Abdallah Zubedi
7/25/2023

0.1 Table of Contents

1 Waze_Project

1.1 Objectives

> - Understanding churn rate
> - Assessing relationships with churn rate and other variables
> - Create a model that could potentially predict churn rate

This project shows the process of data analysis but also includes regression modelling and machine learning techniques to showcase the skillsets I have.

I have added conclusions to each section, some are slighlty repeated as each section can have some overlap.

Even though they can be looked into seperately, I have attempted to create a flow as a case scenerio.

You will also notice library imports happening within the sections it is needed isntead of an initial mass import of libraries and packages.

The source of the data was provided by Google and does not necessesarily represent actual data

1.2 Understanding_the_Dataset

In [579]:
```
# Import packages for data manipulation

import pandas as pd
import numpy as np

# Ignore warnings

import warnings
warnings.filterwarnings('ignore')
```

In [580]:
```
# Load dataset into dataframe
OG_waze = pd.read_csv(r'C:\Users\dalla\My Python Stuff\Waze Data Project\waze_
```

In [581]:
```
# Understand data frame
OG_waze.head(10)
```

Out[581]:

	ID	label	sessions	drives	total_sessions	n_days_after_onboarding	total_navigations_fav1
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 [582]:
```
OG_waze.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
```

In [583]:
```
# Check for number of null rows
num_null_rows = OG_waze.isnull().sum()

print(num_null_rows)
```

```
ID                           0
label                      700
sessions                     0
drives                       0
total_sessions               0
n_days_after_onboarding      0
total_navigations_fav1       0
total_navigations_fav2       0
driven_km_drives             0
duration_minutes_drives      0
activity_days                0
driving_days                 0
device                       0
dtype: int64
```

**Initial summary of dataset

1) There are 14,999 rows and 13 columns 2) There are 3 main data types, (total_sessions, driven_km_drives and duration_minutes_drives are) floats64, (ID, sessions, drives, n_days_after_onboarding, fav1 and fav2 activity days and driving days are) int64, (labels,device) object data types. 3) There are 700 rows with missing values and they are all in the label column

In [584]:
```
# Isolating null values
null_OG = OG_waze[OG_waze['label'].isnull()]

# Display summary stats of rows with null values
null_OG.describe()
```

Out[584]:

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

In [585]:
```
# Isolaing rows without null values
nonull_OG = OG_waze[~OG_waze ['label'].isnull()]

#Display non null values
nonull_OG.describe()
```

Out[585]:

	ID	sessions	drives	total_sessions	n_days_after_onboarding	total_
count	14299.000000	14299.000000	14299.000000	14299.000000	14299.000000	
mean	7503.573117	80.623820	67.255822	189.547409	1751.822505	
std	4331.207621	80.736502	65.947295	136.189764	1008.663834	
min	0.000000	0.000000	0.000000	0.220211	4.000000	
25%	3749.500000	23.000000	20.000000	90.457733	878.500000	
50%	7504.000000	56.000000	48.000000	158.718571	1749.000000	
75%	11257.500000	111.000000	93.000000	253.540450	2627.500000	
max	14998.000000	743.000000	596.000000	1216.154633	3500.000000	

No differences in data between non-null and null values.

In [586]:
```
#Comparing Iphone and Android Null values
null_OG['device'].value_counts()
```

Out[586]:
```
iPhone     447
Android    253
Name: device, dtype: int64
```

Out of the total 700 null values, 447 were using Iphone devices and 253 were using Android devices.

In [587]:
```
# Percentage of Iphone and Andoird null values
null_OG['device'].value_counts(normalize=True)
```

Out[587]:
```
iPhone     0.638571
Android    0.361429
Name: device, dtype: float64
```

In [588]:
```
# Calculate % of iPhone users and Android users in full dataset
OG_waze['device'].value_counts(normalize=True)
```

Out[588]:
```
iPhone     0.644843
Android    0.355157
Name: device, dtype: float64
```

The data is consistent and there is nothing that suggests a patters in terms of non-randomness of the missing data.

In [589]:
```
# Calculate counts of churned vs. retained
print(OG_waze['label'].value_counts())
print()
print(OG_waze['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 82% retained users and 18% churned users.

In [590]:
```
# Median of all columns for retained and churned users

OG_median = OG_waze.groupby('label').median()

print(OG_median)
```

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

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

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

In [591]:
```
percent_OG_median = OG_median.divide(OG_median.sum(axis=1), axis=0) * 100

print(percent_OG_median)
```

```
                ID  sessions    drives  total_sessions  \
label
churned   51.779015  0.408554  0.346232        1.137989
retained  51.279363  0.382427  0.320966        1.076168

          n_days_after_onboarding  total_navigations_fav1  \
label
churned                  9.147453                0.585132
retained                12.585946                0.464376

          total_navigations_fav2  driven_km_drives  duration_minutes_drives
\
label
churned                 0.076171         25.293335                11.129173
retained                0.061461         23.660517                 9.957075

          activity_days  driving_days
label
churned        0.055397      0.041548
retained       0.116094      0.095607
```

Churned users have more total sessions, and usage of waze then retained users in a shorter amount of time (n_days_after_onboarding).

In [592]:
```
# Group data by `label` and calculate the medians
medians_by_label = OG_waze.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']
```

```
Median kilometers per drive:
```

Out[592]:
```
label
churned     73.053113
retained    73.716694
dtype: float64
```

the median for both and retained and churned are basically the same

In [593]:
```
# 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']
```

```
Median kilometers per driving day:
```

Out[593]:
```
label
churned     608.775944
retained    247.477472
dtype: float64
```

In [594]:
```
# 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']
```

```
Median drives per driving day:
```

Out[594]:
```
label
churned     8.333333
retained    3.357143
dtype: float64
```

The median user who churned drove 608 kilometers each day they drove last month, which is almost 250% the per-drive-day distance of retained users. The median churned user had a similarly disproporionate number of drives per drive day compared to retained users.

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

In [595]:
```
# For each label, calculate the number of Android users and iPhone users
OG_waze.groupby(['label', 'device']).size()
```

Out[595]:
```
label     device
churned   Android     891
          iPhone     1645
retained  Android    4183
          iPhone     7580
dtype: int64
```

In [596]:
```
# Percentage of each device in retained and churned
OG_waze.groupby('label')['device'].value_counts(normalize=True)
```

Out[596]:
```
label     device
churned   iPhone     0.648659
          Android    0.351341
retained  iPhone     0.644393
          Android    0.355607
Name: device, dtype: float64
```

in percentages it seems they are both similar in each device if its grouped by churned or retained

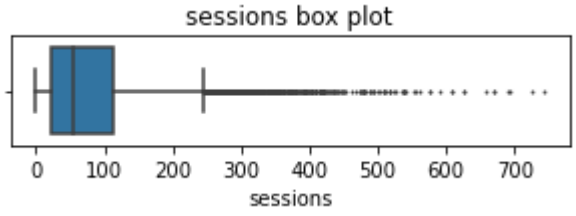
2 Exploratory_Data_Analysis

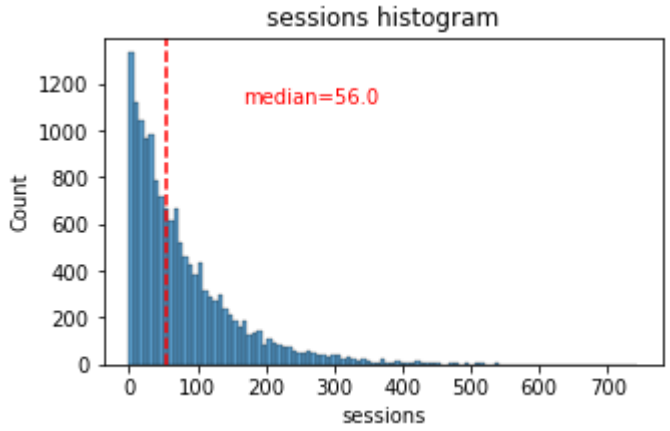
Assessing outliers and existing patterns.

In [597]:
```
# importing more packages for EDA stage
import matplotlib.pyplot as plt
import seaborn as sns
```

In [598]:
```
# Defining a histogram funtion to reduce repetitiveness
# To investigate each column as a historgram and its median

def histogrammer(column_str, median_text=True, **kwargs):    # **kwargs = any
                                                             # from the sns.hi
    median=round(OG_waze[column_str].median(), 1)
    plt.figure(figsize=(5,3))
    ax = sns.histplot(x=OG_waze[column_str], **kwargs)           # Plot the h
    plt.axvline(median, color='red', linestyle='--')         # Plot the median
    if median_text==True:                                    # Add median text
        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');
```

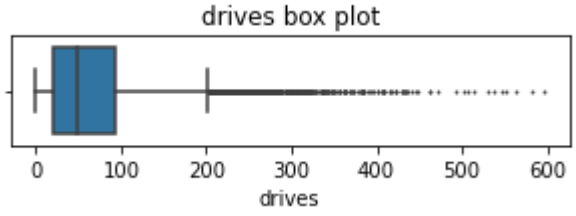
In [599]:
```
# Box plot
plt.figure(figsize=(5,1))
sns.boxplot(x=OG_waze['sessions'], fliersize=1)
plt.title('sessions box plot');

# Histogram for sessions
histogrammer('sessions')
```

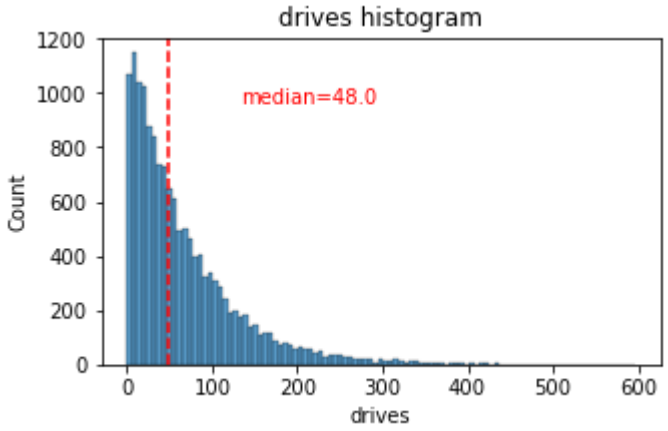




The sessions variable is a right-skewed distribution with half of the observations having 56 or fewer sessions. However, as indicated by the boxplot, some users have more than 700.

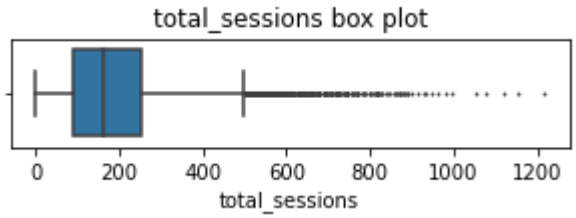
In [600]:
```
#Box plot for sessions
plt.figure(figsize=(5,1))
sns.boxplot(x=OG_waze['drives'], fliersize=1)
plt.title('drives box plot');

#Histogram for drives
histogrammer('drives')
```

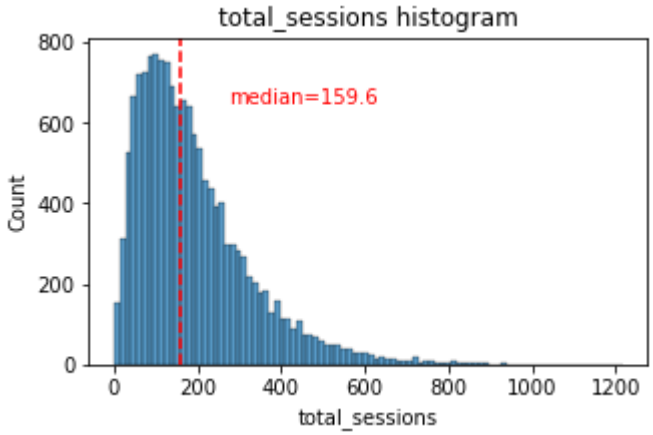




The drives information follows a distribution similar to the sessions variable. It is right-skewed, with a median of 48. However, some drivers had over 400 drives in the last month as seen n the box plot

In [601]:
```
# Box plot
plt.figure(figsize=(5,1))
sns.boxplot(x=OG_waze['total_sessions'], fliersize=1)
plt.title('total_sessions box plot');

#Histogram for total_sessions
histogrammer('total_sessions')
```

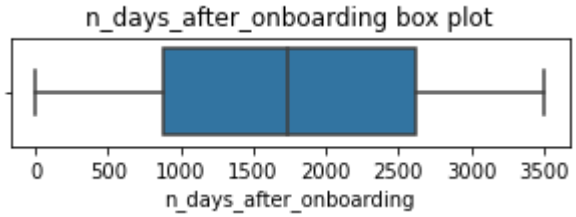


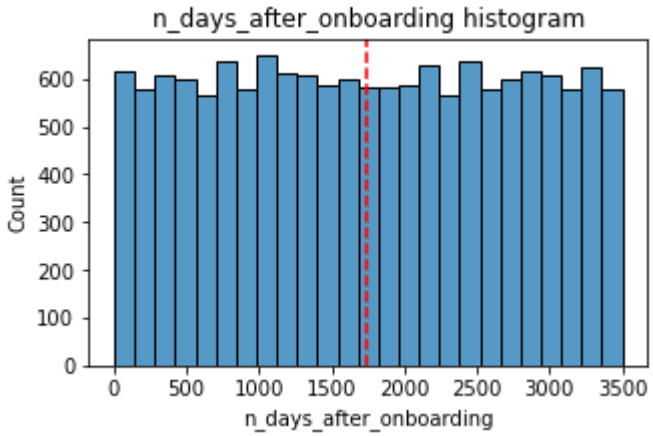


The total sessions are rightly skewed. With the median of total sessions being 159.6. As th emedian drives was 48, comapred to the total sessions median of 159.6, this means that majority of drives occured last month

In [602]:
```
# Box plot for days after onboarding
plt.figure(figsize=(5,1))
sns.boxplot(x=OG_waze['n_days_after_onboarding'], fliersize=1)
plt.title('n_days_after_onboarding box plot');

#Historgram for days after onboarding
histogrammer('n_days_after_onboarding', median_text =False)
```

```
Median: 1741.0
```



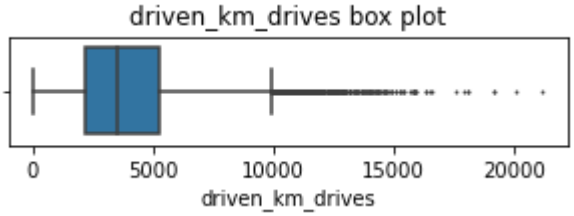


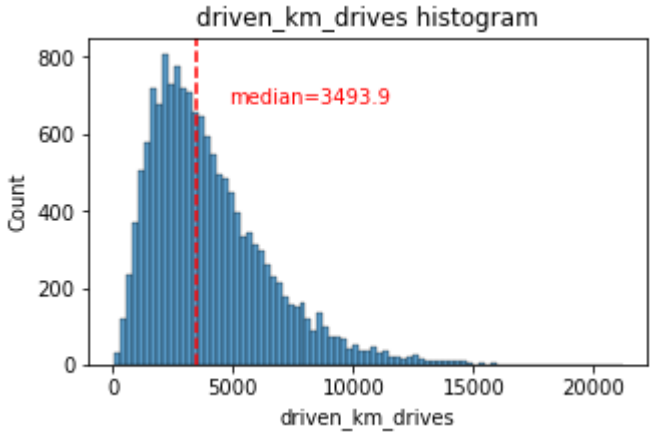
The n_days_after_onboarding is a uniform distribution. starting from almost 0 to 3500 days.

I decided to add on a boxplot function as there was repetition for this, ideally it should have been done earlier. The following function is for a boxplot specifically for the OG_waze dataset.

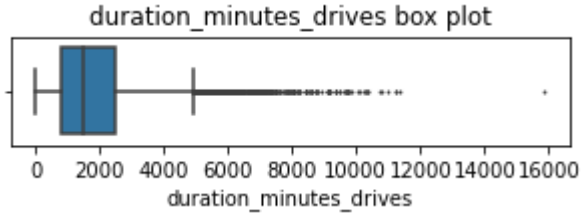
In [603]:
```
# Creating boxplot function and its parameter description.
def boxplotter(column, data=OG_waze, figsize=(5, 1), title=None):
    """
    Function to plot a box plot for a specified column in the data.

    Parameters:
    - column (str): Name of the column to plot.
    - data (DataFrame, optional): Data to plot. Defaults to OG_waze.
    - figsize (tuple, optional): Figure size. Defaults to (5, 1).
    - title (str, optional): Title for the plot. If None, default title will b

    Returns:
    - None
    """
    plt.figure(figsize=figsize)
    sns.boxplot(x=data[column], fliersize=1)
    if title:
        plt.title(title)
    else:
        plt.title(f'{column} box plot')
    plt.show()
```

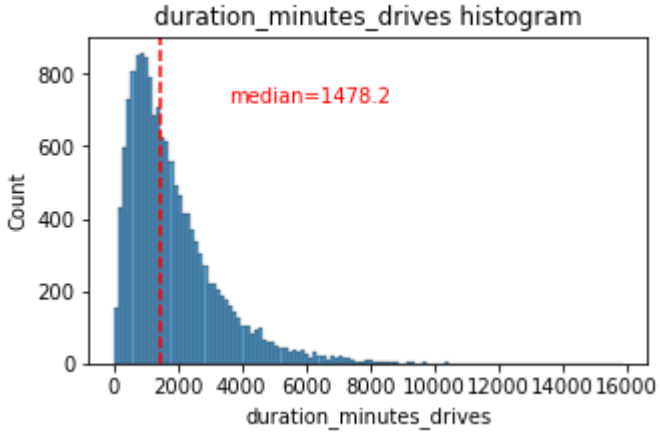
In [604]:
```
boxplotter('driven_km_drives')
histogrammer('driven_km_drives')
```





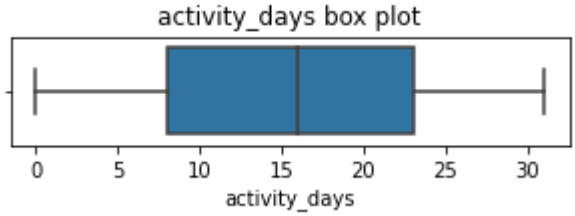
In [605]:
```
# box plit for duration_minutes_drives
boxplotter('duration_minutes_drives')

#Histogram for duration_minutes_drives
histogrammer('duration_minutes_drives')
```

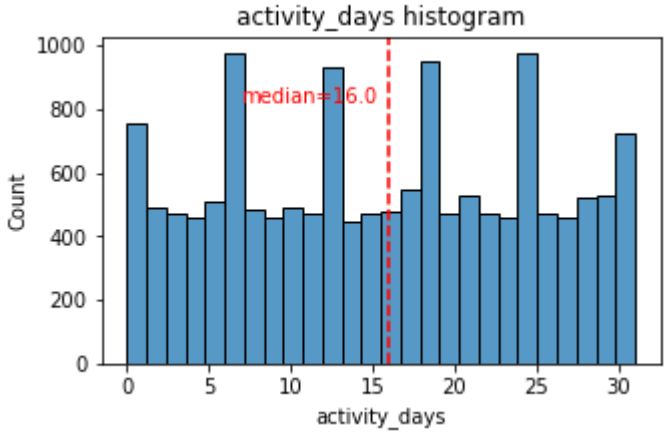




The duration_minutes_drives is skewed to the right. Half of the users drove less then 1482 min.

In [606]:
```
#boxplot for activity_days
boxplotter('activity_days')

#histogram for activity_days
histogrammer('activity_days')
```

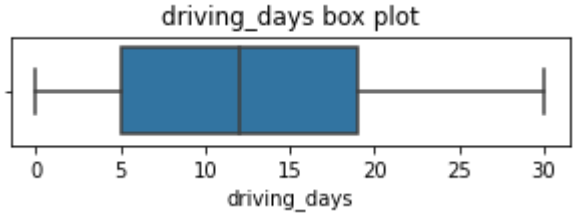


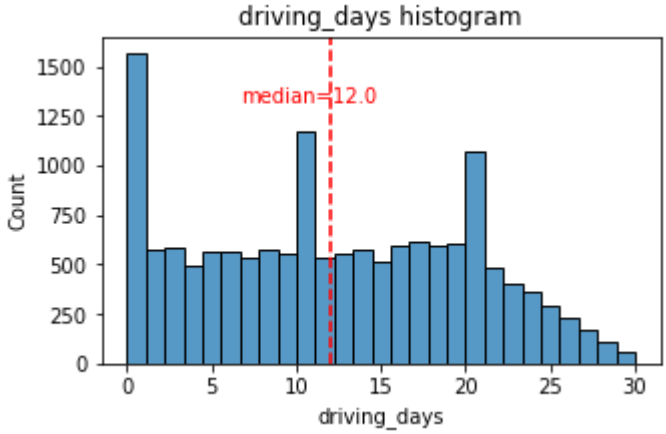


Over the past month, the median frequency of users launching the app stands at 16 times. The box plot indicates a balanced distribution. The histogram illustrates an almost consistent distribution, with around 500 users accessing the app for each specific day count. Intriguingly, about 250 users never accessed the app, while a similar number used it daily.

What's remarkable about this distribution is that it doesn't align with the sessions distribution, even though one might assume they'd be closely related to activity_days.

In [607]:
```
#boxplot for driving_days
boxplotter('driving_days')

#histogram for driving days
histogrammer('driving_days')
```

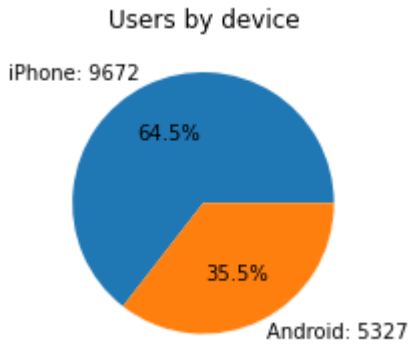




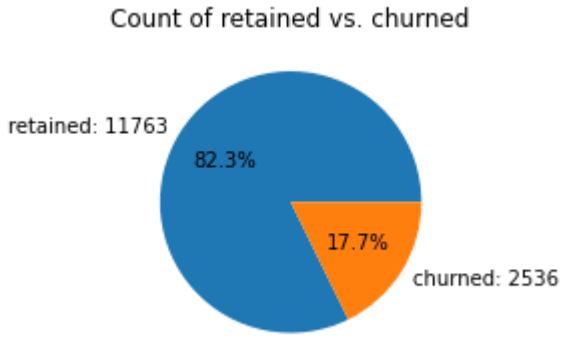
The frequency of days users drove within a month is fairly consistent and largely mirrors the days they accessed the app during the same period. However, a decline is noticed in the distribution of driving_days towards the end.

Interestingly, around 1,000 users didn't drive in the entire month, almost double the figure of 550. This is puzzling when juxtaposed with the data from activity_days, where about 500 users accessed the app on varying day counts. Only approximately 250 of them didn't launch the app at all in the month, while a similar number used it daily. This discrepancy warrants a closer look later on.

In [608]:
```
# Pie chart
fig = plt.figure(figsize=(3,3))
data=OG_waze['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');
```

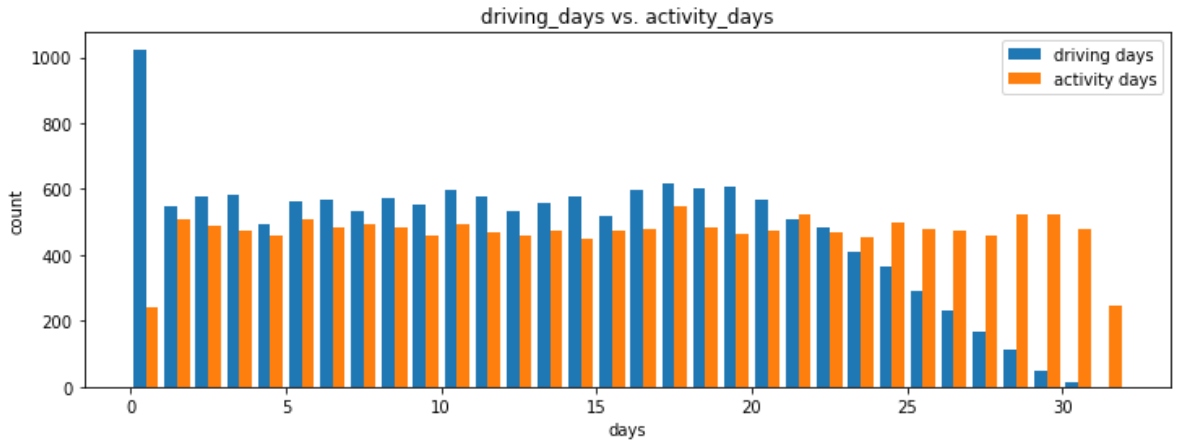


In [609]:
```
# Pie chart
fig = plt.figure(figsize=(3,3))
data=OG_waze['label'].value_counts()
plt.pie(data,
        labels=[f'{data.index[0]}: {data.values[0]}',
                f'{data.index[1]}: {data.values[1]}'],
        autopct='%1.1f%%'
        )
plt.title('Count of retained vs. churned');
```



less than 18% of users churned (17.7%)

In [610]:
```
# Histogram
plt.figure(figsize=(12,4))
label=['driving days', 'activity days']
plt.hist([OG_waze['driving_days'], OG_waze['activity_days']],
         bins=range(0,33),
         label=label)
plt.xlabel('days')
plt.ylabel('count')
plt.legend()
plt.title('driving_days vs. activity_days');
```



You would expect a relationship with driving days and activity days. But as seen in the histogram there are days of the month that has more activity (opening the app) while very little and at times no driving that occurs. This can be simply because users open the app but do not use it to drive, perhaps to look at routes or traffic etc

In [611]:
```
print(OG_waze['driving_days'].max())
print(OG_waze['activity_days'].max())
```

```
30
31
```

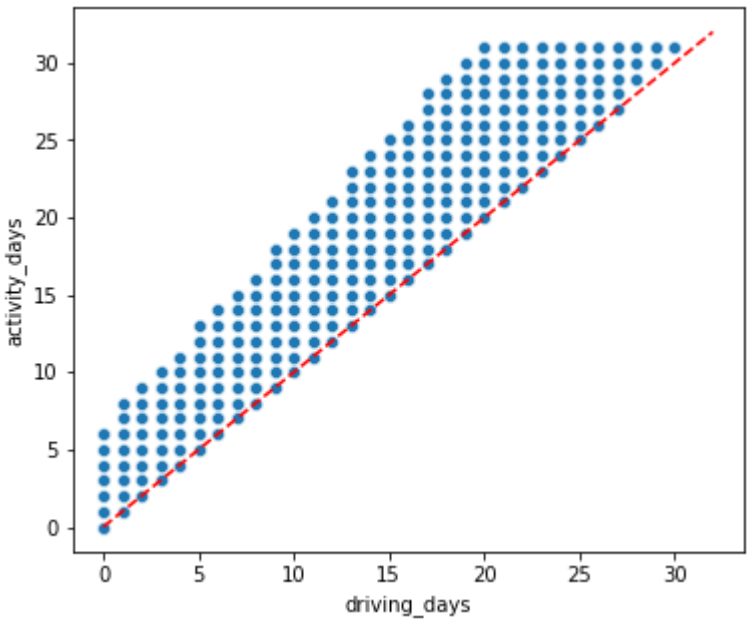
In [612]:
```
print(OG_waze['driving_days'].mean())
print(OG_waze['activity_days'].mean())
```

```
12.179878658577238
15.537102473498233
```

The max for driving days is 30 and activity days is 31. This is unlikely as it means out of the 30 days there was no one who took a drive on the 31st day while there are 14,999 entries in the dataset.

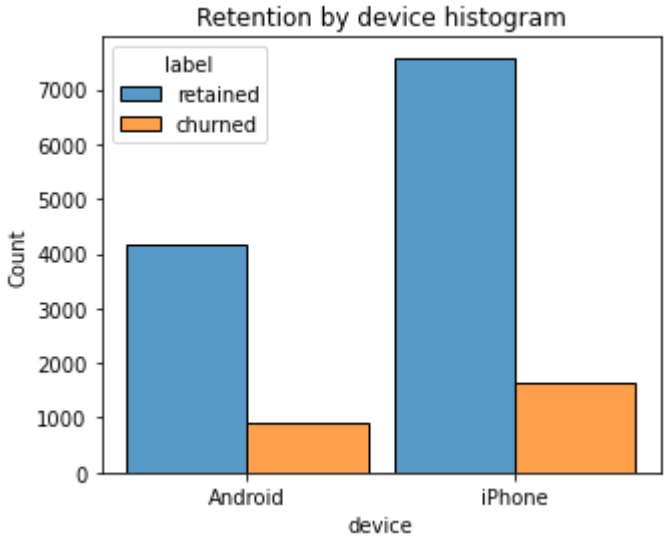
In [613]:
```
#Scatter plot function

def scatterer(x_column, y_column, data=OG_waze, title=None, xlabel=None, ylabe
    plt.figure(figsize=figsize)
    sns.scatterplot(x=data[x_column], y=data[y_column])

    if title:
        plt.title(title)
    if xlabel:
        plt.xlabel(xlabel)
    else:
        plt.xlabel(x_column)
    if ylabel:
        plt.ylabel(ylabel)
    else:
        plt.ylabel(y_column)

    plt.plot([0, 32], [0, 32], color='red', linestyle='--')
    plt.show()
```

In [614]:
```
scatterer('driving_days','activity_days')
```



The data looks good, as you cant have more driving days than activity days, since driving directly results in a activity day.

In [615]:
```
# Histogram showing devices
plt.figure(figsize=(5,4))
sns.histplot(data = OG_waze,
             x='device',
             hue='label',
             multiple='dodge',
             shrink=0.9
             )
plt.title('Retention by device histogram');
```



In [616]:
```
#Create `km_per_driving_day` column
OG_waze['km_per_driving_day'] = OG_waze['driven_km_drives'] / OG_waze['driving

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

Out[616]:
```
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
```

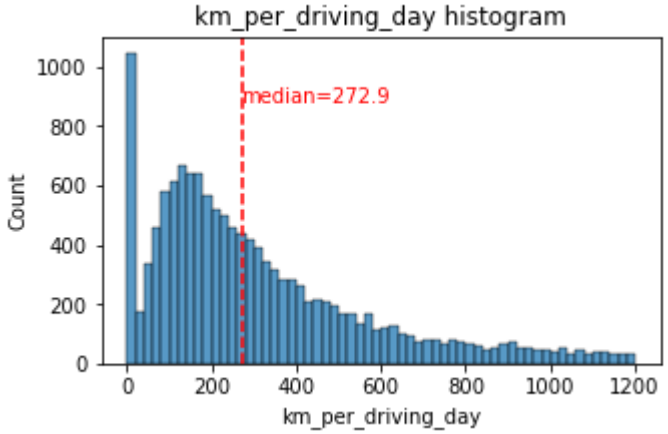
Due to driving days haveing values of zero. Pandas divides with the 0 and result is undefined. Thus converting infinity values to 0.

In [617]:
```
#Convert infinite values to zero
OG_waze.loc[OG_waze['km_per_driving_day']==np.inf, 'km_per_driving_day'] = 0

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

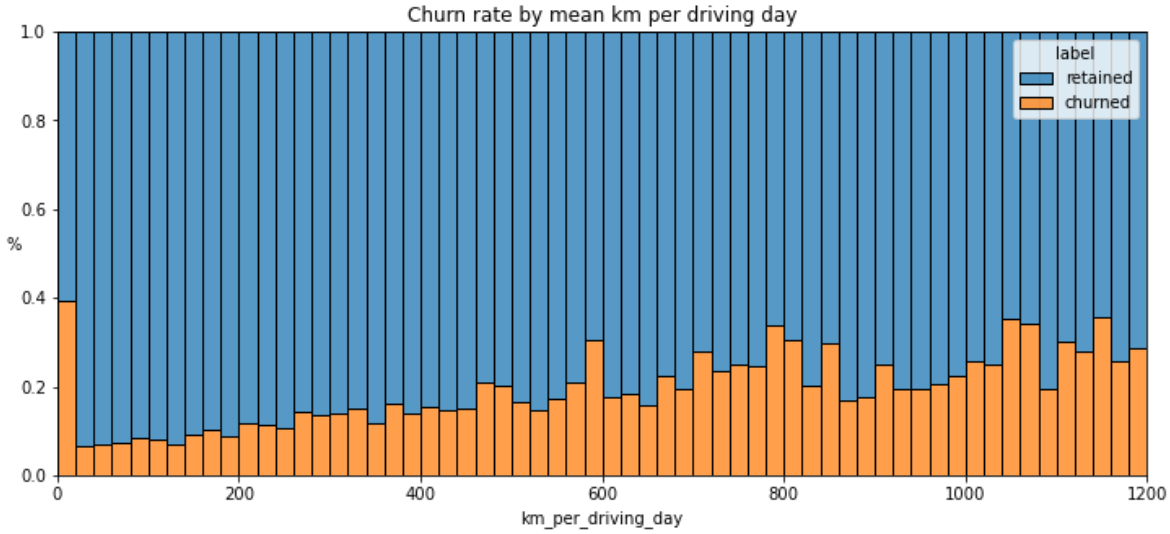
Out[617]:
```
count    14999.000000
mean       578.963113
std       1030.094384
min          0.000000
25%        136.238895
50%        272.889272
75%        558.686918
max      15420.234110
Name: km_per_driving_day, dtype: float64
```

The max of 15,420 km in a day would not be correct. To handle this incorrect data a range will be used to remove impossible distances.

In [618]:
```
histogrammer('km_per_driving_day', bins=range(0,1201,20))
```

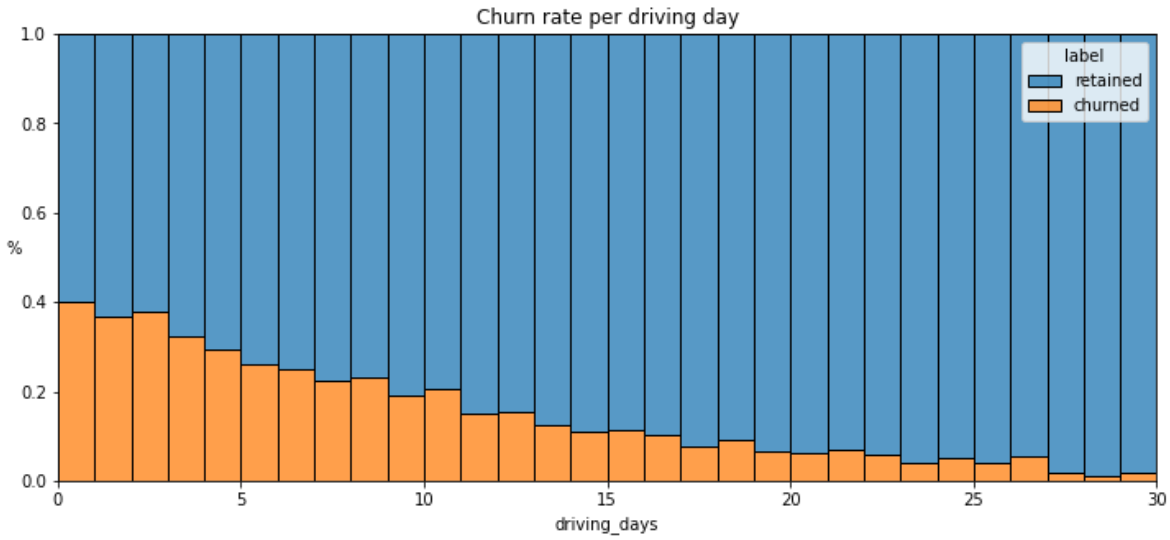


In [619]:
```
# Histogram
plt.figure(figsize=(12,5))
sns.histplot(data=OG_waze,
             x='km_per_driving_day',
             bins=range(0,1201,20),
             hue='label',
             multiple='fill')
plt.ylabel('%', rotation=0)
plt.title('Churn rate by mean km per driving day');
```



From the histogram we can tell that the more KM driven in a day, the more users churned.

In [620]:
```
#Hisotgram showing churned rate vs per driving day
plt.figure(figsize=(12,5))
sns.histplot(data=OG_waze,
             x='driving_days',
             bins=range(0,31),
             hue='label',
             multiple='fill')
plt.ylabel('%', rotation=0)
plt.title('Churn rate per driving day');
```



The churn rate peaks for individuals who barely engaged with Waze over the past month. As their app usage increased, their likelihood to churn decreased. For example, while 40% of those who didn't open the app at all in the previous month opted out, none of the users who accessed it for 30 days did.

This observation aligns with expectations. If frequent users started to opt out, it might suggest they're unhappy with the app. On the other hand, infrequent users leaving could be a reflection of past discontent or perhaps a reduced necessity for a navigation tool. They might've relocated to areas with efficient public transit and no longer need to drive.

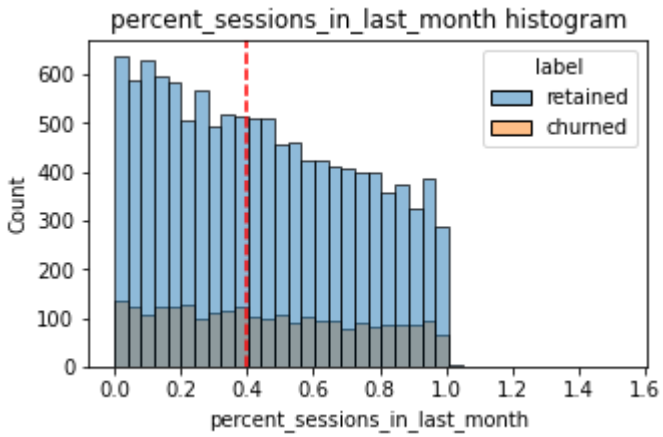
In [621]:
```
OG_waze['percent_sessions_in_last_month'] = OG_waze['sessions'] / OG_waze['tot
```

In [622]:
```
OG_waze['percent_sessions_in_last_month'].median()
```

Out[622]:
```
0.42309702992763176
```

In [623]:
```
# Histogram
histogrammer('percent_sessions_in_last_month',
             hue=OG_waze['label'],
             multiple='layer',
             median_text=False)
```

```
Median: 0.4
```



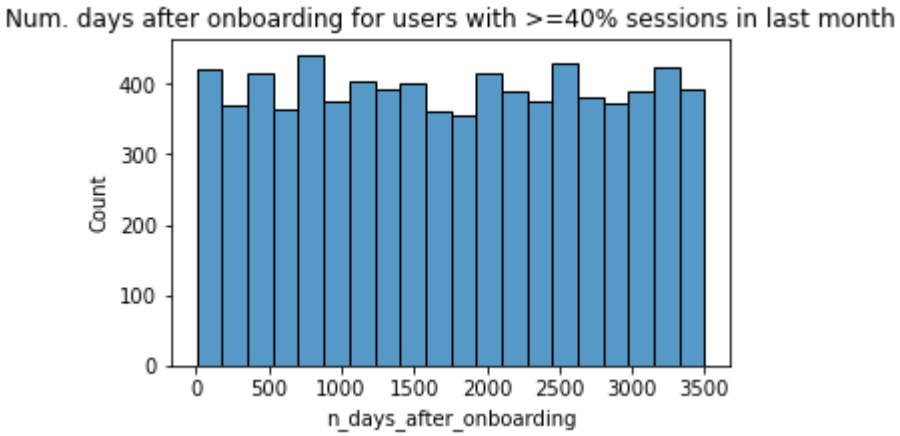
In [624]:
```
#Checking median days after onboarding
OG_waze['n_days_after_onboarding'].median()
```

Out[624]:
```
1741.0
```

Fifty percent of the individuals in the dataset conducted 40% or more of their sessions within the recent month, even though the average time since their initial registration is nearly five years.

Subsequently, I will construct a histogram showcasing the `n_days_after_onboarding` specifically for those who had 40% or above of their complete sessions in the preceding month.

In [625]:
```
# Histogram
data = OG_waze.loc[OG_waze['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 mo
```



This is a uniform distribution. As per project objective, we would need more information from Waze on why long time users have stopped using the app in the last month.

Initially we discovered some outliers in the box plot and also some data that might not be incorrect. The outliers will need to be handled before we continue.

Found a function for exactly this scenerio.

In [626]:
```
def outlier_imputer(column_name, percentile):
    # Calculate threshold
    threshold = OG_waze[column_name].quantile(percentile)
    # Impute threshold for values > than threshold
    OG_waze.loc[OG_waze[column_name] > threshold, column_name] = threshold

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

In [627]:
```
for column in ['sessions', 'drives', 'total_sessions', 'driven_km_drives', 'du
    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: 4668.899348999998
```

In [628]:
```
OG_waze.describe()
```

Out[628]:

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

In [629]:
```
OG_waze['monthly_drives_per_session_ratio'] = (OG_waze['drives']/OG_waze['sess
```

In [630]:
```
OG_waze.head(10)
```

Out[630]:

	ID	label	sessions	drives	total_sessions	n_days_after_onboarding	total_navigations_fav1
0	0	retained	243.0	201.0	296.748273	2276	208
1	1	retained	133.0	107.0	326.896596	1225	19
2	2	retained	114.0	95.0	135.522926	2651	0
3	3	retained	49.0	40.0	67.589221	15	322
4	4	retained	84.0	68.0	168.247020	1562	166
5	5	retained	113.0	103.0	279.544437	2637	0
6	6	retained	3.0	2.0	236.725314	360	185
7	7	retained	39.0	35.0	176.072845	2999	0
8	8	retained	57.0	46.0	183.532018	424	0
9	9	churned	84.0	68.0	244.802115	2997	72

EDA insights:

- Most variables were right-skewed or uniformly distributed.
- While the data was largely consistent, some variables like

3 Hypothesis_Testing

In [631]:
```
#importing packages
from scipy import stats
```

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.

5% as the significance level

In [632]:
```
#Creating `map_dictionary`
map_dictionary = {'Android': 2, 'iPhone': 1}

#Creating new `device_type` column
OG_waze['device_type'] = OG_waze['device']

#Mapping the new column to the dictionary
OG_waze['device_type'] = OG_waze['device_type'].map(map_dictionary)

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

Out[632]:
```
0    2
1    1
2    2
3    1
4    2
Name: device_type, dtype: int64
```

In [633]:
```
OG_waze.groupby('device_type')['drives'].mean()
```

Out[633]:
```
device_type
1    64.446340
2    63.353482
Name: drives, dtype: float64
```

In [634]:
```
#Isolate the `drives` column for iPhone users.
iPhone = OG_waze[OG_waze['device_type'] == 1]['drives']

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

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

Out[634]:
```
Ttest_indResult(statistic=1.164371413602629, pvalue=0.24429844267242234)
```

Since the p-value is larger than the chosen significance level (5%), we fail to reject the null hypothesis. we conclude that there is not a statistically significant difference in the average number of drives between drivers who use iPhones and drivers who use Androids. The key business insight is that drivers who use iPhone devices on average have a similar number of drives as those who use Androids.

Hypothesis testing insights:

- A primary observation is that, on average, drivers using iPhones record a similar drive count as Android users.

- A logical follow-up would be to delve into other elements affecting the drive count and conduct more hypothesis testing to understand user patterns better. Additionally, short-term adjustments in Waze app's marketing or user interface might offer further data to scrutinize churn.

4 Regression_Modelling

In [635]:
```
#importing packages

# 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_s
recall_score, f1_score, confusion_matrix, ConfusionMatrixDisplay
from sklearn.linear_model import LogisticRegression
```

In [636]:
```
# Summary of dataset
print(OG_waze.shape)

OG_waze.info()
```

```
(14999, 17)
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 14999 entries, 0 to 14998
Data columns (total 17 columns):
 #   Column                           Non-Null Count  Dtype
---  ------                           --------------  -----
 0   ID                               14999 non-null  int64
 1   label                            14299 non-null  object
 2   sessions                         14999 non-null  float64
 3   drives                           14999 non-null  float64
 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
 13  km_per_driving_day               14999 non-null  float64
 14  percent_sessions_in_last_month   14999 non-null  float64
 15  monthly_drives_per_session_ratio 14894 non-null  float64
 16  device_type                      14999 non-null  int64
dtypes: float64(8), int64(7), object(2)
memory usage: 1.9+ MB
```

We had implemented a cut off in the monthly drives column which results in 14894. This is not missing data but purposely limited.

The label column as mentioned before is missing 700 values.

In [637]:
```
OG_waze.head()
```

Out[637]:

	ID	label	sessions	drives	total_sessions	n_days_after_onboarding	total_navigations_fav1
0	0	retained	243.0	201.0	296.748273	2276	208
1	1	retained	133.0	107.0	326.896596	1225	19
2	2	retained	114.0	95.0	135.522926	2651	0
3	3	retained	49.0	40.0	67.589221	15	322
4	4	retained	84.0	68.0	168.247020	1562	166

Lets remove columns that are not needed for the analysis.

In [638]:
```
waze_rm = OG_waze.drop('ID', axis=1)
```

In [639]:
```
waze_rm['label'].value_counts(normalize=True)
```

Out[639]:
```
retained    0.822645
churned     0.177355
Name: label, dtype: float64
```

In [640]:
```
waze_rm.describe()
```

Out[640]:

	sessions	drives	total_sessions	n_days_after_onboarding	total_navigations_fa
count	14999.000000	14999.000000	14999.000000	14999.000000	14999.0000
mean	76.568705	64.058204	184.031320	1749.837789	121.6059
std	67.297958	55.306924	118.600463	1008.513876	148.1215
min	0.000000	0.000000	0.220211	4.000000	0.0000
25%	23.000000	20.000000	90.661156	878.000000	9.0000
50%	56.000000	48.000000	159.568115	1741.000000	71.0000
75%	112.000000	93.000000	254.192341	2623.500000	178.0000
max	243.000000	201.000000	454.363204	3500.000000	1236.0000

The features that were previously created were km_per_driving_day, percent_sessions_in_last_month, monthly_drives_per_session_ratio.

The max values for the following columns seem to be outliers:

- sessions
- drives
- total_sessions
- total_navigations_fav1
- total_navigations_fav2
- driven_km_drives
- duration_minutes_drives

In [641]:
```
waze_rm['pro_driver'] = np.where((waze_rm['drives'] >= 60) & (waze_rm['driving
```

In [642]:
```
print(waze_rm['pro_driver'].value_counts())
```

```
0    12405
1     2594
Name: pro_driver, dtype: int64
```

In [643]:
```
waze_rm.groupby(['pro_driver']) ['label'].value_counts(normalize=True)
```

Out[643]:
```
pro_driver  label
0           retained    0.801202
            churned     0.198798
1           retained    0.924437
            churned     0.075563
Name: label, dtype: float64
```

The churn rate for professional drivers is 7.6%, while the churn rate for non-professionals is 19.9%. This seems like it could add predictive signal to the model.

In [644]:
```
waze_rm.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 14999 entries, 0 to 14998
Data columns (total 17 columns):
 #   Column                           Non-Null Count  Dtype
---  ------                           --------------  -----
 0   label                            14299 non-null  object
 1   sessions                         14999 non-null  float64
 2   drives                           14999 non-null  float64
 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  percent_sessions_in_last_month   14999 non-null  float64
 14  monthly_drives_per_session_ratio 14894 non-null  float64
 15  device_type                      14999 non-null  int64
 16  pro_driver                       14999 non-null  int32
dtypes: float64(8), int32(1), int64(6), object(2)
memory usage: 1.9+ MB
```

dropping the missing columns from label

In [645]:
```
waze_rm_dropped = waze_rm.dropna(subset=['label'])
```

In [646]:
```
waze_rm_dropped.describe()
```

Out[646]:

	sessions	drives	total_sessions	n_days_after_onboarding	total_navigations_fa
count	14299.000000	14299.000000	14299.000000	14299.000000	14299.0000
mean	76.539688	64.014546	183.663233	1751.822505	121.7473
std	67.243178	55.251272	118.596924	1008.663834	147.7134
min	0.000000	0.000000	0.220211	4.000000	0.0000
25%	23.000000	20.000000	90.457733	878.500000	10.0000
50%	56.000000	48.000000	158.718571	1749.000000	71.0000
75%	111.000000	93.000000	253.540450	2627.500000	178.0000
max	243.000000	201.000000	454.363204	3500.000000	1236.0000

lets also place threshholds on the columns with the outliers just as we did with monthly drives ratio.

In [647]:
```
waze_rm_dropped = waze_rm_dropped.copy()
```

In [648]:
```
for column in ['sessions', 'drives', 'total_sessions', 'total_navigations_fav1
                'total_navigations_fav2', 'driven_km_drives', 'duration_minutes
    threshold = waze_rm_dropped[column].quantile(0.95)
    waze_rm_dropped.loc[waze_rm_dropped[column] > threshold, column] = thresho
```

In [649]:
```
waze_rm_dropped.describe()
```

Out[649]:

	sessions	drives	total_sessions	n_days_after_onboarding	total_navigations_fa
count	14299.000000	14299.000000	14299.000000	14299.000000	14299.0000
mean	76.539688	63.964683	183.663233	1751.822505	114.5627
std	67.243178	55.127927	118.596924	1008.663834	124.3785
min	0.000000	0.000000	0.220211	4.000000	0.0000
25%	23.000000	20.000000	90.457733	878.500000	10.0000
50%	56.000000	48.000000	158.718571	1749.000000	71.0000
75%	111.000000	93.000000	253.540450	2627.500000	178.0000
max	243.000000	200.000000	454.363204	3500.000000	422.0000

In [650]:
```
#Encoding label from categorical to binary 0 being retained and 1 being churne

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

Out[650]:

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

The assumptions for logistic regression has been met apart from the last 2 which can be done after modelling

Assumptions of logistic regression:

- Independant observations - [x]
- No extreme outliers - [x]
- Little to no multicollinearity among X predictors - [ ]
- Linear relationship between X and the logit of y - [ ]

We will have to set up a correlation matrix to check the correlation with the predictor variables

Instead of using some features that were created in EDA, will be dropping them for modelling.

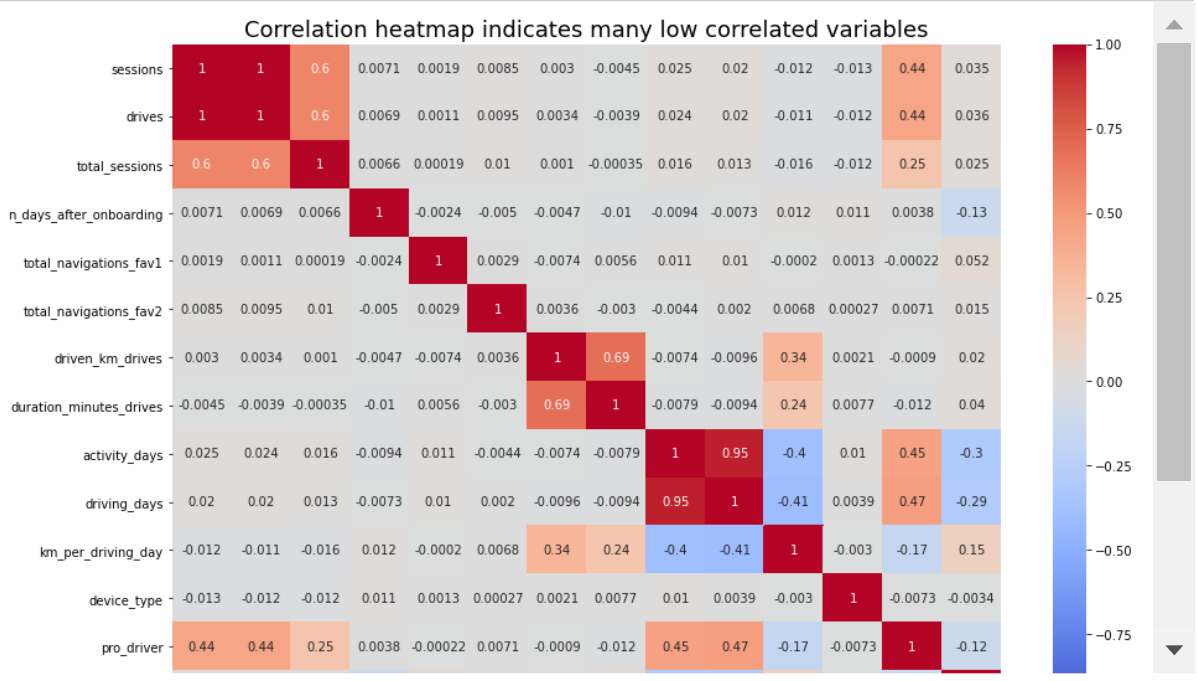
In [651]:
```
waze_rm_dropped = waze_rm_dropped.drop(['percent_sessions_in_last_month', 'mon
```

In [652]:
```
# Generate a correlation matrix
waze_rm_dropped.corr(method='pearson')
```

Out[652]:

	sessions	drives	total_sessions	n_days_after_onboarding	total_nav
sessions	1.000000	0.996942	0.597299	0.007101	
drives	0.996942	1.000000	0.595396	0.006940	
total_sessions	0.597299	0.595396	1.000000	0.006615	
n_days_after_onboarding	0.007101	0.006940	0.006615	1.000000	
total_navigations_fav1	0.001858	0.001058	0.000194	-0.002450	
total_navigations_fav2	0.008536	0.009505	0.010363	-0.004968	
driven_km_drives	0.002995	0.003445	0.001015	-0.004655	
duration_minutes_drives	-0.004545	-0.003889	-0.000345	-0.010167	
activity_days	0.025113	0.024357	0.015757	-0.009418	
driving_days	0.020294	0.019608	0.012957	-0.007321	
km_per_driving_day	-0.011569	-0.010989	-0.016162	0.011764	
device_type	-0.012704	-0.011684	-0.012133	0.011299	
pro_driver	0.443654	0.444425	0.254532	0.003770	
label2	0.034911	0.035865	0.024568	-0.129263	

In [653]:
```
# Plot correlation heatmap
plt.figure(figsize=(15,10))
sns.heatmap(waze_rm_dropped.corr(method='pearson'), vmin=-1, vmax=1, annot=Tru
plt.title('Correlation heatmap indicates many low correlated variables',
          fontsize=18)
plt.show();
```



The variables that have mutlicollinearity are:

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

In [654]:
```
waze_rm_dropped.head()
```

Out[654]:

	label	sessions	drives	total_sessions	n_days_after_onboarding	total_navigations_fav1	to
0	retained	243.0	200.0	296.748273	2276	208.0	
1	retained	133.0	107.0	326.896596	1225	19.0	
2	retained	114.0	95.0	135.522926	2651	0.0	
3	retained	49.0	40.0	67.589221	15	322.0	
4	retained	84.0	68.0	168.247020	1562	166.0	

Creating a new binary column for device to device 2. The column device_type will not be necessary to use here.

In [655]:
```
waze_rm_dropped=waze_rm_dropped.drop('device_type', axis = 1)
```

In [656]:
```
# Creating new `device2` variable
waze_rm_dropped['device2'] = np.where(waze_rm_dropped['device']=='Android', 0,
waze_rm_dropped[['device', 'device2']].tail()
```

Out[656]:

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

In [657]:
```
# Isolate predictor variables
X = waze_rm_dropped.drop(columns = ['label', 'label2', 'device', 'sessions', '
```

In [658]:
```
# Isolating target variable
y = waze_rm_dropped['label2']
```

In [659]:
```
#split_test
# Performing the train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y, stratify=y, random_s
```

In [660]:
```
# Use .head()
X_train.head()
```

Out[660]:

	drives	total_sessions	n_days_after_onboarding	total_navigations_fav1	total_navigations_
152	108.0	186.192746	3116	243.0	1
11899	2.0	3.487590	794	114.0	
10937	139.0	347.106403	331	4.0	
669	108.0	454.363204	2320	11.0	
8406	10.0	89.475821	2478	135.0	

In [661]:
```
# fitting model x train on y train. penalty set to none as predictors are unsc
model = LogisticRegression(penalty='none', max_iter=400)

model.fit(X_train, y_train)
```

Out[661]:
```
LogisticRegression(max_iter=400, penalty='none')
```
In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [662]:
```
pd.Series(model.coef_[0], index=X.columns)
```

Out[662]:
```
drives                    0.001915
total_sessions            0.000328
n_days_after_onboarding  -0.000406
total_navigations_fav1    0.001232
total_navigations_fav2    0.000926
driven_km_drives         -0.000015
duration_minutes_drives   0.000109
activity_days            -0.106024
km_per_driving_day        0.000018
pro_driver               -0.001528
device2                  -0.001041
dtype: float64
```

In [663]:
```
model.intercept_
```

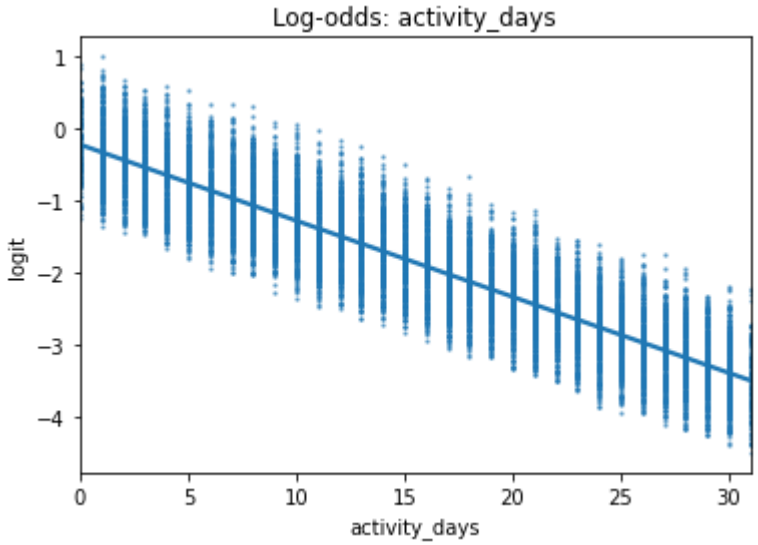
Out[663]:
```
array([-0.00170627])
```

In [664]:
```
# Get the predicted probabilities of the training data
training_probabilities = model.predict_proba(X_train)
training_probabilities
```

Out[664]:
```
array([[0.93965209, 0.06034791],
       [0.61967449, 0.38032551],
       [0.76458201, 0.23541799],
       ...,
       [0.91908213, 0.08091787],
       [0.85095402, 0.14904598],
       [0.93515678, 0.06484322]])
```

In [665]:
```
# 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_probabil
```

In [666]:
```
#  Plot regplot of `activity_days` log-odds
sns.regplot(x='activity_days', y='logit', data=logit_data, scatter_kws={'s': 2
plt.title('Log-odds: activity_days');
```



4.1 Evaluation_of_Model

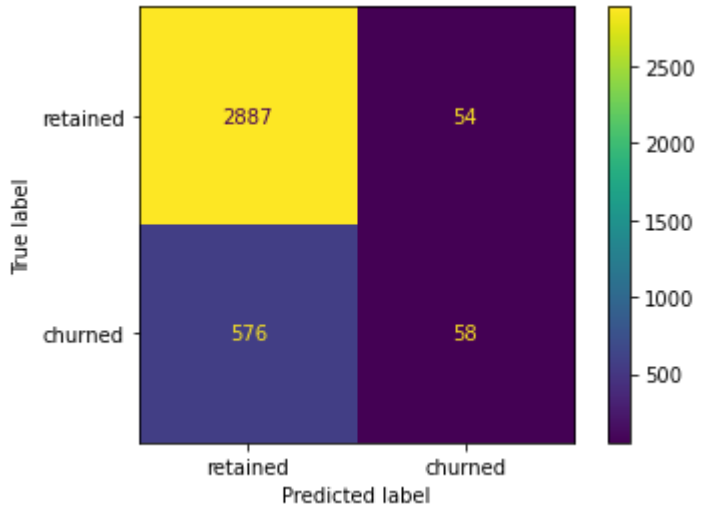
In [667]:
```
#Generate predictions on X test
y_preds = model.predict(X_test)
```

In [668]:
```
# Score the model (accuracy) on the test data
model.score(X_test, y_test)
```

Out[668]:
```
0.8237762237762237
```

In [669]:
```
cm = confusion_matrix(y_test, y_preds)
```

In [670]:
```
disp = ConfusionMatrixDisplay(confusion_matrix=cm,
                              display_labels=['retained', 'churned'],
                              )
disp.plot();
```



In [671]:
```
precision = cm[1,1] / (cm[0,1] + cm[1,1])
precision
```

Out[671]:
```
0.5178571428571429
```

In [672]:
```
recall = cm[1,1] / (cm[1,1] + cm[1,0])
recall
```

Out[672]:
```
0.0914826498422713
```

In [673]:
```
#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
```

- **Overall Model Performance**:
  - Accuracy: 82% on a test dataset of 3,575 instances.
- **Retained Users Classification**:
  - Precision: 83%.
  - Recall: 98%.

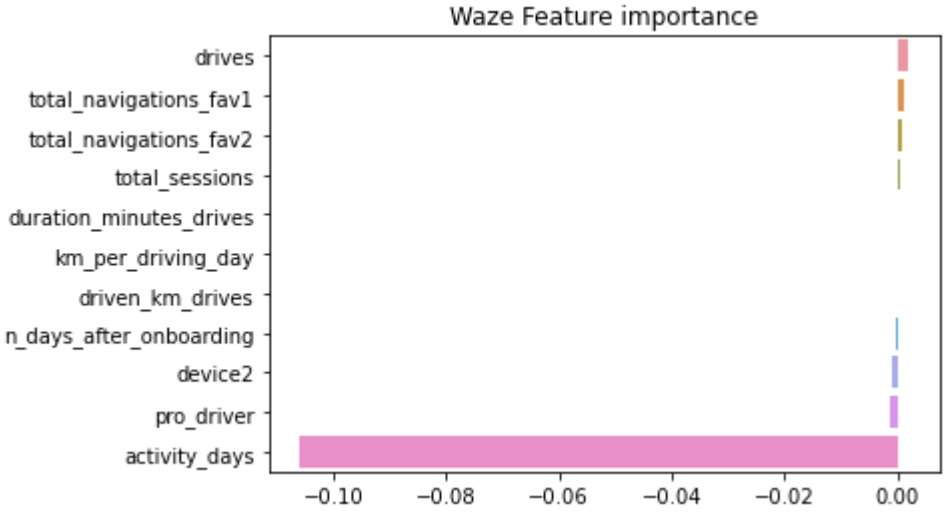
- **Churned Users Classification**:
  - Precision: 52%.
  - Recall: 9%.
- **Insights**:
  - The model is proficient in predicting retained users but struggles with churned users.
  - Despite a decent overall accuracy, the low recall for "churned" indicates a high number of false negatives.
  - The class imbalance between "retained" and "churned" might influence the high accuracy rate.

As the goal is to predict churned users, this model would not be a good choice to use. To improve the model, i decided to take a look at feature importance and see which of the features has the biggest impact.

In [674]:
```
# Create a list of (column_name, coefficient) tuples
waze_feature_importance = list(zip(X_train.columns, model.coef_[0]))

# Sort the list by coefficient value
waze_feature_importance = sorted(waze_feature_importance, key=lambda x: x[1],
waze_feature_importance
```

Out[674]:
```
[('drives', 0.0019146636449751556),
 ('total_navigations_fav1', 0.0012318288515483966),
 ('total_navigations_fav2', 0.00092633468241052),
 ('total_sessions', 0.00032750041378524933),
 ('duration_minutes_drives', 0.00010909313081863752),
 ('km_per_driving_day', 1.8258738149154475e-05),
 ('driven_km_drives', -1.488022450014146e-05),
 ('n_days_after_onboarding', -0.00040649907901463777),
 ('device2', -0.001040753192386349),
 ('pro_driver', -0.0015281741341061981),
 ('activity_days', -0.10602428908502631)]
```

In [675]:
```
# Plot the feature importance
sns.barplot(x=[x[1] for x in waze_feature_importance],
            y=[x[0] for x in waze_feature_importance],
            orient='h')
plt.title('Waze Feature importance');
```




```
<h2 id="Feature_Importance_Summary">Feature_Importance_Summary</h2>

<h3 id="Positive_Influence_on_Target">Positive_Influence_on_Target</h3>

- **drives**: Most significant positive feature with an importance of
0.0019.
- **total_navigations_fav1**: Second most influential positive feature with
a score of 0.0012.
- **total_navigations_fav2**: Positive influence with a score of 0.0009.
- **total_sessions**: Has a minor positive influence of 0.0003.
- **duration_minutes_drives**: A very minor positive influence with a score
of 0.0001.
- **km_per_driving_day**: Holds the least positive influence with an
importance close to 0 (1.83e-05).

<h3 id="Negative_Influence_on_Target">Negative_Influence_on_Target</h3>

- **activity_days**: Most significant negative feature with a score of
-0.1060.
- **n_days_after_onboarding**: Negative influence with a score of -0.0004.
- **device2**: Holds a negative influence of -0.0010.
- **pro_driver**: Another negative influencer with a score of -0.0015.
- **driven_km_drives**: Has a very minor negative influence, close to 0
(-1.49e-05).

`activity_days` has the most substantial influence on user churn among all
features. The more active days a user has, the less likely they are to
churn.
On the other hand, features like `km_per_driving_day` and `driven_km_drives`
have a very minimal influence on the target.
```

```
pro_driver and device2 seem to have protective qualities against churn;
their increase is associated with a reduction in churn.
```

4.2.3 Model_Enhancement_and_Additional_Data_Suggestions

4.2.4 Feature_Engineering_and_Model_Refinement

> - Engineered features, especially with domain knowledge, can enhance predictive capability.
> - The `professional_driver` feature stands out as a key predictive predictor.
> - Scaling predictor variables and re-evaluating model predictors can potentially reduce noise and improve performance.

4.2.5 Desired_Additional_Data

> - Drive-level Information**: Data about drive times and geographic locations of each user.
> - User-App Interaction**: Granularity on how users engage with the app, such as frequency of reporting or confirming road hazards.
> - Travel Patterns**: Monthly count of unique start and end locations input by drivers.

Leveraging the above suggestions could yield a more robust and predictive model.

5 Building_the_Machine_Learning_Model

Goal:

> - To predict if cusotmer will churn or retain

In [676]:
```
#installation of xgboost
# To install xgboost directly to ide
!pip install xgboost

# To view all columns in the frame
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, Precision

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
```

```
Requirement already satisfied: xgboost in c:\users\dalla\anaconda3\lib\site-p
ackages (1.7.6)
Requirement already satisfied: numpy in c:\users\dalla\anaconda3\lib\site-pac
kages (from xgboost) (1.20.1)
Requirement already satisfied: scipy in c:\users\dalla\anaconda3\lib\site-pac
kages (from xgboost) (1.6.2)
```

In [677]:
```
waze_new = waze_rm.append(waze_rm_dropped)
```

In [678]:
```
waze_rm_dropped.head()
```

Out[678]:

	label	sessions	drives	total_sessions	n_days_after_onboarding	total_navigations_fav1	to
0	retained	243.0	200.0	296.748273	2276	208.0	
1	retained	133.0	107.0	326.896596	1225	19.0	
2	retained	114.0	95.0	135.522926	2651	0.0	
3	retained	49.0	40.0	67.589221	15	322.0	
4	retained	84.0	68.0	168.247020	1562	166.0	

In [679]:
```
waze_new = waze_rm_dropped
```

Lets bring the previous features we had dropped. We would have to calculate it again,

In [680]:
```
#Creating `percent_sessions_in_last_month` feature
waze_rm_dropped['percent_sessions_in_last_month'] = waze_rm_dropped['sessions'

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

Out[680]:
```
count    14299.000000
mean         0.444101
std          0.278496
min          0.000000
25%          0.200241
50%          0.427865
75%          0.665012
max          1.530637
Name: percent_sessions_in_last_month, dtype: float64
```

In [681]:
```
waze_rm_dropped.head()
```

Out[681]:

	label	sessions	drives	total_sessions	n_days_after_onboarding	total_navigations_fav1	to
0	retained	243.0	200.0	296.748273	2276	208.0	
1	retained	133.0	107.0	326.896596	1225	19.0	
2	retained	114.0	95.0	135.522926	2651	0.0	
3	retained	49.0	40.0	67.589221	15	322.0	
4	retained	84.0	68.0	168.247020	1562	166.0	

In [682]:
```
# Creating `total_sessions_per_day` feature
waze_rm_dropped['total_sessions_per_day'] = waze_rm_dropped['total_sessions']
```

In [683]:
```
waze_rm_dropped.head()
```

Out[683]:

	label	sessions	drives	total_sessions	n_days_after_onboarding	total_navigations_fav1	to
0	retained	243.0	200.0	296.748273	2276	208.0	
1	retained	133.0	107.0	326.896596	1225	19.0	
2	retained	114.0	95.0	135.522926	2651	0.0	
3	retained	49.0	40.0	67.589221	15	322.0	
4	retained	84.0	68.0	168.247020	1562	166.0	

In [684]:
```
# Create `km_per_hour` feature
waze_rm_dropped['km_per_hour'] = waze_rm_dropped['driven_km_drives'] / waze_rm
waze_rm_dropped['km_per_hour'].describe()
```

Out[684]:
```
count    14299.000000
mean         0.052624
std          0.090946
min          0.020004
25%          0.025802
50%          0.033675
75%          0.053099
max          6.055706
Name: km_per_hour, dtype: float64
```

In [685]:
```
# Creating `km_per_drive` feature
waze_rm_dropped['km_per_drive'] = waze_rm_dropped['driven_km_drives'] / waze_r
waze_rm_dropped['km_per_drive'].describe()
```

Out[685]:
```
count    1.429900e+04
mean              inf
std               NaN
min      1.008775e+00
25%      3.365859e+01
50%      7.429025e+01
75%      1.828194e+02
max               inf
Name: km_per_drive, dtype: float64
```

In [686]:
```
# 1. Convert infinite values to zero
waze_rm_dropped.loc[waze_rm_dropped['km_per_drive']==np.inf, 'km_per_drive'] =

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

Out[686]:
```
count    14299.000000
mean       225.797731
std        572.400481
min          0.000000
25%         32.910489
50%         72.319628
75%        177.431844
max       8889.794236
Name: km_per_drive, dtype: float64
```

In [687]:
```
# Creating `percent_of_sessions_to_favorite` feature
waze_rm_dropped['percent_of_drives_to_favorite'] = (
    waze_rm_dropped['total_navigations_fav1'] + waze_rm_dropped['total_navigat

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

Out[687]:
```
count    14299.000000
mean         1.575282
std          8.243636
min          0.000000
25%          0.212409
50%          0.648292
75%          1.593733
max        668.888397
Name: percent_of_drives_to_favorite, dtype: float64
```

In [688]:
```
waze_rm_dropped['label'].value_counts(normalize=True)
```

Out[688]:
```
retained    0.822645
churned     0.177355
Name: label, dtype: float64
```

Splitting and training the model. We will be using recall to asess the perfomance of the model.

In [689]:
```
# Isolating X variables
X = waze_rm_dropped.drop(columns=['label', 'label2', 'device'])

# Isolating y variable
y = waze_rm_dropped['label2']

# Splitting 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)

# Splitting 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
```

In [690]:
```
for x in [X_train, X_val, X_test]:
    print(len(x))
```

```
8579
2860
2860
```

Now to create a random forest classifier and tune the hyperparemeters that will be included.

In [691]:
```
# 1. Instantiate the random forest classifier
rf = RandomForestClassifier(random_state=42)

# 2. Create a dictionary of hyperparameters to tune
cv_params = {
    'max_depth': [None, 10, 20],
    'max_features': ['auto', 0.5],
    'max_samples': [None, 0.5],
    'min_samples_leaf': [1, 2],
    'min_samples_split': [2, 5],
    'n_estimators': [100, 200]
}

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

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

In [692]:
```
%%time
rf_cv.fit(X_train, y_train)
```

```
Fitting 4 folds for each of 96 candidates, totalling 384 fits
Wall time: 2min 18s
```

Out[692]:
```
GridSearchCV(cv=4, estimator=RandomForestClassifier(random_state=42), n_jobs=
-1,
             param_grid={'max_depth': [None, 10, 20],
                         'max_features': ['auto', 0.5],
                         'max_samples': [None, 0.5], 'min_samples_leaf': [1,
2],
                         'min_samples_split': [2, 5],
                         'n_estimators': [100, 200]},
             refit='recall',
             scoring={'accuracy': 'accuracy', 'f1': 'f1',
                      'precision': 'precision', 'recall': 'recall'},
             verbose=10)
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [693]:
```
# Examine best score
rf_cv.best_score_
```

Out[693]:
```
0.12548003867937563
```

In [694]:
```
rf_cv.best_params_
```

Out[694]:
```
{'max_depth': None,
 'max_features': 0.5,
 'max_samples': None,
 'min_samples_leaf': 1,
 'min_samples_split': 2,
 'n_estimators': 100}
```

In [695]:
```
def make_results(model_name:str, model_object, metric:str):
    '''
    Arguments:
        model_name (string): what you want the model to be called in the outpu
        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 fold
    '''

    # Create dictionary that maps input metric to actual metric name in GridSe
    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]].i

    # 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
```

In [696]:
```
results = make_results('RF cv', rf_cv, 'recall')
results
```

Out[696]:

	model	precision	recall	F1	accuracy
0	RF cv	0.49812	0.12548	0.200423	0.822356

In [697]:

```
# 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': 'accuracy',
    'precision': 'precision',
    'recall': 'recall',
    'f1': 'f1'
}

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

In [698]:
```
%%time
xgb_cv.fit(X_train, y_train)
```

```
Wall time: 33.4 s
```

Out[698]:
```
GridSearchCV(cv=4,
             estimator=XGBClassifier(base_score=None, booster=None,
                                     callbacks=None, colsample_bylevel=None,
                                     colsample_bynode=None,
                                     colsample_bytree=None,
                                     early_stopping_rounds=None,
                                     enable_categorical=False, eval_metric=No
ne,
                                     feature_types=None, gamma=None,
                                     gpu_id=None, grow_policy=None,
                                     importance_type=None,
                                     interaction_constraints=None,
                                     learning_rate=None,...
                                     max_leaves=None, min_child_weight=None,
                                     missing=nan, monotone_constraints=None,
                                     n_estimators=100, n_jobs=None,
                                     num_parallel_tree=None, predictor=None,
                                     random_state=42, ...),
             param_grid={'learning_rate': [0.01, 0.1], 'max_depth': [6, 12],
                         'min_child_weight': [3, 5], 'n_estimators': [300]},
             refit='recall',
             scoring={'accuracy': 'accuracy', 'f1': 'f1',
                      'precision': 'precision', 'recall': 'recall'})
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [699]:
```
# Examine best score
xgb_cv.best_score_
```

Out[699]:
```
0.16886137588064648
```

In [700]:
```
# Examine best parameters
xgb_cv.best_params_
```

Out[700]:
```
{'learning_rate': 0.1,
 'max_depth': 12,
 'min_child_weight': 3,
 'n_estimators': 300}
```

In [701]:
```
# 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[701]:

	model	precision	recall	F1	accuracy
0	RF cv	0.498120	0.125480	0.200423	0.822356
0	XGB cv	0.429664	0.168861	0.242401	0.812798

Selecting champion model

In [702]:
```
# Use random forest model to predict on validation data
rf_val_preds = rf_cv.best_estimator_.predict(X_val)
```

In [703]:
```
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 o
        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 y
    '''
    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 [704]:
```
# 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[704]:

	model	precision	recall	F1	accuracy
0	RF cv	0.498120	0.125480	0.200423	0.822356
0	XGB cv	0.429664	0.168861	0.242401	0.812798
0	RF val	0.478261	0.130178	0.204651	0.820629

XGBoost

In [705]:
```
# 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[705]:

	model	precision	recall	F1	accuracy
0	RF cv	0.498120	0.125480	0.200423	0.822356
0	XGB cv	0.429664	0.168861	0.242401	0.812798
0	RF val	0.478261	0.130178	0.204651	0.820629
0	XGB val	0.407609	0.147929	0.217077	0.810839

Using the champion model above, lets use it to predict.

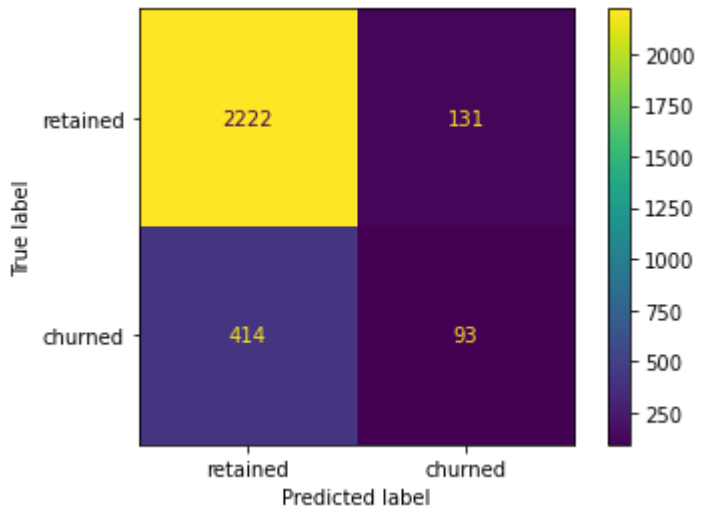
In [706]:
```
# Use XGBoost model to predict on test data
xgb_test_preds = xgb_cv.best_estimator_.predict(X_test)

# 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[706]:

	model	precision	recall	F1	accuracy
0	RF cv	0.498120	0.125480	0.200423	0.822356
0	XGB cv	0.429664	0.168861	0.242401	0.812798
0	RF val	0.478261	0.130178	0.204651	0.820629
0	XGB val	0.407609	0.147929	0.217077	0.810839
0	XGB test	0.415179	0.183432	0.254446	0.809441

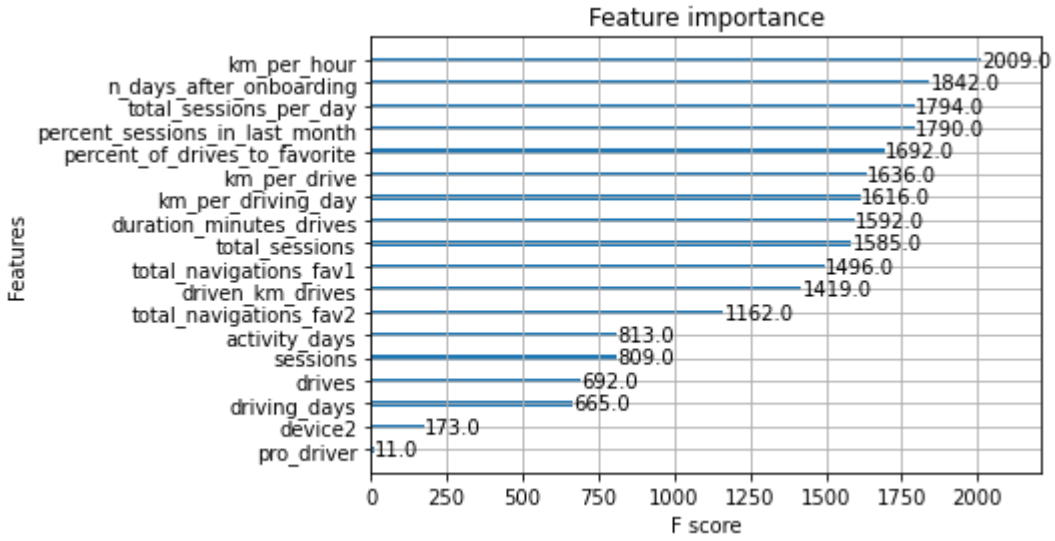
5.1 Confusion_Matrix_for_Champion_Model

In [707]:
```
# 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();
```



Feature importance in the champion model

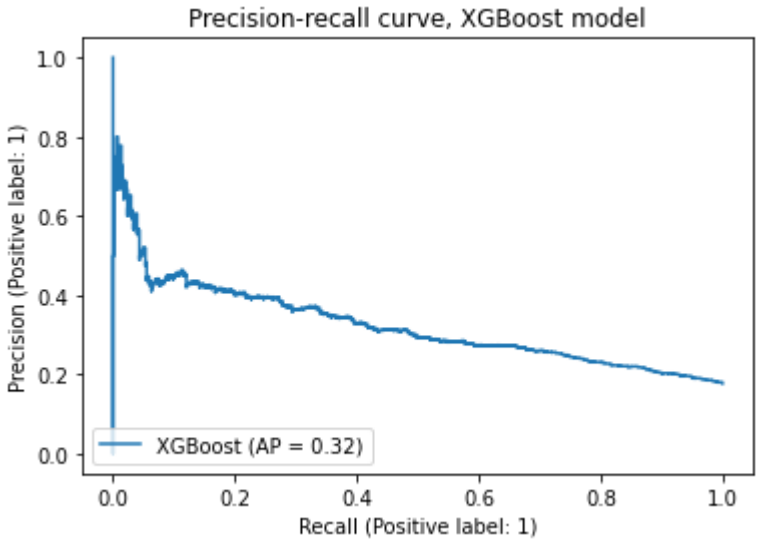
In [708]:
```
plot_importance(xgb_cv.best_estimator_);
```



In [709]:
```
pip install --upgrade scikit-learn
```

```
Requirement already satisfied: scikit-learn in c:\users\dalla\anaconda3\lib\s
ite-packages (1.3.0)
Requirement already satisfied: numpy>=1.17.3 in c:\users\dalla\anaconda3\lib
\site-packages (from scikit-learn) (1.20.1)
Requirement already satisfied: threadpoolctl>=2.0.0 in c:\users\dalla\anacond
a3\lib\site-packages (from scikit-learn) (2.1.0)
Requirement already satisfied: scipy>=1.5.0 in c:\users\dalla\anaconda3\lib\s
ite-packages (from scikit-learn) (1.6.2)
Requirement already satisfied: joblib>=1.1.1 in c:\users\dalla\anaconda3\lib
\site-packages (from scikit-learn) (1.3.2)
Note: you may need to restart the kernel to use updated packages.
```

In [710]:
```
# 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 [711]:
```
# Get predicted probabilities on the test data
predicted_probabilities = xgb_cv.best_estimator_.predict_proba(X_test)
predicted_probabilities
```

Out[711]:
```
array([[0.99251163, 0.00748836],
       [0.74189436, 0.2581056 ],
       [0.99660224, 0.00339775],
       ...,
       [0.82451403, 0.17548597],
       [0.9839575 , 0.01604244],
       [0.7869331 , 0.21306688]], dtype=float32)
```

In [712]:
```
# 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[712]:
```
array([0, 0, 0, ..., 0, 0, 0])
```

In [713]:
```
# Get evaluation metrics for when the threshold is 0.4
get_test_scores('XGB, threshold = 0.4', new_preds, y_test)
```

Out[713]:

	model	precision	recall	F1	accuracy
0	XGB, threshold = 0.4	0.39404	0.234714	0.29419	0.80035

In [714]:
```
results
```

Out[714]:

	model	precision	recall	F1	accuracy
0	RF cv	0.498120	0.125480	0.200423	0.822356
0	XGB cv	0.429664	0.168861	0.242401	0.812798
0	RF val	0.478261	0.130178	0.204651	0.820629
0	XGB val	0.407609	0.147929	0.217077	0.810839
0	XGB test	0.415179	0.183432	0.254446	0.809441

In [715]:
```
def threshold_finder(y_test_data, probabilities, desired_recall):
    '''
    Find the decision threshold that most closely yields a desired recall scor

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

    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 t
        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 gre
    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
```

In [716]:
```
# 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[716]:
```
(0.114, 0.5009861932938856)
```

In [717]:
```
# 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[717]:

	model	precision	recall	F1	accuracy
0	XGB, threshold = 0.124	0.301337	0.489152	0.372932	0.708392

6 Conclusion

In the minimum without the regression model or machine learning, we understand the following:

- Most variables were right-skewed or uniformly distributed.
- While the data was largely consistent, some variables like driven_km_drives had dubious values.
- Discrepancies in monthly metrics (e.g., activity_days vs. driving_days) suggest potential data inconsistencies.
- It's advisable to consult the Waze data team about these inconsistencies and the recent surge in app usage by long-term users.
- User retention stood at ~82% with 18% churning.
- Users driving longer distances per day were more likely to churn; frequent drivers were less likely.
- User tenure in the data ranged from new to ~10 years, evenly represented.

Hypothesis testing insights:

- A primary observation is that, on average, drivers using iPhones record a similar drive count as Android users.

- A logical follow-up would be to delve into other elements affecting the drive count and conduct more hypothesis testing to understand user patterns better. Additionally, short-term adjustments in Waze app's marketing or user interface might offer further data to scrutinize churn.

Regression and machine learning model:

- The model would not be recomended for use commercially or by the business, instead it can be used as a follow up point internally to keep improving the model as more data comes in. A lack of data can impede the process. In this case, I do not have access to waze or am i able to ask them follow up questions from the "understanding the data" section.

- If maximizing recall is more important (i.e., catching as many churn cases as possible), then the previous XGBoost model provided (with a threshold of 0.124) is the best. It has the highest recall at 48.92%, even though its precision is lower and its accuracy is considerably lower than the others.

- I was able to detect feature importance which is vital for following up on the model. We now know activity_days is the most important feature under the regression model section. Whilst in the machine learning model km_per_hour was the most important feature.

- Engineered features accounted for 6 of 10 top features.

- The ensembles of tree-based models in this project milestone are more valuable than a singular logistic regression model because they achieve higher scores across all evaluation metrics and require less preprocessing of the data. However, it is more difficult to understand how they make their