print('\n')
        plt.figure(figsize=(12, 5))
        hist_overall = sns.histplot(source_df[x_column_name],
                                    bins=50,
                                    kde=True)
        plt.title(f'Overall Distribution of {x_column_label}')
        plt.xlabel(x_column_label)
        plt.ylabel(count_label)

        # Format x-axis as percentage (optional)
        if as_percentage:
            plt.gca().xaxis.set_major_formatter(PercentFormatter(xmax=1))
```

```
        # Rotate x-axis labels
        # plt.xticks(rotation=45)

        # Add value labels on top of bars
        for patch in hist_overall.patches:
            height = patch.get_height()
            if height > 0:
                plt.text(patch.get_x() + patch.get_width() / 2, height + 5,␣
↪f'{int(height)}',
                         ha='center', va='bottom', fontsize=8, rotation=45)

        plt.tight_layout()
        plt.show()

        # ********** Show overall statistical summary **********
        print(f'Overall {x_column_label}')
        summary_df = pd.DataFrame([
            {
                'count': source_df[x_column_name].count(),
                'mean': source_df[x_column_name].mean(),
                'median': source_df[x_column_name].median(),
                'min': source_df[x_column_name].min(),
                'max': source_df[x_column_name].max()
            }
        ])
        display(summary_df)
        print('\n')

    # ********** Plot histogram by group **********
    plt.figure(figsize=(12, 5))
    hist_by_group = sns.histplot(data=source_df, x=x_column_name,␣
↪hue=group_by_column_name,
                                 bins=50, kde=True, element='bars',␣
↪stat='count', common_norm=False)
    plt.title(f'Distribution of {x_column_label} by {group_by_column_label}')
    plt.xlabel(x_column_label)
    plt.ylabel(count_label)

    if hist_by_group.get_legend() is not None:
        hist_by_group.get_legend().set_title(group_by_column_label)

    # Format x-axis as percentage (optional)
    if as_percentage:
        plt.gca().xaxis.set_major_formatter(PercentFormatter(xmax=1))

    # Rotate x-axis labels
```

```
    # plt.xticks(rotation=45)

    # Add value labels per hue group
    for container in hist_by_group.containers:
        for bar in container:
            height = bar.get_height()
            if height > 0:
                hist_by_group.annotate(f'{int(height)}', xy=(bar.get_x() + bar.
↪get_width() / 2, height),
                                       xytext=(0, 4), textcoords='offset␣
↪points', ha='center',
                                       va='bottom', fontsize=8, rotation=45)
    plt.tight_layout()
    plt.show()

    # ********** Show statistical summary by group **********
    if grouped_df is None:
        grouped_df = source_df.groupby([group_by_column_name])

    print(f'{x_column_label} by {group_by_column_label}')
    group_summary_df = grouped_df.agg({x_column_name: ['count', 'mean',␣
↪'median', 'min', 'max']})

    # Flatten multi-level columns
    group_summary_df.columns = [f'{stat}' for stat in group_summary_df.columns.
↪get_level_values(1)]
    group_summary_df = group_summary_df.reset_index()
    display(group_summary_df)

def plot_scatter(source_df,
                 x_column_name,
                 x_column_label,
                 y_column_name,
                 y_column_label,
                 hue_column_name=None,
                 hue_column_label=None,
                 size_column=None):

    plt.figure(figsize=(12, 8))

    # Create scatter plot with optional hue and size
    scatter = sns.scatterplot(
        data=source_df,
        x=x_column_name,
        y=y_column_name,
        hue=hue_column_name,
        size=size_column,
```

```
        alpha=0.7
    )

    # Labels and title
    plt.xlabel(x_column_label)
    plt.ylabel(y_column_label)

    title = f'{x_column_label} vs {y_column_label}'
    if hue_column_label:
        title = f'{x_column_label} vs {y_column_label} by {hue_column_label}'
        plt.legend(title=hue_column_label)
    plt.title(title)

    plt.grid(True, linestyle='--', alpha=0.6)
    plt.show()

def plot_heatmap(source_df, column_name_label_map, title):
    plt.figure(figsize=(10, 8))

    # Derive column names
    column_names = list(column_name_label_map.keys())

    # Compute correlation and rename for display
    correlation = source_df[column_names].corr()
    correlation.rename(index=column_name_label_map,␣
↪columns=column_name_label_map, inplace=True)

    sns.heatmap(correlation, annot=True, cmap='coolwarm', fmt='.2f')
    plt.title(title)
    plt.show()
```

4.2.5 Preliminary Data Integrity Issues

Before conducting visual analysis, several inconsistencies were identified in the dataset. These issues affect key usage variables such as `drives`, `sessions`, and `driving_days`, and must be addressed to ensure accurate interpretation and reliable churn modeling.

[26]:
```
summarize_waze_column_stats_multi(['driving_days', 'driven_km_drives'],␣
↪'drives', 'Drives')
```

<IPython.core.display.HTML object>

	total	non-null	null	min	max
driving_days	106	106	0	0.000000	28.00000
driven_km_drives	106	106	0	628.853609	16480.93908

<IPython.core.display.HTML object>

	total	non-null	null	min	max
driving_days	14	14	0	0.000000	17.00000
driven_km_drives	14	14	0	1410.581332	12316.76797

<IPython.core.display.HTML object>

	total	non-null	null	min	max
driving_days	88	88	0	0.000000	28.00000
driven_km_drives	88	88	0	628.853609	16480.93908

<IPython.core.display.HTML object>

	total	non-null	null	min	max
driving_days	4	4	0	4.00000	17.000000
driven_km_drives	4	4	0	1363.20614	5597.490052

Drives = 0 with Distance Logged This finding presents a logical contradiction. A user cannot accumulate hundreds or even thousands of kilometers of distance traveled (`driven_km_drives`) without completing any recorded trips (`drives`). This inconsistency suggests a failure in the trip logging mechanism, where the event counter (`drives`) did not register trips for 106 users while the distance tracker (`driven_km_drives`) continued to capture activity. As a result, this portion of the data is not reliable for straightforward analysis and must be cleaned or corrected before inclusion in any predictive modeling.

[27]:
```
summarize_waze_column_stats_multi(['driven_km_drives', 'activity_days'],␣
↪'sessions', 'Sessions')
```

<IPython.core.display.HTML object>

	total	non-null	null	min	max
driven_km_drives	105	105	0	628.853609	16480.93908
activity_days	105	105	0	0.000000	30.00000

<IPython.core.display.HTML object>

	total	non-null	null	min	max
driven_km_drives	14	14	0	1410.581332	12316.76797
activity_days	14	14	0	0.000000	19.00000

<IPython.core.display.HTML object>

	total	non-null	null	min	max
driven_km_drives	87	87	0	628.853609	16480.93908
activity_days	87	87	0	0.000000	30.00000

<IPython.core.display.HTML object>

	total	non-null	null	min	max
driven_km_drives	4	4	0	1363.20614	5597.490052
activity_days	4	4	0	9.00000	26.000000

Sessions = 0 with Activity Recorded This issue mirrors the previous finding. A user must initiate a session to log driving distance or activity days. It is highly improbable for a user to accumulate more than 600 kilometers and remain active for nearly an entire month without a single recorded session. This points to a flaw in the session recording process: sessions were in fact initiated and activity occurred, but the final session count was incorrectly stored as zero. The evidence confirms that these 105 users were active but were misrepresented by the `sessions` metric.

[28]:
```python
summarize_waze_column_stats_multi(['drives', 'driven_km_drives'], 'driving_days', 'Driving days')
```

<IPython.core.display.HTML object>

	total	non-null	null	min	max
drives	1024	1024	0	0.000000	407.00000
driven_km_drives	1024	1024	0	159.444055	16321.74737

<IPython.core.display.HTML object>

	total	non-null	null	min	max
drives	393	393	0	0.000000	382.00000
driven_km_drives	393	393	0	195.996535	14936.04257

<IPython.core.display.HTML object>

	total	non-null	null	min	max
drives	590	590	0	0.000000	407.00000
driven_km_drives	590	590	0	159.444055	16321.74737

<IPython.core.display.HTML object>

	total	non-null	null	min	max
drives	41	41	0	2.000000	224.00000
driven_km_drives	41	41	0	799.048257	12961.44177

Driving Days = 0 with Trips Logged This is the most significant and widespread data flaw. It is logically impossible for a user to complete hundreds of trips and drive thousands of kilometers without any recorded driving days. This error affects over 1,000 users across both retained and churned groups. Since `driving_days` was previously identified as a strong predictor of churn (with retained users averaging 14 days vs. 6 for churned users), the unreliability of this metric undermines a key variable for prediction.

Overall Conclusion These three findings confirm that essential usage metrics—`drives`, `sessions`, and `driving_days`—are compromised by systematic logging errors. Left uncorrected, they will distort both descriptive analysis and predictive modeling. To address this, zero values that conflict with positive distance (`driven_km_drives`) should either be replaced with `NaN` to exclude them from calculations or adjusted to corrected values inferred from related activity. Cleaning these anomalies is a prerequisite for producing a reliable churn prediction model.

PACE: Analyze Stage Questions **Question 1:** Will the available information be sufficient to achieve the goal based on your intuition and the analysis of the variables?

Question 2: How would you build summary dataframe statistics and assess the min and max range of the data?

Question 3: Do the averages of any of the data variables look unusual? Can you describe the interval data?

Answer 1: > Yes, the dataset contains enough information to build a baseline churn prediction model. However, several integrity issues were identified, such as users with positive kilometers but zero recorded `drives`, `sessions`, or `driving_days`. These anomalies must be addressed through cleaning or validation before modeling. While the available variables are sufficient for initial work, additional historical (multi-month) data would provide deeper behavioral context and improve model reliability.

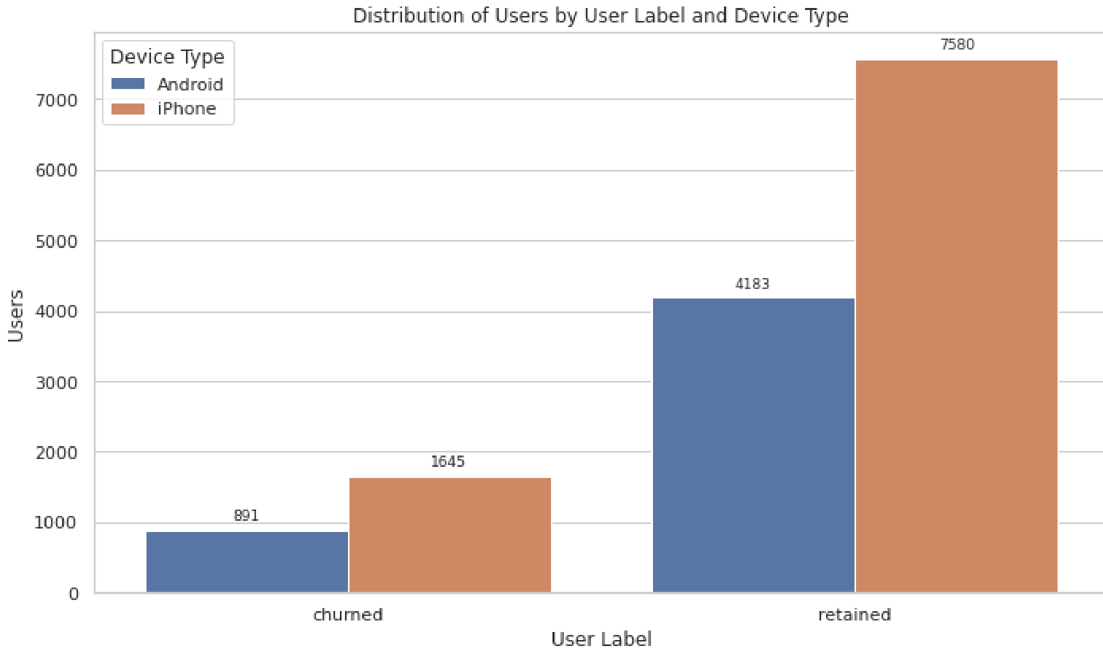
Answer 2: > We would use the `.describe()` method in Pandas. The minimum and maximum values in its output directly show the range for each numerical variable. This allows quick detection of outliers and validation of whether values fall within logical limits.

Answer 3: > The averages themselves are not problematic, but they reveal skewness due to extreme outliers. For example, maximum values for `sessions`, `drives`, and `driven_km_drives` are far higher than their means, suggesting the presence of heavy users. In addition, some averages may be misleading if underlying variables contain contradictions *(e.g., users with kilometers logged but zero drives)*. The numerical variables in this dataset represent ratio data, not interval data. Ratio data has a meaningful zero point *(e.g., zero kilometers means no distance traveled)* and allows for meaningful comparisons using ratios *(e.g., 10 km is twice 5 km)*.

4.2.6 Visualization

Comparison of Churned vs. Retained Users by Device Type

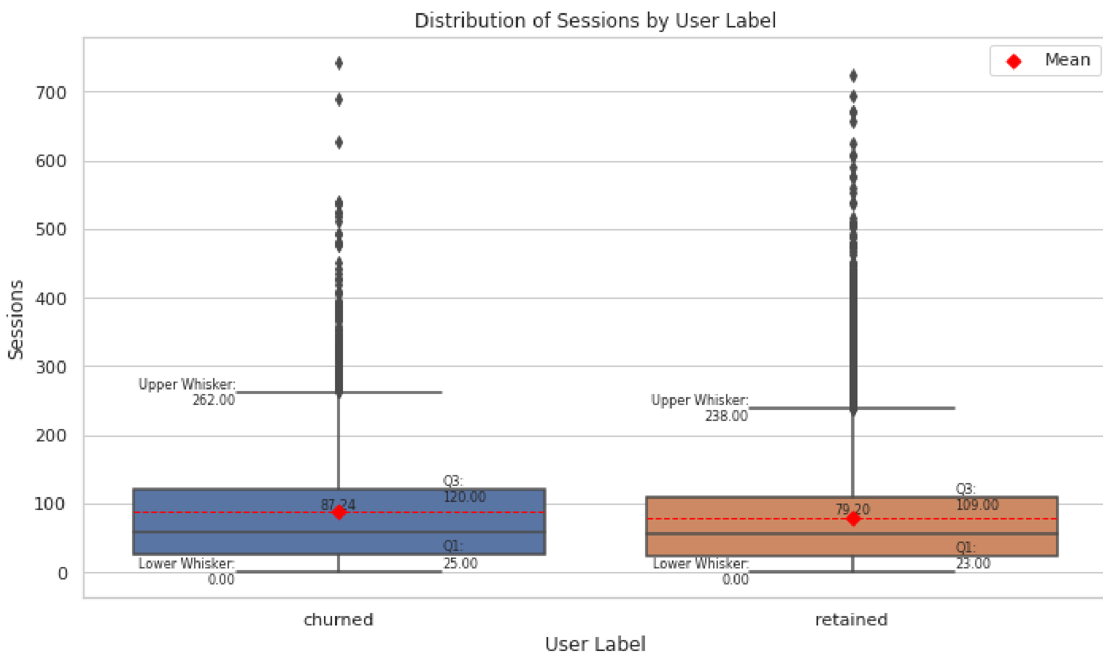
[29]:
```
plot_bar_by_group(waze_users_df, 'label', 'User Label', 'device', 'Device␣
↪Type', 'Users')
```



This chart compares churned and retained users across iPhone and Android devices. The proportions are nearly identical in both groups: approximately 65% iPhone and 35% Android. This indicates that device type is not a meaningful driver of churn, as churn occurs at similar rates across platforms.

Distribution of User Sessions for Churned vs. Retained Groups

[30]:
```
plot_box_with_stats_by_group(waze_users_df, 'sessions', 'Sessions', 'label',␣
↪'User Label', waze_users_label_group, True)
```



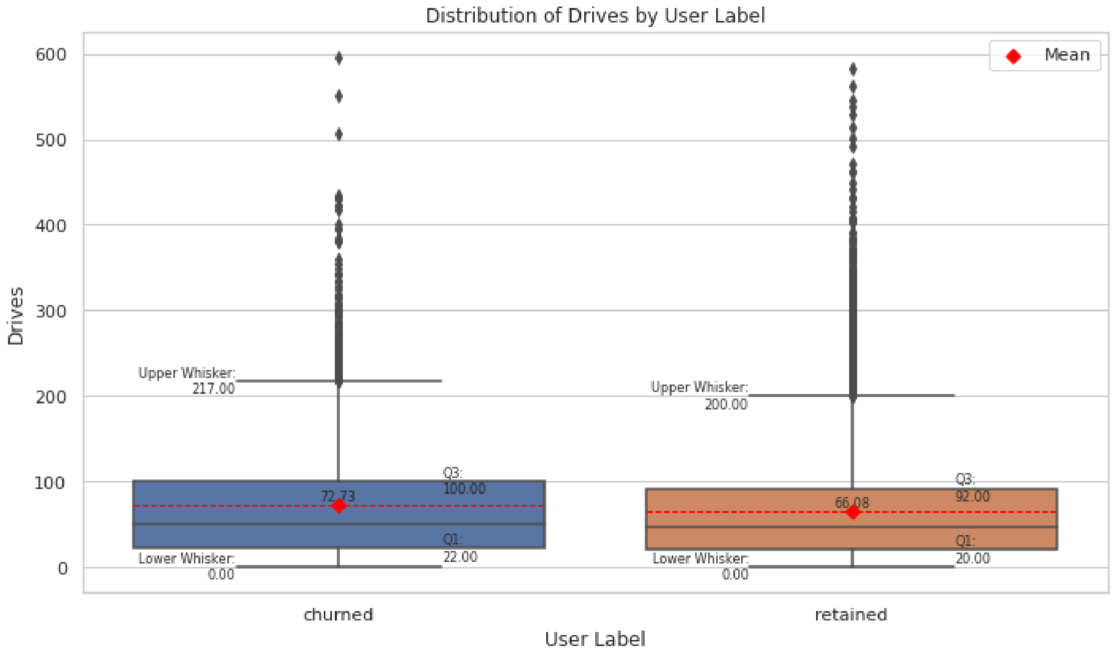
Sessions per User Label

	label	mean	median	Q1	Q3	IQR	min	max
0	churned	87.24	59.0	25.0	120.0	95.0	0	743
1	retained	79.20	56.0	23.0	109.0	86.0	0	725

The distribution of sessions reveals that both churned and retained users have similar medians, but churned users exhibit slightly higher variability. The presence of significant outliers among churned users suggests that while most behave similarly to retained users, a small subset engages in an unusually high number of sessions.

Distribution of User Drives for Churned vs. Retained Groups

[31]:
```
plot_box_with_stats_by_group(waze_users_df, 'drives', 'Drives', 'label', 'User␣
↪Label', waze_users_label_group, True)
```



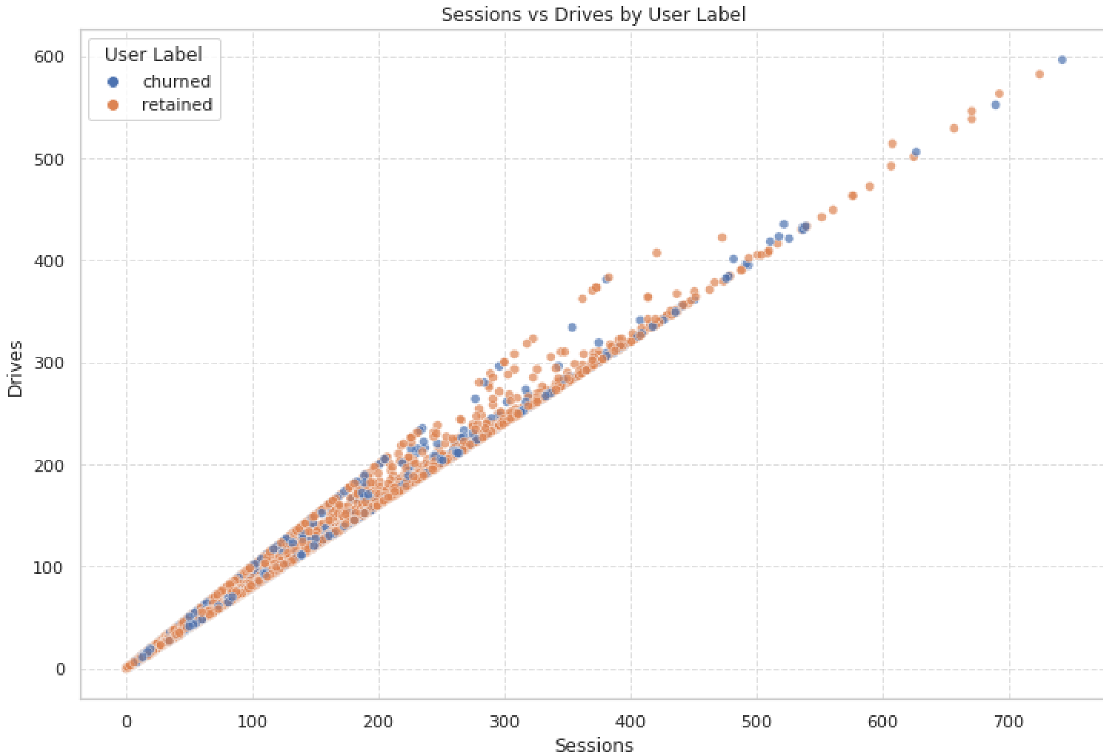

```
Drives per User Label

      label   mean  median    Q1     Q3   IQR  min  max
0   churned  72.73    50.0  22.0  100.0  78.0    0  596
1  retained  66.08    47.0  20.0   92.0  72.0    0  582
```

The boxplot indicates that churned users typically have more drives than retained users, with medians of 50 and 47, respectively. However, churned users also display greater variability and a higher concentration of outliers. This suggests that churned users may exhibit more intense but irregular usage patterns.

Relationship Between Sessions and Drives by User Retention Status

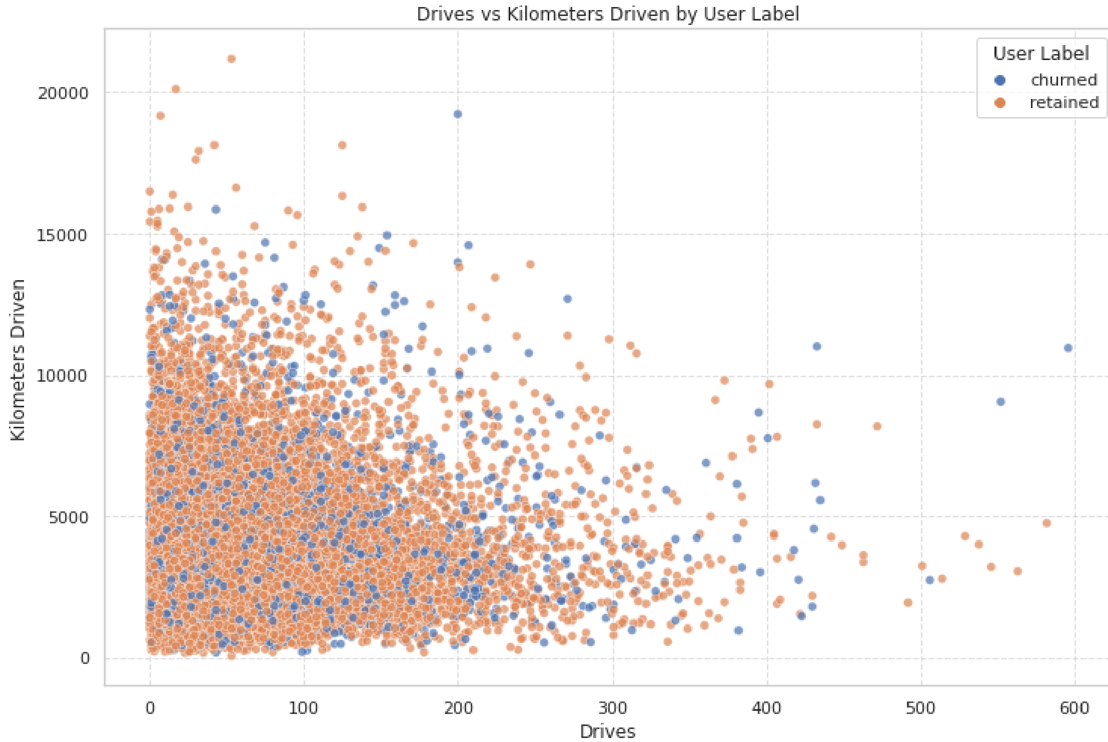
[32]:
```
plot_scatter(waze_users_df, 'sessions', 'Sessions', 'drives', 'Drives',␣
↪'label', 'User Label')
```



The scatter plot highlights a strong positive correlation between sessions and drives for both groups, confirming that more sessions generally translate into more drives. However, churned users appear slightly more concentrated at higher drive counts, suggesting they may complete more trips within a comparable number of sessions.

Relationship Between Drives and Kilometers Driven by User Retention Status

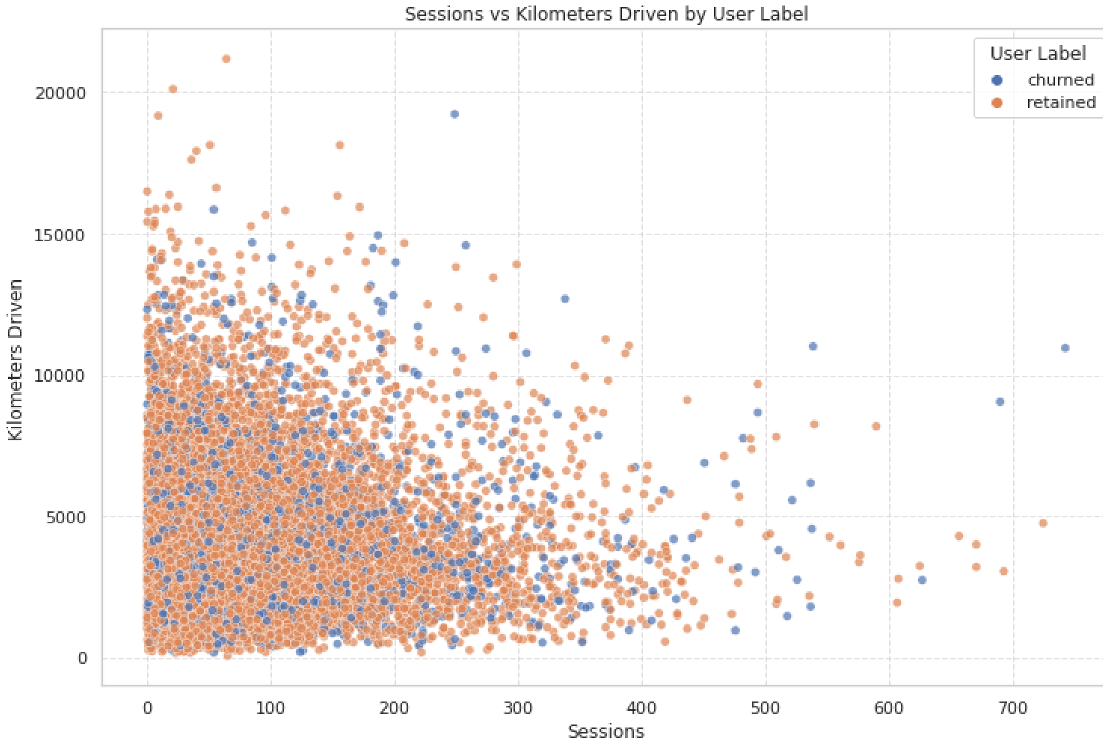
[33]:
```
plot_scatter(waze_users_df, 'drives', 'Drives', 'driven_km_drives', 'Kilometers␣
↪Driven', 'label', 'User Label')
```



This visualization demonstrates a clear positive relationship between the number of drives and total kilometers driven. While both churned and retained users follow the same overall trend, churned users show a tendency toward longer total distances, implying that they may use Waze for longer or more frequent trips.

Relationship Between Sessions and Kilometers Driven by User Retention Status

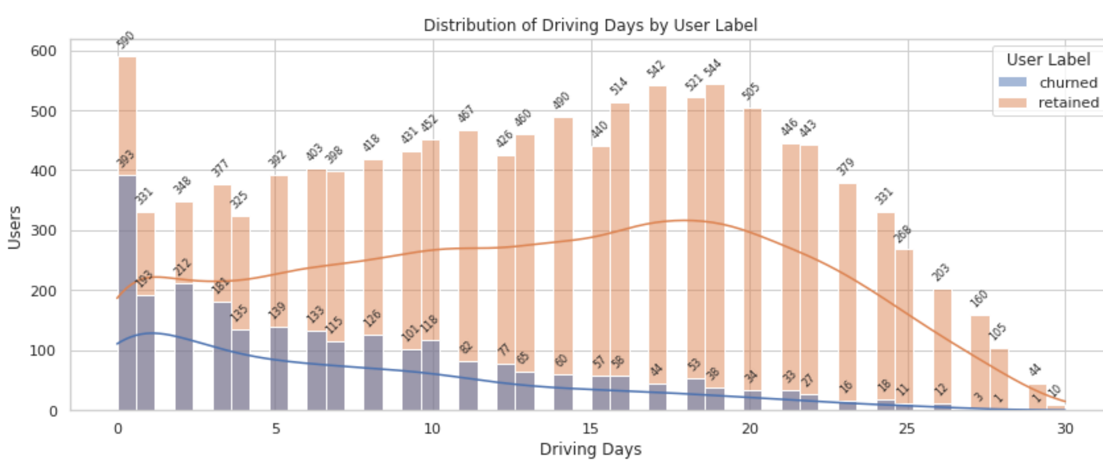
[34]:
```
plot_scatter(waze_users_df, 'sessions', 'Sessions', 'driven_km_drives',␣
↪'Kilometers Driven', 'label', 'User Label')
```



Sessions correlate strongly with kilometers driven, but churned users display higher kilometer counts at comparable session levels. This suggests that churned users engage in longer trips per session, reinforcing the idea that their driving behavior differs in intensity compared to retained users.

Comparison of Driving Days per Month for Churned vs. Retained Users

[35]:
```
plot_histogram_all_and_by_group(waze_users_df, 'driving_days', 'Driving Days',
                                 'label', 'User Label', 'Users',␣
↪waze_users_label_group, include_overall=False)
```



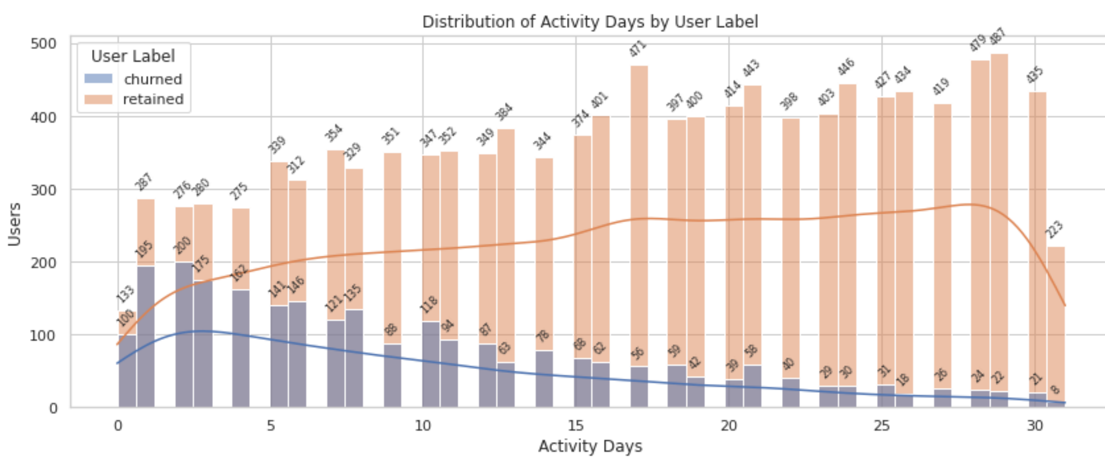
Driving Days by User Label

	label	count	mean	median	min	max
0	churned	2536	7.218060	6.0	0	29
1	retained	11763	13.252827	14.0	0	30

The histogram shows a notable separation between groups. Retained users drive on more days per month (median ~14 days), while churned users drive on fewer (median ~6 days). This finding indicates that higher frequency of app engagement across the month is a key differentiator of retention.

Comparison of Activity Days per Month for Churned vs. Retained Users

[36]:
```
plot_histogram_all_and_by_group(waze_users_df, 'activity_days', 'Activity Days',
                                'label', 'User Label', 'Users',␣
 ↪waze_users_label_group, include_overall=False)
```



Activity Days by User Label

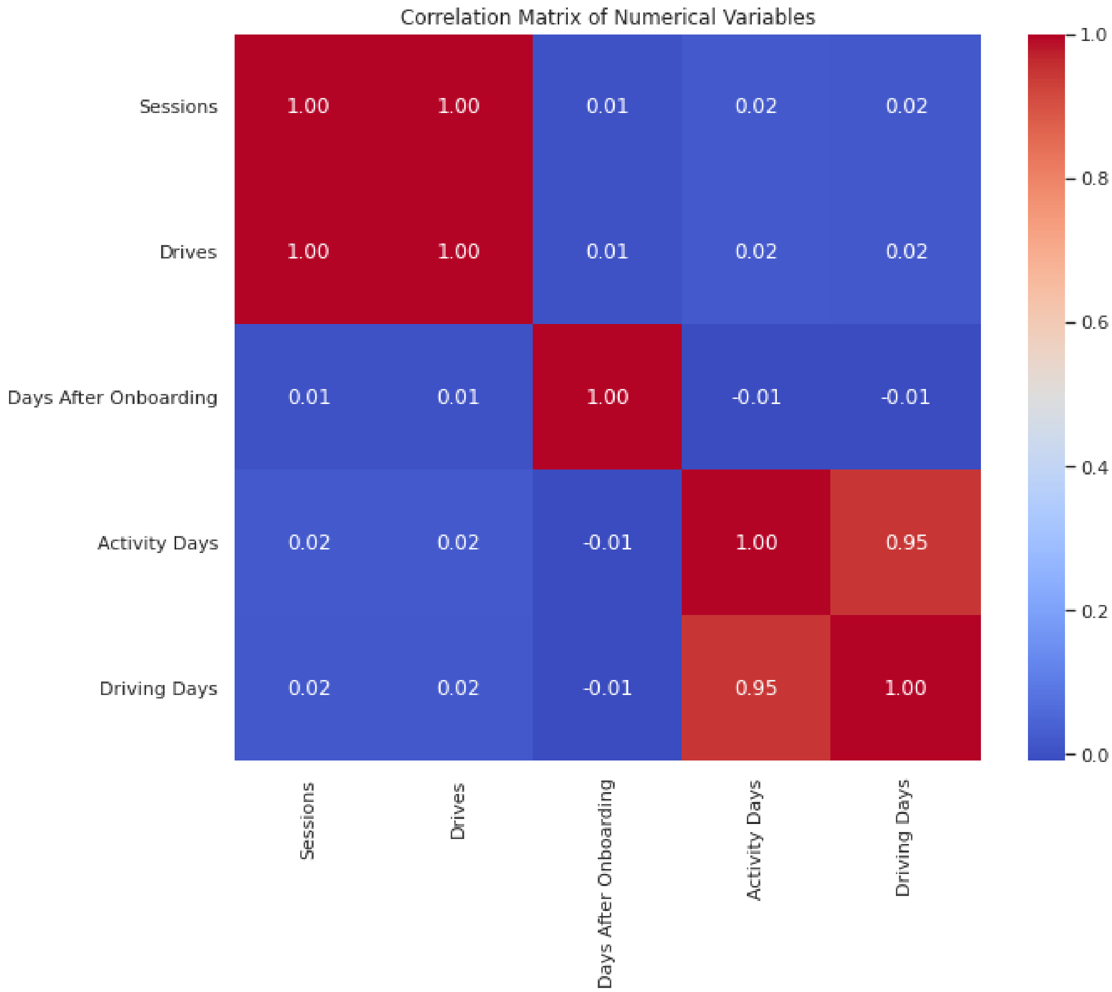
	label	count	mean	median	min	max
0	churned	2536	9.644716	8.0	0	31
1	retained	11763	16.816628	17.0	0	31

Similar to driving days, retained users have significantly more app activity days (median ~17) compared to churned users (median ~8). This reinforces the pattern that consistent and sustained engagement with the app is strongly associated with user retention.

Correlation Matrix of Key User Activity Variables

[37]:

```
plot_heatmap(waze_users_df,
             {
                 'sessions': 'Sessions', 'drives': 'Drives',␣
↪'n_days_after_onboarding': 'Days After Onboarding',
                 'activity_days': 'Activity Days', 'driving_days': 'Driving␣
↪Days'
             },
             'Correlation Matrix of Numerical Variables')
```



The correlation heatmap reveals strong positive relationships among activity-related variables such as sessions, drives, activity days, and driving days, reflecting their shared role in measuring user engagement. By contrast, days after onboarding shows a weaker relationship with current activity, suggesting that longevity alone does not predict engagement or churn.

4.3 PACE: Construct

Note: The Construct stage does not apply to this workflow. The PACE framework can be adapted to fit the specific requirements of any project.

4.4 PACE: Execute

Consider the questions in your PACE Strategy Document and those below to craft your response:

Question 1: Given your current knowledge of the data, what would you initially recommend to your manager to investigate further prior to performing exploratory data analysis?

Question 2: What data initially presents as containing anomalies?

Question 3: What additional types of data could strengthen this dataset?

Answer 1: > We recommend first investigating the significant inconsistencies found in the data. Specifically, many users show a positive value for `driven_km_drives` but have **`zero drives`** or `driving_days`. This is a critical data quality issue that must be addressed before any analysis can be trusted.
In addition, the **700 missing values** in the `label` column also need to be investigated. Since `label` is our target variable for predicting churn, understanding why these values are missing is essential. They may represent new users who have not yet been classified, or they could be the result of data collection errors. Knowing the reason behind these missing values will determine whether these rows should be removed, imputed, or treated as a distinct group. Resolving both the inconsistencies and the missing labels will provide a stronger foundation for reliable analysis and modeling.

Answer 2: > There are outliers in columns such as `sessions`, `drives`, `total_sessions`, `driven_km_drives`, and `duration_minutes_drives` that could skew the analysis and model performance. In addition, some columns show inconsistencies. For example, a few users have zero `drives` or `sessions` but still show positive driving days or kilometers driven, while over 1,000 users have significant drives and kilometers but zero driving days. These are not just outliers, since these fields are expected to align, the mismatches suggest potential data errors that need to be validated and cleaned before modeling.

Answer 3:
> 1. Having historical data of users, rather than just a single monthly snapshot, would likely improve the prediction accuracy of our machine learning model. It would let us see trends in user activity leading up to churn, like a gradual decrease in sessions or drives. This helps us understand the "why" much better than a single point in time. 2. Having data on app and feature usage would also be valuable. Beyond just monthly totals, more granular data—such as the frequency of app use within a day or week, or engagement with specific Waze features—could provide deeper understanding. For example, user interactions like actively reporting traffic, using the carpool feature, connecting with friends, or submitting bug reports or contacting customer support (a potential indicator of frustration leading to churn) can reveal important behavioral patterns. 3. Having user profile data—such as age, general location, or the types of routes they typically take (e.g., daily commutes vs. long-distance travel)—could also provide valuable context for understanding user behavior.

4.4.1 Task 3. Conclusion

Recall that your supervisor, May Santer, asked you to share your findings with the data team in an executive summary. Consider the following questions as you prepare to write your summary. Think about key points you may want to share with the team, and what information is most relevant to the user churn project.

Questions:

1. Did the data contain any missing values? How many, and which variables were affected? Was there a pattern to the missing data?
   > Yes, the data had missing values. The `label` column, which tells us if a user has churned or retained, had **700 missing values**. Earlier analysis show that there didn't seem to be a strong pattern to the missing data. The average user behavior for those with a missing label was very similar to the rest of the dataset.
   In addition to missing labels, we also found significant inconsistencies across related columns. For example, some users recorded positive kilometers driven (`driven_km_drives`) despite having zero `drives` or `driving_days`, while over 1,000 users had substantial `drives` and `driven_km_drives` but zero `driving_days`. Since these fields are supposed to align, these mismatches suggest potential data quality issues that will need to be validated and cleaned before modeling.

2. What is a benefit of using the median value of a sample instead of the mean?
   > A big benefit of using the median is that it's not affected by outliers. The mean can get skewed pretty easily by extremely high or low values. For example, if one user drove 20,000 km in a month, that would pull the average (mean) way up. The median, on the other hand, would stay closer to the middle of the pack, giving us a more representative idea of a "typical" user's behavior.

3. Did your investigation give rise to further questions that you would like to explore or ask the Waze team about?
   > Yes, these questions are:

- **Why is the `label` missing for 700 users?**
  We found that their behavior is pretty standard, so it's not because they're new or inactive. We need to know if there's a technical reason for this missing data or if these users were intentionally left out.
- **Why are `km_per_driving_day` and `drives_per_driving_day` so high for churned users?**
  The median for churned users is significantly higher than for retained users. This is a very interesting pattern. Are these users driving long distances for work (e.g., couriers) and perhaps find another app more suitable, or are they a different type of user altogether? We need to dig into this to understand the "why."

- **What's the relationship between sessions and total_sessions?**
  The descriptive stats show that some users have a low number of sessions this month but a very high total number of sessions. This suggests they might be long-time users who are becoming less active. This could be a very strong predictor for churn.
- **Why do some users record distance and trips without corresponding sessions,**

drives, or driving days?
Several inconsistencies were identified where users accumulated kilometers or trips while the related counters remained at zero. These contradictions point to possible data logging failures. It would be critical to know whether these errors stem from system design, technical glitches, or post-processing steps, since they impact key variables used in churn modeling.

4. What percentage of the users in the dataset were Android users and what percentage were iPhone users?
   > The percentages for the entire dataset are very close to the percentages of those with a null label, as well as the percentages within the churned and retained groups.
   Overall:
   iPhone users: **64.48%** | Android users: **35.52%**
   Null `label`:
   iPhone users: **63.86%** | Android users: **36.14%**
   Churned users:
   iPhone users: **64.87%** | Android users: **35.13%**
   Retained users:
   iPhone users: **64.44%** | Android users: **35.56%**

5. What were some distinguishing characteristics of users who churned vs. users who were retained?
   > **Churned users** have a much higher median `km_per_driving_day` (about 698 km) compared to retained users (about 290 km). Similarly, they have a much higher median `drives_per_driving_day` (10 drives) compared to retained users (4 drives).This suggests that users who churn may be more intense, frequent drivers on the days they're using the app.
   Retained users on the other hand, have a higher median `n_days_after_onboarding` (1843 days) compared to churned users (1321 days), which hints that they are generally longer-term users. They also have more `activity_days` and `driving_days` during the month.

6. Was there an appreciable difference in churn rate between iPhone users vs. Android users?
   > No, there was not. The percentage of iPhone users and Android users within the churned group (64.87% and 35.13% respectively) is almost identical to their proportion in the entire dataset and the retained group. This means that a user's device type doesn't appear to be a strong predictor of churn.

Overall:
The dataset provides a strong basis for churn analysis, but two major data quality issues stand out: the 700 missing labels and the inconsistencies across related activity columns. Both need to be resolved to ensure reliable results.

---

[38]:
```
waze_users_df.head(10)
```

[38]:
```
   id     label  sessions  drives  total_sessions  n_days_after_onboarding  \
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

   total_navigations_fav1  total_navigations_fav2  driven_km_drives  \
0                     208                       0       2628.845068
1                      19                      64      13715.920550
2                       0                       0       3059.148818
3                     322                       7        913.591123
4                     166                       5       3950.202008
5                       0                       0        901.238699
6                     185                      18       5249.172828
7                       0                       0       7892.052468
8                       0                      26       2651.709764
9                      72                       0       6043.460295

   duration_minutes_drives  activity_days  driving_days   device  \
0              1985.775061             28            19  Android
1              3160.472914             13            11   iPhone
2              1610.735904             14             8  Android
3               587.196542              7             3   iPhone
4              1219.555924             27            18  Android
5               439.101397             15            11   iPhone
6               726.577205             28            23   iPhone
7              2466.981741             22            20   iPhone
8              1594.342984             25            20  Android
9              2341.838528              7             3   iPhone

   km_per_drive  km_per_driving_day  drives_per_driving_day
0     11.632058          138.360267               11.894737
1    128.186173         1246.901868                9.727273
2     32.201567          382.393602               11.875000
3     22.839778          304.530374               13.333333
4     58.091206          219.455667                3.777778
5      8.749890           81.930791                9.363636
6   2624.586414          228.224906                0.086957
7    225.487213          394.602623                1.750000
8     57.645864          132.585488                2.300000
9     88.874416         2014.486765               22.666667
```

[39]:
```
waze_users_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 14999 entries, 0 to 14998
```

```
Data columns (total 16 columns):
 #   Column                   Non-Null Count  Dtype
---  ------                   --------------  -----
 0   id                       14999 non-null  int64
 1   label                    14299 non-null  category
 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  category
 13  km_per_drive             14999 non-null  float64
 14  km_per_driving_day       14999 non-null  float64
 15  drives_per_driving_day   14992 non-null  float64
dtypes: category(2), float64(6), int64(8)
memory usage: 1.6 MB
```

[40]:
```
waze_users_df.describe()
```

[40]:
```
                 id      sessions        drives  total_sessions  \
count  14999.000000  14999.000000  14999.000000    14999.000000
mean    7499.000000     80.633776     67.281152      189.964447
std     4329.982679     80.699065     65.913872      136.405128
min        0.000000      0.000000      0.000000        0.220211
25%     3749.500000     23.000000     20.000000       90.661156
50%     7499.000000     56.000000     48.000000      159.568115
75%    11248.500000    112.000000     93.000000      254.192341
max    14998.000000    743.000000    596.000000     1216.154633

       n_days_after_onboarding  total_navigations_fav1  \
count             14999.000000            14999.000000
mean               1749.837789              121.605974
std                1008.513876              148.121544
min                   4.000000                0.000000
25%                 878.000000                9.000000
50%                1741.000000               71.000000
75%                2623.500000              178.000000
max                3500.000000             1236.000000

       total_navigations_fav2  driven_km_drives  duration_minutes_drives  \
count            14999.000000      14999.000000             14999.000000
mean                29.672512       4039.340921              1860.976012
```

```
std               45.394651      2502.149334            1446.702288
min                0.000000        60.441250              18.282082
25%                0.000000      2212.600607             835.996260
50%                9.000000      3493.858085            1478.249859
75%               43.000000      5289.861262            2464.362632
max              415.000000     21183.401890           15851.727160

       activity_days  driving_days  km_per_drive  km_per_driving_day  \
count   14999.000000  14999.000000  1.499900e+04        1.499900e+04
mean       15.537102     12.179879           inf                 inf
std         9.004655      7.824036           NaN                 NaN
min         0.000000      0.000000  1.008775e+00        3.022063e+00
25%         8.000000      5.000000  3.323065e+01        1.672804e+02
50%        16.000000     12.000000  7.488006e+01        3.231459e+02
75%        23.000000     19.000000  1.854667e+02        7.579257e+02
max        31.000000     30.000000           inf                 inf

       drives_per_driving_day
count            1.499200e+04
mean                      inf
std                       NaN
min              0.000000e+00
25%              1.800000e+00
50%              4.666667e+00
75%              1.216667e+01
max                       inf
```

4.4.2 Data Analyst Notes

`inf` and `NaN` Values of Derived Columns In the resulting DataFrame after my initial cleaning, I've added three computed columns to aid in my analysis: * **`km_per_drive`** = `driven_km_drives` / `drives` * **`km_per_driving_day`** = `driven_km_drives` / `driving_days` * **`drives_per_driving_day`** = `drives` / `driving_days`

Using `pandas`' `describe()`, the values for `mean`, standard deviation (`std`), and `max` are `inf` and `NaN` for these computed columns. It was observed, that in `drives` and `driving_days` there are zero values (0.0) present. These columns are being used as divisors for our computed columns. In `pandas`, when a floating-point number is divided by zero, the result is positive or negative infinity (`inf`). The `NaN` values for the standard deviation (`std`) in these columns are a direct consequence of the `inf` values. It's not possible to compute the standard deviation when a dataset contains infinite values.

`inf` Explanation

Python Behavior In standard Python, dividing by zero raises a `ZeroDivisionError`. This is because the result is mathematically undefined and cannot be represented as a finite number. This forces you to handle the error explicitly instead of allowing invalid results to continue.

`NumPy` and `Pandas` Behavior In contrast, when working with `NumPy` arrays or `Pandas` DataFrames that store values as floating-point numbers (float64), division follows the **IEEE 754 standard** for floating-point arithmetic. This standard defines special values for division by zero:

- Positive number / 0.0 → positive infinity (`inf`)
- Negative number / 0.0 → negative infinity (`-inf`)
- Zero / 0.0 → "Not a Number" (`NaN`).

When calculating the derived ratio columns in `waze_users_df`, any division by zero results in `inf` or `NaN` values instead of raising an error. This allows `Pandas` to compute the entire column without interruption, but it also means that these special values can appear in the results. In turn, statistical summaries such as `mean` or `std` may return `NaN` if the column contains `inf` values, because meaningful variance calculations cannot be performed with infinite values.

[41]:
```python
# This will raise a ZeroDivisionError
try:
    py_result = 50.0 / 0.0
except ZeroDivisionError as error:
    print(f'Python behavior: Error: {error}')

# This will NOT raise an error and will return inf
numpy_result = np.divide(50.0, 0.0)
print(f'NumPy behavior: {numpy_result}')

# This is what happens inside a Pandas DataFrame column
numerator_se = pd.Series([50.0, 10.0])
denominator_se = pd.Series([0.0, 2.0])
result_se = numerator_se / denominator_se
print(f'Result:\n{result_se}')
```

```
Python behavior: Error: float division by zero
NumPy behavior: inf
Result:
0    inf
1    5.0
dtype: float64
```

`inf` values in `max` The maximum value appears as `inf` because, under Python's comparison rules and the IEEE 754 floating-point standard, positive infinity is greater than any finite number. When `pandas.describe()` calculates a column's maximum, it compares each value to the current highest; since `inf` is always larger than any finite value, it becomes the reported maximum.

`NaN` Explanation The standard deviation becomes `NaN` when a column contains `inf` values because the calculation cannot be performed if the `mean` is infinite. In statistics, standard deviation measures the "spread around a mean", meaning how far the values vary from the average. A small standard deviation means values are close to the mean, while a large one means they are more scattered.

[42]:
```
# Create a boolean mask to get inf values in km_per_drive column
km_per_drive_inf = np.isinf(waze_users_df['km_per_drive'])

# Apply the mask to our DataFrame
km_per_drive_inf_df = waze_users_df[km_per_drive_inf]

km_per_drive_inf_df[['driven_km_drives', 'drives', 'km_per_drive']].head(5)
```

[42]:

	driven_km_drives	drives	km_per_drive
25	5702.339466	0	inf
97	6668.844350	0	inf
217	6103.881670	0	inf
339	2520.850896	0	inf
485	1363.206140	0	inf

[43]:
```
km_per_drive_df = waze_users_df[['km_per_drive']]
print(f"Mean\n{km_per_drive_df.mean(numeric_only=True)}")
print(f"\nStandard Deviation\n{km_per_drive_df.std(numeric_only=True)}")
```

```
Mean
km_per_drive    inf
dtype: float64

Standard Deviation
km_per_drive   NaN
dtype: float64
```

[44]:
```
# Demontrating the result by creating a very small sample
data_with_inf = np.array([10, 20, 30, float('inf')])

# Calculate the mean
mean_value = np.mean(data_with_inf)
print(f"The mean of the array is: {mean_value}")

# Calculate the standard deviation
std_value = np.std(data_with_inf)
print(f"The standard deviation of the array is: {std_value}")
```

```
The mean of the array is: inf
The standard deviation of the array is: nan
```

The standard deviation is calculated using the following formula:

$$\sigma = \sqrt{\frac{\sum(x_i - \mu)^2}{N}}$$

- x_i is each individual value in the dataset.
- is the mean of the dataset.
- ***N*** is the number of values.

Given observations: `[10, 20, 30, inf]`

This formula fails when your data contains `inf`:

1. **Calculate the Mean ( )**: The mean is the sum of all values divided by the count. When you add a finite number to `inf`, the result is `inf`.

```
mean = (10 + 20 + 30 + inf) / 4
     = inf / 4
     = inf
```

2. **Calculate the Deviation from the Mean** $(x_i -)$:

```
-inf = 10 - inf
-inf = 20 - inf
-inf = 30 - inf
NaN = inf - inf
```

3. **Calculate the Sum of Squared Deviations (** $(x_i -)^2$):

```
inf = (-inf)^2
NaN = NaN^2

NaN = (inf + inf + inf + NaN)
```

Since the numerator of the formula becomes `NaN`, the final result of the standard deviation calculation is also `NaN`.

Mitigation Strategies Division by zero in derived columns can produce `inf` or `NaN` values, which may distort statistical summaries. Common approaches include:

- Filtering out rows where the denominator is zero, removing them entirely from the dataset.
- Replacing `inf` and `NaN` with a suitable value (such as 0).
- Using `NaN` for invalid results to retain the rows while ensuring they are excluded from calculations.

These steps help maintain the accuracy of summary statistics and visualizations.

Best practice: For most analyses, using `NaN` is recommended. It preserves data integrity, keeps the dataset complete, and ensures statistical functions automatically ignore invalid values.

Why Our Most Active Users Might Be Leaving

The churned group racks up more sessions and drives than those who stay — and it may not be a coincidence.

Analysis of the available data shows that churned users have, on average, higher sessions, drives, and related activity metrics compared to retained users. One possible explanation — though not directly verifiable with the current dataset — is that heavy usage of Waze could lead to increased battery drain and data consumption, which might prompt some users to switch to alternative navigation solutions. However, because the dataset does not include direct measures of device battery usage, data consumption, or user feedback, this interpretation remains a hypothesis rather than a confirmed cause of churn.

Congratulations! You've completed this lab. However, you may not notice a green check mark next to this item on Coursera's platform. Please continue your progress regardless of the check mark. Just click on the "save" icon at the top of this notebook to ensure your work has been logged.