Waze Project

Course 5 - Regression analysis: Simplify complex data relationships

Your team is more than halfway through their user churn project. Earlier, you completed a project proposal, used Python to explore and analyze Waze's user data, created data visualizations, and conducted a hypothesis test. Now, leadership wants your team to build a regression model to predict user churn based on a variety of variables.

You check your inbox and discover a new email from Ursula Sayo, Waze's Operations Manager. Ursula asks your team about the details of the regression model. You also notice two follow-up emails from your supervisor, May Santner. The first email is a response to Ursula, and says that the team will build a binomial logistic regression model. In her second email, May asks you to help build the model and prepare an executive summary to share your results.

A notebook was structured and prepared to help you in this project. Please complete the following questions and prepare an executive summary.

Course 5 End-of-course project: Regression modeling

In this activity, you will build a binomial logistic regression model. As you have learned, logistic regression helps you estimate the probability of an outcome. For data science professionals, this is a useful skill because it allows you to consider more than one variable against the variable you're measuring against. This opens the door for much more thorough and flexible analysis to be completed.

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

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

This activity has three parts:

Part 1: EDA & Checking Model Assumptions

- What are some purposes of EDA before constructing a binomial logistic regression model?

Part 2: Model Building and Evaluation

- What resources do you find yourself using as you complete this stage?

Part 3: Interpreting Model Results

- What key insights emerged from your model(s)?
- What business recommendations do you propose based on the models built?

Follow the instructions and answer the question below to complete the activity. Then, you will complete an executive summary using the questions listed on the PACE Strategy Document.

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

Build a regression model

PACE stages

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

PACE: Plan

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

Task 1. Imports and data loading

Import the data and packages that you've learned are needed for building logistic regression models.

In [1]:
```
# Packages for numerics + dataframes
### YOUR CODE HERE ###
import pandas as pd
import numpy as np
# Packages for visualization
### YOUR CODE HERE ###
import matplotlib.pyplot as plt
import seaborn as sns
# Packages for Logistic Regression & Confusion Matrix
### YOUR CODE HERE ###
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
```

Import the dataset.

Note: As shown in this cell, the dataset has been automatically loaded in for you. You do not need to download the .csv file, or provide more code, in order to access the dataset and

In [2]:
```
# Load the dataset by running this cell

df = pd.read_csv('waze_dataset.csv')
```

PACE: Analyze

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

In this stage, consider the following question:

- What are some purposes of EDA before constructing a binomial logistic regression model?

==> ENTER YOUR RESPONSE HERE: To clean the data, to understand the data and columns within it. To identify dependent and independent variables.

Task 2a. Explore data with EDA

Analyze and discover data, looking for correlations, missing data, potential outliers, and/or duplicates.

Start with `.shape` and `info()`.

In [4]:
```
### YOUR CODE HERE ###
print(df.shape)
print(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
None
```

Question: Are there any missing values in your data?

==> ENTER YOUR RESPONSE HERE: Yes in the label column.

Use `.head()`.

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

Out[5]:

	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

Use `.drop()` to remove the ID column since we don't need this information for your analysis.

In [6]:
```
### YOUR CODE HERE ###
df = df.drop(["ID"], axis=1)
```

Now, check the class balance of the dependent (target) variable, `label`.

In [10]:
```
### YOUR CODE HERE ###
class_balance = df["label"].value_counts()
class_balance
```

Out[10]:
```
retained    11763
churned      2536
Name: label, dtype: int64
```

Call `.describe()` on the data.

In [11]:
```
### YOUR CODE HERE ###
df.describe()
```

Out[11]:

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

Question: Are there any variables that could potentially have outliers just by assessing at the quartile values, standard deviation, and max values?

==> ENTER YOUR RESPONSE HERE: Yes there are multiple columns with outliers.

Task 2b. Create features

Create features that may be of interest to the stakeholder and/or that are needed to address the business scenario/problem.

`km_per_driving_day`

You know from earlier EDA that churn rate correlates with distance driven per driving day in the last month. It might be helpful to engineer a feature that captures this information.

1. Create a new column in `df` called `km_per_driving_day`, which represents the mean distance driven per driving day for each user.

In [12]:
```
# 1. Create `km_per_driving_day` column
### YOUR CODE HERE ###
df["km_per_driving_day"] = df["driven_km_drives"] / df["driving_days"]
# 2. Call `describe()` on the new column
### YOUR CODE HERE ###
df["km_per_driving_day"].describe()
```

Out[12]:
```
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. Pandas imputes a value of infinity in the corresponding rows of the new column because division by zero is undefined.

1. Convert these values from infinity to zero. You can use `np.inf` to refer to a value of infinity.
2. Call `describe()` on the `km_per_driving_day` column to verify that it worked.

In [13]:
```
# 1. Convert infinite values to zero
### YOUR CODE HERE ###
df["km_per_driving_day"].replace([np.inf, -np.inf], 0, inplace=True)
# 2. Confirm that it worked
### YOUR CODE HERE ###
df["km_per_driving_day"].describe()
```

Out[13]:
```
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`

Create a new, binary feature called `professional_driver` that is a 1 for users who had 60 or more drives **and** drove on 15+ days in the last month.

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

To create this column, use the `np.where()` (https://numpy.org/doc/stable/reference/generated/numpy.where.html) function. This function accepts as arguments:

1. A condition
2. What to return when the condition is true
3. What to return when the condition is false

```
Example:
x = [1, 2, 3]
x = np.where(x > 2, 100, 0)
x
array([  0,   0, 100])
```

In [14]:
```
# Create `professional_driver` column
### YOUR CODE HERE ###
df['professional_driver'] = np.where((df['drives'] >= 60) & (df['driving_days'
```

Perform a quick inspection of the new variable.

1. Check the count of professional drivers and non-professionals
2. Within each class (professional and non-professional) calculate the churn rate

In [15]:
```
# 1. Check count of professionals and non-professionals
### YOUR CODE HERE ###
print(df["professional_driver"].value_counts())
# 2. Check in-class churn rate
### YOUR CODE HERE ###
df.groupby(["professional_driver"])["label"].value_counts(normalize=True)
```

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

Out[15]:
```
professional_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.

PACE: Construct

After analysis and deriving variables with close relationships, it is time to begin constructing the model.

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

In this stage, consider the following question:

- Why did you select the X variables you did?

==> ENTER YOUR RESPONSE HERE: Since we know the label is the column that is supposed to be predicted and other columns that we have made are supposed to be used as X variables and all columns that have high correlation with the variable that is to be predicted.

Task 3a. Preparing variables

Call `info()` on the dataframe to check the data type of the `label` variable and to verify if there are any missing values.

In [16]:
```
### YOUR CODE HERE ###
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
```

Because you know from previous EDA that there is no evidence of a non-random cause of the 700 missing values in the `label` column, and because these observations comprise less than 5% of the data, use the `dropna()` method to drop the rows that are missing this data.

In [17]:
```
# Drop rows with missing data in `label` column
### YOUR CODE HERE ###
df = df.dropna(subset=["label"])
```

Impute outliers

You rarely want to drop outliers, and generally will not do so unless there is a clear reason for it (e.g., typographic errors).

At times outliers can be changed to the **median, mean, 95th percentile, etc.**

Previously, you determined that seven of the variables had clear signs of containing outliers:

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

For this analysis, impute the outlying values for these columns. Calculate the **95th percentile** of each column and change to this value any value in the column that exceeds it.

In [22]:
```
# Impute outliers
### YOUR CODE HERE ###
for i in ["sessions", "drives", "total_sessions", "total_navigations_fav1", "t
    percentile_95 = df[i].quantile(0.95)
    df.loc[df[i] > percentile_95, i] = percentile_95
```

Call `describe()`.

In [23]:
```
### YOUR CODE HERE ###
df.describe()
```

Out[23]:

	sessions	drives	total_sessions	n_days_after_onboarding	total_navigations_fav
count	14299.000000	14299.000000	14299.000000	14299.000000	14299.00000
mean	76.539688	63.964683	183.716684	1751.822505	114.56276
std	67.243178	55.127927	118.719100	1008.663834	124.37855
min	0.000000	0.000000	0.220211	4.000000	0.00000
25%	23.000000	20.000000	90.457733	878.500000	10.00000
50%	56.000000	48.000000	158.718571	1749.000000	71.00000
75%	111.000000	93.000000	253.540450	2627.500000	178.00000
max	243.000000	200.000000	455.427084	3500.000000	422.00000

Encode categorical variables

Change the data type of the `label` column to be binary. This change is needed to train a logistic regression model.

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

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

Save this variable as `label2` as to not overwrite the original `label` variable.

Note: There are many ways to do this. Consider using `np.where()` as you did earlier in this notebook.

In [24]:
```
# Create binary `label2` column
### YOUR CODE HERE ###
df["label2"] = np.where(df["label"] == "churned", 1, 0)
df[["label", "label2"]].tail()
```

Out[24]:

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

Task 3b. Determine whether assumptions have been met

The following are the assumptions for logistic regression:

- Independent observations (This refers to how the data was collected.)
- No extreme outliers
- Little to no multicollinearity among X predictors
- Linear relationship between X and the **logit** of y

For the first assumption, you can assume that observations are independent for this project.

The second assumption has already been addressed.

The last assumption will be verified after modeling.

Note: In practice, modeling assumptions are often violated, and depending on the specifics of your use case and the severity of the violation, it might not affect your model much at all or it will result in a failed model.

Collinearity

Check the correlation among predictor variables. First, generate a correlation matrix.

In [26]:
```
# Generate a correlation matrix
### YOUR CODE HERE ###
correlation_matrix = df.corr(method = 'pearson')
correlation_matrix
```

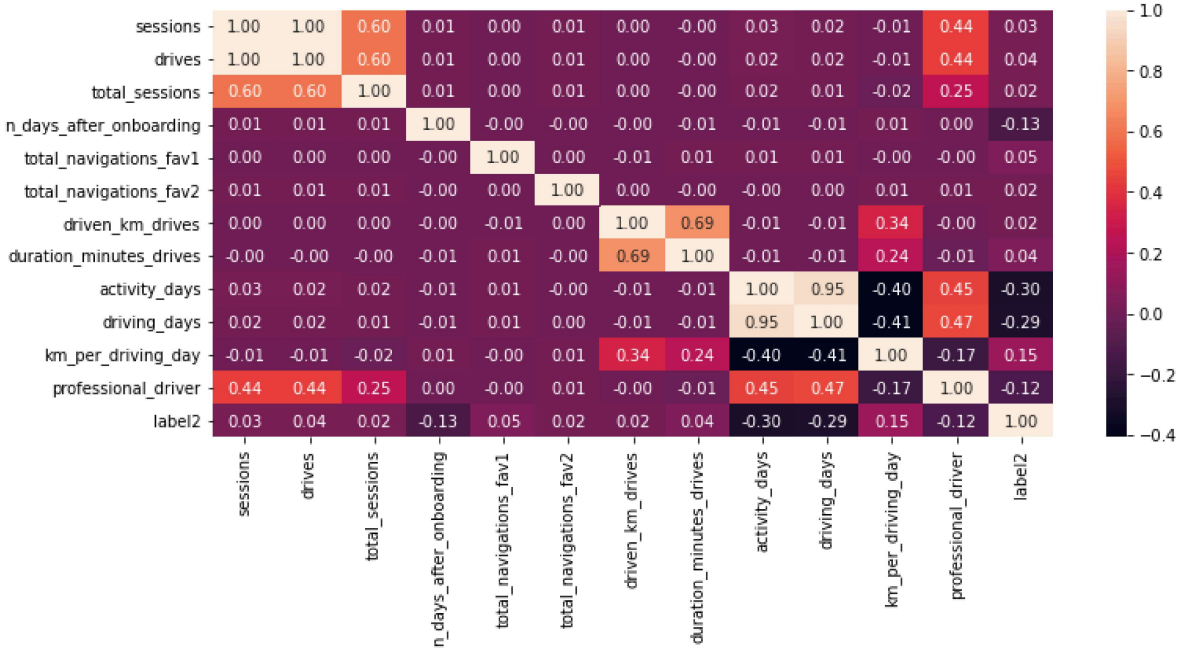
Out[26]:

	sessions	drives	total_sessions	n_days_after_onboarding	total_nav
sessions	1.000000	0.996942	0.597190	0.007101	
drives	0.996942	1.000000	0.595286	0.006940	
total_sessions	0.597190	0.595286	1.000000	0.006596	
n_days_after_onboarding	0.007101	0.006940	0.006596	1.000000	
total_navigations_fav1	0.001858	0.001058	0.000187	-0.002450	
total_navigations_fav2	0.008536	0.009505	0.010371	-0.004968	
driven_km_drives	0.002996	0.003445	0.001016	-0.004652	
duration_minutes_drives	-0.004545	-0.003889	-0.000338	-0.010167	
activity_days	0.025113	0.024357	0.015755	-0.009418	
driving_days	0.020294	0.019608	0.012953	-0.007321	
km_per_driving_day	-0.011569	-0.010989	-0.016167	0.011764	
professional_driver	0.443654	0.444425	0.254434	0.003770	
label2	0.034911	0.035865	0.024568	-0.129263	

Now, plot a correlation heatmap.

In [40]:
```
# Plot correlation heatmap
### YOUR CODE HERE ###
plt.figure(figsize = (12,5))
sns.heatmap(correlation_matrix,annot = True,fmt = '.2f')
```

Out[40]: <matplotlib.axes._subplots.AxesSubplot at 0x7257c955a1d0>



If there are predictor variables that have a Pearson correlation coefficient value greater than the **absolute value of 0.7**, these variables are strongly multicollinear. Therefore, only one of these variables should be used in your model.

Note: 0.7 is an arbitrary threshold. Some industries may use 0.6, 0.8, etc.

Question: Which variables are multicollinear with each other?

==> ENTER YOUR RESPONSE HERE: All variables with itself.

Task 3c. Create dummies (if necessary)

If you have selected `device` as an X variable, you will need to create dummy variables since this variable is categorical.

In cases with many categorical variables, you can use pandas built-in `pd.get_dummies()` (https://pandas.pydata.org/docs/reference/api/pandas.get_dummies.html), or you can use scikit-learn's `OneHotEncoder()` (https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.OneHotEncoder.html) function.

Note: Variables with many categories should only be dummied if absolutely necessary. Each category will result in a coefficient for your model which can lead to overfitting.

Because this dataset only has one remaining categorical feature (`device`), it's not necessary to use one of these special functions. You can just implement the transformation directly.

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

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

In [41]:
```
# Create new `device2` variable
### YOUR CODE HERE ###
df["device2"] = np.where(df["device"] == "Android", 0, 1)
df[["device", "device2"]].tail()
```

Out[41]:

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

Task 3d. Model building

Assign predictor variables and target

To build your model you need to determine what X variables you want to include in your model to predict your target— `label2` .

Drop the following variables and assign the results to `X` :

- `label` (this is the target)
- `label2` (this is the target)
- `device` (this is the non-binary-encoded categorical variable)
- `sessions` (this had high multicollinearity)
- `driving_days` (this had high multicollinearity)

Note: Notice that `sessions` and `driving_days` were selected to be dropped, rather than `drives` and `activity_days` . The reason for this is that the features that were kept for modeling had slightly stronger correlations with the target variable than the features that were dropped.

In [42]:
```
# Isolate predictor variables
### YOUR CODE HERE ###
x = df.drop(columns=["label", "label2", "device", "sessions", "driving_days"])
```

Now, isolate the dependent (target) variable. Assign it to a variable called `y` .

In [43]:
```
# Isolate target variable
### YOUR CODE HERE ###
y = df["label2"]
```

Split the data

Use scikit-learn's `train_test_split()` (https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html) function to perform a train/test split on your data using the X and y variables you assigned above.

Note 1: It is important to do a train test to obtain accurate predictions. You always want to fit your model on your training set and evaluate your model on your test set to avoid data leakage.

Note 2: Because the target class is imbalanced (82% retained vs. 18% churned), you want to make sure that you don't get an unlucky split that over- or under-represents the frequency of the minority class. Set the function's `stratify` parameter to `y` to ensure that the minority class appears in both train and test sets in the same proportion that it does in the overall dataset.

In [44]:
```
# Perform the train-test split
### YOUR CODE HERE ###
x_train, x_test, y_train, y_test = train_test_split(x, y, stratify=y, random_s
```

In [45]:
```
# Use .head()
### YOUR CODE HERE ###
x_train.head()
```

Out[45]:

	drives	total_sessions	n_days_after_onboarding	total_navigations_fav1	total_navigations_f
152	108	186.192746	3116	243	
11899	2	3.487590	794	114	
10937	139	347.106403	331	4	
669	108	455.427084	2320	11	
8406	10	89.475821	2478	135	

Use scikit-learn to instantiate a logistic regression model. Add the argument `penalty = None`.

It is important to add `penalty = None` since your predictors are unscaled.

Refer to scikit-learn's logistic regression (https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html) documentation for more information.

Fit the model on `X_train` and `y_train`.

In [46]:
```
### YOUR CODE HERE ###
model = LogisticRegression(penalty="none", max_iter=400)
model.fit(x_train, y_train)
```

Out[46]:
```
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)
```

Call the `.coef_` attribute on the model to get the coefficients of each variable. The coefficients are in order of how the variables are listed in the dataset. Remember that the coefficients represent the change in the **log odds** of the target variable for **every one unit increase in X**.

If you want, create a series whose index is the column names and whose values are the coefficients in `model.coef_`.

In [47]:
```
### YOUR CODE HERE ###
pd.Series(model.coef_[0], index=x.columns)
```

Out[47]:
```
drives                    0.001913
total_sessions            0.000328
n_days_after_onboarding  -0.000407
total_navigations_fav1    0.001231
total_navigations_fav2    0.000930
driven_km_drives         -0.000015
duration_minutes_drives   0.000109
activity_days            -0.106030
km_per_driving_day        0.000018
professional_driver      -0.001529
device2                  -0.001041
dtype: float64
```

Call the model's `intercept_` attribute to get the intercept of the model.

In [48]:
```
### YOUR CODE HERE ###
model.intercept_
```

Out[48]:
```
array([-0.00170675])
```

Check final assumption

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

Call the model's `predict_proba()` method to generate the probability of response for each sample in the training data. (The training data is the argument to the method.) Assign the result to a variable called `training_probabilities`. This results in a 2-D array where each row represents a user in `X_train`. The first column is the probability of the user not churning, and the second column is the probability of the user churning.

In [49]:
```
# Get the predicted probabilities of the training data
### YOUR CODE HERE ###
training_probabilities = model.predict_proba(x_train)
training_probabilities
```

Out[49]:
```
array([[0.93964519, 0.06035481],
       [0.6196859 , 0.3803141 ],
       [0.76460081, 0.23539919],
       ...,
       [0.91909159, 0.08090841],
       [0.85093128, 0.14906872],
       [0.93515753, 0.06484247]])
```

In logistic regression, the relationship between a predictor variable and the dependent variable does not need to be linear, however, the log-odds (a.k.a., logit) of the dependent variable with respect to the predictor variable should be linear. Here is the formula for calculating log-odds, where *p* is the probability of response:

$$logit(p) = ln(\frac{p}{1-p})$$

1. Create a dataframe called `logit_data` that is a copy of `df` .
2. Create a new column called `logit` in the `logit_data` dataframe. The data in this column should represent the logit for each user.

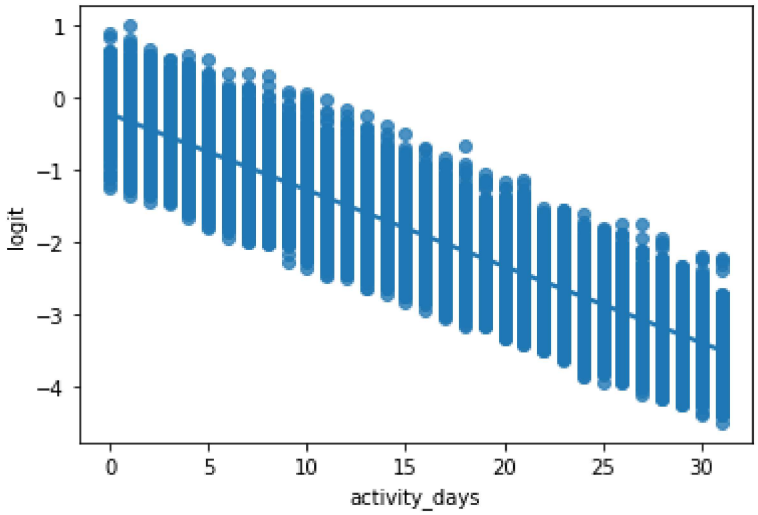
In [50]:
```
# 1. Copy the `X_train` dataframe and assign to `logit_data`
### YOUR CODE HERE ###
logit_data = x_train.copy()
# 2. Create a new `logit` column in the `logit_data` df
### YOUR CODE HERE ###
logit_data['logit'] = [np.log(prob[1] / prob[0]) for prob in training_probabil
```

Plot a regplot where the x-axis represents an independent variable and the y-axis represents the log-odds of the predicted probabilities.

In an exhaustive analysis, this would be plotted for each continuous or discrete predictor variable. Here we show only `driving_days` .

In [63]:
```
# Plot regplot of `activity_days` log-odds
### YOUR CODE HERE ###
sns.regplot(x="activity_days", y="logit", data=logit_data)
```

Out[63]: <matplotlib.axes._subplots.AxesSubplot at 0x7257c8c96050>



PACE: Execute

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

Task 4a. Results and evaluation

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

Use the code block below to make predictions on the test data.

In [52]:
```
# Generate predictions on X_test
### YOUR CODE HERE ###
y_preds = model.predict(x_test)
```

Now, use the `score()` method on the model with `X_test` and `y_test` as its two arguments. The default score in scikit-learn is **accuracy**. What is the accuracy of your model?

Consider: Is accuracy the best metric to use to evaluate this model?

In [53]:
```
# Score the model (accuracy) on the test data
### YOUR CODE HERE ###
model.score(x_test, y_test)
```

Out[53]: 0.8237762237762237

Task 4b. Show results with a confusion matrix

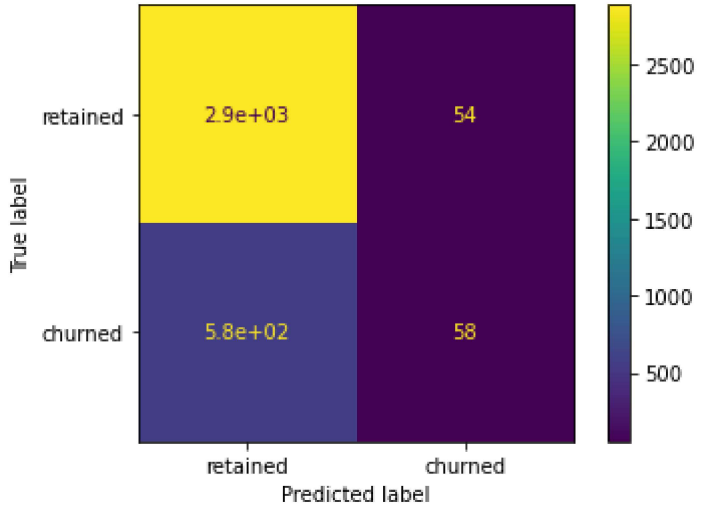
Use the `confusion_matrix` function to obtain a confusion matrix. Use `y_test` and `y_preds` as arguments.

In [54]:
```
### YOUR CODE HERE ###
confusion_matrix = confusion_matrix(y_test, y_preds)
```

Next, use the `ConfusionMatrixDisplay()` function to display the confusion matrix from the above cell, passing the confusion matrix you just created as its argument.

In [56]:
```
### YOUR CODE HERE ###
disp = ConfusionMatrixDisplay(confusion_matrix=confusion_matrix,
                              display_labels=["retained", "churned"])
disp.plot()
```

Out[56]:
```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x7257c8e49
d90>
```



You can use the confusion matrix to compute precision and recall manually. You can also use scikit-learn's `classification_report()` (https://scikit-learn.org/stable/modules/generated/sklearn.metrics.classification_report.html) function to generate a table from `y_test` and `y_preds`.

In [57]:
```
# Calculate precision manually
### YOUR CODE HERE ###
precision = confusion_matrix[1,1] / (confusion_matrix[0, 1] + confusion_matrix
precision
```

Out[57]:
```
0.5178571428571429
```

In [58]:
```
# Calculate recall manually
### YOUR CODE HERE ###
recall = confusion_matrix[1,1] / (confusion_matrix[1, 0] + confusion_matrix[1,
recall
```

Out[58]:
```
0.0914826498422713
```

In [59]:
```
# Create a classification report
### YOUR CODE HERE ###
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
```

Note: The model has decent precision but very low recall, which means that it makes a lot of false negative predictions and fails to capture users who will churn.

BONUS

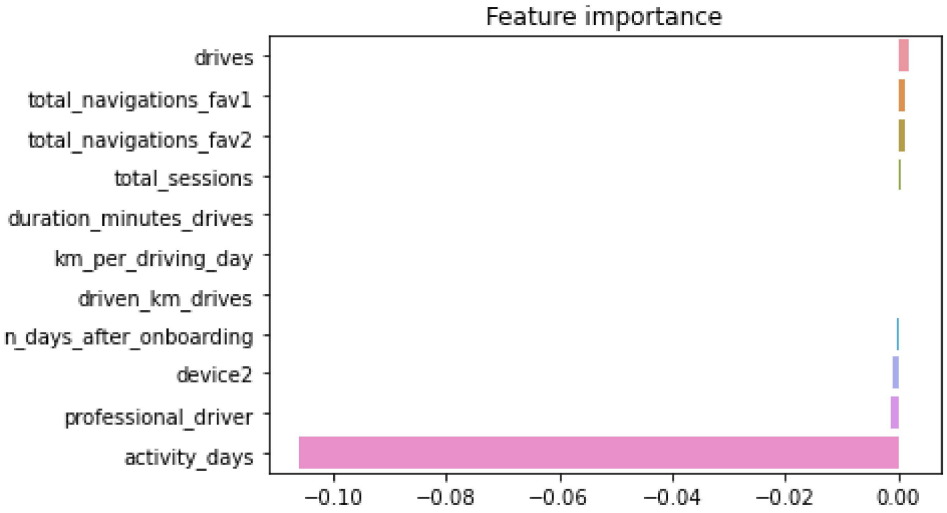
Generate a bar graph of the model's coefficients for a visual representation of the importance of the model's features.

In [60]:
```
# Create a list of (column_name, coefficient) tuples
### YOUR CODE HERE ###
importance = list(zip(x_train.columns, model.coef_[0]))
# Sort the list by coefficient value
### YOUR CODE HERE ###
importance = sorted(importance, key=lambda x: x[1], reverse=True)
importance
```

Out[60]:
```
[('drives', 0.0019132432505302545),
 ('total_navigations_fav1', 0.0012314938307091327),
 ('total_navigations_fav2', 0.0009304689580629822),
 ('total_sessions', 0.00032779487873899814),
 ('duration_minutes_drives', 0.00010910494608116396),
 ('km_per_driving_day', 1.8231952961715533e-05),
 ('driven_km_drives', -1.4874276552052445e-05),
 ('n_days_after_onboarding', -0.0004065049051799866),
 ('device2', -0.00104115847957279),
 ('professional_driver', -0.0015285951346215329),
 ('activity_days', -0.10603026704831042)]
```

In [61]:
```
# Plot the feature importances
### YOUR CODE HERE ###
sns.barplot(x=[x[1] for x in importance],
            y=[x[0] for x in importance],
            orient='h')
plt.title("Feature importance")
```

Out[61]: Text(0.5, 1.0, 'Feature importance')



Task 4c. Conclusion

Now that you've built your regression model, the next step is to share your findings with the Waze leadership team. Consider the following questions as you prepare to write your executive summary. Think about key points you may want to share with the team, and what information is most relevant to the user churn project.

Questions:

1. What variable most influenced the model's prediction? How? Was this surprising?
2. Were there any variables that you expected to be stronger predictors than they were?
3. Why might a variable you thought to be important not be important in the model?
4. Would you recommend that Waze use this model? Why or why not?
5. What could you do to improve this model?
6. What additional features would you like to have to help improve the model?

==> ENTER YOUR RESPONSES TO QUESTIONS 1-6 HERE

1. "activity day"
2. "km_per_driving_day"
3. "Cause initally it was used more and by the end of the project it was not used much."
4. "Completely depends on WAZE and the situation."
5. "Build more models based on other such variables."
6. "We can add more varaibles in order to improve the model."

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