Waze Project Proposal

Overview

Waze leadership has asked the data team to build a machine learning model to predict user churn. The model is based on data collected from users of the Waze app.

Milestones	Tasks	Deliverables/Reports
1	Establish structure for project workflow (PACE)	• Global-level project document
1a	Write a project proposal	
2	Compile summary information about the data	Data files ready for EDA
2a	Begin exploring the data	
3	Data exploration and cleaning	EDA report
3a	Visualization building	Tableau dashboard/visualizations
4	Compute descriptive statistics	Analysis of testing results between two important variables
4a	Conduct hypothesis testing	
5	Build a regression model	

5a	**Evaluate the model**	Determine the success of the model
6	**Build a machine learning model**	Final model
6a	**Communicate final insights with stakeholders**	Report to all stakeholders

WAZE Data Dictionary:

This project uses a dataset called waze_dataset.csv. It contains synthetic data created for this project in partnership with Waze.

The dataset contains:

14,999 rows – each row represents one unique user

12 columns

Column name	Type	Description
label	obj	Binary target variable (“retained” vs “churned”) for if a user has churned anytime during the course of the month
sessions	int	The number of occurrence of a user opening the app during the month
drives	int	An occurrence of driving at least 1 km during the month
device	obj	The type of device a user starts a session with
total_sessions	float	A model estimate of the total number of sessions since a user has onboarded
n_days_after_onboarding	int	The number of days since a user signed up for the app
total_navigations_fav1	int	Total navigations since onboarding to the user’s favorite place 1
total_navigations_fav2	int	Total navigations since onboarding to the user’s favorite place 2
driven_km_drives	float	Total kilometers driven during the month
duration_minutes_drives	float	Total duration driven in minutes during the month
activity_days	int	Number of days the user opens the app during the month
driving_days	int	Number of days the user drives (at least 1 km) during the month

Waze Project

Milestone 2 / 2a - Compile information about the data. Begin exploring the data.

Inspect and analyze data

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

The goal is to use a dataframe contructed within Python to perform a cursory inspection of the provided dataset.

This notebook has two parts:

Part 1: Summary Information

Part 2: Initial Churned vs. Retained exploration

Identify data types and compile summary information

Imports and data loading

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

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

Summary information

In [3]:
```
df.head(10)
```

Out[3]:

	ID	label	sessions	drives	total_sessions	n_days_after_onboarding	total_navigations_fav1	total_navigati
0	0	retained	283	226	296.748273	2276	208	
1	1	retained	133	107	326.896596	1225	19	
2	2	retained	114	95	135.522926	2651	0	
3	3	retained	49	40	67.589221	15	322	
4	4	retained	84	68	168.247020	1562	166	
5	5	retained	113	103	279.544437	2637	0	
6	6	retained	3	2	236.725314	360	185	
7	7	retained	39	35	176.072845	2999	0	
8	8	retained	57	46	183.532018	424	0	
9	9	churned	84	68	244.802115	2997	72	

In [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
```

Null values and summary statistics

In [5]:
```
# Isolate rows with null values
null_df = df[df['label'].isnull()]
# Display summary stats of rows with null values
null_df.describe()
```

Out[5]:

	ID	sessions	drives	total_sessions	n_days_after_onboarding	total_navigations_fav1	t
count	700.000000	700.000000	700.000000	700.000000	700.000000	700.000000	
mean	7405.584286	80.837143	67.798571	198.483348	1709.295714	118.717143	
std	4306.900234	79.987440	65.271926	140.561715	1005.306562	156.308140	
min	77.000000	0.000000	0.000000	5.582648	16.000000	0.000000	
25%	3744.500000	23.000000	20.000000	94.056340	869.000000	4.000000	
50%	7443.000000	56.000000	47.500000	177.255925	1650.500000	62.500000	
75%	11007.000000	112.250000	94.000000	266.058022	2508.750000	169.250000	
max	14993.000000	556.000000	445.000000	1076.879741	3498.000000	1096.000000	

In [6]:
```
# Isolate rows without null values
not_null_df = df[~df['label'].isnull()]
# Display summary stats of rows without null values
not_null_df.describe()
```

Out[6]:

	ID	sessions	drives	total_sessions	n_days_after_onboarding	total_navigations_fav
count	14299.000000	14299.000000	14299.000000	14299.000000	14299.000000	14299.00000
mean	7503.573117	80.623820	67.255822	189.547409	1751.822505	121.74739
std	4331.207621	80.736502	65.947295	136.189764	1008.663834	147.71342
min	0.000000	0.000000	0.000000	0.220211	4.000000	0.00000
25%	3749.500000	23.000000	20.000000	90.457733	878.500000	10.00000
50%	7504.000000	56.000000	48.000000	158.718571	1749.000000	71.00000
75%	11257.500000	111.000000	93.000000	253.540450	2627.500000	178.00000

max	14998.000000	743.000000	596.000000	1216.154633	3500.000000	1236.00000

Null values - device counts

In [7]:
```
# Get count of null values by device
null_df['device'].value_counts()
```

Out[7]:
```
iPhone     447
Android    253
Name: device, dtype: int64
```

Of the 700 rows with null values, 447 were iPhone users and 253 were Android users.

In [8]:
```
# Calculate % of iPhone nulls and Android nulls
null_df['device'].value_counts(normalize=True)
```

Out[8]:
```
iPhone     0.638571
Android    0.361429
Name: device, dtype: float64
```

In [9]:
```
# Calculate % of iPhone users and Android users in full dataset
df['device'].value_counts(normalize=True)
```

Out[9]:
```
iPhone     0.644843
Android    0.355157
Name: device, dtype: float64
```

The distribution of missing values across different devices aligns with their overall presence in the data, suggesting no indication of a systematic reason behind the missing data.

Churned vs. Retained

In [10]:
```
# Calculate counts of churned vs. retained
print(df['label'].value_counts())
print()
print(df['label'].value_counts(normalize=True))
```

```
retained    11763
churned      2536
Name: label, dtype: int64

retained    0.822645
churned     0.177355
Name: label, dtype: float64
```

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

In [11]:
```
# Calculate median values of all columns for churned and retained users
df.groupby('label').median(numeric_only=True)
```

Out[11]:

	ID	sessions	drives	total_sessions	n_days_after_onboarding	total_navigations_fav1	total_naviga
label							
churned	7477.5	59.0	50.0	164.339042	1321.0	84.5	
retained	7509.0	56.0	47.0	157.586756	1843.0	68.0	

A few interesting observations jump out from this quick comparions.

Churned users averaged significantly fewer activity days and driving days than the retained users, yet they also averaged slightly more drives, kms driven, and minutes driven.

Churned vs. Retained - drive comparisons

In [12]:
```
# Group data by `label` and calculate the medians
medians_by_label = df.groupby('label').median(numeric_only=True)
print('Median kilometers per drive:')
# Divide the median distance by median number of drives
medians_by_label['driven_km_drives'] / medians_by_label['drives']
```

Out[12]:
```
Median kilometers per drive:
label
churned     73.053113
retained    73.716694
dtype: float64
```

There is not a significant difference between churned and retained median kilometers per dri. They both averaged ~73 km/drive. How many kilometers per driving day was this?

In [13]:
```
# Divide the median distance by median number of driving days
print('Median kilometers per driving day:')
medians_by_label['driven_km_drives'] / medians_by_label['driving_days']
```

Out[13]:
```
Median kilometers per driving day:
label
churned     608.775944
retained    247.477472
dtype: float64
```

Calculate the median number of drives per driving day for each group.

In [14]:
```
# Divide the median number of drives by median number of driving days
print('Median drives per driving day:')
medians_by_label['drives'] / medians_by_label['driving_days']
```

Out[14]:
```
Median drives per driving day:
label
churned     8.333333
retained    3.357143
dtype: float64
```

The median churned user traveled an average of 608 kilometers per driving day last month, which is nearly 2.5 times the distance covered by retained users on each drive day. Additionally, the median churned user had a disproportionately higher number of drives per drive day compared to retained users.

Churned vs. Retained - device type comparison

In [15]:
```
# For each label, calculate the number of Android users and iPhone users
df.groupby(['label', 'device']).size()
```

Out[15]:
```
label     device
churned   Android     891
          iPhone     1645
retained  Android    4183
          iPhone     7580
dtype: int64
```

In [16]:
```
# For each label, calculate the percentage of Android users and iPhone users
df.groupby('label')['device'].value_counts(normalize=True)
```

Out[16]:
```
label     device
churned   iPhone     0.648659
          Android    0.351341
retained  iPhone     0.644393
          Android    0.355607
Name: device, dtype: float64
```

The proportion of iPhone users to Android users remains consistent within both the churned and retained groups, and these ratios align with the overall dataset.

Conclusion

- **The dataset contains 700 missing values, and there is no discernible pattern to these missing values.**
- **Within the dataset, around 36% of the users were Android users, whereas approximately 64% were iPhone users.**
- **The median churned user traveled an average of 608 kilometers per driving day last month, which is nearly 2.5 times the distance covered by retained users on each drive day.**
- **The median churned user had a disproportionately higher number of drives per drive day compared to retained users.**
- **In general, churned users covered similar distances but had longer durations of driving within a shorter span of days compared to retained users.**
- **Churned users utilized the app approximately half as frequently as retained users during the same time frame**
- **Churn rate for both iPhone and Android users differed by less than one percentage point. There is no indication of any correlation between device type and churn, suggesting that device choice does not play a significant role in the churn rate.**

Waze Project

Milestone 3 / 3a - Data exploration and cleaning. Visualization building

Exploratory data analysis

The purpose of this project is to conduct exploratory data analysis (EDA) on a provided dataset.

The goal is to continue the examination of the data, adding relevant visualizations that help communicate the story that the data tells.

This notebook has 4 parts:

Part 1: Imports, links, and loading

Part 2: Data Cleaning and Exploration

Part 3: Building visualizations

Part 4: Evaluating and Conclusion

Imports and data loading

In [1]:
```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
import seaborn as sns
```

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

Data cleaning and exploration

In [3]:
```
df.head(10)
```

Out[3]:

	ID	label	sessions	drives	total_sessions	n_days_after_onboarding	total_navigations_fav1	total_navigati
0	0	retained	283	226	296.748273	2276	208	
1	1	retained	133	107	326.896596	1225	19	
2	2	retained	114	95	135.522926	2651	0	
3	3	retained	49	40	67.589221	15	322	
4	4	retained	84	68	168.247020	1562	166	
5	5	retained	113	103	279.544437	2637	0	
6	6	retained	3	2	236.725314	360	185	
7	7	retained	39	35	176.072845	2999	0	
8	8	retained	57	46	183.532018	424	0	

9	9	churned	84	68	244.802115	2997	72

In [5]:
```
df.size
```

Out[5]:
```
194987
```

In [6]:
```
df.describe()
```

Out[6]:

	ID	sessions	drives	total_sessions	n_days_after_onboarding	total_navigations_fav
count	14999.000000	14999.000000	14999.000000	14999.000000	14999.000000	14999.00000
mean	7499.000000	80.633776	67.281152	189.964447	1749.837789	121.60597
std	4329.982679	80.699065	65.913872	136.405128	1008.513876	148.12154
min	0.000000	0.000000	0.000000	0.220211	4.000000	0.00000
25%	3749.500000	23.000000	20.000000	90.661156	878.000000	9.00000
50%	7499.000000	56.000000	48.000000	159.568115	1741.000000	71.00000
75%	11248.500000	112.000000	93.000000	254.192341	2623.500000	178.00000
max	14998.000000	743.000000	596.000000	1216.154633	3500.000000	1236.00000

In [7]:
```
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
```

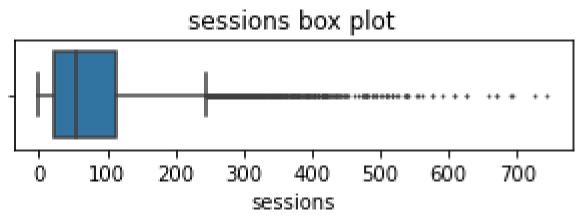
Visualizations

'Sessions' EDA

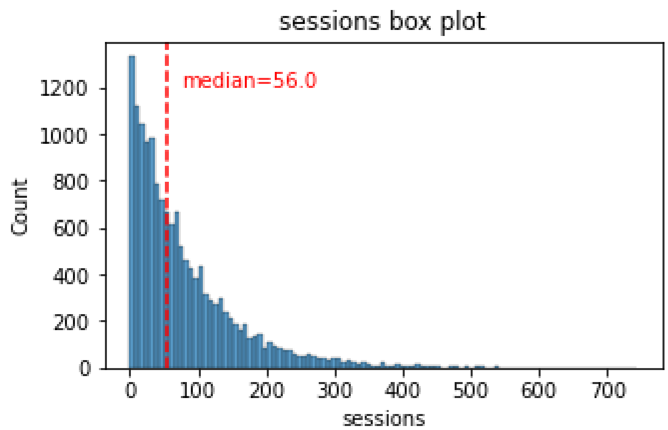
`sessions`

The number of occurrence of a user opening the app during the month

In [8]:
```
# Box plot
plt.figure(figsize=(5,1))
sns.boxplot(x=df['sessions'], fliersize=1)
plt.title('sessions box plot');
```



In [9]:
```
# Histogram
plt.figure(figsize=(5,3))
sns.histplot(x=df['sessions'])
median = df['sessions'].median()
plt.axvline(median, color='red', linestyle='--')
plt.text(75,1200, 'median=56.0', color='red')
plt.title('sessions box plot');
```



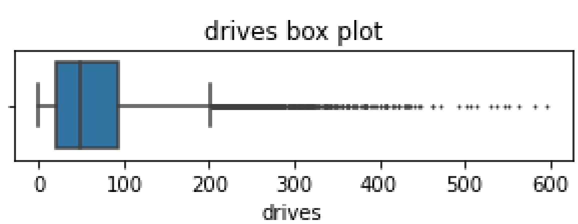
The `sessions` variable exhibits a skewed distribution to the right, where approximately 50% of the observations consist of 56 sessions or fewer. However, the boxplot reveals that a subset of users has more than 700 sessions.

'Drives' EDA

`drives`

An occurrence of driving at least 1 km during the month

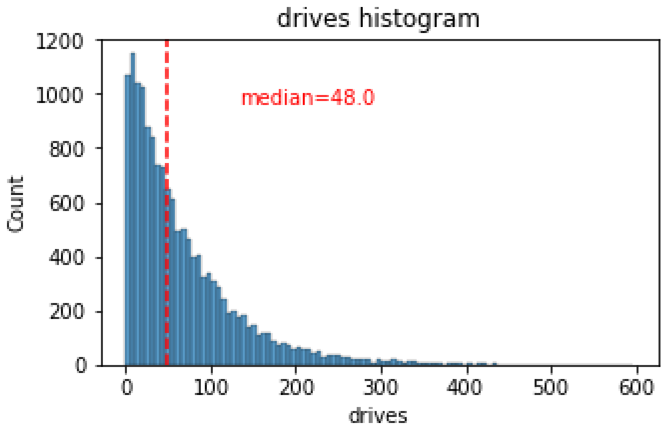
In [10]:
```
# Box plot
plt.figure(figsize=(5,1))
sns.boxplot(x=df['drives'], fliersize=1)
plt.title('drives box plot');
```



In [11]:
```
# Helper function to plot histograms based on the
# format of the `sessions` histogram
def histogrammer(column_str, median_text=True, **kwargs):    # **kwargs = any keyword ar
                                                             # from the sns.histplot() f
    median=round(df[column_str].median(), 1)
    plt.figure(figsize=(5,3))
    ax = sns.histplot(x=df[column_str], **kwargs)            # Plot the histogram
    plt.axvline(median, color='red', linestyle='--')         # Plot the median line
    if median_text==True:                                    # Add median text unless se
```

```
        ax.text(0.25, 0.85, f'median={median}', color='red',
            ha="left", va="top", transform=ax.transAxes)
    else:
        print('Median:', median)
    plt.title(f'{column_str} histogram');
```

In [12]:
```
# Histogram
histogrammer('drives')
```



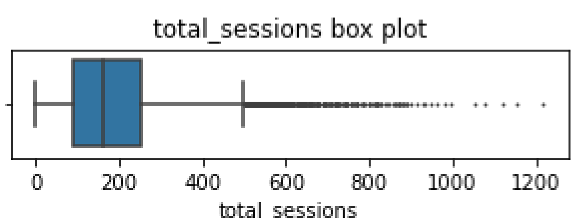
The **`drives`** data exhibits a distribution resembling that of the **`sessions`** variable. It is right-skewed, resembles a log-normal distribution, with a median of 48. However, a subset of drivers recorded over 400 drives in the last month.

'Total Sessions' EDA

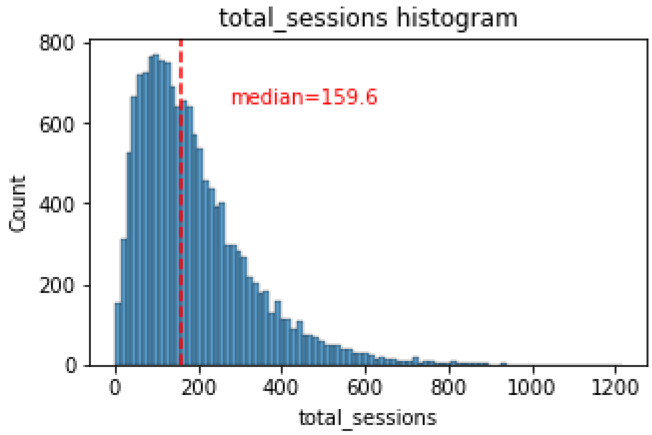
`total_sessions`

A model estimate of the total number of sessions since a user has onboarded

In [13]:
```
# Box plot
plt.figure(figsize=(5,1))
sns.boxplot(x=df['total_sessions'], fliersize=1)
plt.title('total_sessions box plot');
```



In [14]:
```
# Histogram
histogrammer('total_sessions')
```



The distribution of `total_sessions` is right-skewed, appearing closer to a normal distribution compared to the previous variables. The median total number of sessions is approximately 159.6. This observation is noteworthy because if the median number of sessions in the last month was 48 and the median total sessions was around 160, it suggests that a significant proportion of a user's overall sessions possibly occurred within the last month.

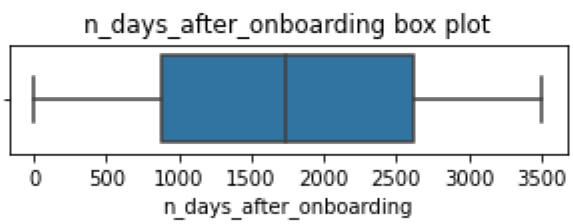
'n Days After Onboarding' EDA

`n_days_after_onboarding`

The number of days since a user signed up for the app

In [15]:

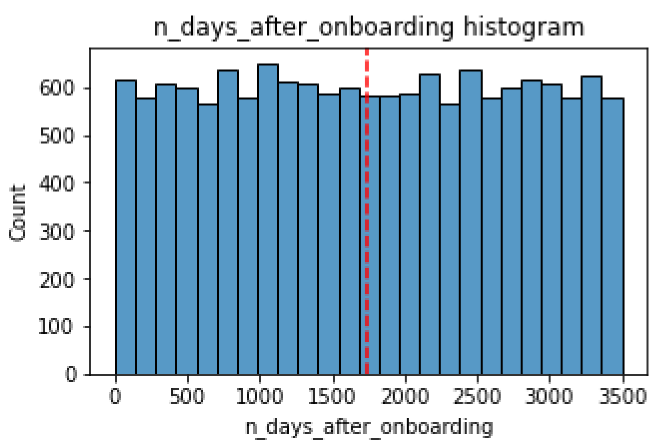
```
# Box plot
plt.figure(figsize=(5,1))
sns.boxplot(x=df['n_days_after_onboarding'], fliersize=1)
plt.title('n_days_after_onboarding box plot');
```



In [16]:

```
# Histogram
histogrammer('n_days_after_onboarding', median_text=False)
```

```
Median: 1741.0
```



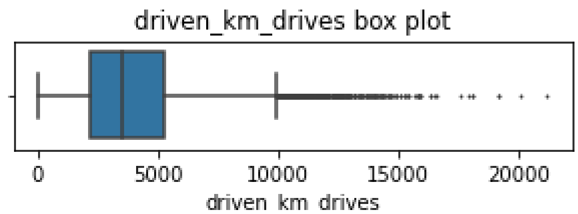
The total user tenure is a uniform distribution with values ranging from near-zero to ~3,500 days, or roughly 9.5 years.

'Driven KM Drives' EDA

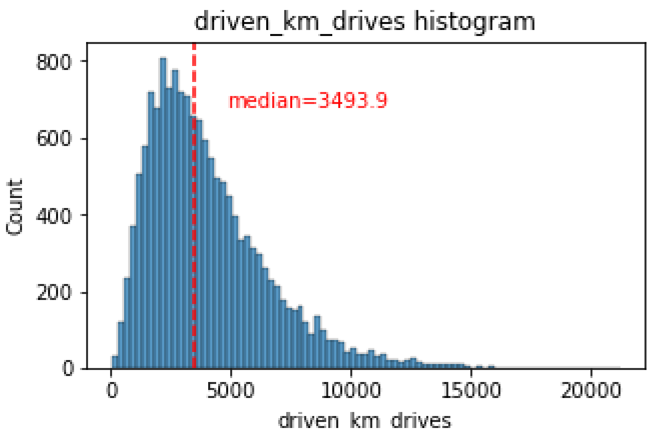
`driven_km_drives`

Total kilometers driven during the month

In [17]:
```
# Box plot
plt.figure(figsize=(5,1))
sns.boxplot(x=df['driven_km_drives'], fliersize=1)
plt.title('driven_km_drives box plot');
```



In [18]:
```
# Histogram
histogrammer('driven_km_drives')
```



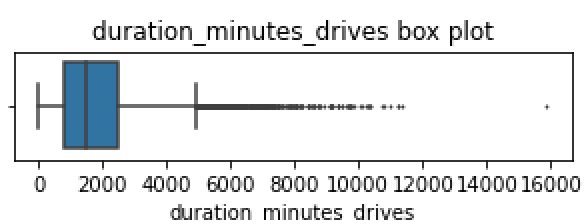
The distribution of drives completed by each user in the last month exhibits right-skewed normal distribution. Roughly 50% of users drove fewer than 3,495 kilometers during that period.

'Duration Minutes Drives' EDA

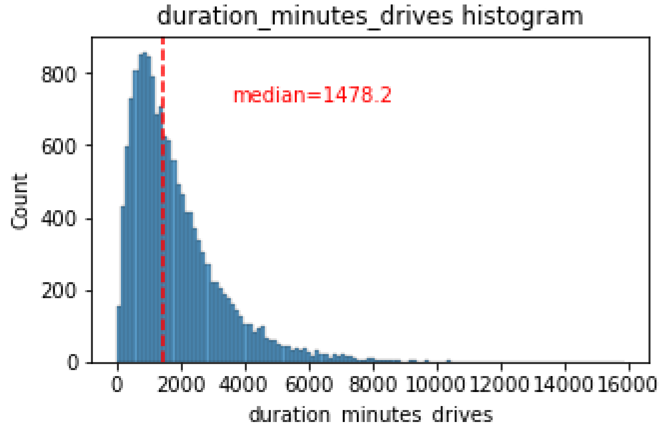
`duration_minutes_drives`

Total duration driven in minutes during the month

In [19]:
```
# Box plot
plt.figure(figsize=(5,1))
sns.boxplot(x=df['duration_minutes_drives'], fliersize=1)
plt.title('duration_minutes_drives box plot');
```



In [20]:
```
# Histogram
histogrammer('duration_minutes_drives')
```



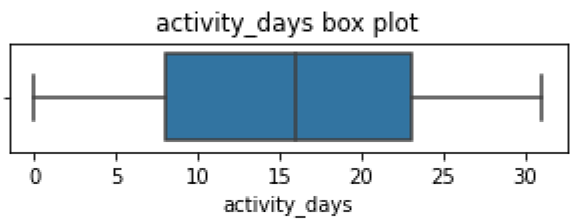
The `duration_minutes_drives` variable has a normalish distribution with a heavily skewed right tail. Around 50% of the users had a driving duration of less than 1,478 minutes (equivalent to about 25 hours), while certain users recorded over 250 hours of driving time throughout the month.

'Activity Days' EDA

`activity_days`

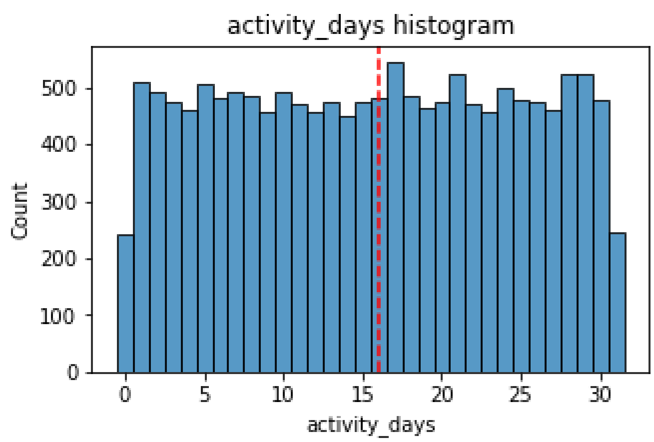
Number of days the user opens the app during the month

In [21]:
```
# Box plot
plt.figure(figsize=(5,1))
sns.boxplot(x=df['activity_days'], fliersize=1)
plt.title('activity_days box plot');
```



In [22]:
```
# Histogram
histogrammer('activity_days', median_text=False, discrete=True)
```

```
Median: 16.0
```



In the past month, users had a median of 16 app openings. The box plot displays a distribution that is centered. The histogram indicates a relatively uniform pattern with approximately 500 individuals opening the app on each day count. However, there are approximately 250 users who did not open the app at all, while another 250 users opened it every day throughout the month.

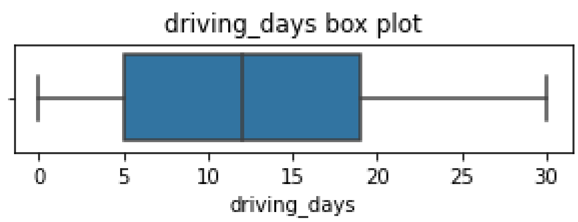
This distribution is of interest because it does not align with the distribution of `sessions`, which one might assume would be closely related to `activity_days`.

'Driving Days' EDA

`driving_days`

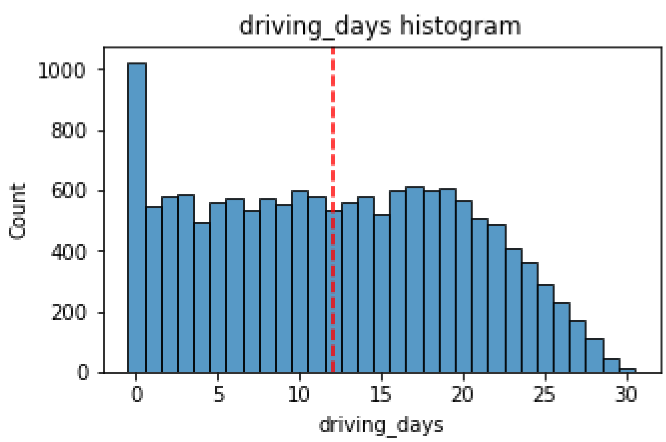
Number of days the user drives (at least 1 km) during the month

In [23]:
```
# Box plot
plt.figure(figsize=(5,1))
sns.boxplot(x=df['driving_days'], fliersize=1)
plt.title('driving_days box plot');
```



In [24]:
```
# Histogram
histogrammer('driving_days', median_text=False, discrete=True)
```

```
Median: 12.0
```



The frequency of users driving each month shows a relatively uniform pattern, closely aligned with the number of days they accessed the app within the same period. However, it's worth noting that the distribution of `driving_days` skews towards lower values.

Interestingly, there were nearly twice as many users (~1,000 versus ~550) who didn't engage in any driving activity throughout the month. This is interesting when considering the information provided about `activity_days`.

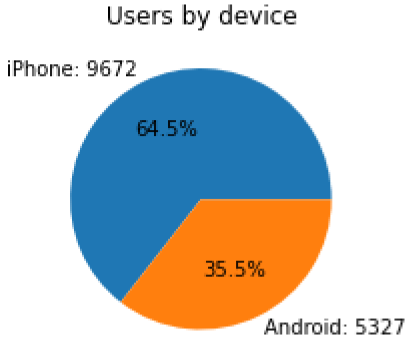
'Device' EDA

`device`

The type of device a user starts a session with

In [25]:
```
# Pie chart
fig = plt.figure(figsize=(3,3))
data=df['device'].value_counts()
plt.pie(data,
```

```
        labels=[f'{data.index[0]}: {data.values[0]}',
                f'{data.index[1]}: {data.values[1]}'],
        autopct='%1.1f%%'
        )
plt.title('Users by device');
```



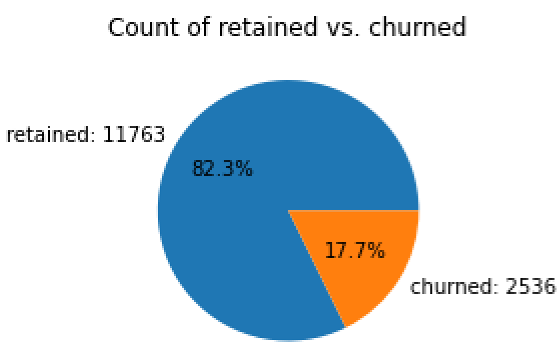
There are almost twice as many iPhone users as Android users.

'Label' EDA

`label`

Binary target variable ("retained" vs "churned") for if a user has churned anytime during the course of the month

In [26]:
```
# Pie chart
fig = plt.figure(figsize=(3,3))
data=df['label'].value_counts()
plt.pie(data,
        labels=[f'{data.index[0]}: {data.values[0]}',
                f'{data.index[1]}: {data.values[1]}'],
        autopct='%1.1f%%'
        )
plt.title('Count of retained vs. churned');
```



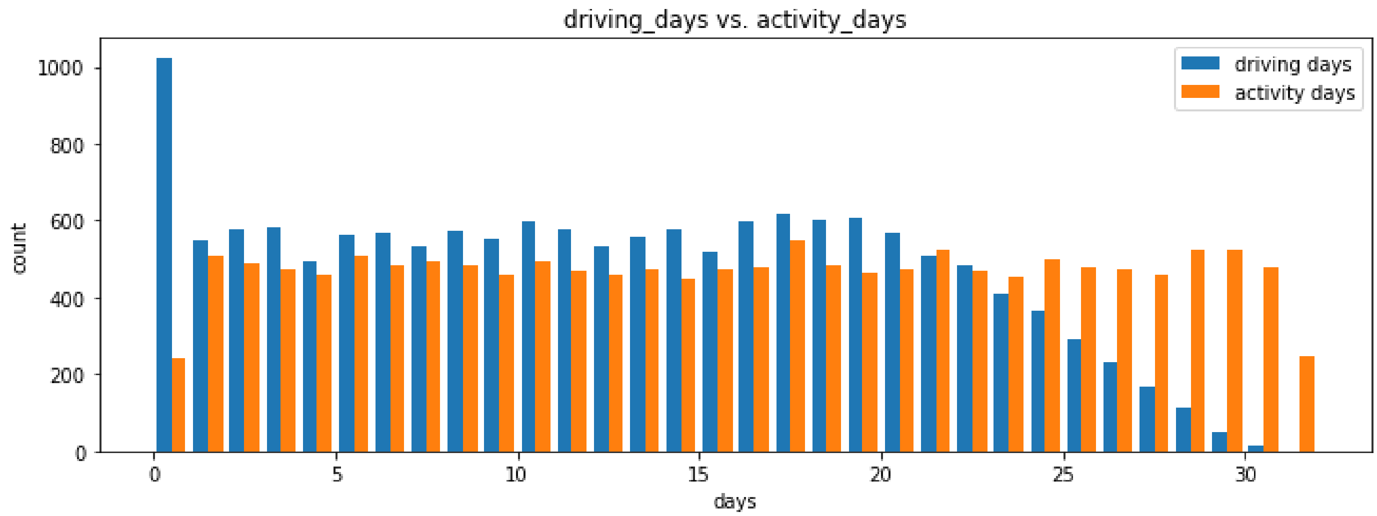
Most of the users were retained. Less than 18% of the users churned.

Driving Days vs Activity Days EDA

`driving days` vs. `activity days`

In [27]:
```
# Histogram
plt.figure(figsize=(12,4))
label=['driving days', 'activity days']
plt.hist([df['driving_days'], df['activity_days']],
         bins=range(0,33),
```

```
             label=label)
plt.xlabel('days')
plt.ylabel('count')
plt.legend()
plt.title('driving_days vs. activity_days');
```

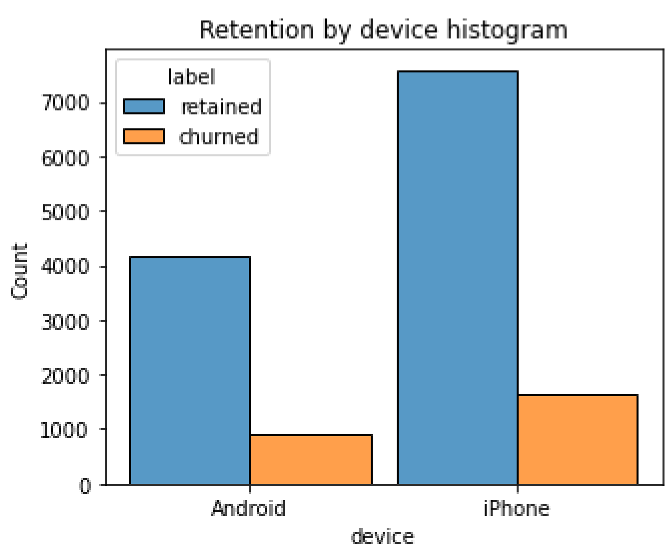


This is interesting. Initially, more users had an increase in `driving_days` compared to `activity_days`. They two stayed fairly consistent through until around day 21. Then, `driving_days` steadily declined, while `activity_days` remained near its previous levels. This would suggest that though users weren't driving as much, they were still opening and using the app.

Retention by device EDA

`Device` : iPhone vs Android

In [30]:
```
# Histogram
plt.figure(figsize=(5,4))
sns.histplot(data=df,
             x='device',
             hue='label',
             multiple='dodge',
             shrink=0.9
             )
plt.title('Retention by device histogram');
```



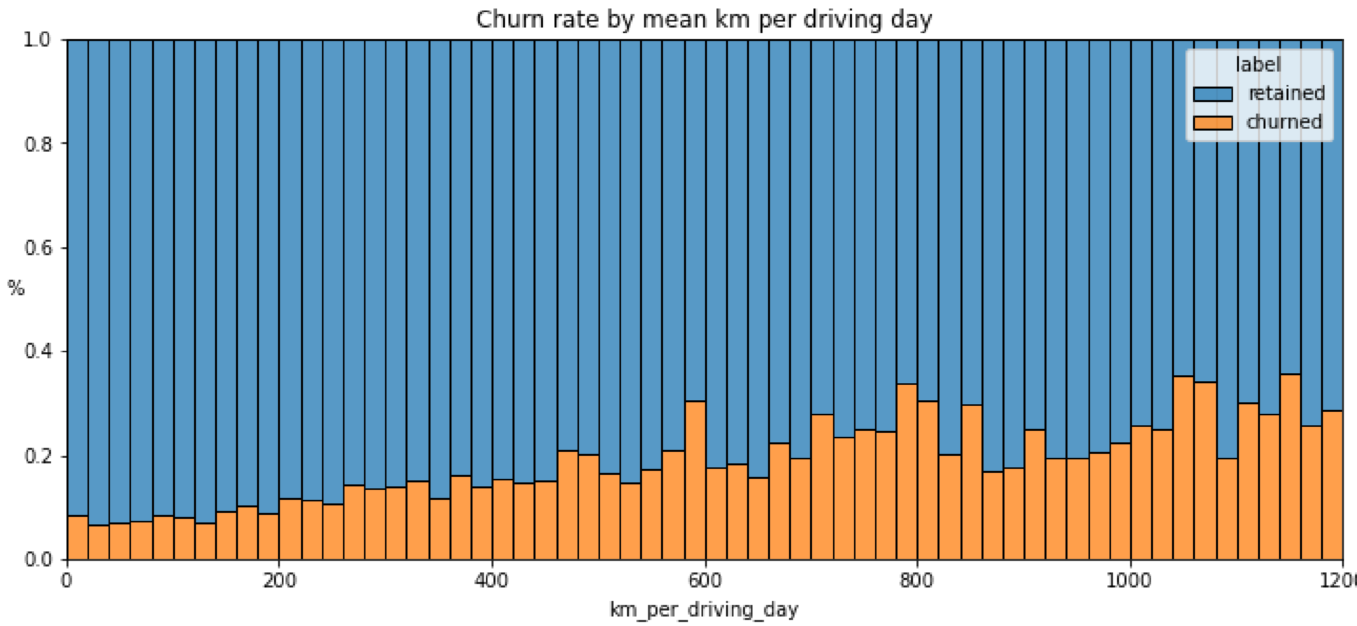
The ratio of users who churned to those who were retained remains consistent across both Android and iPhone devices. It is worth noting that iPhone users had higher numbers of churn and retention, thought that is likely due to the popularity of the iPhone.

Retention by kilometers driven per driving day EDA

`km_per_driving_day`

In [ ]:
```
# 1. Create `km_per_driving_day` column
df['km_per_driving_day'] = df['driven_km_drives'] / df['driving_days']
```

In [32]:
```
# Histogram
plt.figure(figsize=(12,5))
sns.histplot(data=df,
             x='km_per_driving_day',
             bins=range(0,1201,20),
             hue='label',
             multiple='fill')
plt.ylabel('%', rotation=0)
plt.title('Churn rate by mean km per driving day');
```

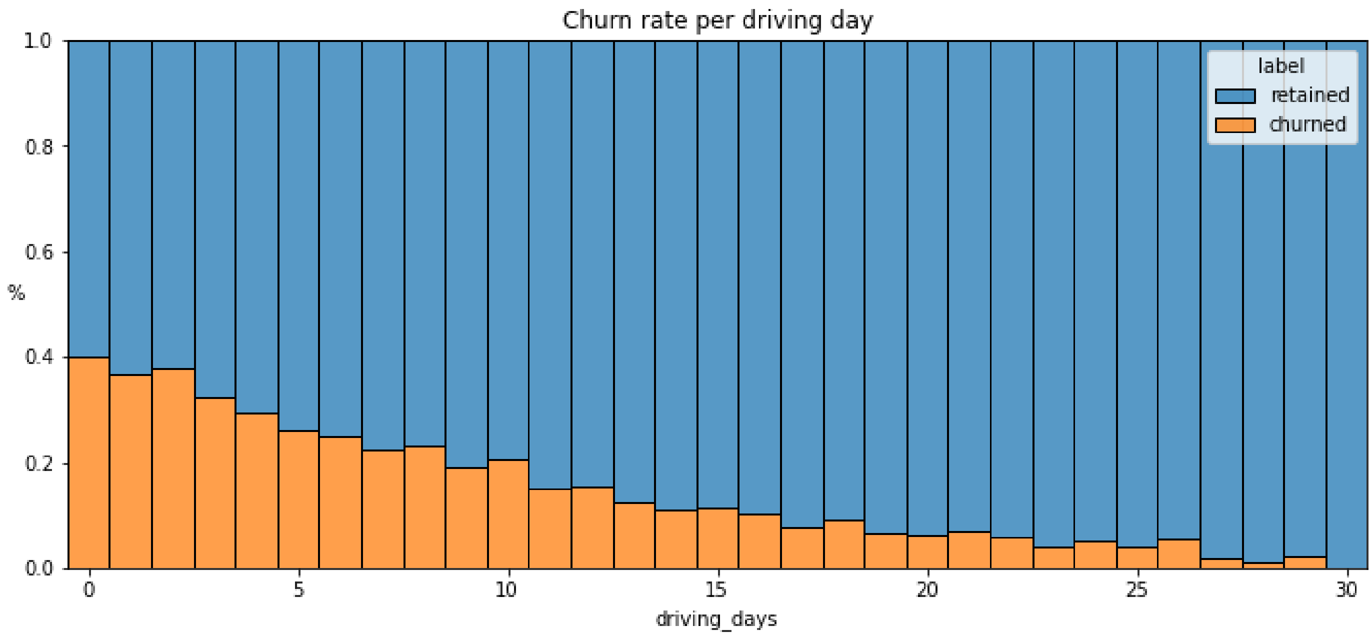


As the average daily distance driven increases, the churn rate also tends to rise. It would be valuable to delve deeper into the reasons why users who cover longer distances choose to discontinue using the app.

Churn rate per number of driving days EDA

`driving days`

In [33]:
```
# Histogram
plt.figure(figsize=(12,5))
sns.histplot(data=df,
             x='driving_days',
             bins=range(1,32),
             hue='label',
             multiple='fill',
             discrete=True)
plt.ylabel('%', rotation=0)
plt.title('Churn rate per driving day');
```



The likelihood of churn decreased as the frequency of app usage increased. Among users who did not use the app at all in the last month, 40% churned, whereas none of the users who used the app for 30 days experienced churn.

Proportion of sessions that occurred in the last month EDA

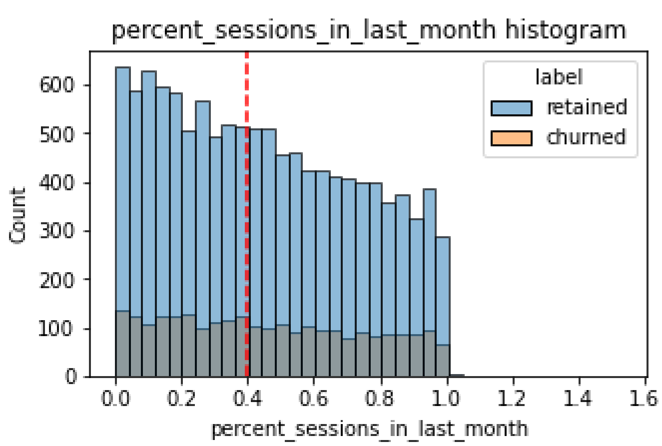
In [34]:
```
df['percent_sessions_in_last_month'] = df['sessions'] / df['total_sessions']
```

In [35]:
```
df['percent_sessions_in_last_month'].median()
```

Out[35]:
```
0.42309702992763176
```

In [36]:
```
# Histogram
histogrammer('percent_sessions_in_last_month',
             hue=df['label'],
             multiple='layer',
             median_text=False)
```

```
Median: 0.4
```

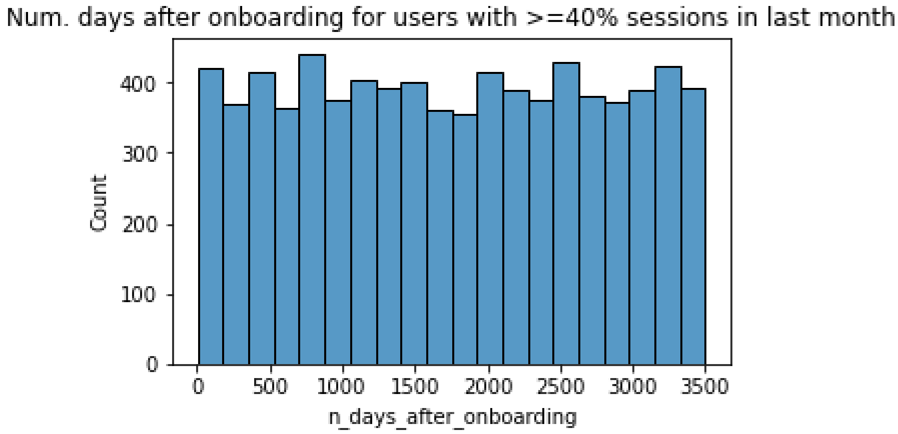


In [37]:
```
df['n_days_after_onboarding'].median()
```

Out[37]:
```
1741.0
```

Around half of the users included in the dataset had 40% or more of their sessions concentrated solely in the last month. Despite this, the median time elapsed since their initial onboarding is 4.77 years.

In [38]:
```
# Histogram
data = df.loc[df['percent_sessions_in_last_month']>=0.4]
plt.figure(figsize=(5,3))
sns.histplot(x=data['n_days_after_onboarding'])
plt.title('Num. days after onboarding for users with >=40% sessions in last month');
```



The number of days since users onboarded, who have experienced 40% or more of their total sessions within the last month, conforms to a uniform distribution. This is an interesting observation. Why the sudden surge in app usage by these longstanding users during the recent month?

Outliers due to skew

In [39]:
```
def outlier_imputer(column_name, percentile):
    # Calculate threshold
    threshold = df[column_name].quantile(percentile)
    # Impute threshold for values > than threshold
    df.loc[df[column_name] > threshold, column_name] = threshold

    print('{:>25} | percentile: {} | threshold: {}'.format(column_name, percentile, thre
```

In [40]:
```
for column in ['sessions', 'drives', 'total_sessions',
               'driven_km_drives', 'duration_minutes_drives']:
               outlier_imputer(column, 0.95)
```

```
                 sessions | percentile: 0.95 | threshold: 243.0
                   drives | percentile: 0.95 | threshold: 201.0
           total_sessions | percentile: 0.95 | threshold: 454.3632037399997
         driven_km_drives | percentile: 0.95 | threshold: 8889.7942356
  duration_minutes_drives | percentile: 0.95 | threshold: 4668.899348999999
```

In [41]:
```
df.describe()
```

Out[41]:

	ID	sessions	drives	total_sessions	n_days_after_onboarding	total_navigations_fav
count	14999.000000	14999.000000	14999.000000	14999.000000	14999.000000	14999.00000
mean	7499.000000	76.568705	64.058204	184.031320	1749.837789	121.60597
std	4329.982679	67.297958	55.306924	118.600463	1008.513876	148.12154
min	0.000000	0.000000	0.000000	0.220211	4.000000	0.00000
25%	3749.500000	23.000000	20.000000	90.661156	878.000000	9.00000
50%	7499.000000	56.000000	48.000000	159.568115	1741.000000	71.00000
75%	11248.500000	112.000000	93.000000	254.192341	2623.500000	178.00000
max	14998.000000	243.000000	201.000000	454.363204	3500.000000	1236.00000

Conclusion

Types of distributions noticed in the variables:

- The majority of variables displayed either a strong right-skewness or a uniform distribution. In the case of right-skewed distributions, this indicates that a significant portion of users had values concentrated towards the lower end of the variable's range. Conversely, for variables exhibiting a uniform distribution, users had an approximately equal likelihood of possessing values across the entire range of that variable.

Indications the data may be erroneous or problematic:

- The majority of the data exhibited no issues, and there was no clear indication that any particular variable was entirely erroneous. However, a few variables contained highly unlikely or potentially impossible outlier values, such as driven_km_drives. Additionally, certain monthly variables, such as activity_days and driving_days, raise concerns as they possess conflicting maximum values of 31 and 30, respectively. This discrepancy suggests that data collection might not have been conducted within the same month for both of these variables, warranting further investigation.

Further questions that need to be explored or asked to the Waze team:

- I would like to inquire with the Waze data team to validate whether the monthly variables were collected within the same month, considering the discrepancy in maximum values—some variables indicating 30 days while others reflecting 31 days. Furthermore, I am interested in understanding the underlying reasons behind the sudden surge in app usage by a significant number of long-time users specifically within the last month. It would be valuable to investigate whether any changes occurred during that period that could have triggered such behavioral shifts.

Percentage of users churned and what percentage were retained:

- The churn rate among users was below 18%, while the majority, approximately 82%, were retained.

Factors that correlated with user churn?

- There was a positive correlation between the distance driven per driving day and user churn. In other words, the farther a user drove on each driving day, the higher the likelihood of churn. Conversely, the number of driving days exhibited a negative correlation with churn. Users who had a higher frequency of driving days within the last month were less likely to churn.

Representation of varying tenure lengths in the dataset:

- The data includes users spanning a range of tenures, from brand new to approximately 10 years, and they are fairly evenly represented. This observation is supported by the histogram depicting the distribution of n_days_after_onboarding, which demonstrates a uniform pattern for this variable.

Waze Project

Milestone 4 / 4a - Compute descriptive statistics. Conduct hypothesis testing

Data exploration and hypothesis testing

The purpose of this project is to compute descriptive statistics and conduct a two-sample hypothesis test.

The goal is to apply descriptive statistics and hypothesis testing in Python.

This notebook has four parts:

Part 1: Imports and data loading

Part 2: Data exploration

Part 3: Conduct hypothesis testing

Part 3: Communicate insights

Data exploration and hypothesis testing

"Do drivers who open the application using an iPhone have the same number of drives on average as drivers who use Android devices?"

Task 1. Imports and data loading

In [1]:
```
# Import any relevant packages or libraries
import pandas as pd
from scipy import stats
```

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

Task 2. Data exploration

Using descriptive statistics to conduct exploratory data analysis (EDA).

In [3]:
```
# 1. Create `map_dictionary`
map_dictionary = {'Android': 2, 'iPhone': 1}

# 2. Create new `device_type` column
df['device_type'] = df['device']

# 3. Map the new column to the dictionary
df['device_type'] = df['device_type'].map(map_dictionary)

df['device_type'].head()
```

Out[3]:
```
0    2
```

```
1    1
2    2
3    1
4    2
Name: device_type, dtype: int64
```

Average number of drives for each device type

In [4]:
```
df.groupby('device_type')['drives'].mean()
```

Out[4]:
```
device_type
1    67.859078
2    66.231838
Name: drives, dtype: float64
```

Given the displayed averages, it seems that iPhone device users tend to have a higher average number of drives when interacting with the application. However, it's important to consider that this disparity may be a result of random sampling rather than an actual difference in the number of drives. To determine if the distinction is statistically significant, we can perform a hypothesis test.

Task 3. Hypothesis testing

The goal is to conduct a two-sample t-test.

1. State the null hypothesis and the alternative hypothesis
2. Choose a signficance level
3. Find the p-value
4. Reject or fail to reject the null hypothesis

Note: This is a t-test for two independent samples. This is the appropriate test since the two groups are independent (Android users vs. iPhone users).

Hypotheses:

$H0$: There is no difference in average number of drives between drivers who use iPhone devices and drivers who use Androids.

HA : There is a difference in average number of drives between drivers who use iPhone devices and drivers who use Androids.

Two-sample test with 5% as the significance level with a two-sample t-test.

In [5]:
```
# 1. Isolate the `drives` column for iPhone users.
iPhone = df[df['device_type'] == 1]['drives']

# 2. Isolate the `drives` column for Android users.
Android = df[df['device_type'] == 2]['drives']

# 3. Perform the t-test
stats.ttest_ind(a=iPhone, b=Android, equal_var=False)
```

Out[5]:
```
Ttest_indResult(statistic=1.4635232068852353, pvalue=0.1433519726802059)
```

p Value = 0.143...

As the p-value exceeds the selected significance level of 5%, we fail to reject the null hypothesis. This indicates that there is no statistically significant distinction in the average number of drives between iPhone users and Android users.

Task 4. Insights

The significant business insight is that, on average, drivers who utilize iPhone devices have a comparable number of drives to those using Androids.

One potential subsequent action is to investigate additional factors that influence the variation in the number of drives. Conducting additional hypothesis tests can help gain further insights into user behavior. Temporary alterations in marketing strategies or user interface for the Waze app could yield more data to examine churn patterns.

Waze Project

Milestone 5 / 5a - Regression analysis: Build a regression model. Evaluate the model

Regression modeling

The purpose of this project is to conduct exploratory data analysis (EDA) and build a binomial logistic regression model.

The goal is to build a binomial logistic regression model and evaluate the model's performance.

This notebook has three parts:

Part 1: EDA & Checking Model Assumptions

Part 2: Model Building, Results, and Evaluation

Part 3: Conclusions, Insights, and Recommendations

Imports and data loading

In [1]:
```
# Packages for numerics + dataframes
import pandas as pd
import numpy as np

# Packages for visualization
import matplotlib.pyplot as plt
import seaborn as sns

# Packages for Logistic Regression & Confusion Matrix
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, accuracy_score, precision_score, \
recall_score, f1_score, confusion_matrix, ConfusionMatrixDisplay
from sklearn.linear_model import LogisticRegression
```

In [4]:
```
# Load the dataset by running this cell
df = pd.read_csv('https://raw.githubusercontent.com/adacert/waze/main/Synthetic_Waze_Dat
```

Part 1. Explore data with EDA & Checking model assumptions

In [5]:
```
print(df.shape)

df.info()
```

```
(14999, 13)
<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
```

The label column is missing 700 values

In [6]:
```
df.head()
```

Out[6]:

	ID	label	sessions	drives	total_sessions	n_days_after_onboarding	total_navigations_fav1	total_navigati
0	0	retained	283	226	296.748273	2276	208	
1	1	retained	133	107	326.896596	1225	19	
2	2	retained	114	95	135.522926	2651	0	
3	3	retained	49	40	67.589221	15	322	
4	4	retained	84	68	168.247020	1562	166	

Remove the ID column since we don't need this information.

In [7]:
```
df = df.drop('ID', axis=1)
```

Class balance of the dependent (target) variable, `label`.

In [8]:
```
df['label'].value_counts(normalize=True)
```

Out[8]:
```
retained    0.822645
churned     0.177355
Name: label, dtype: float64
```

In [9]:
```
df.describe()
```

Out[9]:

	sessions	drives	total_sessions	n_days_after_onboarding	total_navigations_fav1	total_naviga
count	14999.000000	14999.000000	14999.000000	14999.000000	14999.000000	14!
mean	80.633776	67.281152	189.964447	1749.837789	121.605974	
std	80.699065	65.913872	136.405128	1008.513876	148.121544	
min	0.000000	0.000000	0.220211	4.000000	0.000000	
25%	23.000000	20.000000	90.661156	878.000000	9.000000	
50%	56.000000	48.000000	159.568115	1741.000000	71.000000	
75%	112.000000	93.000000	254.192341	2623.500000	178.000000	
max	743.000000	596.000000	1216.154633	3500.000000	1236.000000	[illegible]

The following columns all seem to have outliers:

sessions, drives, total_sessions, total_navigations_fav1, total_navigations_fav2, driven_km_drives, duration_minutes_drives

The maximum values of all these columns surpass the 75th percentile by multiple standard deviations, suggesting the presence of potential outliers in these variables.

Create features

In [10]:
```
# 1. Create `km_per_driving_day` column
df['km_per_driving_day'] = df['driven_km_drives'] / df['driving_days']

# 2. Call `describe()` on the new column
df['km_per_driving_day'].describe()
```

Out[10]:
```
count    1.499900e+04
mean              inf
std               NaN
min      3.022063e+00
25%      1.672804e+02
50%      3.231459e+02
75%      7.579257e+02
max               inf
Name: km_per_driving_day, dtype: float64
```

Note that some values are infinite. This is the result of there being values of zero in the `driving_days` column.

In [11]:
```
# 1. Convert infinite values to zero
df.loc[df['km_per_driving_day']==np.inf, 'km_per_driving_day'] = 0

# 2. Confirm that it worked
df['km_per_driving_day'].describe()
```

Out[11]:
```
count    14999.000000
mean       578.963113
std       1030.094384
min          0.000000
25%        136.238895
50%        272.889272
75%        558.686918
max      15420.234110
Name: km_per_driving_day, dtype: float64
```

`professional_driver`

Creates a new, binary feature called `professional_driver` that is a 1 for users who had 100 or more drives **and** drove on 20+ days in the last month.

Note: The objective is to create a new feature that separates professional drivers from other drivers.

In [12]:
```
# Create `professional_driver` column
df['professional_driver'] = np.where((df['drives'] >= 60) & (df['driving_days'] >= 15),
```

In [13]:
```
# 1. Check count of professionals and non-professionals
print(df['professional_driver'].value_counts())

# 2. Check in-class churn rate
df.groupby(['professional_driver'])['label'].value_counts(normalize=True)
```

```
0    12405
```

```
1     2594
Name: professional_driver, dtype: int64
```

Out[13]:

```
professional_driver  label   
0                    retained    0.801202
                     churned     0.198798
1                    retained    0.924437
                     churned     0.075563
Name: label, dtype: float64
```

The churn rate among professional drivers stands at 7.6%, whereas non-professionals experience a churn rate of 19.9%. This observation appears to contribute a valuable predictive signal to the model.

Preparing variables

In [14]:

```python
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 14999 entries, 0 to 14998
Data columns (total 14 columns):
 #   Column                   Non-Null Count  Dtype  
---  ------                   --------------  -----  
 0   label                    14299 non-null  object 
 1   sessions                 14999 non-null  int64  
 2   drives                   14999 non-null  int64  
 3   total_sessions           14999 non-null  float64
 4   n_days_after_onboarding  14999 non-null  int64  
 5   total_navigations_fav1   14999 non-null  int64  
 6   total_navigations_fav2   14999 non-null  int64  
 7   driven_km_drives         14999 non-null  float64
 8   duration_minutes_drives  14999 non-null  float64
 9   activity_days            14999 non-null  int64  
 10  driving_days             14999 non-null  int64  
 11  device                   14999 non-null  object 
 12  km_per_driving_day       14999 non-null  float64
 13  professional_driver      14999 non-null  int64  
dtypes: float64(4), int64(8), object(2)
memory usage: 1.6+ MB
```

In [15]:

```python
# Drop rows with missing data in `label` column
df = df.dropna(subset=['label'])
```

Impute outliers

Calculate the **95th percentile** of each column and change to this value any value in the column that exceeds it.

In [16]:

```python
# Impute outliers
for column in ['sessions', 'drives', 'total_sessions', 'total_navigations_fav1',
               'total_navigations_fav2', 'driven_km_drives', 'duration_minutes_drives']:
    threshold = df[column].quantile(0.95)
    df.loc[df[column] > threshold, column] = threshold
```

In [17]:

```python
df.describe()
```

Out[17]:

	sessions	drives	total_sessions	n_days_after_onboarding	total_navigations_fav1	total_naviga
count	14299.000000	14299.000000	14299.000000	14299.000000	14299.000000	14:
mean	76.539688	63.964683	183.717304	1751.822505	114.562767	
std	67.243178	55.127927	118.720520	1008.663834	124.378550	
min	0.000000	0.000000	0.220211	4.000000	0.000000	

25%	23.000000	20.000000	90.457733	878.500000	10.000000
50%	56.000000	48.000000	158.718571	1749.000000	71.000000
75%	111.000000	93.000000	253.540450	2627.500000	178.000000
max	243.000000	200.000000	455.439492	3500.000000	422.000000

Encode categorical variables

In [18]:

```
# Create binary `label2` column
df['label2'] = np.where(df['label']=='churned', 1, 0)
df[['label', 'label2']].tail()
```

Out[18]:

	label	label2
14994	retained	0
14995	retained	0
14996	retained	0
14997	churned	1
14998	retained	0

Checking assumptions

The following are the assumptions for this logistic regression:

- Independent observations
- No extreme outliers
- Little to no multicollinearity among X predictors
- Linear relationship between X and the **logit** of y

Collinearity

In [20]:

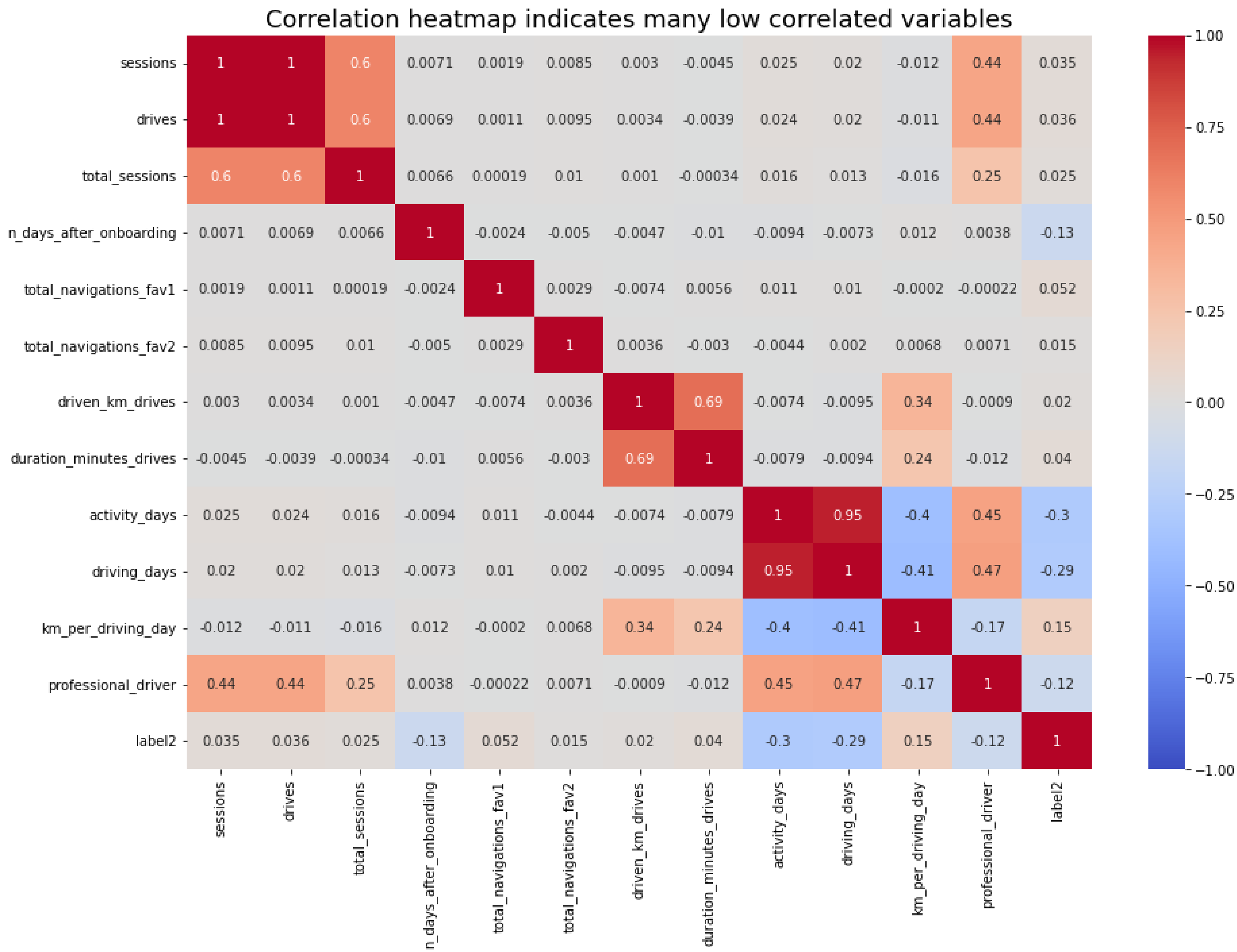
```
# Generate a correlation matrix
df.corr(method='pearson')
```

Out[20]:

	sessions	drives	total_sessions	n_days_after_onboarding	total_navigations_fav1
sessions	1.000000	0.996942	0.597189	0.007101	0.001858
drives	0.996942	1.000000	0.595285	0.006940	0.001058
total_sessions	0.597189	0.595285	1.000000	0.006596	0.000187
n_days_after_onboarding	0.007101	0.006940	0.006596	1.000000	-0.002450
total_navigations_fav1	0.001858	0.001058	0.000187	-0.002450	1.000000
total_navigations_fav2	0.008536	0.009505	0.010371	-0.004968	0.002866
driven_km_drives	0.002996	0.003445	0.001016	-0.004652	-0.007368
duration_minutes_drives	-0.004545	-0.003889	-0.000338	-0.010167	0.005646
activity_days	0.025113	0.024357	0.015755	-0.009418	0.010902
driving_days	0.020294	0.019608	0.012953	-0.007321	0.010419
km_per_driving_day	-0.011569	-0.010989	-0.016167	0.011764	-0.000197

professional_driver	0.443654	0.444425	0.254433	0.003770	-0.000224
label2	0.034911	0.035865	0.024568	-0.129263	0.052322

In [22]:
```
# Plot correlation heatmap
plt.figure(figsize=(15,10))
sns.heatmap(df.corr(method='pearson'), vmin=-1, vmax=1, annot=True, cmap='coolwarm')
plt.title('Correlation heatmap indicates many low correlated variables',
          fontsize=18)
plt.show();
```



Variables that are multicollinear with each other?

- sessions and drives: 1.0
- driving_days and activity_days: 0.95

Create dummies

Creates a new, binary column called `device2` that encodes user devices as follows:

- `Android` -> `0`
- `iPhone` -> `1`

In [23]:
```
# Create new `device2` variable
df['device2'] = np.where(df['device']=='Android', 0, 1)
df[['device', 'device2']].tail()
```

Out[23]:

	device	device2
14994	iPhone	1
14995	Android	0
14996	iPhone	1
14997	iPhone	1
14998	iPhone	1

Part 2. Model building, Results, and Evaluation

Assign predictor variables and target

In [24]:
```
# Isolate predictor variables
X = df.drop(columns = ['label', 'label2', 'device', 'sessions', 'driving_days'])
```

In [25]:
```
# Isolate target variable
y = df['label2']
```

Split the data

In [26]:
```
# Perform the train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y, stratify=y, random_state=42)
```

In [27]:
```
# Use .head()
X_train.head()
```

Out[27]:

	drives	total_sessions	n_days_after_onboarding	total_navigations_fav1	total_navigations_fav2	driven_kı
152	108	186.192746	3116	243	124	889
11899	2	3.487590	794	114	18	328
10937	139	347.106403	331	4	7	740
669	108	455.439492	2320	11	4	656
8406	10	89.475821	2478	135	0	127

Instantiate a logistic regression model

Add the argument `penalty = None`.

We add `penalty = None` since the predictors are unscaled.

In [30]:
```
model = LogisticRegression(penalty='none', max_iter=400)

model.fit(X_train, y_train)
```

Out[30]:
```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, l1_ratio=None, max_iter=400,
                   multi_class='auto', n_jobs=None, penalty='none',
                   random_state=None, solver='lbfgs', tol=0.0001, verbose=0,
                   warm_start=False)
```

In [31]:
```
pd.Series(model.coef_[0], index=X.columns)
```

Out[31]:
```
drives                    0.001913
total_sessions            0.000327
```

```
n_days_after_onboarding   -0.000406
total_navigations_fav1     0.001232
total_navigations_fav2     0.000931
driven_km_drives          -0.000015
duration_minutes_drives    0.000109
activity_days             -0.106032
km_per_driving_day         0.000018
professional_driver       -0.001529
device2                   -0.001041
dtype: float64
```

In [32]:
```python
model.intercept_
```

Out[32]:
```
array([-0.00170675])
```

Check final assumption

Verifies the linear relationship between X and the estimated log odds (known as logits) by making a regplot.

In [33]:
```python
# Get the predicted probabilities of the training data
training_probabilities = model.predict_proba(X_train)
training_probabilities
```

Out[33]:
```
array([[0.93963483, 0.06036517],
       [0.61967304, 0.38032696],
       [0.76463181, 0.23536819],
       ...,
       [0.91909641, 0.08090359],
       [0.85092112, 0.14907888],
       [0.93516293, 0.06483707]])
```

Below creates a dataframe called `logit_data` that is a copy of `df`.

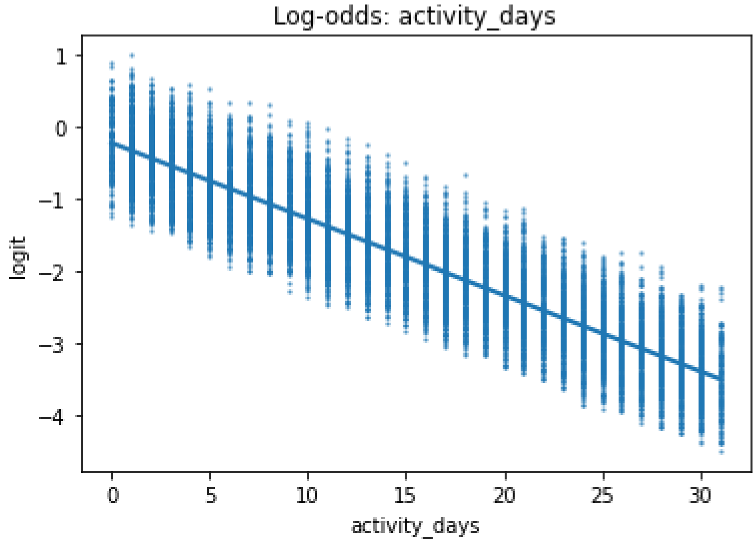
Below also creates a new column called `logit` in the `logit_data` dataframe. The data in this column should represent the logit for each user.

In [34]:
```python
# 1. Copy the `X_train` dataframe and assign to `logit_data`
logit_data = X_train.copy()

# 2. Create a new `logit` column in the `logit_data` df
logit_data['logit'] = [np.log(prob[1] / prob[0]) for prob in training_probabilities]
```

Below creates a dataframe called `logit_data` that is a copy of `df`.

Below also creates a new column called `logit` in the `logit_data` dataframe. The data in this column should represent the logit for each user.

In [35]:
```python
# Plot regplot of `activity_days` log-odds
sns.regplot(x='activity_days', y='logit', data=logit_data, scatter_kws={'s': 2, 'alpha':
plt.title('Log-odds: activity_days');
```



Results and evaluation

If the logistic assumptions are met, the model results can be appropriately interpreted.

Below we will make predictions on the test data.

In [36]:
```
# Generate predictions on X_test
y_preds = model.predict(X_test)
```

Accuracy of the model

In [37]:
```
# Score the model (accuracy) on the test data
model.score(X_test, y_test)
```

Out[37]:
```
0.8237762237762237
```

Results shown with a confusion matrix

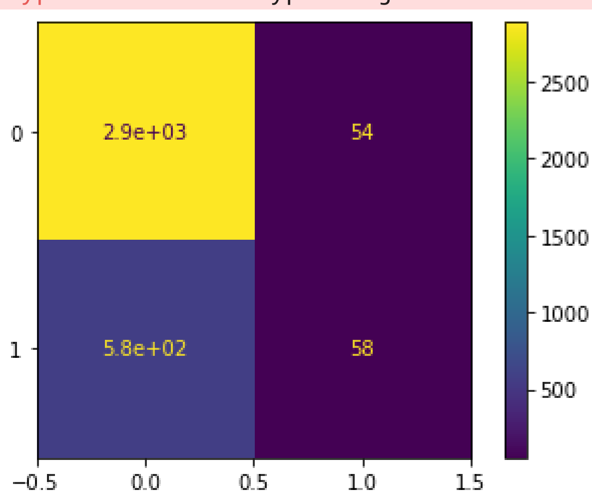
In [53]:
```
cm = confusion_matrix(y_test, y_preds)
```

The below confusion matrix shows an error, but displays correctly.

In [54]:
```
disp = ConfusionMatrixDisplay(confusion_matrix=cm,display_labels=None)
disp.plot()
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-54-5be7a6a26f01> in <module>
      1 disp = ConfusionMatrixDisplay(confusion_matrix=cm,display_labels=None)
----> 2 disp.plot()

/opt/conda/lib/python3.7/site-packages/sklearn/metrics/_plot/confusion_matrix.py in plot
(self, include_values, cmap, xticks_rotation, values_format, ax)
    107                yticklabels=self.display_labels,
    108                ylabel="True label",
--> 109                xlabel="Predicted label")
    110
    111         ax.set_ylim((n_classes - 0.5, -0.5))

/opt/conda/lib/python3.7/site-packages/matplotlib/artist.py in set(self, **kwargs)
   1099             sorted(kwargs.items(), reverse=True,
   1100                    key=lambda x: (self._prop_order.get(x[0], 0), x[0])))
```

```
-> 1101         return self.update(props)
   1102
   1103     def findobj(self, match=None, include_self=True):

/opt/conda/lib/python3.7/site-packages/matplotlib/artist.py in update(self, props)
   1004
   1005         with cbook._setattr_cm(self, eventson=False):
-> 1006             ret = [_update_property(self, k, v) for k, v in props.items()]
   1007
   1008         if len(ret):

/opt/conda/lib/python3.7/site-packages/matplotlib/artist.py in <listcomp>(.0)
   1004
   1005         with cbook._setattr_cm(self, eventson=False):
-> 1006             ret = [_update_property(self, k, v) for k, v in props.items()]
   1007
   1008         if len(ret):

/opt/conda/lib/python3.7/site-packages/matplotlib/artist.py in _update_property(self, k,
 v)
   1001                         raise AttributeError('{!r} object has no property {!r}'
   1002                                              .format(type(self).__name__, k))
-> 1003                 return func(v)
   1004
   1005         with cbook._setattr_cm(self, eventson=False):

/opt/conda/lib/python3.7/site-packages/matplotlib/axes/_base.py in set_yticklabels(self,
 labels, fontdict, minor, **kwargs)
   3774             kwargs.update(fontdict)
   3775         return self.yaxis.set_ticklabels(labels,
-> 3776                                          minor=minor, **kwargs)
   3777
   3778     def xaxis_date(self, tz=None):

/opt/conda/lib/python3.7/site-packages/matplotlib/axis.py in set_ticklabels(self, tickla
bels, minor, *args, **kwargs)
   1714                 "3.1; passing them will raise a TypeError in Matplotlib 3.3.")
   1715         get_labels = []
-> 1716         for t in ticklabels:
   1717             # try calling get_text() to check whether it is Text object
   1718             # if it is Text, get label content

TypeError: 'NoneType' object is not iterable
```



Precision

In [55]:
```
# Calculate precision manually
precision = cm[1,1] / (cm[0, 1] + cm[1, 1])
precision
```

Out[55]: 0.5178571428571429

Recall

In [56]:
```
# Calculate recall manually
recall = cm[1,1] / (cm[1, 0] + cm[1, 1])
recall
```

Out[56]: 0.0914826498422713

Classification Report

In [57]:
```
# Create a classification report
target_labels = ['retained', 'churned']
print(classification_report(y_test, y_preds, target_names=target_labels))
```

```
              precision    recall  f1-score   support

    retained       0.83      0.98      0.90      2941
     churned       0.52      0.09      0.16       634

    accuracy                           0.82      3575
   macro avg       0.68      0.54      0.53      3575
weighted avg       0.78      0.82      0.77      3575
```

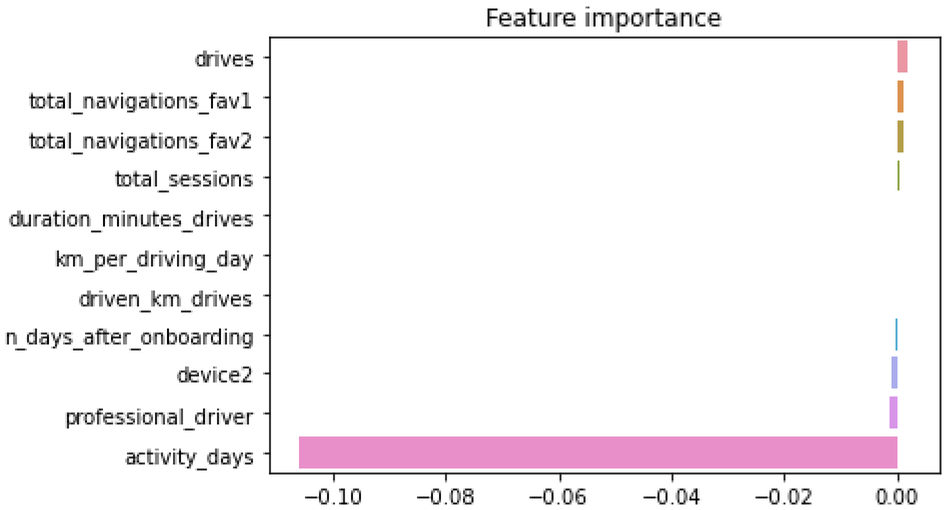
Although the model demonstrates reasonable precision, its recall is extremely low, indicating a high number of false negative predictions. Consequently, it fails to identify and capture users who are likely to churn.

Visual representation of the importance of the model's features

In [58]:
```
# Create a list of (column_name, coefficient) tuples
feature_importance = list(zip(X_train.columns, model.coef_[0]))

# Sort the list by coefficient value
feature_importance = sorted(feature_importance, key=lambda x: x[1], reverse=True)
feature_importance
```

Out[58]:
```
[('drives', 0.001913369447769776),
 ('total_navigations_fav1', 0.001231754741616306),
 ('total_navigations_fav2', 0.0009314786513814626),
 ('total_sessions', 0.00032707088819142904),
 ('duration_minutes_drives', 0.00010909343558951453),
 ('km_per_driving_day', 1.8223094015325207e-05),
 ('driven_km_drives', -1.4860453424647997e-05),
 ('n_days_after_onboarding', -0.00040647763730561445),
 ('device2', -0.0010412175209008018),
 ('professional_driver', -0.0015285041567402024),
 ('activity_days', -0.10603196504385491)]
```

In [59]:
```
# Plot the feature importances
import seaborn as sns
sns.barplot(x=[x[1] for x in feature_importance],
            y=[x[0] for x in feature_importance],
            orient='h')
plt.title('Feature importance');
```



Part 3: Conclusions, Insights, and Recommendations

Variables that most influenced the model's prediction:

- Among all the features in the model, "activity_days" emerged as the most significant one, exhibiting a negative correlation with user churn. This finding is not unexpected since "activity_days" is highly correlated with "driving_days," which was already identified during the exploratory data analysis (EDA) to have a negative correlation with churn.

Variables expected to be stronger predictors than they were:

- During the exploratory data analysis (EDA), it was observed that the user churn rate rose in conjunction with increasing values in "km_per_driving_day." The correlation heatmap in this notebook further confirmed this observation, indicating that this variable exhibited the highest positive correlation with churn among all the predictor variables, surpassing others by a significant margin. Surprisingly, in the model, "km_per_driving_day" ranked as the second-least important variable.

Why might a variable thought to be important not be important in the model?

- In a multiple logistic regression model, the presence of feature interactions can lead to relationships that may appear counterintuitive. This phenomenon represents both a strength and a weakness of predictive models. On one hand, capturing these interactions enhances the predictive capabilities of the model. On the other hand, it complicates the model's interpretability, making it more challenging to explain the underlying relationships.

Is it recommended that Waze use this model?

- The usefulness of the model depends on its intended purpose. If the model is employed to inform critical business decisions, its performance may not be sufficiently strong, particularly evident from its low recall score. However, if the model is primarily utilized to guide further exploratory efforts and provide insights, it can still offer value in that context.

Steps that can be taken to improve this model:

- By leveraging domain knowledge, it is possible to engineer new features aimed at improving predictive signal. In the context of this model, one of the engineered features, namely "professional_driver," emerged as the third-most influential predictor. Additionally, scaling the predictor variables and reconstructing the model using different combinations of predictors can be beneficial in minimizing noise stemming from unpromising features.

Additional features that would be needed to help improve the model:

- It would be beneficial to possess drive-level specifics for individual users, such as drive times and geographic locations. Furthermore, obtaining more detailed information regarding how users engage with the app would likely provide valuable insights. For instance, understanding the frequency at which they report or confirm road hazard alerts. Finally, having knowledge of the monthly count of distinct starting and ending locations inputted by each driver could offer valuable additional information.

Waze Project

Milestone 6 / 6A - Build a machine learning model. Communicate final insights

Build a machine learning model

The purpose of this model is to find factors that drive user churn.

The goal of this model is to predict whether or not a Waze user is retained or churned.

This notebook has four parts:

Part 1: Imports and Data Loading

Part 2: Feature engineering

Part 3: Modeling

Part 4: Insights and Conclusion

Part 1: Imports and data loading

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

# Import packages for data visualization
import matplotlib.pyplot as plt

# This lets us see all of the columns, preventing Juptyer from redacting them.
pd.set_option('display.max_columns', None)

# Import packages for data modeling
from sklearn.model_selection import GridSearchCV, train_test_split
from sklearn.metrics import roc_auc_score, roc_curve, auc
from sklearn.metrics import accuracy_score, precision_score, recall_score,\
f1_score, confusion_matrix, ConfusionMatrixDisplay, RocCurveDisplay, PrecisionRecallDisp

from sklearn.ensemble import RandomForestClassifier
from xgboost import XGBClassifier

# This is the function that helps plot feature importance
from xgboost import plot_importance

# This module lets us save our models once we fit them.
import pickle

# from google.colab import drive
# drive.mount('/content/drive', force_remount=True)
```

In [2]:
```
# Import dataset
df0 = pd.read_csv('waze_dataset.csv')
```

In [3]:
```
# Inspect the first five rows
df0.head()
```

Out[3]:

	ID	label	sessions	drives	total_sessions	n_days_after_onboarding	total_navigations_fav1	total_navigati
0	0	retained	283	226	296.748273	2276	208	
1	1	retained	133	107	326.896596	1225	19	
2	2	retained	114	95	135.522926	2651	0	
3	3	retained	49	40	67.589221	15	322	
4	4	retained	84	68	168.247020	1562	166	

Part 2: Feature engineering

In [4]:
```
# Copy the df0 dataframe
df = df0.copy()
```

In [5]:
```
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
```

`km_per_driving_day`

Creates a feature representing the mean number of kilometers driven on each driving day in the last month for each user.

In [6]:
```
# 1. Create `km_per_driving_day` feature
df['km_per_driving_day'] = df['driven_km_drives'] / df['driving_days']

# 2. Get descriptive stats
df['km_per_driving_day'].describe()
```

Out[6]:
```
count    1.499900e+04
mean              inf
std               NaN
min      3.022063e+00
25%      1.672804e+02
50%      3.231459e+02
75%      7.579257e+02
max               inf
Name: km_per_driving_day, dtype: float64
```

In [7]:
```
# 1. Convert infinite values to zero
df.loc[df['km_per_driving_day']==np.inf, 'km_per_driving_day'] = 0
```

```
# 2. Confirm that it worked
df['km_per_driving_day'].describe()
```

Out[7]:

```
count    14999.000000
mean       578.963113
std       1030.094384
min          0.000000
25%        136.238895
50%        272.889272
75%        558.686918
max      15420.234110
Name: km_per_driving_day, dtype: float64
```

`percent_sessions_in_last_month`

Creates a new column `percent_sessions_in_last_month` that represents the percentage of each user's total sessions that were logged in their last month of use.

In [8]:

```
# 1. Create `percent_sessions_in_last_month` feature
df['percent_sessions_in_last_month'] = df['sessions'] / df['total_sessions']

# 2. Get descriptive stats
df['percent_sessions_in_last_month'].describe()
```

Out[8]:

```
count    14999.000000
mean         0.449255
std          0.286919
min          0.000000
25%          0.196221
50%          0.423097
75%          0.687216
max          1.530637
Name: percent_sessions_in_last_month, dtype: float64
```

`professional_driver`

Creates a new, binary feature called `professional_driver` that is a 1 for users who had 100 or more drives **and** drove on 20+ days in the last month.

In [9]:

```
# Create `professional_driver` feature
df['professional_driver'] = np.where((df['drives'] >= 60) & (df['driving_days'] >= 15),
```

`total_sessions_per_day`

Creates a new column that represents the mean number of sessions per day *since onboarding*.

In [10]:

```
# Create `total_sessions_per_day` feature
df['total_sessions_per_day'] = df['total_sessions'] / df['n_days_after_onboarding']
```

In [11]:

```
# Get descriptive stats
df['total_sessions_per_day'].describe()
```

Out[11]:

```
count    14999.000000
mean         0.338698
std          1.314333
min          0.000298
25%          0.051037
50%          0.100775
75%          0.216269
```

```
max          39.763874
Name: total_sessions_per_day, dtype: float64
```

`km_per_hour`

Creates a column representing the mean kilometers per hour driven in the last month.

In [12]:
```
# Create `km_per_hour` feature
df['km_per_hour'] = df['driven_km_drives'] / df['duration_minutes_drives'] / 60
df['km_per_hour'].describe()
```

Out[12]:
```
count    14999.000000
mean         0.052887
std          0.092965
min          0.020004
25%          0.025196
50%          0.033995
75%          0.053647
max          6.567478
Name: km_per_hour, dtype: float64
```

`km_per_drive`

Creates a column representing the mean number of kilometers per drive made in the last month for each user.

In [13]:
```
# Create `km_per_drive` feature
df['km_per_drive'] = df['driven_km_drives'] / df['drives']
df['km_per_drive'].describe()
```

Out[13]:
```
count    1.499900e+04
mean              inf
std               NaN
min      1.008775e+00
25%      3.323065e+01
50%      7.488006e+01
75%      1.854667e+02
max               inf
Name: km_per_drive, dtype: float64
```

In [14]:
```
# 1. Convert infinite values to zero
df.loc[df['km_per_drive']==np.inf, 'km_per_drive'] = 0

# 2. Confirm that it worked
df['km_per_drive'].describe()
```

Out[14]:
```
count    14999.000000
mean       232.817946
std        620.622351
min          0.000000
25%         32.424301
50%         72.854343
75%        179.347527
max      15777.426560
Name: km_per_drive, dtype: float64
```

`percent_of_sessions_to_favorite`

Creates a new column that represents the percentage of total sessions that were used to navigate to one of the users' favorite places.

This serves as a substitute indicator for the percentage of all drives that are made to a preferred location.

As the dataset lacks information on the total number of drives since the initial use, the total number of sessions can be considered a reasonable estimate.

Individuals who have a higher proportion of drives to non-preferred destinations in relation to their total trips may exhibit a lower likelihood of churn, as they are driving to unfamiliar places more frequently.

In [15]:
```
# Create `percent_of_sessions_to_favorite` feature
df['percent_of_drives_to_favorite'] = (
    df['total_navigations_fav1'] + df['total_navigations_fav2']) / df['total_sessions']

# Get descriptive stats
df['percent_of_drives_to_favorite'].describe()
```

Out[15]:
```
count    14999.000000
mean         1.665439
std          8.865666
min          0.000000
25%          0.203471
50%          0.649818
75%          1.638526
max        777.563629
Name: percent_of_drives_to_favorite, dtype: float64
```

Drop missing values

In [16]:
```
# Drop rows with missing values
df = df.dropna(subset=['label'])
```

Outliers

Tree-based models are resilient to outliers, so there is no need to make any imputations.

Variable encoding

Dummying features

Creates a new, binary column called `device2` that encodes user devices as follows:

- `Android` -> `0`
- `iPhone` -> `1`

In [17]:
```
# Create new `device2` variable
df['device2'] = np.where(df['device']=='Android', 0, 1)
df[['device', 'device2']].tail()
```

Out[17]:

	device	device2
14994	iPhone	1
14995	Android	0
14996	iPhone	1
14997	iPhone	1
14998	iPhone	1

Target encoding

Changes the data type of the `label` column to be binary. This change is needed to train the models.

Assigns a `0` for all `retained` users.

Assigns a `1` for all `churned` users.

Variables saved as `label2` so as not to overwrite the original `label` variable.

In [18]:
```
# Create binary `label2` column
df['label2'] = np.where(df['label']=='churned', 1, 0)
df[['label', 'label2']].tail()
```

Out[18]:

	label	label2
14994	retained	0
14995	retained	0
14996	retained	0
14997	churned	1
14998	retained	0

Feature selection

The only feature that can be cut is `ID`, since it doesn't contain any information relevant to churn.

`device` won't be used simply because it's a copy of `device2`.

Drops `ID` from the `df` dataframe.

In [19]:
```
# Drop `ID` column
df = df.drop(['ID'], axis=1)
```

Evaluation metric

Examines the class balance of the target variable.

In [20]:
```
# Get class balance of 'label' col
df['label'].value_counts(normalize=True)
```

Out[20]:
```
label
retained    0.822645
churned     0.177355
Name: proportion, dtype: float64
```

Around 18% of the users included in this dataset experienced churn. Although the dataset is imbalanced, it can be still modeled without requiring any class rebalancing.

We will select the model based on recall.

Modeling workflow and model selection process

The final modeling dataset contains 14,299 samples. This is towards the lower end of what might be considered sufficient to conduct a robust model selection process, but still doable.

1. Split the data into train/validation/test sets (60/20/20)

2. Fit models and tune hyperparameters on the training set
3. Perform final model selection on the validation set
4. Assess the champion model's performance on the test set

Split the data

1. Defines a variable `X` that isolates the features.

2. Defines a variable `y` that isolates the target variable (`label2`).

3. Splits the data 80/20 into an interim training set and a test set.

4. Splits the interim training set 75/25 into a training set and a validation set, yielding a final ratio of 60/20/20 for training/validation/test sets.

In [21]:
```
# 1. Isolate X variables
X = df.drop(columns=['label', 'label2', 'device'])

# 2. Isolate y variable
y = df['label2']

# 3. Split into train and test sets
X_tr, X_test, y_tr, y_test = train_test_split(X, y, stratify=y,
                                              test_size=0.2, random_state=42)

# 4. Split into train and validate sets
X_train, X_val, y_train, y_val = train_test_split(X_tr, y_tr, stratify=y_tr,
                                                  test_size=0.25, random_state=42)
```

In [22]:
```
for x in [X_train, X_val, X_test]:
    print(len(x))
```

```
8579
2860
2860
```

This is consistent with what was expected.

Part 3: Modeling

Random forest

Begin with using `GridSearchCV` to tune a random forest model.

1. Instantiates the random forest classifier `rf` and sets the random state.

2. Creates a dictionary `cv_params` of any of the following hyperparameters and their corresponding values to tune.

   - `max_depth`
   - `max_features`
   - `max_samples`
   - `min_samples_leaf`
   - `min_samples_split`
   - `n_estimators`

3. Defines a dictionary `scoring` of scoring metrics for GridSearch to capture (precision, recall, F1 score, and accuracy).

4. Instantiates the `GridSearchCV` object `rf_cv`. Passes to it as arguments:

   - estimator= `rf`
   - param_grid= `cv_params`
   - scoring= `scoring`
   - cv: define the number of cross-validation folds you want (`cv=_`)
   - refit: indicate which evaluation metric you want to use to select the model (`refit=_`)

   `refit` should be set to `'recall'`.

In [23]:
```
# 1. Instantiate the random forest classifier
rf = RandomForestClassifier(random_state=42)

# 2. Create a dictionary of hyperparameters to tune
cv_params = {'max_depth': [None],
             'max_features': [1.0],
             'max_samples': [1.0],
             'min_samples_leaf': [2],
             'min_samples_split': [2],
             'n_estimators': [300],
             }

# 3. Define a dictionary of scoring metrics to capture
scoring = {'accuracy', 'precision', 'recall', 'f1'}

# 4. Instantiate the GridSearchCV object
rf_cv = GridSearchCV(rf, cv_params, scoring=scoring, cv=4, refit='recall')
```

In [24]:
```
%%time
rf_cv.fit(X_train, y_train)
```

```
CPU times: user 1min 56s, sys: 27.3 ms, total: 1min 56s
Wall time: 1min 56s
```

Out[24]:
```
GridSearchCV
estimator: RandomForestClassifier
RandomForestClassifier
```

The best average score across all the validation folds.

In [25]:
```
# Examine best score
rf_cv.best_score_
```

Out[25]:
```
0.12678201409034398
```

The best combination of hyperparameters.

In [26]:
```
# Examine best hyperparameter combo
rf_cv.best_params_
```

Out[26]:
```
{'max_depth': None,
 'max_features': 1.0,
 'max_samples': 1.0,
 'min_samples_leaf': 2,
```

```
 'min_samples_split': 2,
 'n_estimators': 300}
```

Creates a `make_results()` function to output all of the scores of the model.

In [27]:
```
def make_results(model_name:str, model_object, metric:str):
    '''
    Arguments:
        model_name (string): what you want the model to be called in the output table
        model_object: a fit GridSearchCV object
        metric (string): precision, recall, f1, or accuracy

    Returns a pandas df with the F1, recall, precision, and accuracy scores
    for the model with the best mean 'metric' score across all validation folds.
    '''

    # Create dictionary that maps input metric to actual metric name in GridSearchCV
    metric_dict = {'precision': 'mean_test_precision',
                   'recall': 'mean_test_recall',
                   'f1': 'mean_test_f1',
                   'accuracy': 'mean_test_accuracy',
                   }

    # Get all the results from the CV and put them in a df
    cv_results = pd.DataFrame(model_object.cv_results_)

    # Isolate the row of the df with the max(metric) score
    best_estimator_results = cv_results.iloc[cv_results[metric_dict[metric]].idxmax(), :

    # Extract accuracy, precision, recall, and f1 score from that row
    f1 = best_estimator_results.mean_test_f1
    recall = best_estimator_results.mean_test_recall
    precision = best_estimator_results.mean_test_precision
    accuracy = best_estimator_results.mean_test_accuracy

    # Create table of results
    table = pd.DataFrame({'model': [model_name],
                          'precision': [precision],
                          'recall': [recall],
                          'F1': [f1],
                          'accuracy': [accuracy],
                          },
                         )

    return table
```

Passes the `GridSearch` object to the `make_results()` function.

In [28]:
```
results = make_results('RF cv', rf_cv, 'recall')
results
```

Out[28]:

	model	precision	recall	F1	accuracy
0	RF cv	0.458198	0.126782	0.198534	0.818626

Apart from the accuracy, the scores are not particularly impressive. It is worth noting that with the previously constructed logistic regression model, the recall was approximately 0.09. This indicates that the current model exhibits a 33% improvement in recall while maintaining a similar level of accuracy, despite being trained on a smaller dataset.

We could fine-tune the hyperparameters in an attempt to achieve a higher score. There is a possibility of making slight improvements to the model.

XGBoost

1. Instantiates the XGBoost classifier `xgb` and set `objective='binary:logistic'`. Also sets the random state.

2. Creates a dictionary `cv_params` of the following hyperparameters and their corresponding values to tune:

   - `max_depth`
   - `min_child_weight`
   - `learning_rate`
   - `n_estimators`
3. Defines a dictionary `scoring` of scoring metrics for grid search to capture (precision, recall, F1 score, and accuracy).

4. Instantiates the `GridSearchCV` object `xgb_cv`. Passes to it as arguments:

   - estimator= `xgb`
   - param_grid= `cv_params`
   - scoring= `scoring`
   - cv: define the number of cross-validation folds you want (`cv=_`)
   - refit: indicate which evaluation metric you want to use to select the model (`refit='recall'`)

In [29]:
```
# 1. Instantiate the XGBoost classifier
xgb = XGBClassifier(objective='binary:logistic', random_state=42)

# 2. Create a dictionary of hyperparameters to tune
cv_params = {'max_depth': [6, 12],
             'min_child_weight': [3, 5],
             'learning_rate': [0.01, 0.1],
             'n_estimators': [300]
             }

# 3. Define a dictionary of scoring metrics to capture
scoring = {'accuracy', 'precision', 'recall', 'f1'}

# 4. Instantiate the GridSearchCV object
xgb_cv = GridSearchCV(xgb, cv_params, scoring=scoring, cv=4, refit='recall')
```

Fits the model to the `X_train` and `y_train` data.

In [30]:
```
%%time
xgb_cv.fit(X_train, y_train)
```

```
CPU times: user 4min 14s, sys: 1.9 s, total: 4min 16s
Wall time: 2min 10s
```

Out[30]:
```
▸ GridSearchCV
▸ estimator: XGBClassifier
  ▸ XGBClassifier
```

The best score from this model.

In [31]:
```
# Examine best score
```

```
xgb_cv.best_score_
```

Out[31]:
```
0.1734683657963807
```

The best parameters.

In [32]:
```
# Examine best parameters
xgb_cv.best_params_
```

Out[32]:
```
{'learning_rate': 0.1,
 'max_depth': 12,
 'min_child_weight': 3,
 'n_estimators': 300}
```

Uses the `make_results()` function to output all of the scores of the model.

In [33]:
```
# Call 'make_results()' on the GridSearch object
xgb_cv_results = make_results('XGB cv', xgb_cv, 'recall')
results = pd.concat([results, xgb_cv_results], axis=0)
results
```

Out[33]:

	model	precision	recall	F1	accuracy
0	RF cv	0.458198	0.126782	0.198534	0.818626
0	XGB cv	0.442586	0.173468	0.248972	0.814780

This model not only outperformed the random forest model in terms of data fitting, but it also achieved a recall score that is nearly twice as high as the recall score obtained by the logistic regression model. It also demonstrates an improvement of almost 50% in recall compared to the random forest model, while maintaining similar levels of accuracy and precision.

Model selection

Random forest

In [34]:
```
# Use random forest model to predict on validation data
rf_val_preds = rf_cv.best_estimator_.predict(X_val)
```

Uses the `get_test_scores()` function to generate a table of scores from the predictions on the validation data.

In [35]:
```
def get_test_scores(model_name:str, preds, y_test_data):
    '''
    Generate a table of test scores.

    In:
        model_name (string): Your choice: how the model will be named in the output tabl
        preds: numpy array of test predictions
        y_test_data: numpy array of y_test data

    Out:
        table: a pandas df of precision, recall, f1, and accuracy scores for your model
    '''
    accuracy = accuracy_score(y_test_data, preds)
    precision = precision_score(y_test_data, preds)
    recall = recall_score(y_test_data, preds)
    f1 = f1_score(y_test_data, preds)
```

```
    table = pd.DataFrame({'model': [model_name],
                          'precision': [precision],
                          'recall': [recall],
                          'F1': [f1],
                          'accuracy': [accuracy]
                          })

    return table
```

In [36]:
```
# Get validation scores for RF model
rf_val_scores = get_test_scores('RF val', rf_val_preds, y_val)

# Append to the results table
results = pd.concat([results, rf_val_scores], axis=0)
results
```

Out[36]:

	model	precision	recall	F1	accuracy
0	RF cv	0.458198	0.126782	0.198534	0.818626
0	XGB cv	0.442586	0.173468	0.248972	0.814780
0	RF val	0.445255	0.120316	0.189441	0.817483

The scores experienced a slight decrease compared to the training scores across all metrics, though with minimal deviation. This suggests that the model did not exhibit overfitting to the training data.

XGBoost

In [37]:
```
# Use XGBoost model to predict on validation data
xgb_val_preds = xgb_cv.best_estimator_.predict(X_val)

# Get validation scores for XGBoost model
xgb_val_scores = get_test_scores('XGB val', xgb_val_preds, y_val)

# Append to the results table
results = pd.concat([results, xgb_val_scores], axis=0)
results
```

Out[37]:

	model	precision	recall	F1	accuracy
0	RF cv	0.458198	0.126782	0.198534	0.818626
0	XGB cv	0.442586	0.173468	0.248972	0.814780
0	RF val	0.445255	0.120316	0.189441	0.817483
0	XGB val	0.430769	0.165680	0.239316	0.813287

Just like the random forest model, the XGBoost model exhibited slightly lower validation scores. However, it still emerges as the clear champion.

Using the champion model(XGBoost) to predict on test data

In [38]:
```
# Use XGBoost model to predict on test data
xgb_test_preds = xgb_cv.best_estimator_.predict(X_test)

# Get test scores for XGBoost model
xgb_test_scores = get_test_scores('XGB test', xgb_test_preds, y_test)

# Append to the results table
```

```
results = pd.concat([results, xgb_test_scores], axis=0)
results
```

Out[38]:

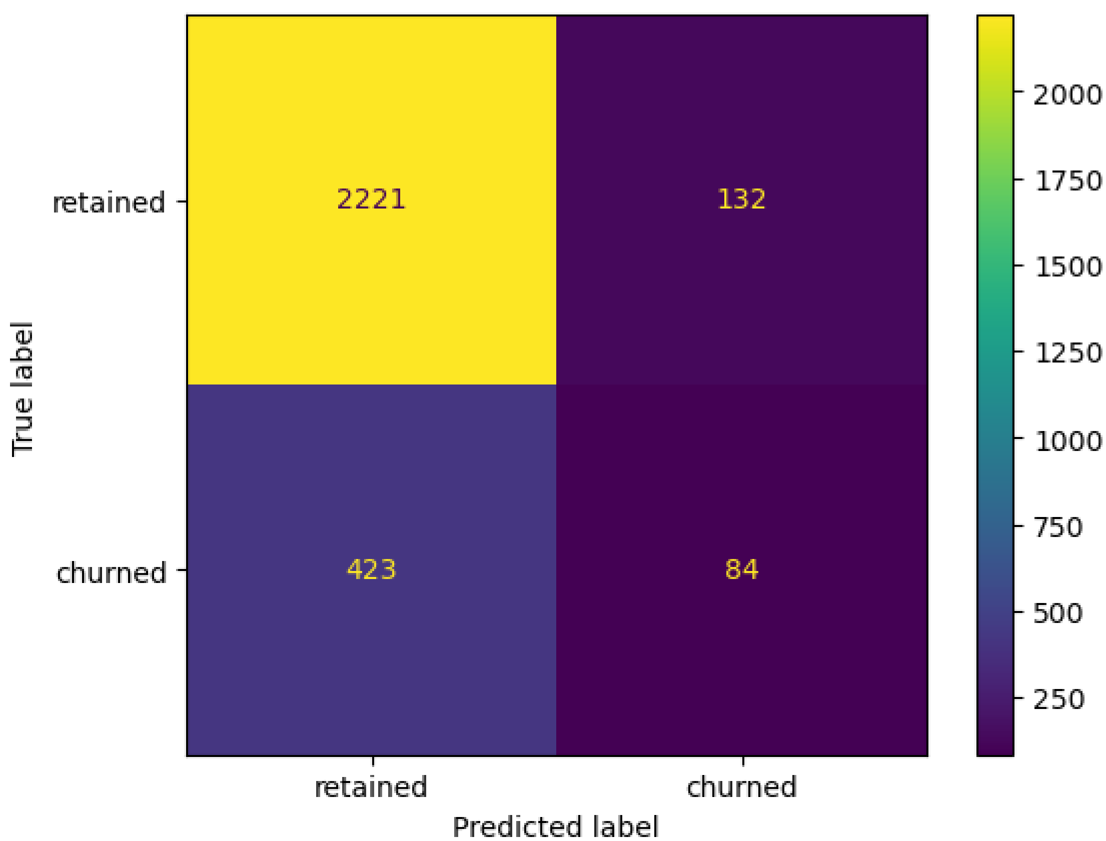
	model	precision	recall	F1	accuracy
0	RF cv	0.458198	0.126782	0.198534	0.818626
0	XGB cv	0.442586	0.173468	0.248972	0.814780
0	RF val	0.445255	0.120316	0.189441	0.817483
0	XGB val	0.430769	0.165680	0.239316	0.813287
0	XGB test	0.388889	0.165680	0.232365	0.805944

The recall remained unchanged from the validation data, while the precision experienced a significant decline, resulting in a slight drop in all other scores. Nevertheless, these variations fall within an acceptable range for performance disparities between validation and test scores.

Task 13. Confusion matrix

In [39]:

```
# Generate array of values for confusion matrix
cm = confusion_matrix(y_test, xgb_test_preds, labels=xgb_cv.classes_)

# Plot confusion matrix
disp = ConfusionMatrixDisplay(confusion_matrix=cm,
                              display_labels=['retained', 'churned'])
disp.plot();
```

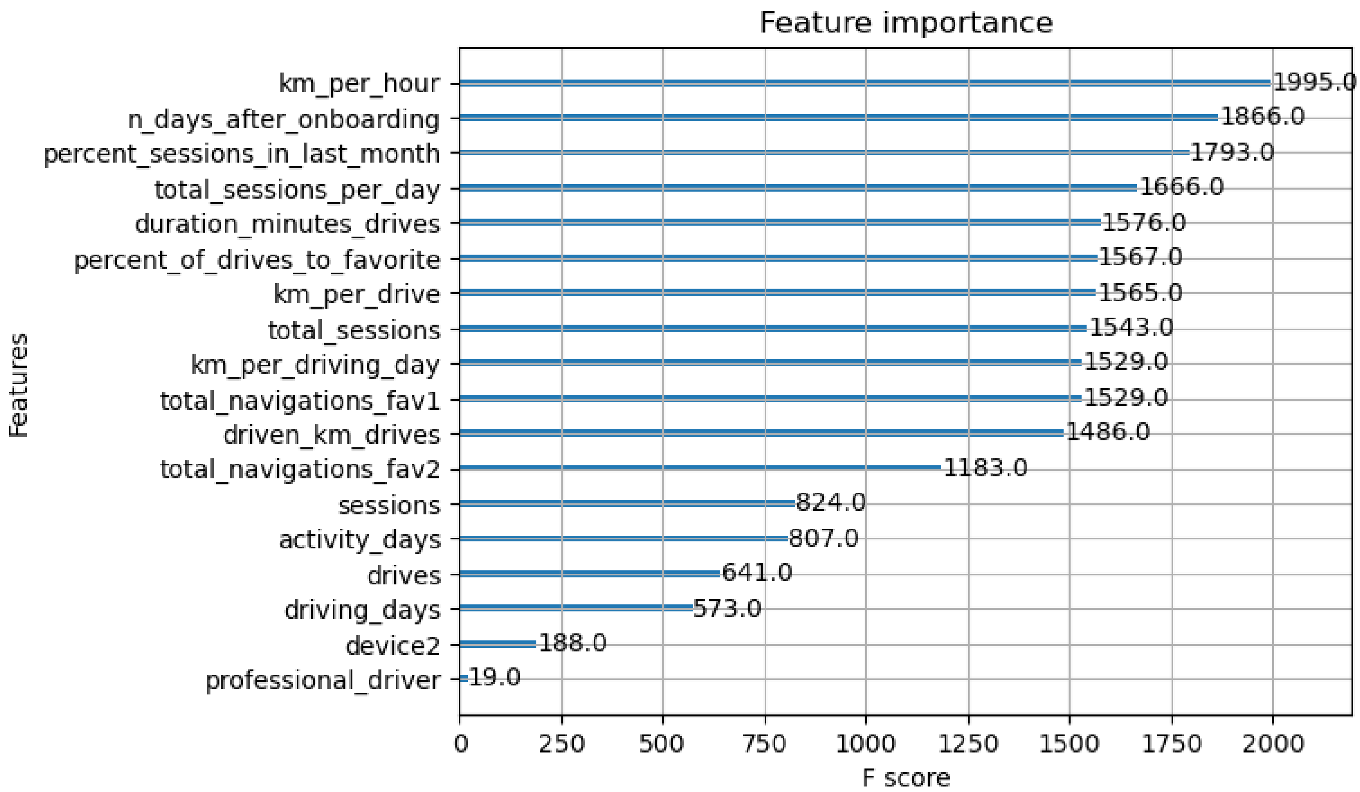


The model's false negatives outnumbered false positives by a factor of three, and it accurately identified only 16.6% of the users who churned.

Feature importance

Uses the `plot_importance` function to inspect the most important features of the final model.

In [40]:
```
plot_importance(xgb_cv.best_estimator_);
```



The XGBoost model utilized a greater number of features compared to the logistic regression model. In particular, the logistic regression model heavily relied on a single feature, namely "activity_days," for its final prediction.

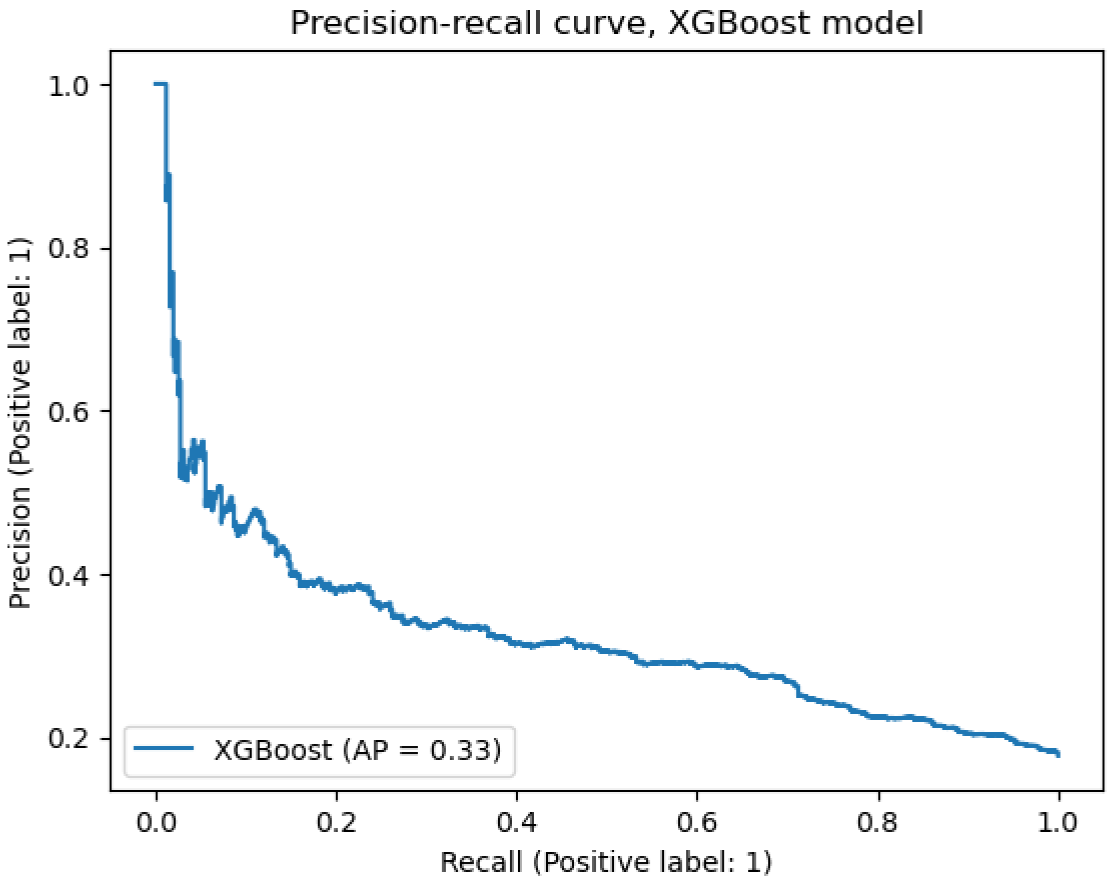
This further emphasizes the significance of feature engineering, as the engineered features played a significant role. They comprised six out of the top 10 features, including three out of the top five.

Additionally, it is worth noting that the selection of important features can vary between different models. Such disparities in selected features are often a result of intricate interactions among features, highlighting the complexity involved in feature selection.

Finding threshold to increase recall

Identify an optimal decision threshold

In [41]:
```
# Plot precision-recall curve
display = PrecisionRecallDisplay.from_estimator(
    xgb_cv.best_estimator_, X_test, y_test, name='XGBoost'
    )
plt.title('Precision-recall curve, XGBoost model');
```



In [42]:
```
# Get predicted probabilities on the test data
predicted_probabilities = xgb_cv.best_estimator_.predict_proba(X_test)
predicted_probabilities
```

Out[42]:
```
array([[0.9765248 , 0.0234752 ],
       [0.5623678 , 0.43763223],
       [0.9964199 , 0.00358006],
       ...,
       [0.80931014, 0.19068986],
       [0.9623124 , 0.03768761],
       [0.64760244, 0.35239756]], dtype=float32)
```

The `predict_proba()` method returns a 2-D array of probabilities where each row represents a user. The first number in the row is the probability of belonging to the negative class, the second number in the row is the probability of belonging to the positive class. (Notice that the two numbers in each row are complimentary to each other and sum to one.)

You can generate new predictions based on this array of probabilities by changing the decision threshold for what is considered a positive response. For example, the following code converts the predicted probabilities to {0, 1} predictions with a threshold of 0.4. In other words, any users who have a value ≥ 0.4 in the second column will get assigned a prediction of `1`, indicating that they churned.

In [43]:
```
# Create a list of just the second column values (probability of target)
probs = [x[1] for x in predicted_probabilities]

# Create an array of new predictions that assigns a 1 to any value >= 0.4
new_preds = np.array([1 if x >= 0.4 else 0 for x in probs])
new_preds
```

Out[43]:
```
array([0, 1, 0, ..., 0, 0, 0])
```

Evaluation metrics when threshold is 0.4

In [44]:
```python
# Get evaluation metrics for when the threshold is 0.4
get_test_scores('XGB, threshold = 0.4', new_preds, y_test)
```

Out[44]:

	model	precision	recall	F1	accuracy
0	XGB, threshold = 0.4	0.383333	0.226824	0.285006	0.798252

Previous models for comparison.

In [45]:
```python
results
```

Out[45]:

	model	precision	recall	F1	accuracy
0	RF cv	0.458198	0.126782	0.198534	0.818626
0	XGB cv	0.442586	0.173468	0.248972	0.814780
0	RF val	0.445255	0.120316	0.189441	0.817483
0	XGB val	0.430769	0.165680	0.239316	0.813287
0	XGB test	0.388889	0.165680	0.232365	0.805944

Recall and F1 score increased significantly, while precision and accuracy decreased.

In [46]:
```python
def threshold_finder(y_test_data, probabilities, desired_recall):
    '''
    Find the threshold that most closely yields a desired recall score.

    Inputs:
        y_test_data: Array of true y values
        probabilities: The results of the `predict_proba()` model method
        desired_recall: The recall that you want the model to have

    Outputs:
        threshold: The threshold that most closely yields the desired recall
        recall: The exact recall score associated with `threshold`
    '''
    probs = [x[1] for x in probabilities]  # Isolate second column of `probabilities`
    thresholds = np.arange(0, 1, 0.001)    # Set a grid of 1,000 thresholds to test

    scores = []
    for threshold in thresholds:
        # Create a new array of {0, 1} predictions based on new threshold
        preds = np.array([1 if x >= threshold else 0 for x in probs])
        # Calculate recall score for that threshold
        recall = recall_score(y_test_data, preds)
        # Append the threshold and its corresponding recall score as a tuple to `scores`
        scores.append((threshold, recall))

    distances = []
    for idx, score in enumerate(scores):
        # Calculate how close each actual score is to the desired score
        distance = abs(score[1] - desired_recall)
        # Append the (index#, distance) tuple to `distances`
        distances.append((idx, distance))

    # Sort `distances` by the second value in each of its tuples (least to greatest)
    sorted_distances = sorted(distances, key=lambda x: x[1], reverse=False)
    # Identify the tuple with the actual recall closest to desired recall
    best = sorted_distances[0]
    # Isolate the index of the threshold with the closest recall score
    best_idx = best[0]
    # Retrieve the threshold and actual recall score closest to desired recall
    threshold, recall = scores[best_idx]
```

```python
    return threshold, recall
```

Tests the function to find the threshold that results in a recall score closest to 0.5.

In [47]:
```python
# Get the predicted probabilities from the champion model
probabilities = xgb_cv.best_estimator_.predict_proba(X_test)

# Call the function
threshold_finder(y_test, probabilities, 0.5)
```

Out[47]: (0.124, 0.5029585798816568)

By establishing a threshold of 0.124, the recall comes in at 0.503.

According to the precision-recall curve, a recall score of 0.5 should correspond to a precision value of approximately 0.3.

In [48]:
```python
# Create an array of new predictions that assigns a 1 to any value >= 0.124
new_preds = np.array([1 if x >= 0.124 else 0 for x in probs])

# Get evaluation metrics for when the threshold is 0.124
get_test_scores('XGB, threshold = 0.124', new_preds, y_test)
```

Out[48]:

	model	precision	recall	F1	accuracy
0	XGB, threshold = 0.124	0.304296	0.502959	0.379182	0.708042

Part 4: Insights and Conclusion

Questions:

Recommendation to use or not use this model for churn prediction:

- If the model is utilized for significant business decisions, then it falls short in being a robust predictor, as evidenced by its low recall score. However, if the model is solely employed to guide exploratory efforts, it can provide value.

Tradeoffs made by splitting the data into training, validation, and test sets as opposed to just training and test sets:

- Although dividing the data into three sets results in less data available for model training compared to a two-way split, conducting model selection on a separate validation set allows for testing the champion model exclusively on the test set. This approach provides a better estimation of future performance compared to a two-way split where the champion model is selected based on performance on the test data.

Benefits of using a logistic regression model over an ensemble of tree-based models for classification tasks:

- Logistic regression models offer easier interpretability due to the assignment of coefficients to predictor variables. This reveals not only the most influential features in the final predictions but also the directionality of their impact. It indicates whether each feature is positively or negatively correlated with the target in the model's final prediction.

Benefits of using an ensemble of tree-based models over a logistic regression model for classification tasks:

- Tree-based model ensembles generally excel in predictive power. If the primary concern is the model's predictive performance, tree-based modeling tends to outperform logistic regression. Tree-based models also require less data cleaning and make fewer assumptions about the underlying distributions of predictor variables, making them more convenient to work with.

Improvements that could be made to this model:

- Introducing new features could enhance the model's predictive capabilities, particularly when domain knowledge is leveraged. In the case of this model, engineered features accounted for over half of the top 10 most-predictive features employed by the model. Reconstructing the model using different combinations of predictor variables can help reduce noise originating from non-predictive features.

Additional features that could help improve the model:

- Having drive-level information for each user, such as drive times and geographic locations, would be beneficial. More detailed data providing insights into user interactions with the app, such as the frequency of reporting or confirming road hazard alerts, would be valuable. Also, knowing the monthly count of unique starting and ending locations provided by each driver could offer further assistance.

USER CHURN ANALYSIS

EDA AND MACHINE LEARNING MODELLING

PROJECT OVERVIEW AND GOALS

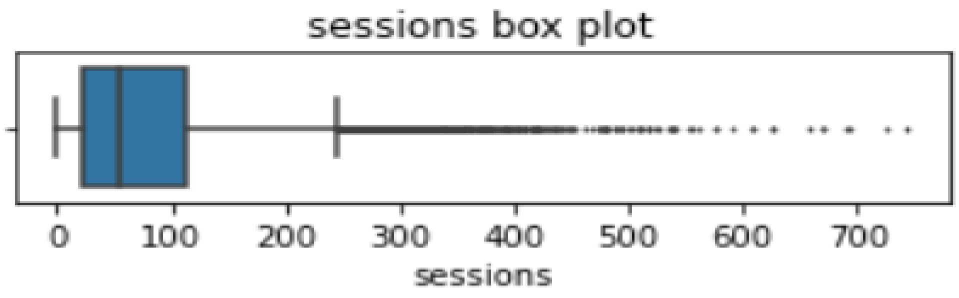
- Waze leadership has asked the data team to build a machine learning model to predict user churn. The model is based on data collected from users of the Waze app.
- We will achieve this through a series of milestones:
  - EDA and Data Visualizations
  - Computing descriptive statistics and conducting hypothesis testing
  - Building a regression model(for comparison) and evaluating that model
  - Building a machine learning model

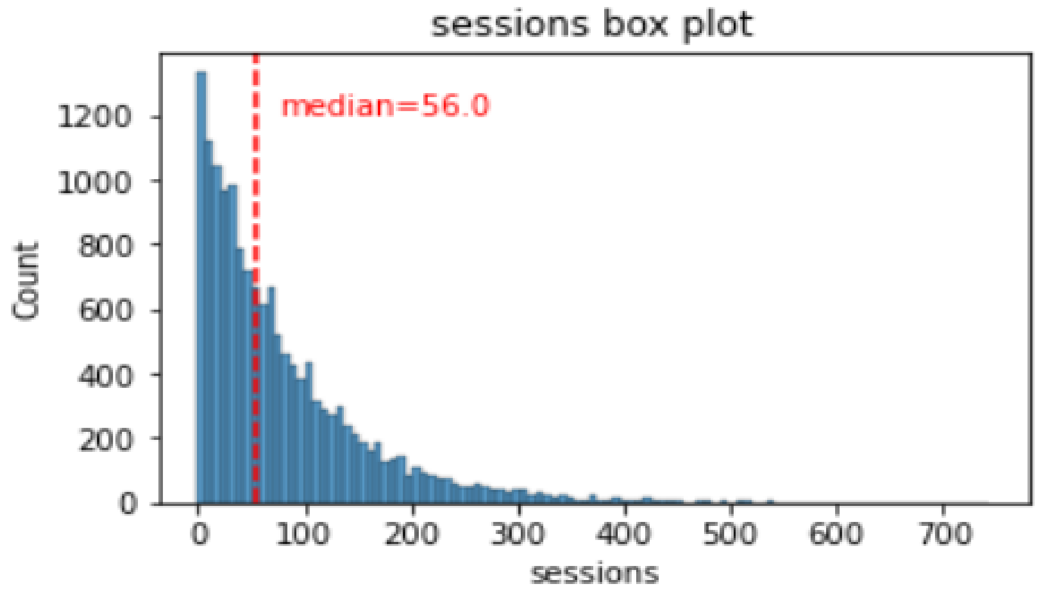
- Based on the data, communicate final insights and any recommendations

METHODOLOGY AND TECHNOLOGY

- **Data Sources:**
  - Waze User Data(one-month) via waze_dataset.csv
- **Data Cleaning:**
  - Dataset was cleaned using Python *pandas* and *numpy*
- **Exploratory Data Analysis:**
  - EDA performed using Python *pandas, numpy, pyplot,* and *seaborn*
- **Hypothesis Testing:**
  - Hypothesis testing performed with Python *pandas* and *scipy stats*
- **Model Building and Evaluation:**
  - Models built using Python *sklearn.linear_model, RandomForestClassifier, XGBClassifier*

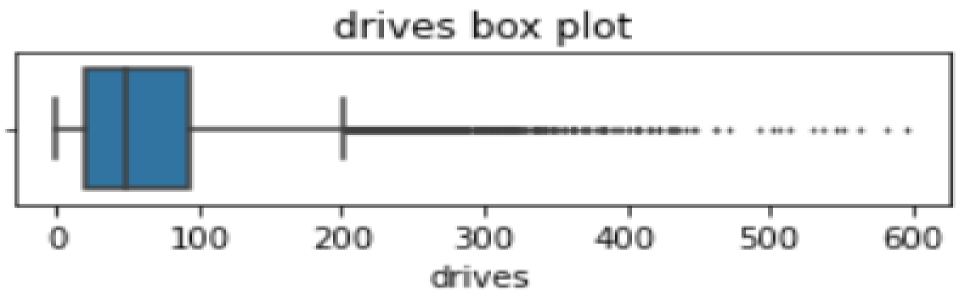
SESSIONS

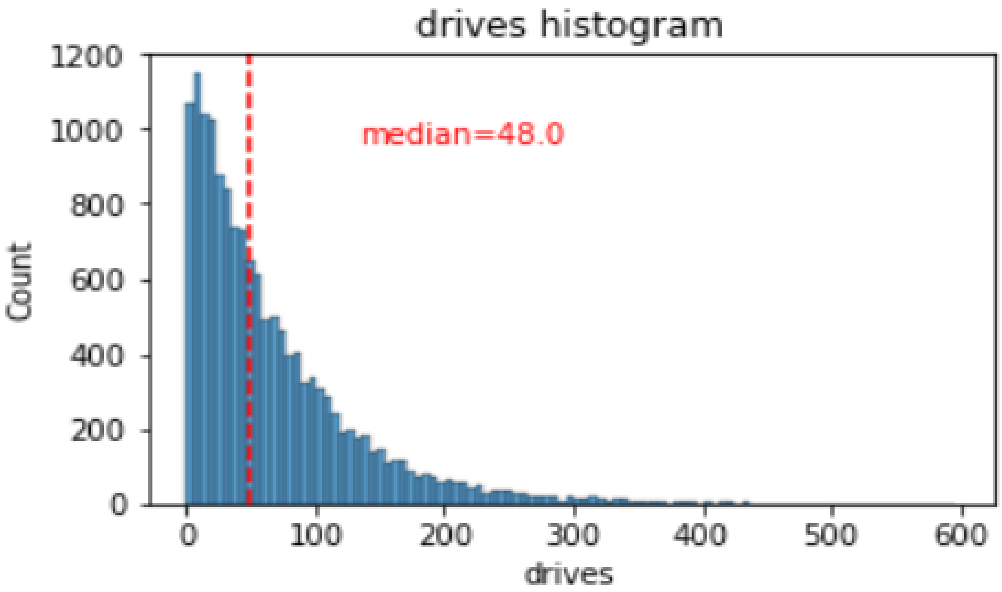




- The boxplot reveals that a **subset of users** has **more than 700 sessions**.
- The **median** number of session is 56.
- The sessions variable exhibits a **skewed distribution to the right**, where approximately **50%** of the **observations consist of 56 sessions or fewer.**

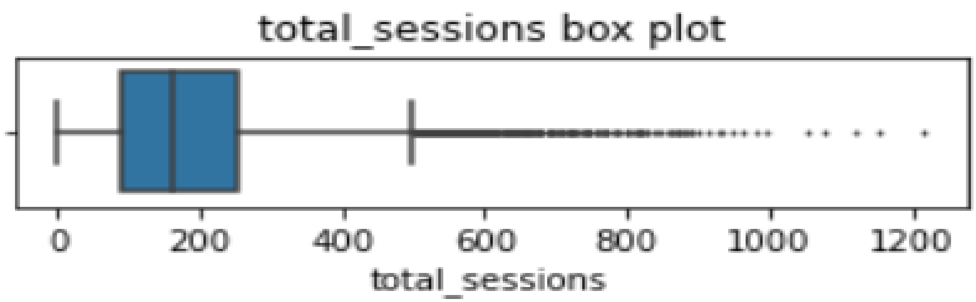
DRIVES

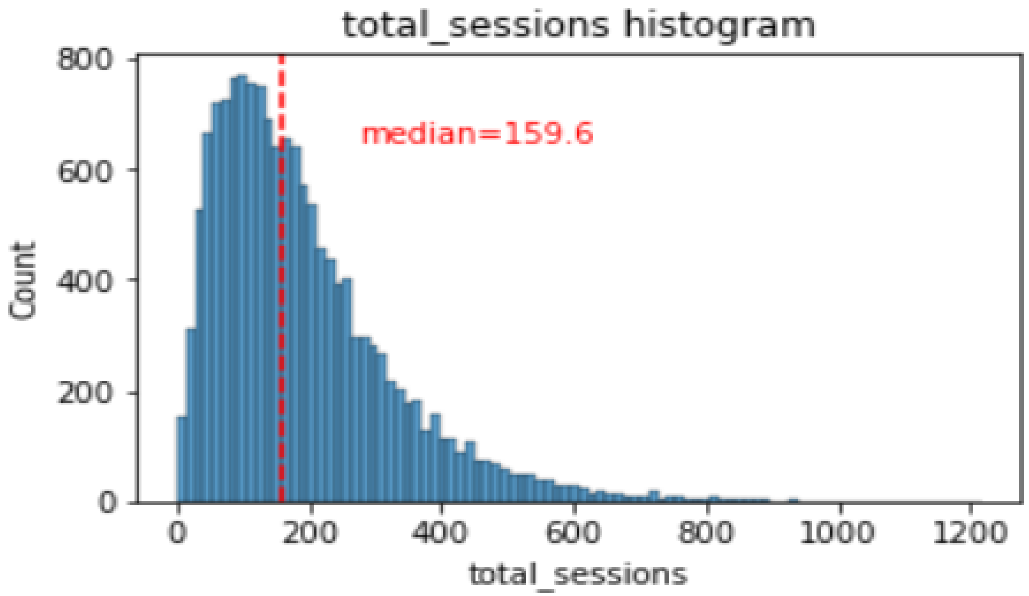




- The drives data exhibits a distribution resembling that of the 'sessions' variable.
- It is **right-skewed**, resembles a **log-normal distribution**, with a **median** of 48 **drives.**
- However, a **subset of drivers** recorded **over 400 drives in the last month.**

TOTAL SESSIONS



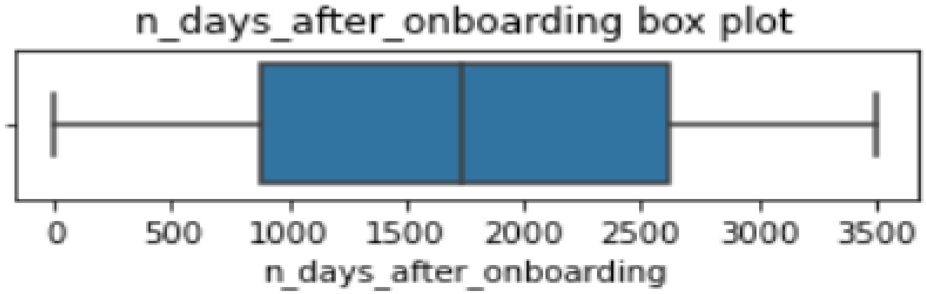


- The distribution of total_sessions is **right-skewed,** appearing closer to a normal distribution compared to the previous variables.

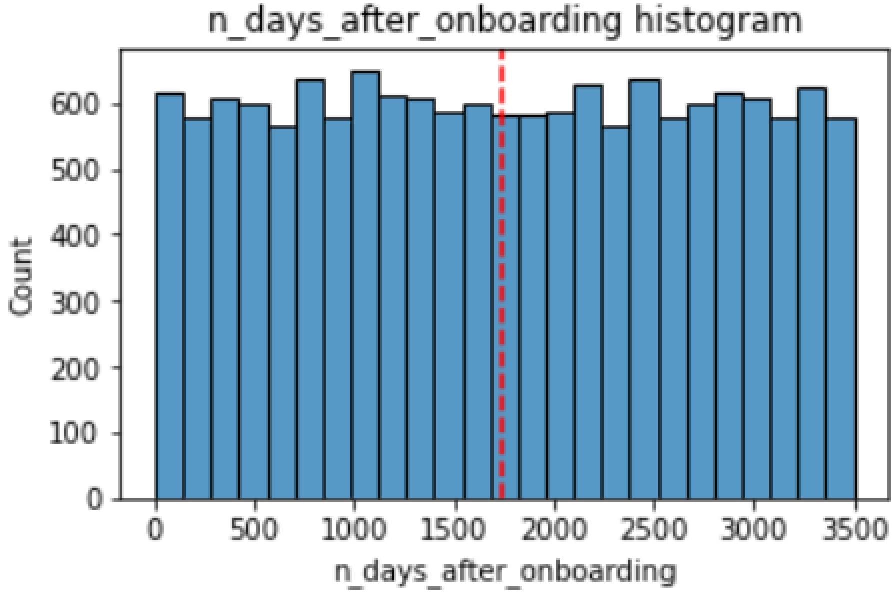
- The **median** total number of sessions is approximately **159.6.**

- If the median number of sessions in the last month was 48 and the median total sessions was around 160, it suggests that a **significant proportion of a user's overall sessions possibly occurred within the last month.**

NUMBER OF DAYS AFTER ONBOARDING

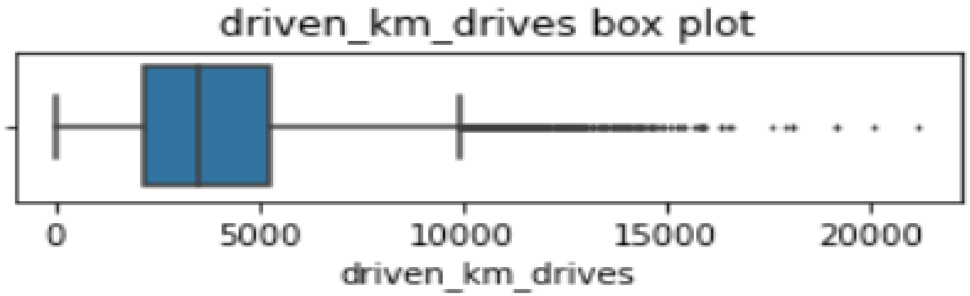


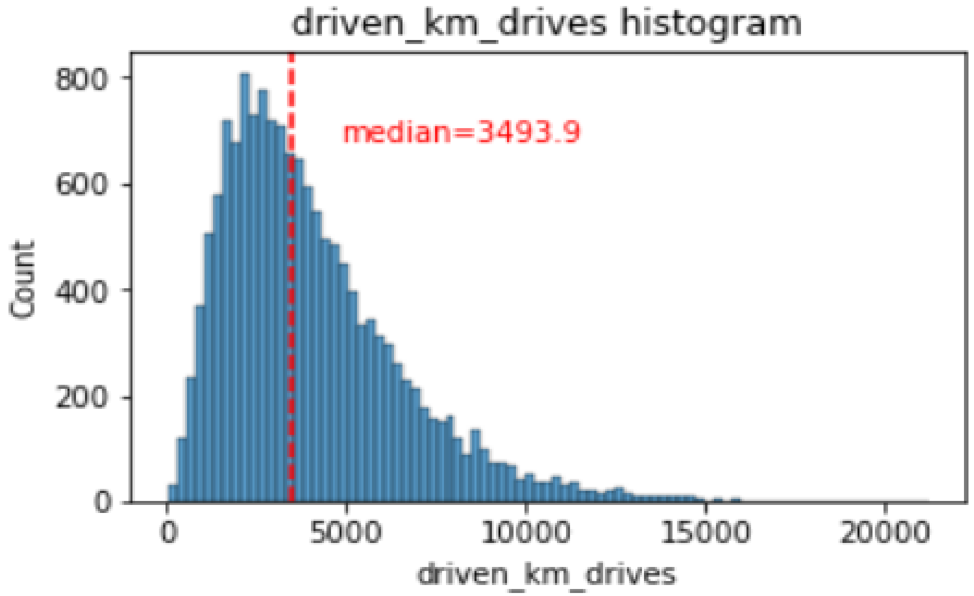
Median: 1741.0



- The total user tenure is a **uniform distribution** with values rangin from near-zero to ~3500 days, or roughly **9.5 years.**
- The **median** number of days since a user signed up for the app is 1741 days, or roughly **4.8 years.**

TOTAL KM DRIVEN DURING THE MONTH



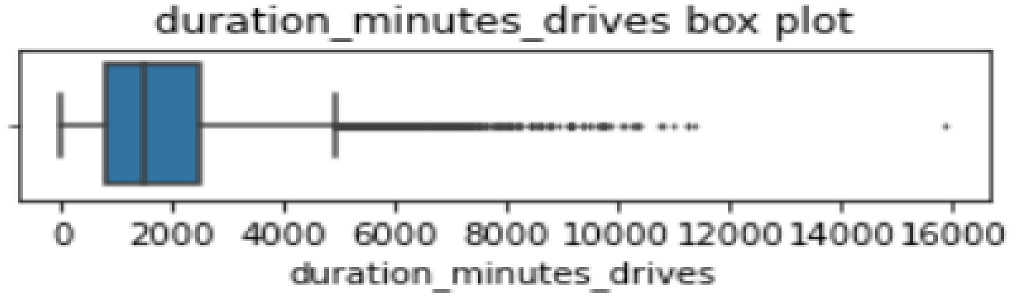


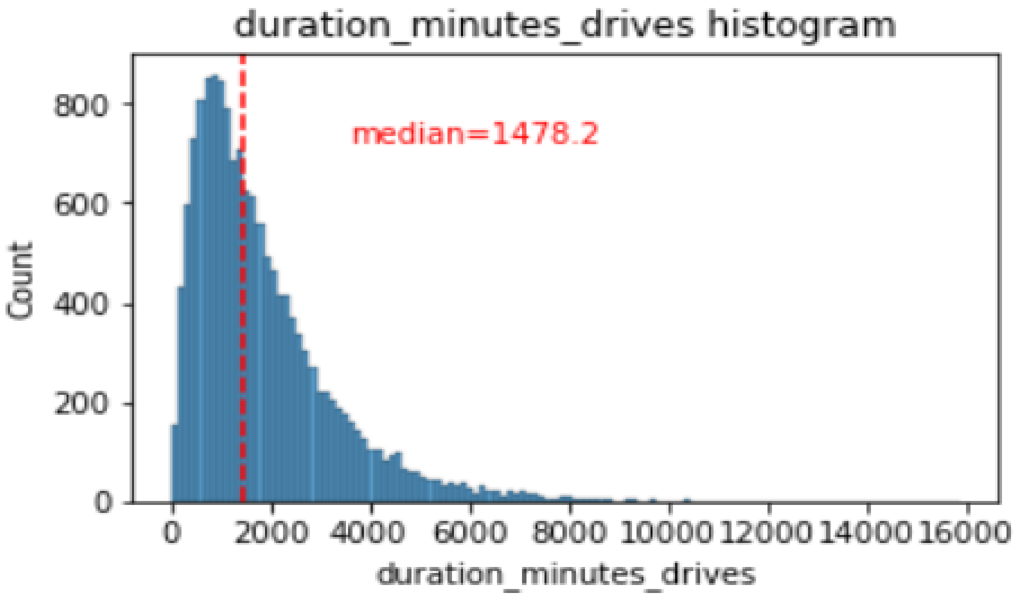
- The distribution of drives completed by each user in the last month exhibits **right-skewed normal distribution.**

- Roughly **50% of users drove fewer than 3,495 kilometers** during that period.

- The **median** number of total kilometers driven during the month 3494 **km.**

TOTAL DURATION DRIVEN DURING THE MONTH

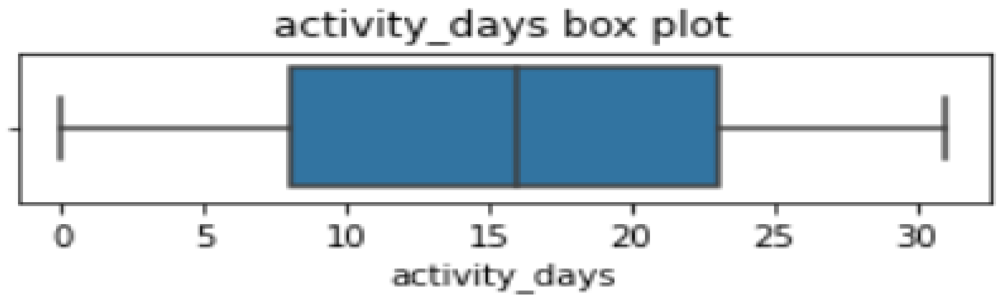




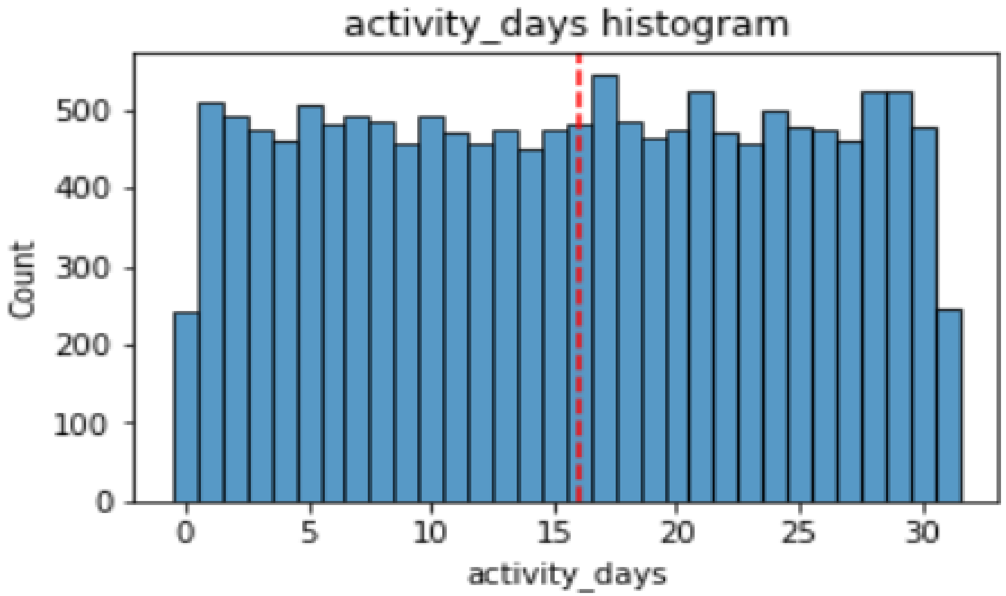
- The duration_minutes_drives variable has a **normalish distribution** with a heavily **skewed right tail.**

- Around **50%** of the users had a driving duration of **less than** the **median** of **1,478 minutes** (equivalent to about 25 hours), while **certain users recorded over 250 hours** of driving time throughout the month.

ACTIVITY DAYS

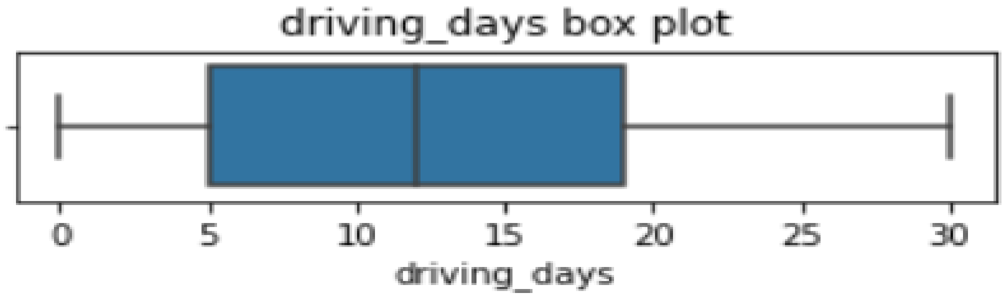


Median: 16.0

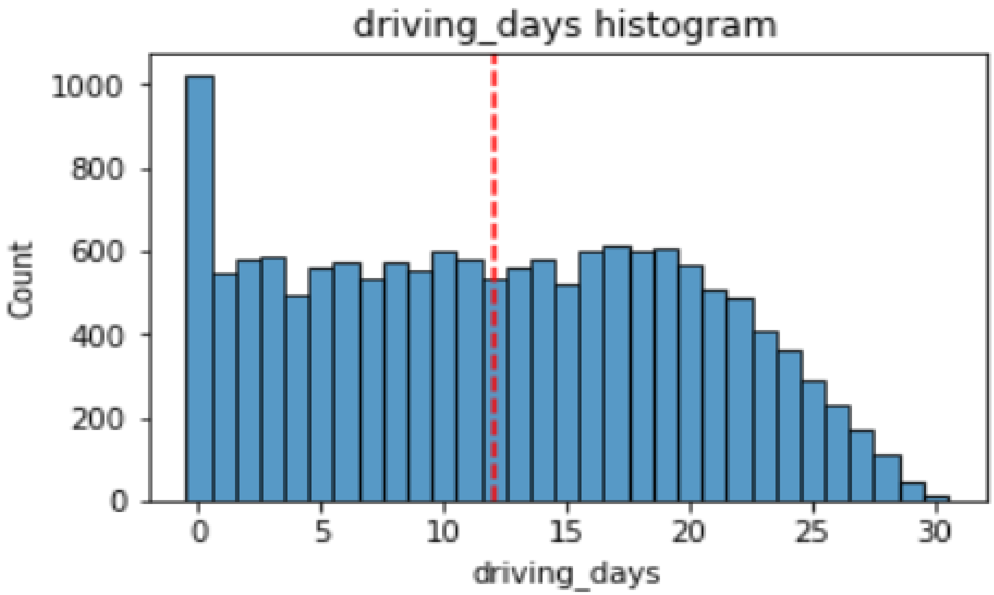


- In the past month, users had a **median of 16 app openings.**
- The box plot displays a **distribution that is centered**.
- The histogram indicates a **relatively uniform pattern** with approximately **500 individuals opening the app on each day.**
- However, there are approximately **250 users who did not open the app at all**, while **another 250 users opened it every day** throughout the month.

DRIVING DAYS

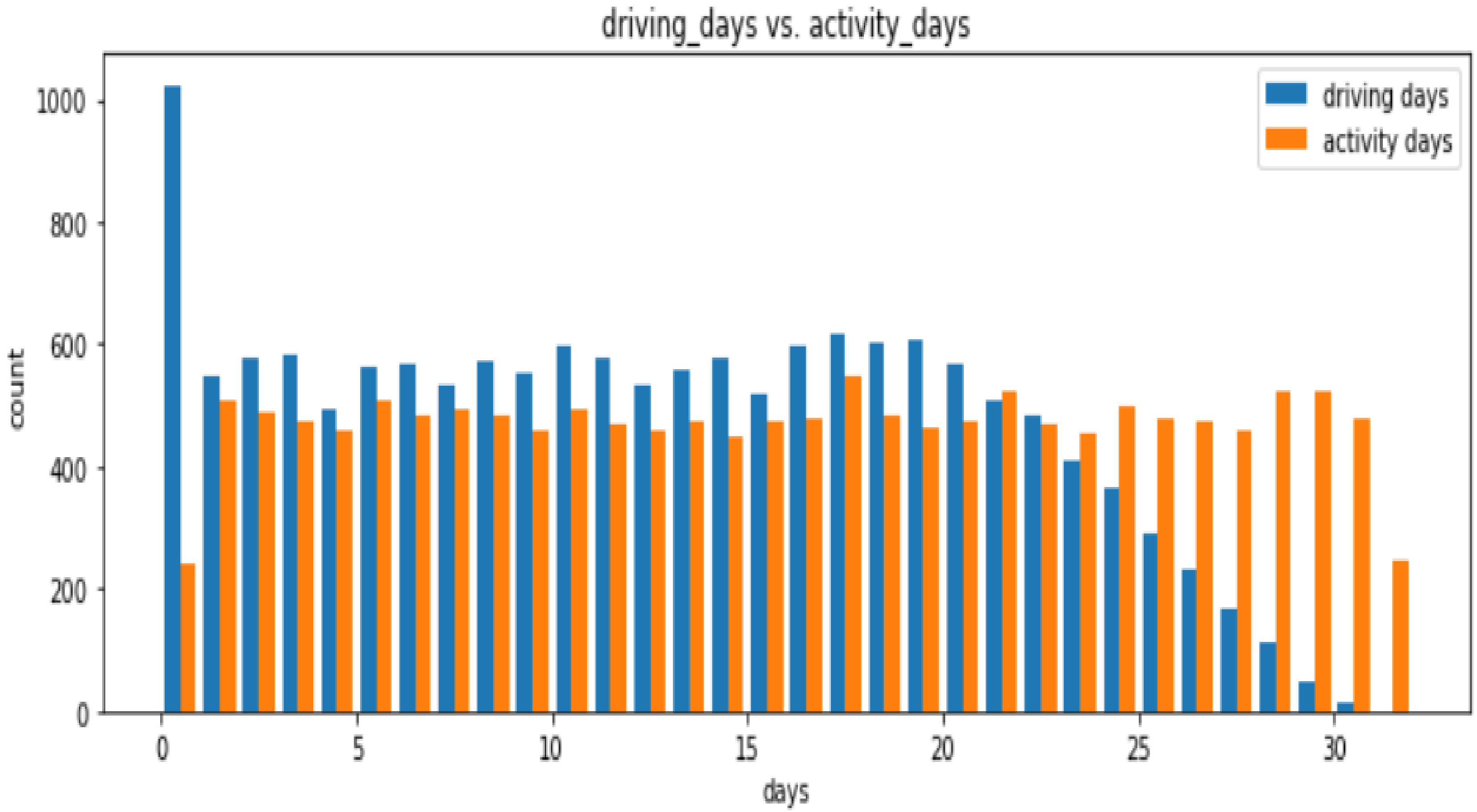


Median: 12.0



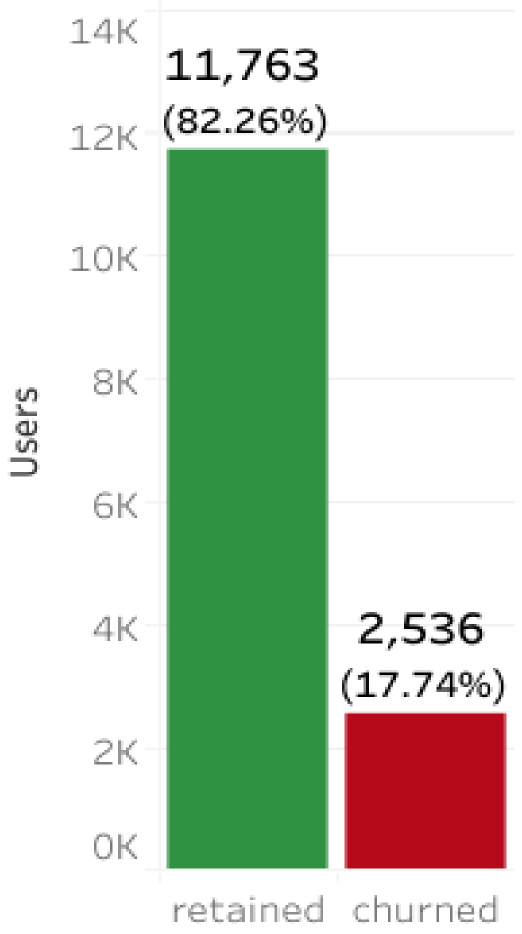
- The **median** number of days the users drove in the last month is **12 days**.
- The frequency of users driving each month shows a **relatively uniform pattern**, closely aligned with the number of days they accessed the app within the same period.
- The **distribution** of driving_days **skews towards lower values.**
- Interestingly, there were nearly **twice as many users** (~1,000 versus ~550) who **didn't engage in any driving** activity throughout the month..

DRIVING DAYS VS. ACTIVITY DAYS



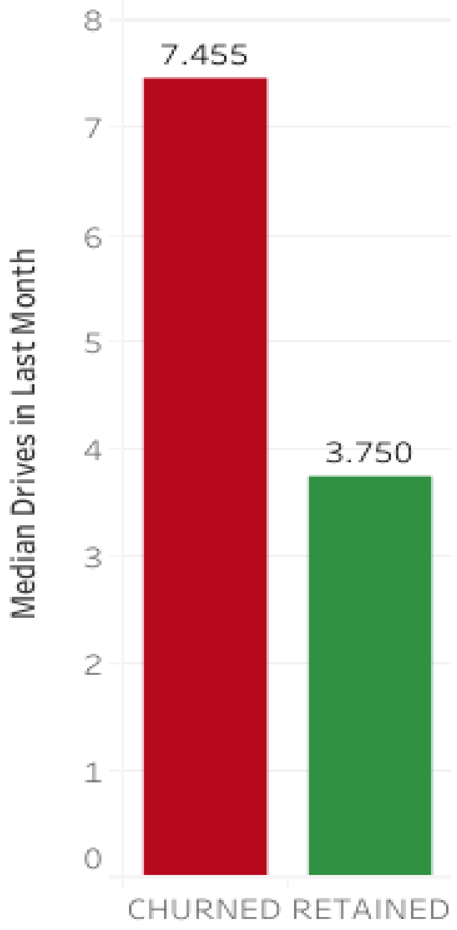
- Initially, more users had an increase in driving_days.
- The two variables stayed fairly consistent until around day 21.
- After day 21, driving_days steadily declined, while activity_days remained near its previous levels.
- This would suggest that though users weren't driving as much, they were still opening and using the app.

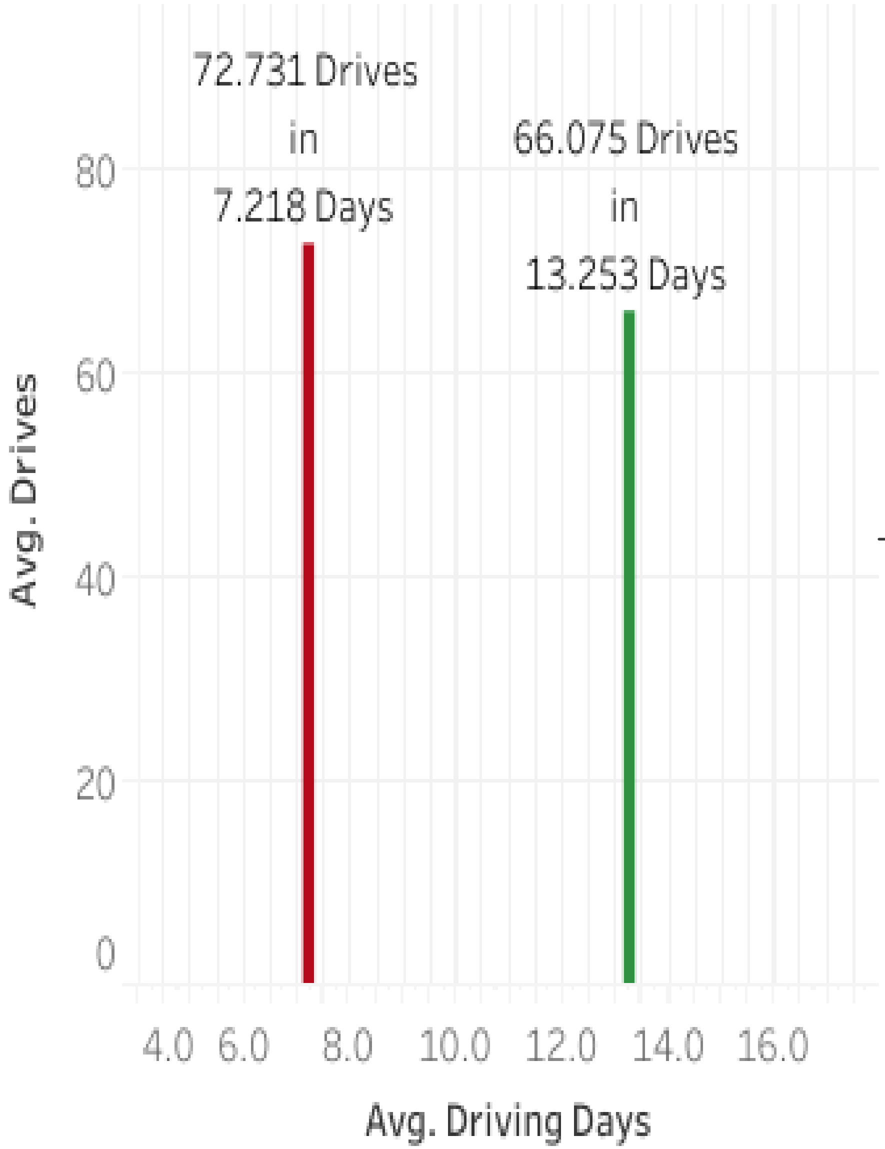
CHURN VS. RETAINED USERS



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

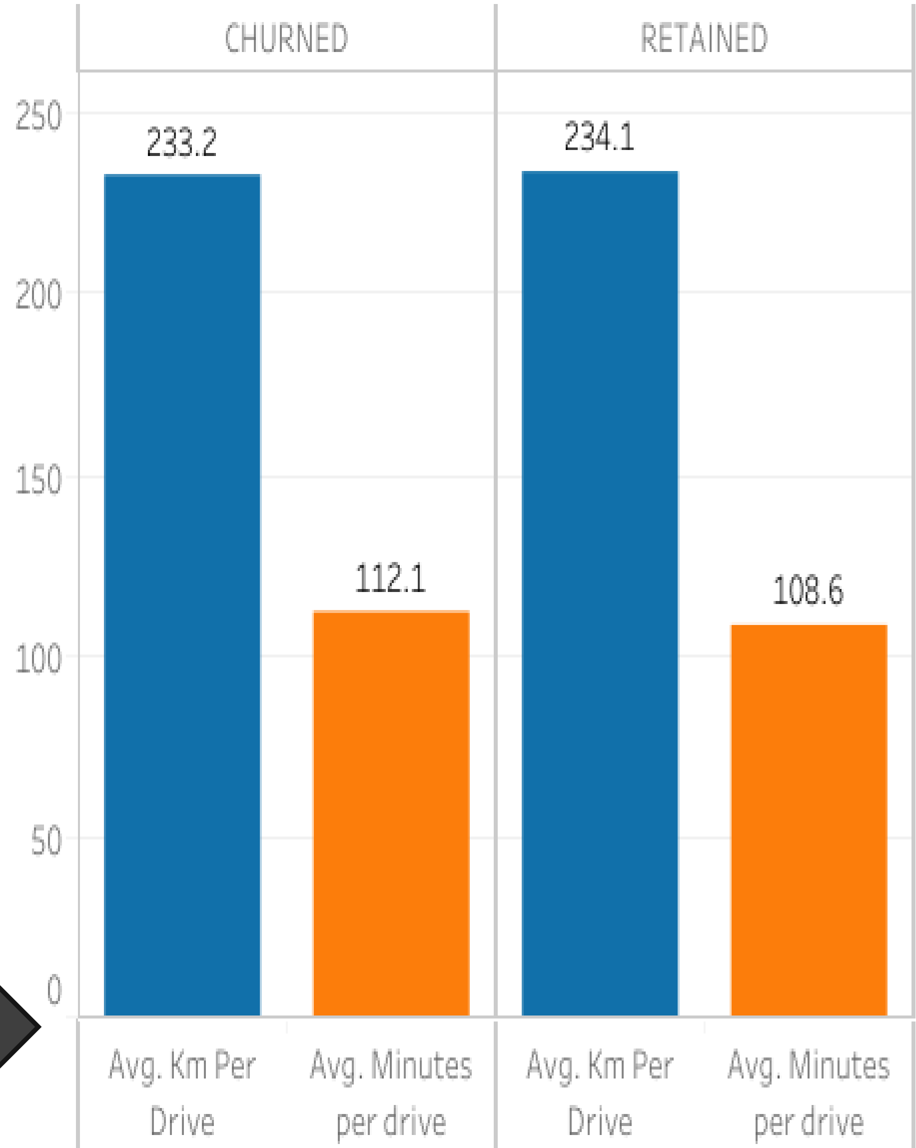
Churned users averaged ~3 more drives in the last month than retained users.



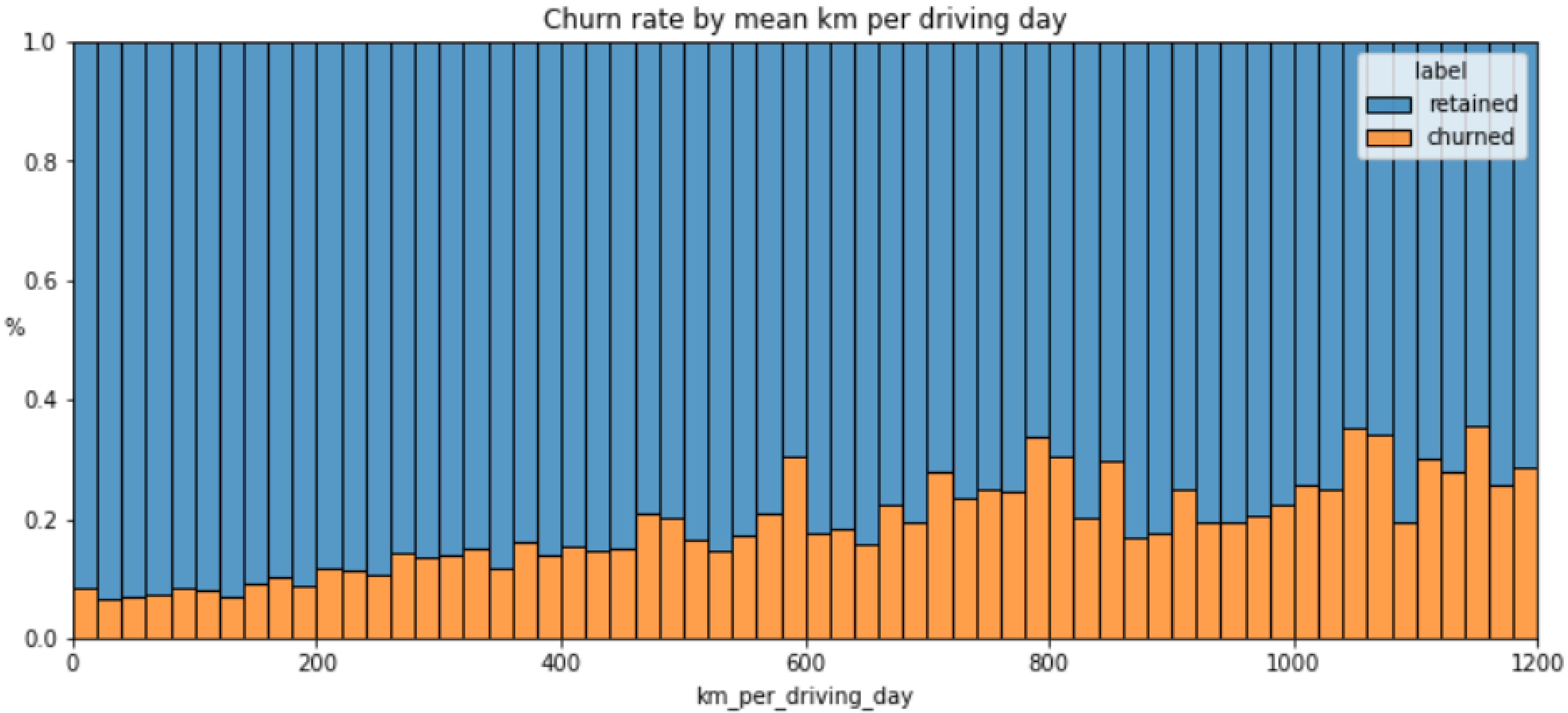


Churned users had **more drives** in **fewer days.**

Churned users trips were similar in length but slightly **longer in duration.**

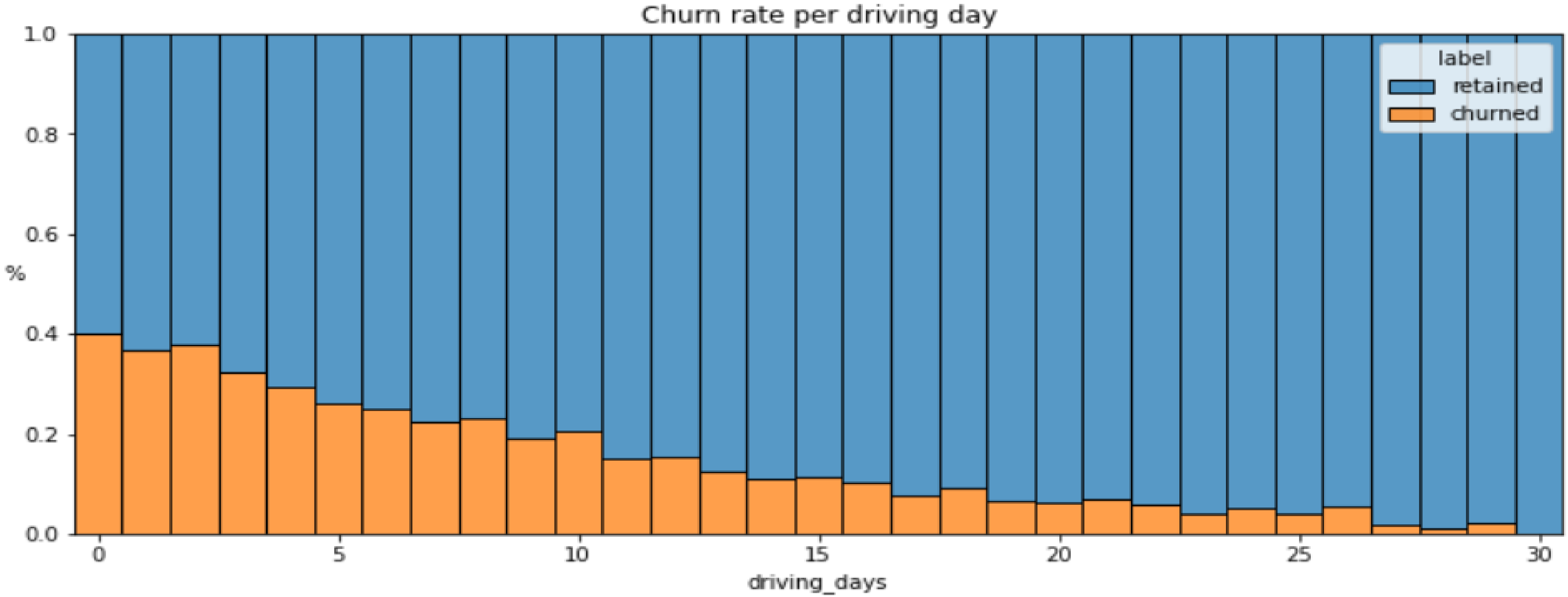


RETENTION BY KM DRIVEN PER DRIVING DAY



As the average daily distance driven increases, the churn rate also tends to rise.

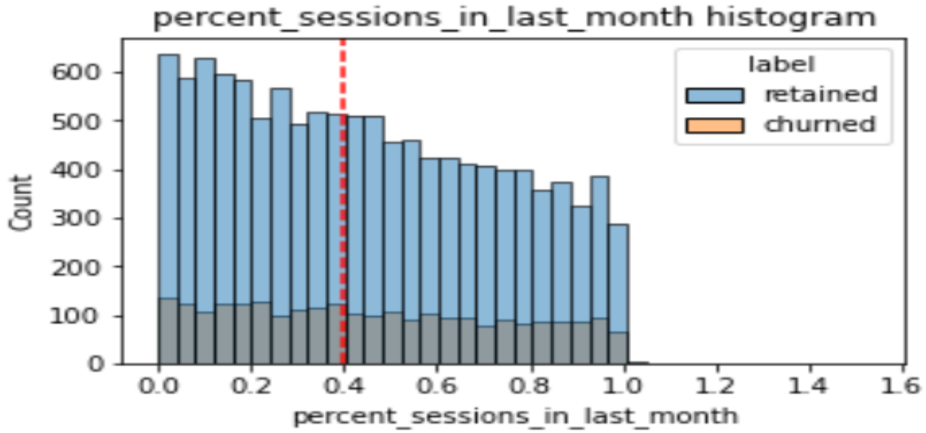
CHURN RATE PER NUMBER OF DRIVING DAYS

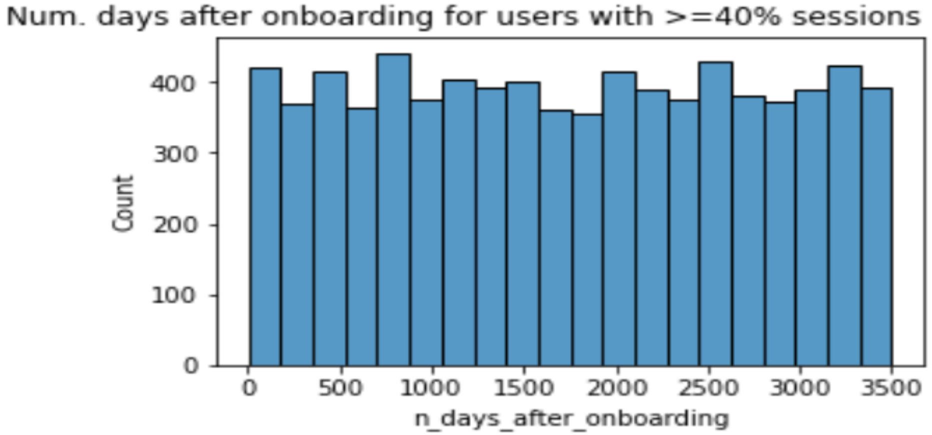


The likelihood of **churn decreased as the frequency of app usage increased**. Among users who did not use the app at all in the last month, 40% churned, whereas **none of the users who used the app for 30 days experienced churn.**

SESSIONS PROPORTIONS AND SURGE IN ACTIVITY FOR LONGSTANDING USERS

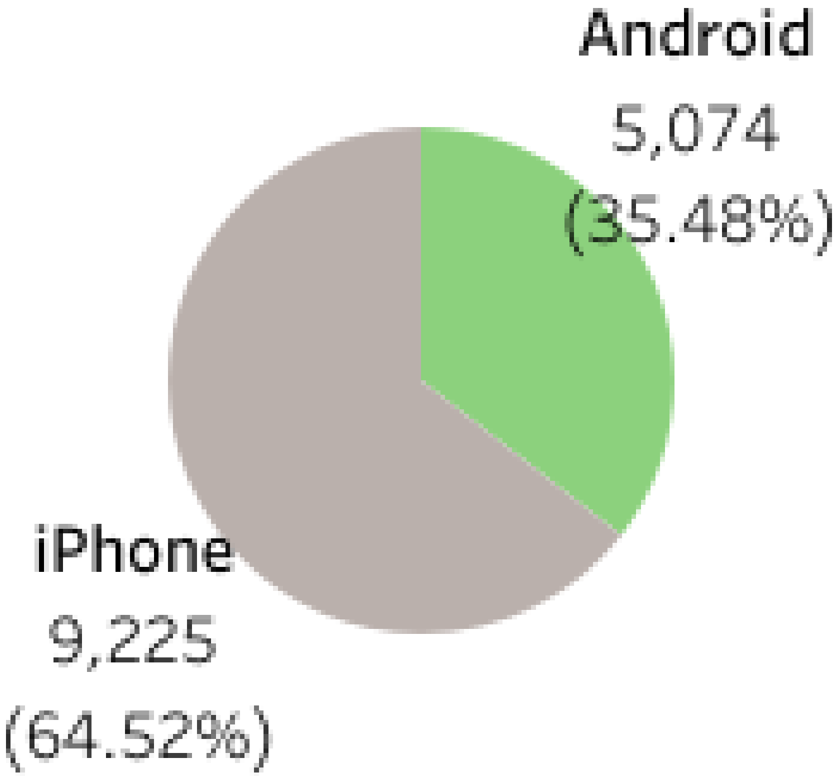
Median: 0.4





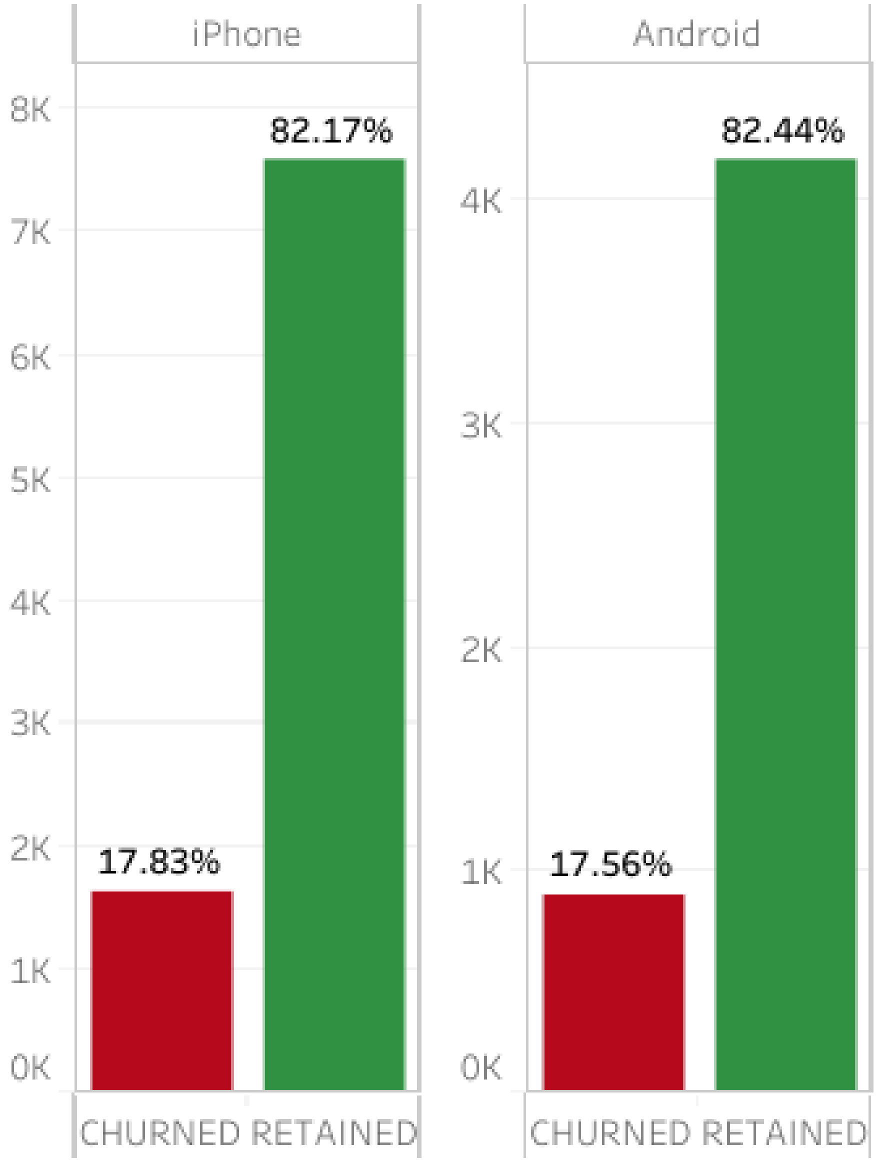
- Around **half of the users** included in the dataset had **40% or more of their sessions** concentrated solely **in the last month.**
- The number of days since users onboarded, who have experienced 40% or more of their total sessions within the last month, conforms to a **uniform distribution**.
- **Why the sudden surge in app usage by these longstanding users during the recent month?**

DEVICES: ANDROID VS. IPHONE



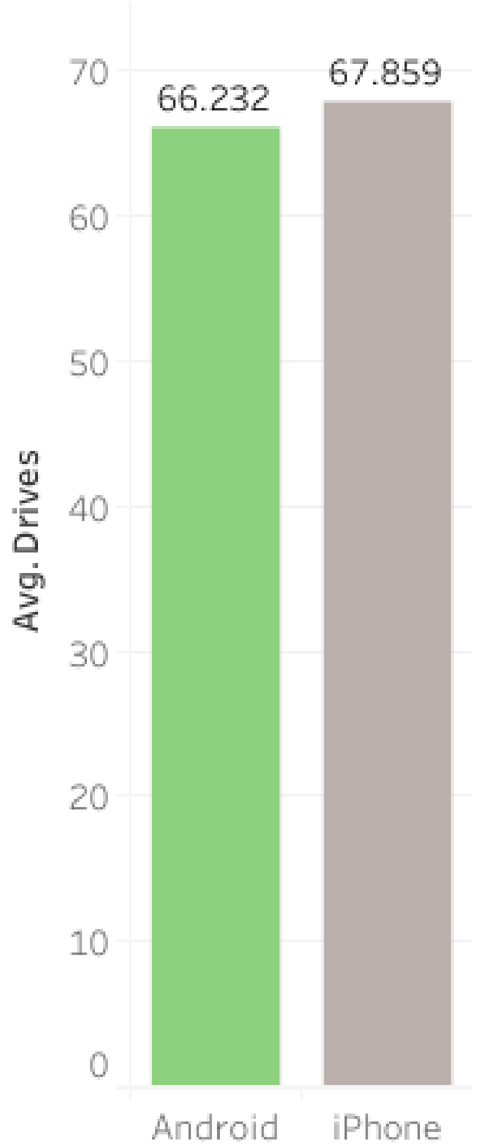
- **iPhone devices** make up a **majority** of the users in this dataset.

- **Android devices** account for roughly **a third** of all users.



- The **proportion** of iPhone users to Android users remains **consistent** within both the churned and retained user groups.

- There is **no indication of any correlation** between device type and churn.



- Given the displayed averages, it seems that iPhone device users tend to have a higher average number of drives when using the application.

- However, it's important to consider that this disparity may be a result of random sampling rather than an actual difference in the number of drives.

- To determine if the distinction is statistically significant, I performed a hypothesis test.

DEVICE HYPOTHESIS TESTING

Hypotheses:

- $H0$: There is no difference in average number of drives between drivers who use iPhone devices and drivers who use Androids.

- HA : There is a difference in average number of drives between drivers who use iPhone devices and drivers who use Androids.

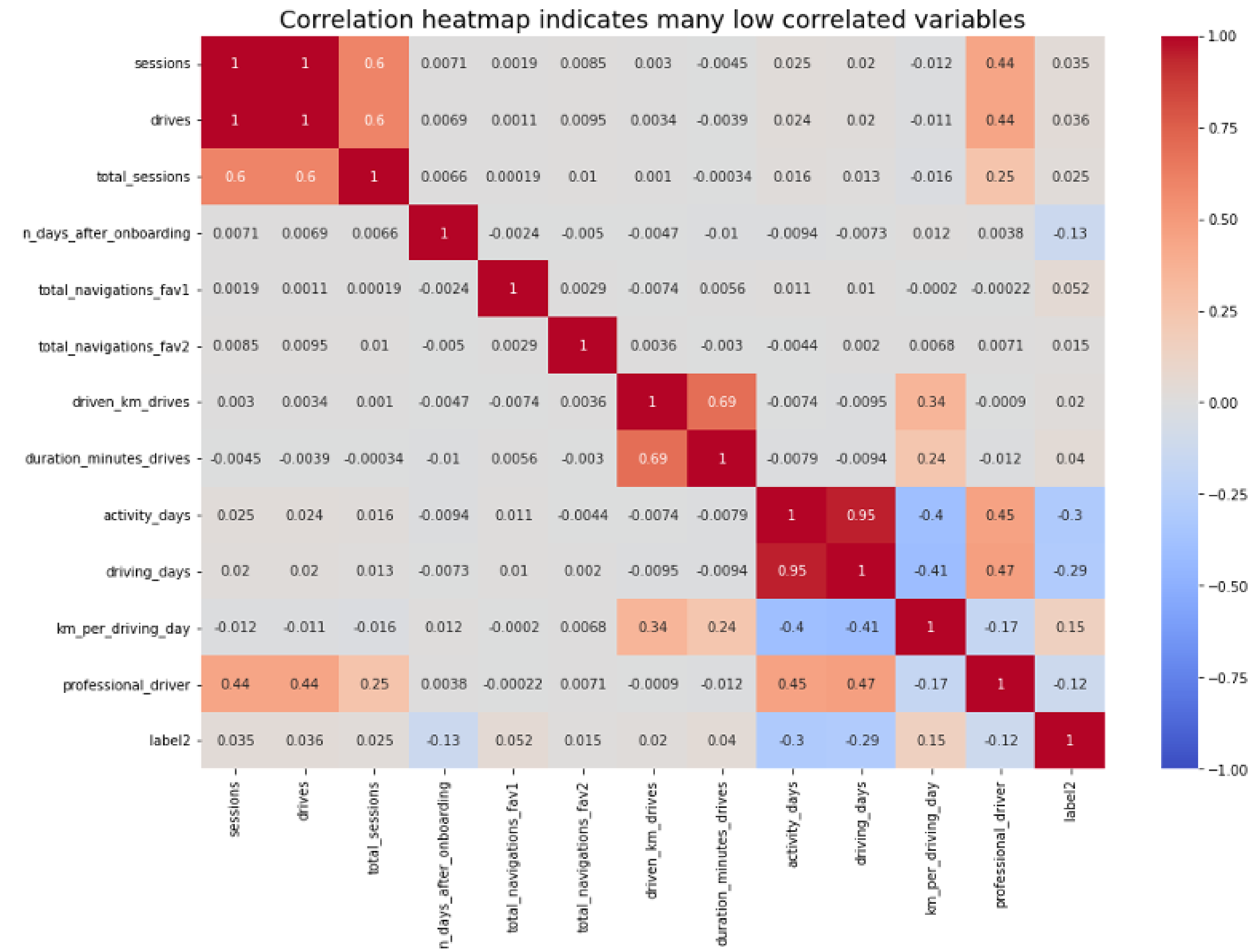
Two-sample test with 5% as the significance level with a two-sample t-test.

```
# 1. Isolate the `drives` column for iPhone users.
iPhone = df[df['device_type'] == 1]['drives']

# 2. Isolate the `drives` column for Android users.
Android = df[df['device_type'] == 2]['drives']

# 3. Perform the t-test
stats.ttest_ind(a=iPhone, b=Android, equal_var=False)
```

```
Ttest_indResult(statistic=1.4635232068852353, pvalue=0.1433519726802059)
```

p Value = 0.143...

As the p-value exceeds the selected significance level of 5%, we fail to reject the null hypothesis. This indicates that there is **no statistically significant distinction in the average number of drives between iPhone users and Android users.**



Collinearity

As title suggests, the correlation heatmap indicates many low correlated variables.

Variables that are multicollinear with each other:

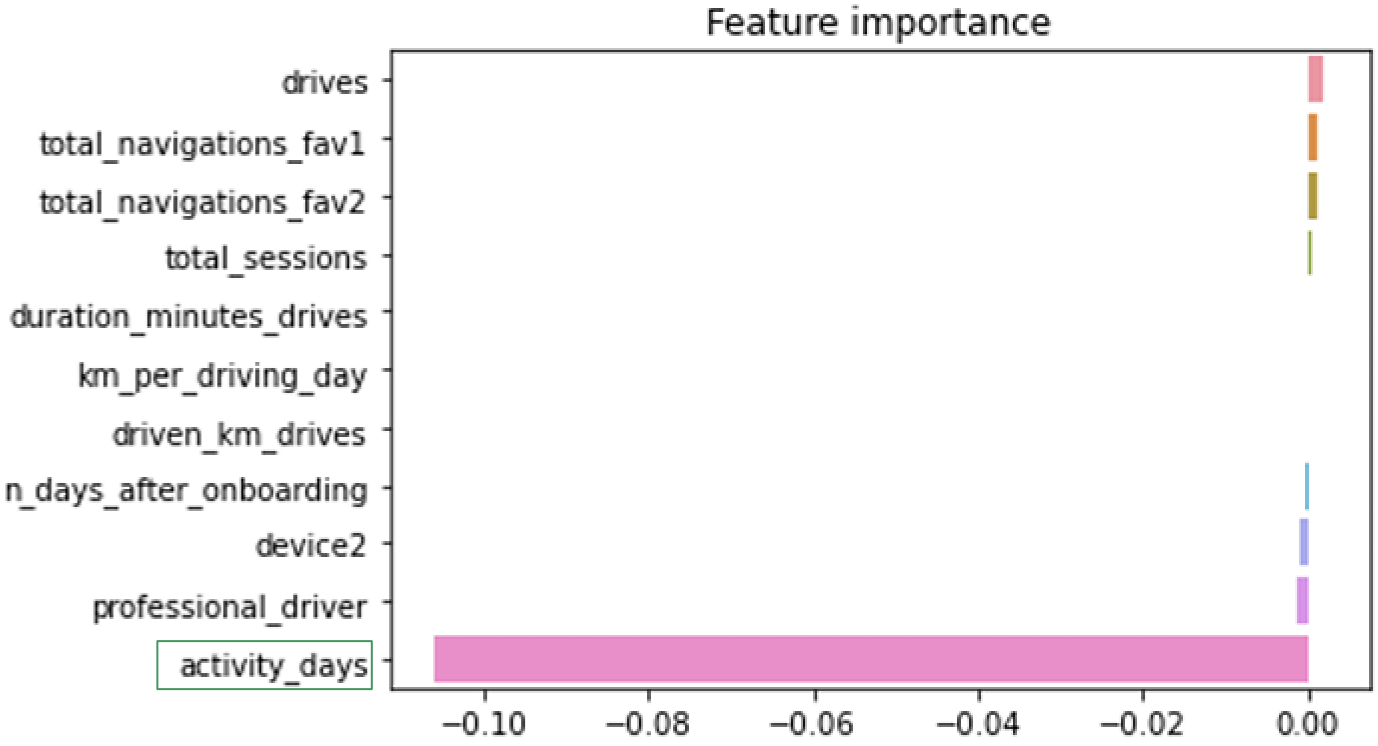
- sessions and drives: 1.0
- driving_days and activity_days: 0.95

LOGISTIC REGRESSION MODEL

```
drives                      0.001913
total_sessions              0.000327
n_days_after_onboarding    -0.000406
total_navigations_fav1      0.001232
total_navigations_fav2      0.000931
driven_km_drives           -0.000015
duration_minutes_drives     0.000109
activity_days              -0.106032
km_per_driving_day          0.000018
professional_driver        -0.001529
device2                    -0.001041
dtype: float64
```

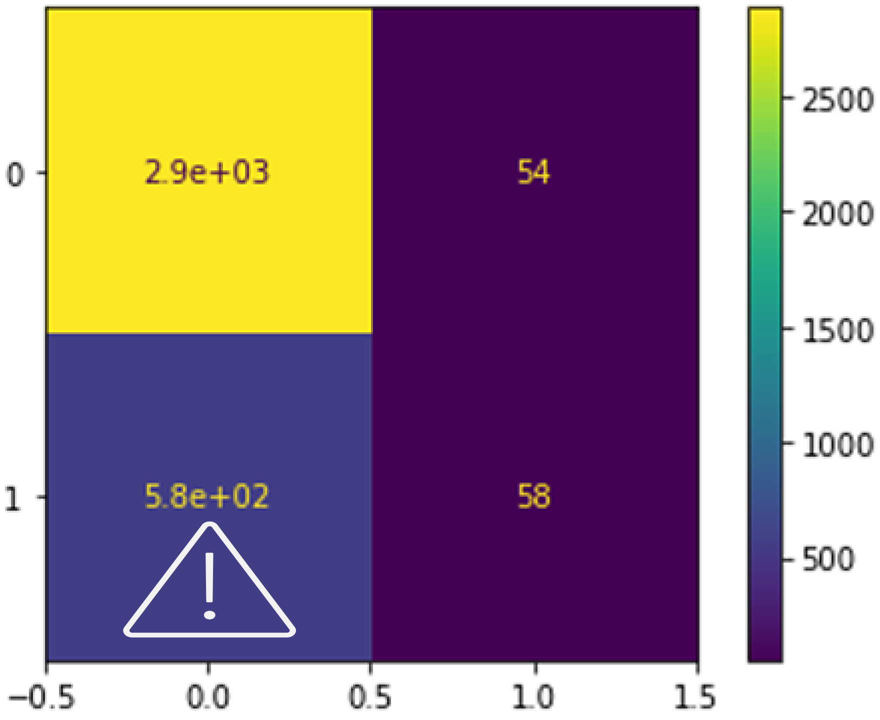
```
model.intercept_
```

```
array([-0.00170675])
```



Among all the features in the model, "activity_days" emerged as the most significant one, exhibiting a negative correlation with user churn.

LOGISTIC REGRESSION MODEL



	precision	recall	f1-score
retained	0.83	0.98	0.90
churned	0.52	0.09	0.16
accuracy			0.82
macro avg	0.68	0.54	0.53
weighted avg	0.78	0.82	0.77

Although the model demonstrates reasonable precision, its recall is extremely low, indicating a **high number of false negative predictions.**

Consequently, **it fails to identify and capture users who are likely to churn.**

LOGISTIC REGRESSION MODEL INSIGHTS

- **“Activity_days" emerged as the most significant feature**, exhibiting a negative correlation with user churn.
    - This finding is not unexpected since "activity_days" is highly correlated with "driving_days," which was already identified to have a negative correlation with churn.
- During EDA, the user churn rate rose in conjunction with increasing values in **"km_per_driving_day."**
    - The correlation heatmap confirmed this observation, indicating that this variable exhibited the highest positive correlation with churn among all the predictor variables.
    - Surprisingly, **in the model, "km_per_driving_day" ranked as the second-least important variable.**

LOGISTIC REGRESSION MODEL IMPROVEMENTS

- By leveraging domain knowledge, it is possible to engineer new features aimed at improving predictive signal.
  - In the context of this model, one of the engineered features, namely "professional_driver," emerged as the third-most influential predictor.
  - Scaling the predictor variables and reconstructing the model using different combinations of predictors can be beneficial in minimizing noise stemming from unpromising features.

- Possessing drive-level specifics for individual users, such as drive times and geographic locations would be beneficial.

- Obtaining more detailed information regarding how users engage with the app would likely provide valuable insights.

- Having knowledge of the monthly count of distinct starting and ending locations inputted by each driver could offer valuable additional information.

LOGISTIC REGRESSION MODEL RECOMMENDATION

The usefulness of the model depends on its intended purpose.

- If the model is employed to inform critical business decisions, its performance may not be sufficiently strong, particularly evident from its low recall score.

- If the model is primarily utilized to guide further exploratory efforts and provide insights, it can still offer value in that context.

MACHINE LEARNING MODEL
RANDOMFOREST VS. XGBOOST

	model	precision	recall	F1	accuracy
0	RF cv	0.458198	0.126782	0.198534	0.818626
0	XGB cv	0.442586	0.173468	0.248972	0.814780

- The XGBoost model not only outperformed the random forest model in terms of data fitting, but it also achieved a recall score that is nearly twice as high as the recall score obtained by the logistic regression model.
- It also demonstrates an improvement of almost 50% in recall compared to the random forest model, while maintaining similar levels of accuracy and precision.

MACHINE LEARNING MODEL
VALIDATION AND TEST

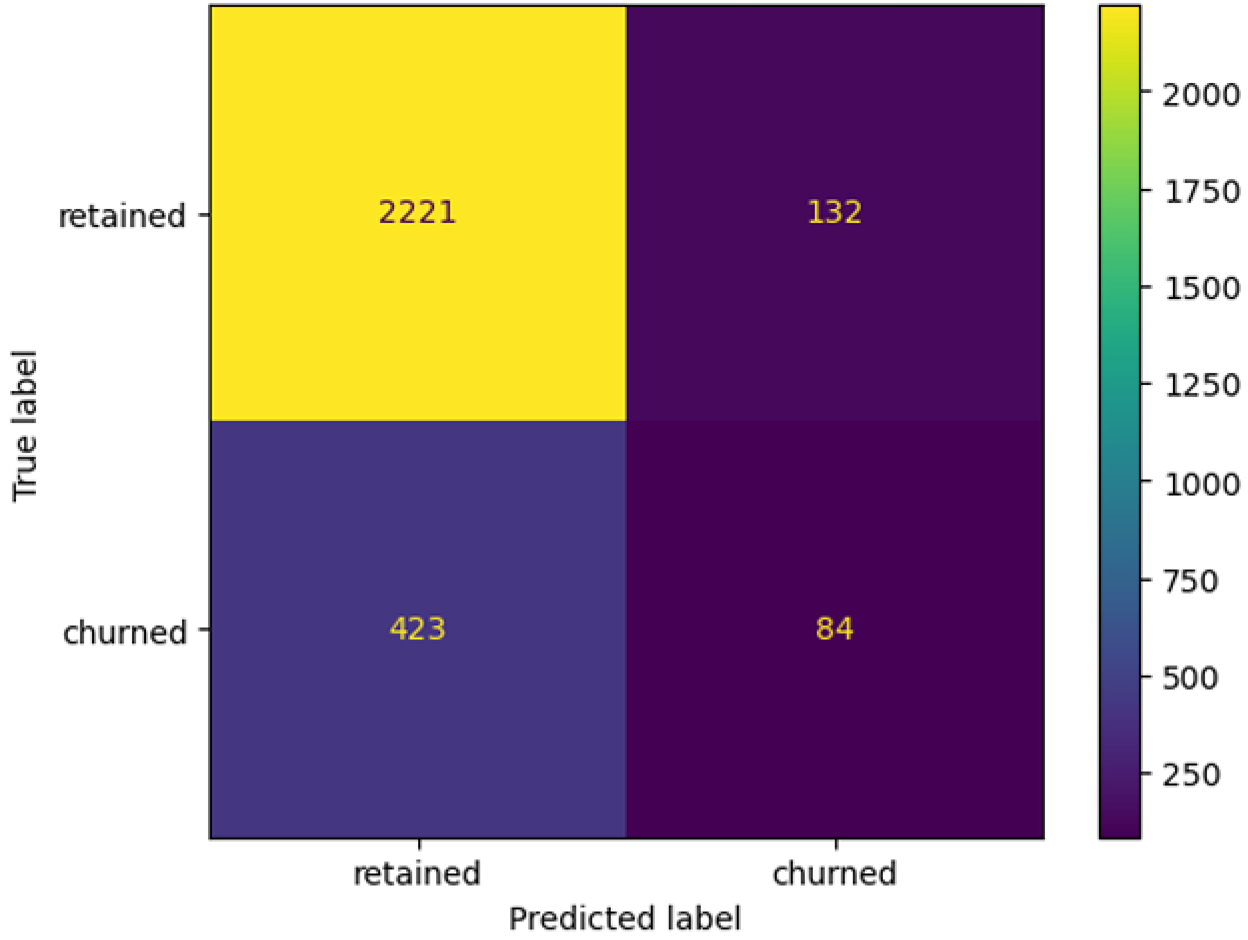
	model	precision	recall	F1	accuracy
0	RF cv	0.458198	0.126782	0.198534	0.818626
0	XGB cv	0.442586	0.173468	0.248972	0.814780
0	RF val	0.445255	0.120316	0.189441	0.817483
0	XGB val	0.430769	0.165680	0.239316	0.813287
0	XGB test	0.388889	0.165680	0.232365	0.805944

- The recall remained unchanged from the validation data, while the precision experienced a significant decline, resulting in a slight drop in all other scores.

- Nevertheless, these variations fall within an acceptable range for performance disparities between validation and test scores.

MACHINE LEARNING MODEL

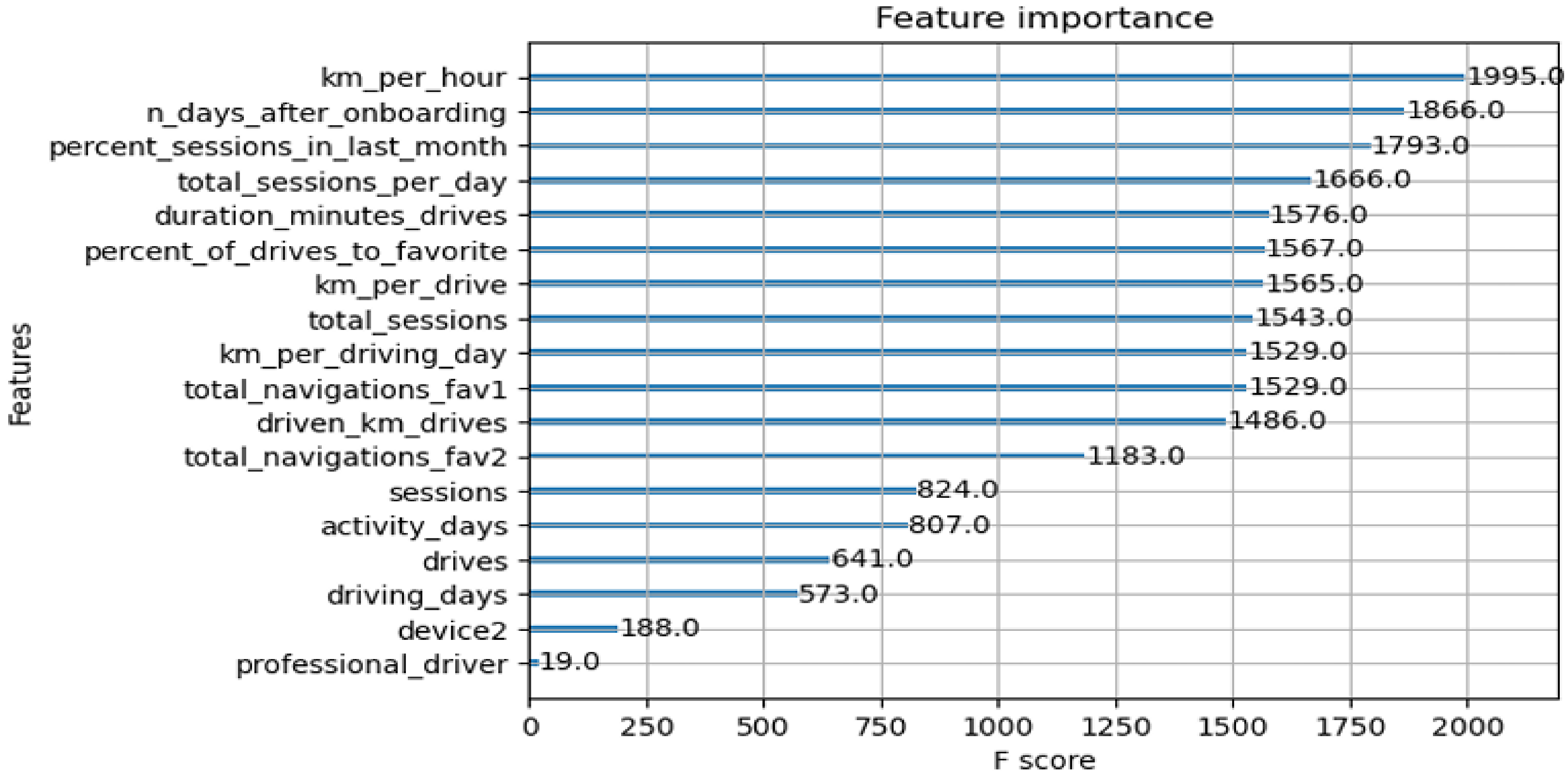
VALIDATION AND TEST



- The model's false negatives outnumbered false positives by a factor of three.
- It accurately identified only 16.6% of the users who churned.

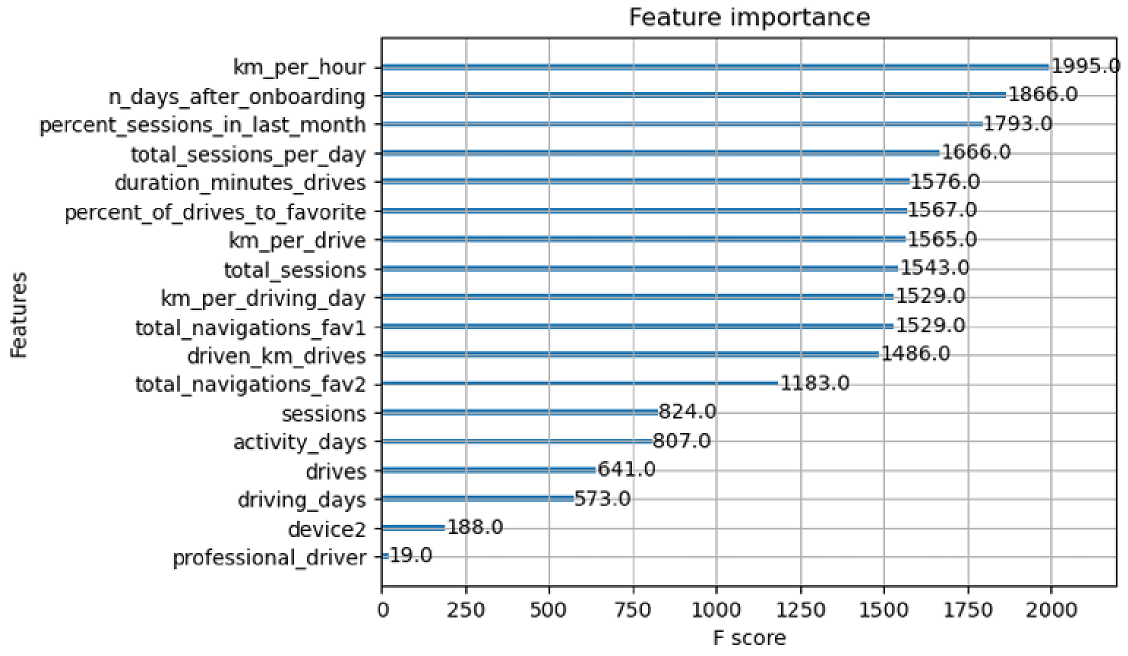
MACHINE LEARNING MODEL

FEATURE IMPORTANCE



Top Five Most Important Features That Impact Churn:

1. km_per_hour
2. n_days_after_onboarding
3. percent_sessions_in_last_month
4. total_sessions_per_day
5. duration_minutes_drives



- The XGBoost model utilized a greater number of features compared to the logistic regression model.
- Engineered features comprised six out of the top 10 features, including three out of the top five.
- It is worth noting that the selection of important features can vary between different models due to the complexity involved in feature selection.

MACHINE LEARNING MODEL

IMPROVEMENTS THAT CAN BE MADE

- Introducing new features could enhance the model's predictive capabilities, particularly with better domain knowledge.

- In the case of this model, engineered features accounted for over half of the top 10 most-predictive features employed by the model.

- Reconstructing the model using different combinations of predictor variables can help reduce noise originating from non-predictive features.

MACHINE LEARNING MODEL
ADDITIONAL FEATURES THAT COULD HELP IMPROVE THE MODEL

- Having drive-level information for each user, such as drive times and geographic locations, would be beneficial.
- More detailed data providing insights into user interactions with the app would be valuable.
- Knowing the monthly count of unique starting and ending locations provided by each driver could offer further assistance.

FINAL RECOMMENDATION

- If the model is to be utilized for significant business decisions, then it falls short in being an ideal predictor, as evidenced by its low recall score.
- If the model is solely employed to guide exploratory efforts, it can provide value.
- The model could be more predictive if we gather more drive level data as mentioned previously, as well as exploring different engineered features.

```python
### ALL WAZE USER CHURN CODE

# Waze 2 code-----------------------------------------------------------------

# Import packages for data manipulation
import pandas as pd
import numpy as np

# Load dataset into dataframe
df = pd.read_csv('waze_dataset.csv')

df.head(10)

df.info()

# Isolate rows with null values
null_df = df[df['label'].isnull()]
# Display summary stats of rows with null values
null_df.describe()

# Isolate rows without null values
not_null_df = df[~df['label'].isnull()]
# Display summary stats of rows without null values
not_null_df.describe()

# Get count of null values by device
null_df['device'].value_counts()

# Calculate % of iPhone nulls and Android nulls
null_df['device'].value_counts(normalize=True)

# Calculate % of iPhone users and Android users in full dataset
df['device'].value_counts(normalize=True)

# Calculate counts of churned vs. retained
print(df['label'].value_counts())
print()
print(df['label'].value_counts(normalize=True))

# Calculate median values of all columns for churned and retained users
df.groupby('label').median(numeric_only=True)

# Group data by `label` and calculate the medians
medians_by_label = df.groupby('label').median(numeric_only=True)
print('Median kilometers per drive:')
# Divide the median distance by median number of drives
medians_by_label['driven_km_drives'] / medians_by_label['drives']

# Divide the median distance by median number of driving days
print('Median kilometers per driving day:')
medians_by_label['driven_km_drives'] / medians_by_label['driving_days']


# Divide the median number of drives by median number of driving days
print('Median drives per driving day:')
medians_by_label['drives'] / medians_by_label['driving_days']

# For each label, calculate the number of Android users and iPhone users
df.groupby(['label', 'device']).size()

# For each label, calculate the percentage of Android users and iPhone users
df.groupby('label')['device'].value_counts(normalize=True)

# Waze 3 code-----------------------------------------------------------------
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
import seaborn as sns

# Load the dataset into a dataframe
df = pd.read_csv('waze_dataset.csv')

df.head(10)

df.size

df.describe()

df.info()

# Box plot
plt.figure(figsize=(5,1))
sns.boxplot(x=df['sessions'], fliersize=1)
plt.title('sessions box plot');


# Histogram
```

```python
plt.figure(figsize=(5,3))
sns.histplot(x=df['sessions'])
median = df['sessions'].median()
plt.axvline(median, color='red', linestyle='--')
plt.text(75,1200, 'median=56.0', color='red')
plt.title('sessions box plot');

# Box plot
plt.figure(figsize=(5,1))
sns.boxplot(x=df['drives'], fliersize=1)
plt.title('drives box plot');


# Helper function to plot histograms based on the
# format of the `sessions` histogram
def histogrammer(column_str, median_text=True, **kwargs):    # **kwargs = any keyword arguments
                                                             # from the sns.histplot() function
    median=round(df[column_str].median(), 1)
    plt.figure(figsize=(5,3))
    ax = sns.histplot(x=df[column_str], **kwargs)            # Plot the histogram
    plt.axvline(median, color='red', linestyle='--')         # Plot the median line
    if median_text==True:                                    # Add median text unless set to False
        ax.text(0.25, 0.85, f'median={median}', color='red',
            ha="left", va="top", transform=ax.transAxes)
    else:
        print('Median:', median)
    plt.title(f'{column_str} histogram');


# Histogram
histogrammer('drives')


# Box plot
plt.figure(figsize=(5,1))
sns.boxplot(x=df['total_sessions'], fliersize=1)
plt.title('total_sessions box plot');


# Histogram
histogrammer('total_sessions')

# Box plot
plt.figure(figsize=(5,1))
sns.boxplot(x=df['n_days_after_onboarding'], fliersize=1)
plt.title('n_days_after_onboarding box plot');

# Histogram
histogrammer('n_days_after_onboarding', median_text=False)

# Box plot
plt.figure(figsize=(5,1))
sns.boxplot(x=df['driven_km_drives'], fliersize=1)
plt.title('driven_km_drives box plot');

# Histogram
histogrammer('driven_km_drives')

# Box plot
plt.figure(figsize=(5,1))
sns.boxplot(x=df['duration_minutes_drives'], fliersize=1)
plt.title('duration_minutes_drives box plot');

# Histogram
histogrammer('duration_minutes_drives')

# Box plot
plt.figure(figsize=(5,1))
sns.boxplot(x=df['activity_days'], fliersize=1)
plt.title('activity_days box plot');

# Histogram
histogrammer('activity_days', median_text=False, discrete=True)

# Box plot
plt.figure(figsize=(5,1))
sns.boxplot(x=df['driving_days'], fliersize=1)
plt.title('driving_days box plot');

# Histogram
histogrammer('driving_days', median_text=False, discrete=True)

# Pie chart
fig = plt.figure(figsize=(3,3))
data=df['device'].value_counts()
plt.pie(data,
        labels=[f'{data.index[0]}: {data.values[0]}',
```

```python
                f'{data.index[1]}: {data.values[1]}'],
        autopct='%1.1f%%'
        )
plt.title('Users by device');

# Pie chart
fig = plt.figure(figsize=(3,3))
data=df['label'].value_counts()
plt.pie(data,
        labels=[f'{data.index[0]}: {data.values[0]}',
                f'{data.index[1]}: {data.values[1]}'],
        autopct='%1.1f%%'
        )
plt.title('Count of retained vs. churned');

# Histogram
plt.figure(figsize=(12,4))
label=['driving days', 'activity days']
plt.hist([df['driving_days'], df['activity_days']],
         bins=range(0,33),
         label=label)
plt.xlabel('days')
plt.ylabel('count')
plt.legend()
plt.title('driving_days vs. activity_days');

# Histogram
plt.figure(figsize=(5,4))
sns.histplot(data=df,
             x='device',
             hue='label',
             multiple='dodge',
             shrink=0.9
             )
plt.title('Retention by device histogram');

# 1. Create `km_per_driving_day` column
df['km_per_driving_day'] = df['driven_km_drives'] / df['driving_days']

# Histogram
plt.figure(figsize=(12,5))
sns.histplot(data=df,
             x='km_per_driving_day',
             bins=range(0,1201,20),
             hue='label',
             multiple='fill')
plt.ylabel('%', rotation=0)
plt.title('Churn rate by mean km per driving day');

# Histogram
plt.figure(figsize=(12,5))
sns.histplot(data=df,
             x='driving_days',
             bins=range(1,32),
             hue='label',
             multiple='fill',
             discrete=True)
plt.ylabel('%', rotation=0)
plt.title('Churn rate per driving day');

df['percent_sessions_in_last_month'] = df['sessions'] / df['total_sessions']

df['percent_sessions_in_last_month'].median()

# Histogram
histogrammer('percent_sessions_in_last_month',
             hue=df['label'],
             multiple='layer',
             median_text=False)

df['n_days_after_onboarding'].median()

# Histogram
data = df.loc[df['percent_sessions_in_last_month']>=0.4]
plt.figure(figsize=(5,3))
sns.histplot(x=data['n_days_after_onboarding'])
plt.title('Num. days after onboarding for users with >=40% sessions in last month');

def outlier_imputer(column_name, percentile):
    # Calculate threshold
    threshold = df[column_name].quantile(percentile)
    # Impute threshold for values > than threshold
    df.loc[df[column_name] > threshold, column_name] = threshold

    print('{:>25} | percentile: {} | threshold: {}'.format(column_name, percentile, threshold))

for column in ['sessions', 'drives', 'total_sessions',
```

```python
              'driven_km_drives', 'duration_minutes_drives']:
              outlier_imputer(column, 0.95)

df.describe()

# Waze 4 code-----------------------------------------------------------------
import pandas as pd
from scipy import stats

# Load dataset into dataframe
df = pd.read_csv('waze_dataset.csv')

# 1. Create `map_dictionary`
map_dictionary = {'Android': 2, 'iPhone': 1}

# 2. Create new `device_type` column
df['device_type'] = df['device']

# 3. Map the new column to the dictionary
df['device_type'] = df['device_type'].map(map_dictionary)

df['device_type'].head()

df.groupby('device_type')['drives'].mean()

# 1. Isolate the `drives` column for iPhone users.
iPhone = df[df['device_type'] == 1]['drives']

# 2. Isolate the `drives` column for Android users.
Android = df[df['device_type'] == 2]['drives']

# 3. Perform the t-test
stats.ttest_ind(a=iPhone, b=Android, equal_var=False)

# Waze 5 code-----------------------------------------------------------------
import pandas as pd
import numpy as np

# Packages for visualization
import matplotlib.pyplot as plt
import seaborn as sns

# Packages for Logistic Regression & Confusion Matrix
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, accuracy_score, precision_score, \
recall_score, f1_score, confusion_matrix, ConfusionMatrixDisplay
from sklearn.linear_model import LogisticRegression


# Load the dataset by running this cell
df = pd.read_csv('https://raw.githubusercontent.com/adacert/waze/main/Synthetic_Waze_Data_14999%20-%20Fictional_Waze_Data_14999.csv')

print(df.shape)

df.info()

df.head()

df = df.drop('ID', axis=1)

df['label'].value_counts(normalize=True)

df.describe()

# 1. Create `km_per_driving_day` column
df['km_per_driving_day'] = df['driven_km_drives'] / df['driving_days']

# 2. Call `describe()` on the new column
df['km_per_driving_day'].describe()

# 1. Convert infinite values to zero
df.loc[df['km_per_driving_day']==np.inf, 'km_per_driving_day'] = 0

# 2. Confirm that it worked
df['km_per_driving_day'].describe()

# Create `professional_driver` column
df['professional_driver'] = np.where((df['drives'] >= 60) & (df['driving_days'] >= 15), 1, 0)

# 1. Check count of professionals and non-professionals
print(df['professional_driver'].value_counts())

# 2. Check in-class churn rate
df.groupby(['professional_driver'])['label'].value_counts(normalize=True)

df.info()
```

```python
# Drop rows with missing data in `label` column
df = df.dropna(subset=['label'])

# Impute outliers
for column in ['sessions', 'drives', 'total_sessions', 'total_navigations_fav1',
               'total_navigations_fav2', 'driven_km_drives', 'duration_minutes_drives']:
    threshold = df[column].quantile(0.95)
    df.loc[df[column] > threshold, column] = threshold

df.describe()

# Create binary `label2` column
df['label2'] = np.where(df['label']=='churned', 1, 0)
df[['label', 'label2']].tail()

# Generate a correlation matrix
df.corr(method='pearson')

# Plot correlation heatmap
plt.figure(figsize=(15,10))
sns.heatmap(df.corr(method='pearson'), vmin=-1, vmax=1, annot=True, cmap='coolwarm')
plt.title('Correlation heatmap indicates many low correlated variables',
          fontsize=18)
plt.show();

# Create new `device2` variable
df['device2'] = np.where(df['device']=='Android', 0, 1)
df[['device', 'device2']].tail()

# Isolate predictor variables
X = df.drop(columns = ['label', 'label2', 'device', 'sessions', 'driving_days'])

# Isolate target variable
y = df['label2']

# Perform the train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y, stratify=y, random_state=42)

# Use .head()
X_train.head()

model = LogisticRegression(penalty='none', max_iter=400)

model.fit(X_train, y_train)


pd.Series(model.coef_[0], index=X.columns)

model.intercept_

# Get the predicted probabilities of the training data
training_probabilities = model.predict_proba(X_train)
training_probabilities

# 1. Copy the `X_train` dataframe and assign to `logit_data`
logit_data = X_train.copy()

# 2. Create a new `logit` column in the `logit_data` df
logit_data['logit'] = [np.log(prob[1] / prob[0]) for prob in training_probabilities]

# Plot regplot of `activity_days` log-odds
sns.regplot(x='activity_days', y='logit', data=logit_data, scatter_kws={'s': 2, 'alpha': 0.5})
plt.title('Log-odds: activity_days');

# Generate predictions on X_test
y_preds = model.predict(X_test)

# Score the model (accuracy) on the test data
model.score(X_test, y_test)

cm = confusion_matrix(y_test, y_preds)

disp = ConfusionMatrixDisplay(confusion_matrix=cm,display_labels=None)
disp.plot()

# Calculate precision manually
precision = cm[1,1] / (cm[0, 1] + cm[1, 1])
precision

# Calculate recall manually
recall = cm[1,1] / (cm[1, 0] + cm[1, 1])
recall

# Create a classification report
target_labels = ['retained', 'churned']
print(classification_report(y_test, y_preds, target_names=target_labels))
```

```
# Create a list of (column_name, coefficient) tuples
feature_importance = list(zip(X_train.columns, model.coef_[0]))

# Sort the list by coefficient value
feature_importance = sorted(feature_importance, key=lambda x: x[1], reverse=True)
feature_importance

# Plot the feature importances
import seaborn as sns
sns.barplot(x=[x[1] for x in feature_importance],
            y=[x[0] for x in feature_importance],
            orient='h')
plt.title('Feature importance');

# Waze 6 code-----------------------------------------------------------------
import numpy as np
import pandas as pd

# Import packages for data visualization
import matplotlib.pyplot as plt

# This lets us see all of the columns, preventing Juptyer from redacting them.
pd.set_option('display.max_columns', None)

# Import packages for data modeling
from sklearn.model_selection import GridSearchCV, train_test_split
from sklearn.metrics import roc_auc_score, roc_curve, auc
from sklearn.metrics import accuracy_score, precision_score, recall_score,\
f1_score, confusion_matrix, ConfusionMatrixDisplay, RocCurveDisplay, PrecisionRecallDisplay

from sklearn.ensemble import RandomForestClassifier
from xgboost import XGBClassifier

# This is the function that helps plot feature importance
from xgboost import plot_importance

# This module lets us save our models once we fit them.
import pickle

# from google.colab import drive
# drive.mount('/content/drive', force_remount=True)

# Import dataset
df0 = pd.read_csv('waze_dataset.csv')

# Inspect the first five rows
df0.head()

# Copy the df0 dataframe
df = df0.copy()

df.info()

# 1. Create `km_per_driving_day` feature
df['km_per_driving_day'] = df['driven_km_drives'] / df['driving_days']

# 2. Get descriptive stats
df['km_per_driving_day'].describe()

# 1. Convert infinite values to zero
df.loc[df['km_per_driving_day']==np.inf, 'km_per_driving_day'] = 0

# 2. Confirm that it worked
df['km_per_driving_day'].describe()

# 1. Create `percent_sessions_in_last_month` feature
df['percent_sessions_in_last_month'] = df['sessions'] / df['total_sessions']

# 2. Get descriptive stats
df['percent_sessions_in_last_month'].describe()

# Create `professional_driver` feature
df['professional_driver'] = np.where((df['drives'] >= 60) & (df['driving_days'] >= 15), 1, 0)

# Create `total_sessions_per_day` feature
df['total_sessions_per_day'] = df['total_sessions'] / df['n_days_after_onboarding']

# Get descriptive stats
df['total_sessions_per_day'].describe()

# Create `km_per_hour` feature
df['km_per_hour'] = df['driven_km_drives'] / df['duration_minutes_drives'] / 60
df['km_per_hour'].describe()

# Create `km_per_drive` feature
df['km_per_drive'] = df['driven_km_drives'] / df['drives']
```

```python
df['km_per_drive'].describe()

# 1. Convert infinite values to zero
df.loc[df['km_per_drive']==np.inf, 'km_per_drive'] = 0

# 2. Confirm that it worked
df['km_per_drive'].describe()

# Create `percent_of_sessions_to_favorite` feature
df['percent_of_drives_to_favorite'] = (
    df['total_navigations_fav1'] + df['total_navigations_fav2']) / df['total_sessions']

# Get descriptive stats
df['percent_of_drives_to_favorite'].describe()

# Drop rows with missing values
df = df.dropna(subset=['label'])

# Create new `device2` variable
df['device2'] = np.where(df['device']=='Android', 0, 1)
df[['device', 'device2']].tail()

# Create binary `label2` column
df['label2'] = np.where(df['label']=='churned', 1, 0)
df[['label', 'label2']].tail()

# Drop `ID` column
df = df.drop(['ID'], axis=1)

# Get class balance of 'label' col
df['label'].value_counts(normalize=True)

# 1. Isolate X variables
X = df.drop(columns=['label', 'label2', 'device'])

# 2. Isolate y variable
y = df['label2']

# 3. Split into train and test sets
X_tr, X_test, y_tr, y_test = train_test_split(X, y, stratify=y,
                                              test_size=0.2, random_state=42)

# 4. Split into train and validate sets
X_train, X_val, y_train, y_val = train_test_split(X_tr, y_tr, stratify=y_tr,
                                                  test_size=0.25, random_state=42)

for x in [X_train, X_val, X_test]:
    print(len(x))

# 1. Instantiate the random forest classifier
rf = RandomForestClassifier(random_state=42)

# 2. Create a dictionary of hyperparameters to tune
cv_params = {'max_depth': [None],
             'max_features': [1.0],
             'max_samples': [1.0],
             'min_samples_leaf': [2],
             'min_samples_split': [2],
             'n_estimators': [300],
             }

# 3. Define a dictionary of scoring metrics to capture
scoring = {'accuracy', 'precision', 'recall', 'f1'}

# 4. Instantiate the GridSearchCV object
rf_cv = GridSearchCV(rf, cv_params, scoring=scoring, cv=4, refit='recall')

%%time
rf_cv.fit(X_train, y_train)

# Examine best score
rf_cv.best_score_

# Examine best hyperparameter combo
rf_cv.best_params_

def make_results(model_name:str, model_object, metric:str):
    '''
    Arguments:
        model_name (string): what you want the model to be called in the output table
        model_object: a fit GridSearchCV object
        metric (string): precision, recall, f1, or accuracy

    Returns a pandas df with the F1, recall, precision, and accuracy scores
    for the model with the best mean 'metric' score across all validation folds.
    '''
```

```python
    # Create dictionary that maps input metric to actual metric name in GridSearchCV
    metric_dict = {'precision': 'mean_test_precision',
                   'recall': 'mean_test_recall',
                   'f1': 'mean_test_f1',
                   'accuracy': 'mean_test_accuracy',
                   }

    # Get all the results from the CV and put them in a df
    cv_results = pd.DataFrame(model_object.cv_results_)

    # Isolate the row of the df with the max(metric) score
    best_estimator_results = cv_results.iloc[cv_results[metric_dict[metric]].idxmax(), :]

    # Extract accuracy, precision, recall, and f1 score from that row
    f1 = best_estimator_results.mean_test_f1
    recall = best_estimator_results.mean_test_recall
    precision = best_estimator_results.mean_test_precision
    accuracy = best_estimator_results.mean_test_accuracy

    # Create table of results
    table = pd.DataFrame({'model': [model_name],
                          'precision': [precision],
                          'recall': [recall],
                          'F1': [f1],
                          'accuracy': [accuracy],
                          },
                         )

    return table

results = make_results('RF cv', rf_cv, 'recall')
results

# 1. Instantiate the XGBoost classifier
xgb = XGBClassifier(objective='binary:logistic', random_state=42)

# 2. Create a dictionary of hyperparameters to tune
cv_params = {'max_depth': [6, 12],
             'min_child_weight': [3, 5],
             'learning_rate': [0.01, 0.1],
             'n_estimators': [300]
             }

# 3. Define a dictionary of scoring metrics to capture
scoring = {'accuracy', 'precision', 'recall', 'f1'}

# 4. Instantiate the GridSearchCV object
xgb_cv = GridSearchCV(xgb, cv_params, scoring=scoring, cv=4, refit='recall')

%%time
xgb_cv.fit(X_train, y_train)

# Examine best score
xgb_cv.best_score_

# Examine best parameters
xgb_cv.best_params_

# Call 'make_results()' on the GridSearch object
xgb_cv_results = make_results('XGB cv', xgb_cv, 'recall')
results = pd.concat([results, xgb_cv_results], axis=0)
results

# Use random forest model to predict on validation data
rf_val_preds = rf_cv.best_estimator_.predict(X_val)

def get_test_scores(model_name:str, preds, y_test_data):
    '''
    Generate a table of test scores.

    In:
        model_name (string): Your choice: how the model will be named in the output table
        preds: numpy array of test predictions
        y_test_data: numpy array of y_test data

    Out:
        table: a pandas df of precision, recall, f1, and accuracy scores for your model
    '''
    accuracy = accuracy_score(y_test_data, preds)
    precision = precision_score(y_test_data, preds)
    recall = recall_score(y_test_data, preds)
    f1 = f1_score(y_test_data, preds)

    table = pd.DataFrame({'model': [model_name],
                          'precision': [precision],
                          'recall': [recall],
                          'F1': [f1],
```

```python
                          'accuracy': [accuracy]
                          })

    return table

# Get validation scores for RF model
rf_val_scores = get_test_scores('RF val', rf_val_preds, y_val)

# Append to the results table
results = pd.concat([results, rf_val_scores], axis=0)
results

# Use XGBoost model to predict on validation data
xgb_val_preds = xgb_cv.best_estimator_.predict(X_val)

# Get validation scores for XGBoost model
xgb_val_scores = get_test_scores('XGB val', xgb_val_preds, y_val)

# Append to the results table
results = pd.concat([results, xgb_val_scores], axis=0)
results

# Use XGBoost model to predict on test data
xgb_test_preds = xgb_cv.best_estimator_.predict(X_test)

# Get test scores for XGBoost model
xgb_test_scores = get_test_scores('XGB test', xgb_test_preds, y_test)

# Append to the results table
results = pd.concat([results, xgb_test_scores], axis=0)
results

# Generate array of values for confusion matrix
cm = confusion_matrix(y_test, xgb_test_preds, labels=xgb_cv.classes_)

# Plot confusion matrix
disp = ConfusionMatrixDisplay(confusion_matrix=cm,
                              display_labels=['retained', 'churned'])
disp.plot();

plot_importance(xgb_cv.best_estimator_);

# Plot precision-recall curve
display = PrecisionRecallDisplay.from_estimator(
    xgb_cv.best_estimator_, X_test, y_test, name='XGBoost'
    )
plt.title('Precision-recall curve, XGBoost model');

# Get predicted probabilities on the test data
predicted_probabilities = xgb_cv.best_estimator_.predict_proba(X_test)
predicted_probabilities

# Create a list of just the second column values (probability of target)
probs = [x[1] for x in predicted_probabilities]

# Create an array of new predictions that assigns a 1 to any value >= 0.4
new_preds = np.array([1 if x >= 0.4 else 0 for x in probs])
new_preds

# Get evaluation metrics for when the threshold is 0.4
get_test_scores('XGB, threshold = 0.4', new_preds, y_test)

results

def threshold_finder(y_test_data, probabilities, desired_recall):
    '''
    Find the threshold that most closely yields a desired recall score.

    Inputs:
        y_test_data: Array of true y values
        probabilities: The results of the `predict_proba()` model method
        desired_recall: The recall that you want the model to have

    Outputs:
        threshold: The threshold that most closely yields the desired recall
        recall: The exact recall score associated with `threshold`
    '''
    probs = [x[1] for x in probabilities]  # Isolate second column of `probabilities`
    thresholds = np.arange(0, 1, 0.001)    # Set a grid of 1,000 thresholds to test

    scores = []
    for threshold in thresholds:
        # Create a new array of {0, 1} predictions based on new threshold
        preds = np.array([1 if x >= threshold else 0 for x in probs])
        # Calculate recall score for that threshold
        recall = recall_score(y_test_data, preds)
        # Append the threshold and its corresponding recall score as a tuple to `scores`
```

```python
        scores.append((threshold, recall))

    distances = []
    for idx, score in enumerate(scores):
        # Calculate how close each actual score is to the desired score
        distance = abs(score[1] - desired_recall)
        # Append the (index#, distance) tuple to `distances`
        distances.append((idx, distance))

    # Sort `distances` by the second value in each of its tuples (least to greatest)
    sorted_distances = sorted(distances, key=lambda x: x[1], reverse=False)
    # Identify the tuple with the actual recall closest to desired recall
    best = sorted_distances[0]
    # Isolate the index of the threshold with the closest recall score
    best_idx = best[0]
    # Retrieve the threshold and actual recall score closest to desired recall
    threshold, recall = scores[best_idx]

    return threshold, recall


# Get the predicted probabilities from the champion model
probabilities = xgb_cv.best_estimator_.predict_proba(X_test)

# Call the function
threshold_finder(y_test, probabilities, 0.5)

# Create an array of new predictions that assigns a 1 to any value >= 0.124
new_preds = np.array([1 if x >= 0.124 else 0 for x in probs])

# Get evaluation metrics for when the threshold is 0.124
get_test_scores('XGB, threshold = 0.124', new_preds, y_test)
```