Sparkify Music Service Churn Prediction Project with Spark

Author: Loredana Fattorini

November 6, 2021

Project Definition

Project Overview

Sparkify is a popular (not real!) music service similar to Spotify or Pandora with a subscription-based business model. Each user can listen to their favorite music every day either through the free-tier plan or by using a subscription plan where she pays a fixed monthly fee. Users can upgrade, downgrade, or cancel the service at any time, so it is critical to be sure users love the service.

Every time a user interacts with the service it generates (synthetic) data. Each event (e.g., song played, logout, like, downgrade, ...) is recorded with the corresponding timestamp. All of this information holds the key to keeping users happy and businesses thriving.

Problem Statement

The goal of the problem is to help Sparkify in answering the following question:

Which users are at risk of churn, i.e. downgrade from premium service to free-tier plan or cancellation of service altogether?

By identifying these users before they abandon the service, Sparkify can proactively engage with them by offering some discounts and/or incentives, which can save a lot of time and money in acquiring new users.

We explored several **binary classification models**, where the target variable is 0 if the customer did not churn and 1 otherwise.

To identify the best model that helps us in predicting customer churn, we completed the following tasks:

- Analyze and preprocess the data
- Use Machine learning pipelines

- Train each classifier (Logistic regression, Random forest, Gradient-boosted tree) on the training set
- Test each model on the test set
- Hyperparameter tuning and cross-validation
- Model evaluation

Finally, we built a **web app** using **Flask** web framework on the back-end and **Bootstrap** CSS framework on the front-end.
In the web app, you can **enter the information about the user** and **get her classification as a customer who is likely or not likely to churn** based on the model prediction.

Metrics

Typically, when we look at the results of a classification model, we focus on the correct predictions of all the predictions made by the model, i.e., the accuracy of the model. In our case, **when doing a churn analysis with the goal of predicting customers who churned, we are particularly** interested in having a lot of true positives. However, since there are many customers who have not churned (**highly imbalanced datasets**), the higher the number of true negatives the higher the accuracy, which can be misleading. Thus, **a better measure of model performance in this case is the F1 score**, which is the harmonic mean of **Precision** ('out of all customers who were labeled as "churned," how many did we correctly label as such?') and **Recall** ('out of all customers who were labeled as "churned," how many actually churned?').

Analysis

Data Exploration and Visualization

As input data we used the following datasets that contain Sparkify music events:

- Mini-dataset file (128MB) `*mini_sparkify_event_data.json*`
- Full dataset available (12GB) `*s3n://udacity-dsnd/sparkify/sparkify_event_data.json*`

Description of columns available in both datasets.

Column	Description
artist	Artist name related to the song of the event.
auth	User authentication status (e.g., Logged or Guest).
firstName	User first name.
gender	Gender (F or M).
itemInSession	Item count in a session.
lastName	User last name.
length	Length of song.
level	User plan (e.g., paid or free)
location	User's location at the time of the event
method	HTTP request method.
page	Page name (e.g., 'NextSong')
registration	Registration timestamp (unix timestamp)
sessionId	Session ID
song	Song name related to the event.
status	HTTP status.
ts	Event timestamp (unix timestamp).
userAgent	User's browser agent.
userId	User ID.

We used the **mini-dataset file,** which is a small subset of the full available dataset for **data exploration**. The raw dataset contains **286,500 events** and **18 features**.
All the details of the analysis are reported on the Jupiter Notebook `**Sparkify.ipynb**`.

printSchema

```
root
 |-- artist: string (nullable = true)
 |-- auth: string (nullable = true)
 |-- firstName: string (nullable = true)
 |-- gender: string (nullable = true)
 |-- itemInSession: long (nullable = true)
 |-- lastName: string (nullable = true)
 |-- length: double (nullable = true)
 |-- level: string (nullable = true)
 |-- location: string (nullable = true)
 |-- method: string (nullable = true)
 |-- page: string (nullable = true)
 |-- registration: long (nullable = true)
 |-- sessionId: long (nullable = true)
 |-- song: string (nullable = true)
 |-- status: long (nullable = true)
 |-- ts: long (nullable = true)
 |-- userAgent: string (nullable = true)
 |-- userId: string (nullable = true)
```

Example of a row.

```
Row(artist='Martha Tilston', auth='Logged In', firstName='Colin', gender='M', itemInSession=50, lastName='Freeman', length=27
7.89016, level='paid', location='Bakersfield, CA', method='PUT', page='NextSong', registration=1538173362000, sessionId=29, s
ong='Rockpools', status=200, ts=1538352117000, userAgent='Mozilla/5.0 (Windows NT 6.1; WOW64; rv:31.0) Gecko/20100101 Firefox
/31.0', userId='30')
```

Most of the **events** in the dataset happened between **October** and **November** 2018. The **registrations at the service** in the dataset happened during **March-November** 2018.

Missing values

We explored each column of the dataset and we found the following **number of missing values**:

Column	# missing values
artist	58392
firstName	8346
gender	8346
lastName	8346
length	58392
location	8346
registration	8346
song	58392
userAgent	8346
userId	8346

From the above count, we observe that:

1. There is **missing information for users not logged in**, corresponding to empty userID, firstName, … .
2. There are a lot of **missing values in *song, length, artist* columns** which correspond to the events where the user is not listening to music.

We decided to **focus only on data about users that were logged in** and hence remove rows corresponding to empty ***userId*** or ***firstName***. Indeed, records without any useful information about the user are not useful in predicting customer churn.

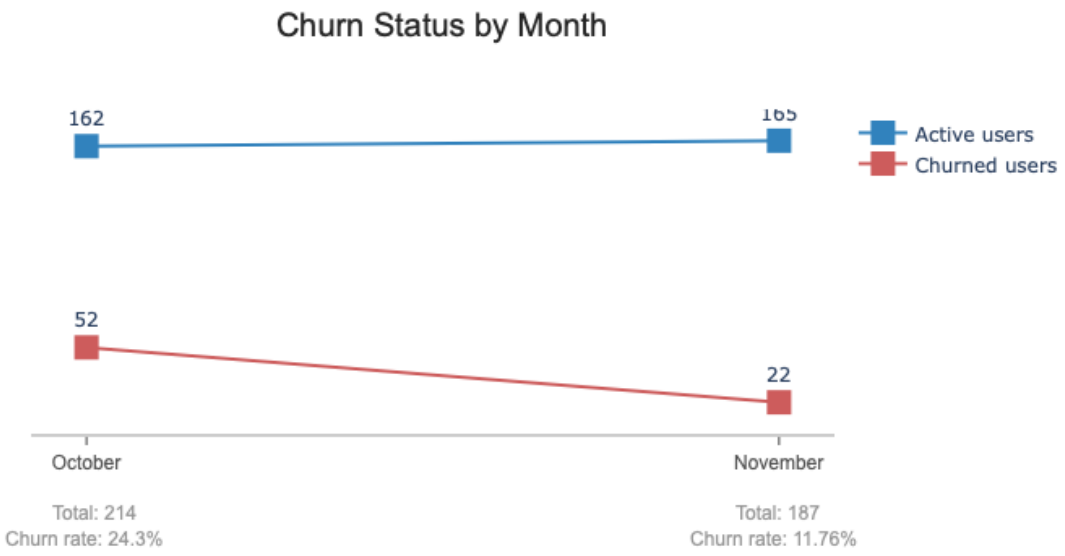
Churn definition

In this project, the **users churn** when they **cancel their subscription** from the platform. We consider the events on the '***Cancellation Confirmation***' page to define our churn, which **occurs for both paid and free plans**.

EDA and Data Visualization: Churned vs Stayed

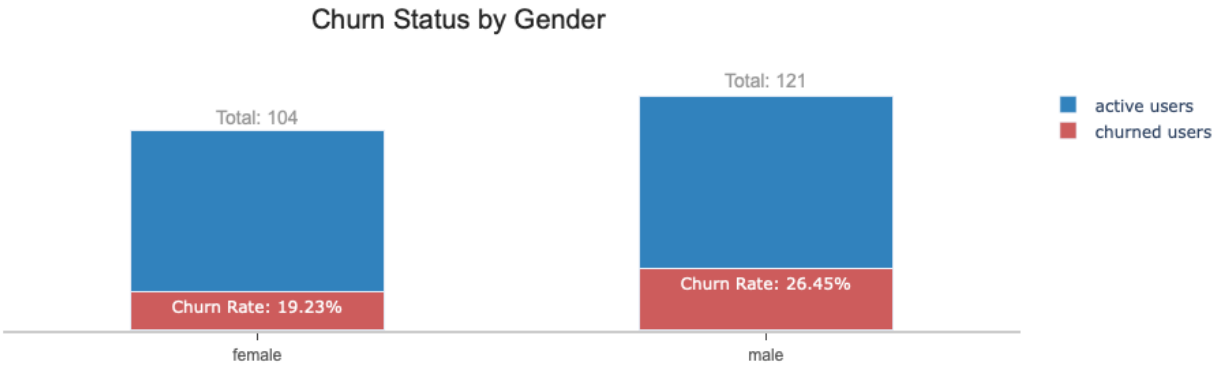
In the cleaned dataset there are **278,154 events** recorded generated by **225 unique users** of which **173 are active** users and **52 are users that cancelled then subscription**.

We have an **unbalanced dataset**: the number of active users are more than three times the number of users who churned.



Source: author's elaboration.

Does the gender affect the user churn status?



Source: author's elaboration.

Comparing female and **male users** we observe that:

- There are **more male users** (121) than female users (104).
- **Most of the users** who have **churned** are **male** (62%).
- Among **female** users **19% have churned**, while among **male** users **26% have churned**.
- **Female users** seem to be **more active**. They have generated **56% of the events**.
- **Female users who churned generated fewer events** than male users who churned.

Does the subscription plan (free vs paid) affect the user churn status?

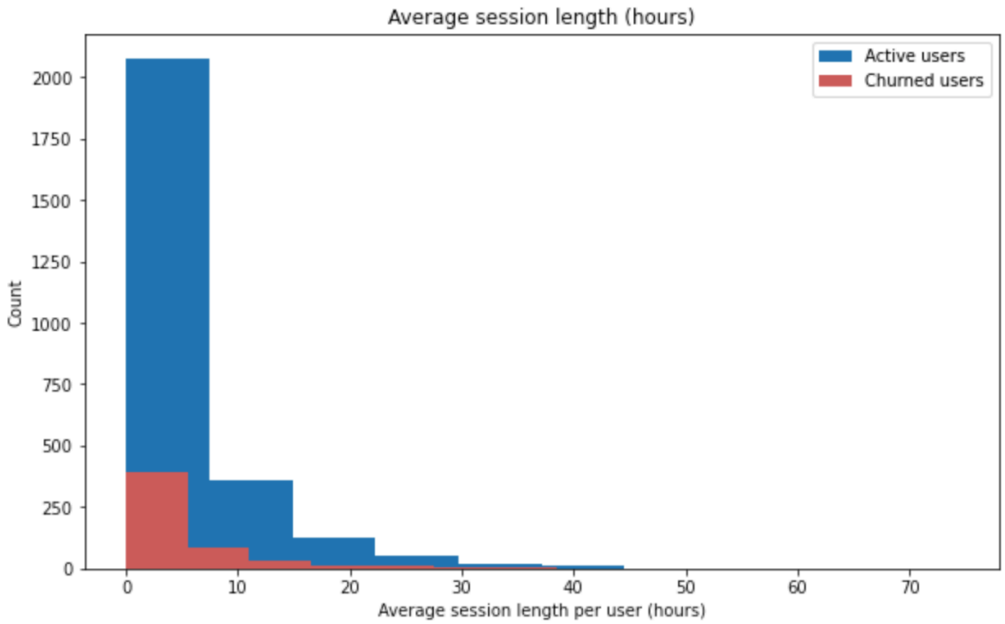
We found that:

- The **total number of paid and free accounts exceeds the total number of unique users**. This may be due to the fact that some of them may have upgraded or downgraded the service during the time period available in the dataset.
- There is **not a big difference between active and cancelled users in terms of the proportion of paid subscription plans**. Overall, **there are more free subscriptions than paid ones**. 44% of churned users were on a paid plan. 46% of active users are on a paid plan.
- **12% of the events** (interactions) are generated **by paying users who churned**.
- **4% of the events** (interactions) are generated **by users with a free subscription** who churned.
- **68% of the events** (interactions) are generated **by paying active users**.
- **16% of the events** (interactions) are generated **by active users with a free subscription**.

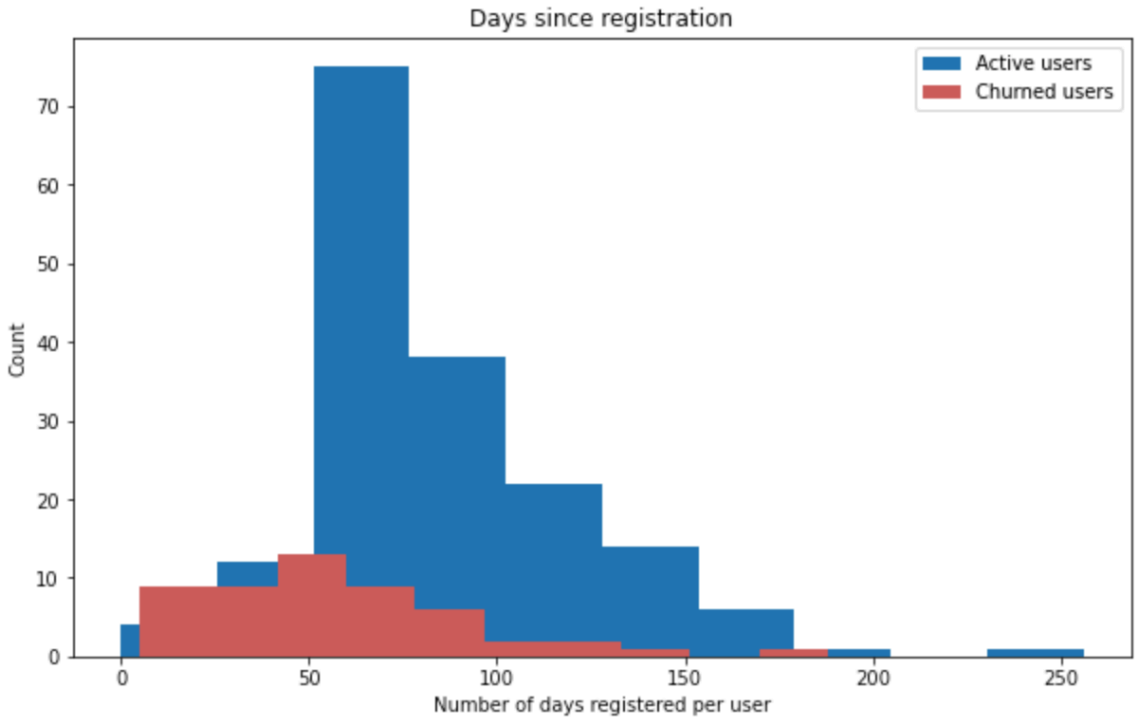
From the above we see that **users who paid for their subscription** are **more active** than those using the free plan, for active users and those who churned.

Explore interactions (events), sessions, days since registration

Variable	Churned (average)	Not churned (average)
Interactions (events) per user	863	1348
Sessions per user	10	15
Items in a session per user	84	88
Session length (hours)	4.72	5.05
Unique artists played	161.6	96.3
Unique songs played	369.56	308.31
Unique artists played per session	15.65	6.31
Unique songs played per session	35.79	20.21
Songs played	699.88	1108.17
Songs played per day	0.81	0.82
Number of days since registration	863	1348



Source: author's elaboration.



Source: author's elaboration.

On average, it seems that:

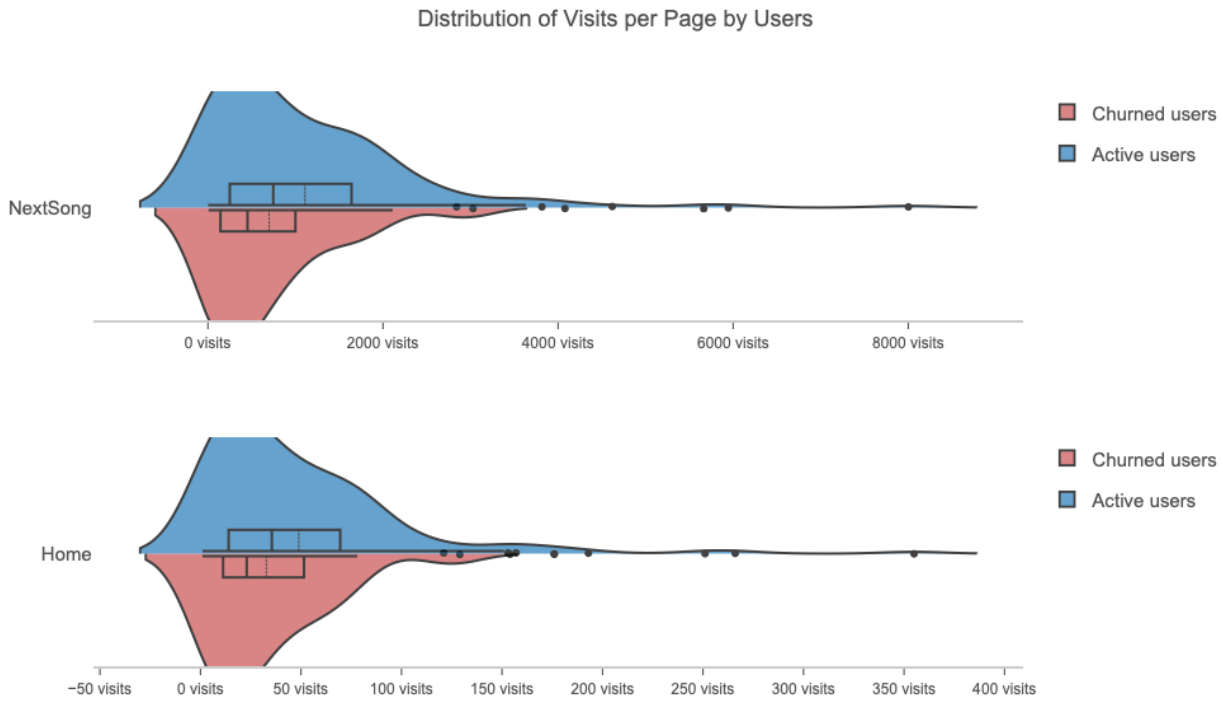
- **Active users interact more with the platform** and **spend more time on it** than churned users.
- Interestingly, **users who have churned have listened to more different artists and songs on average** than active users, while the latter are more active in terms of total songs played. It may be that churned users explore the service more.
- **Active users have been registered onto the platform for a longer time** than churned users.

Explore visits distribution per page by user

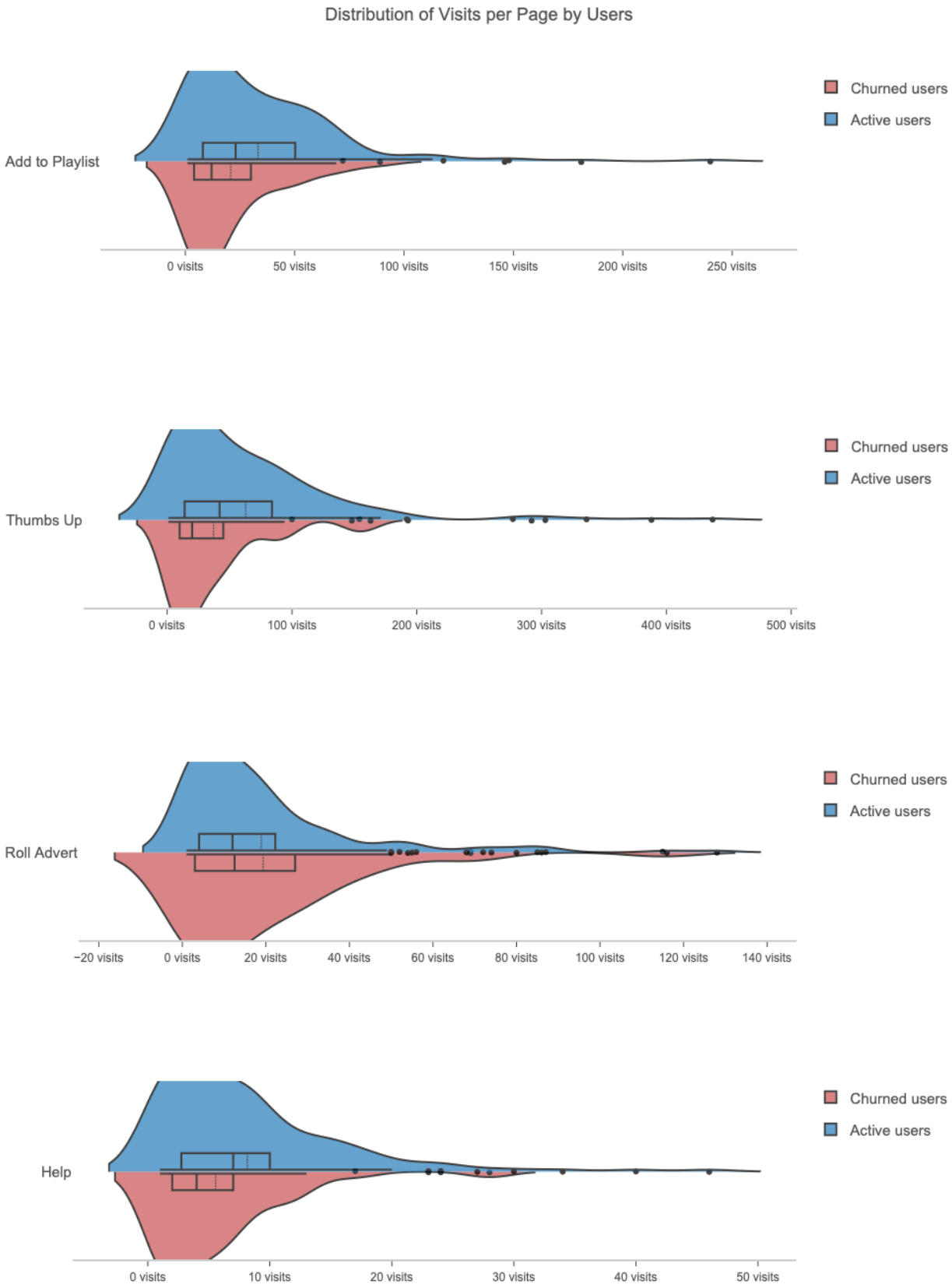
Distribution of events through page visited.

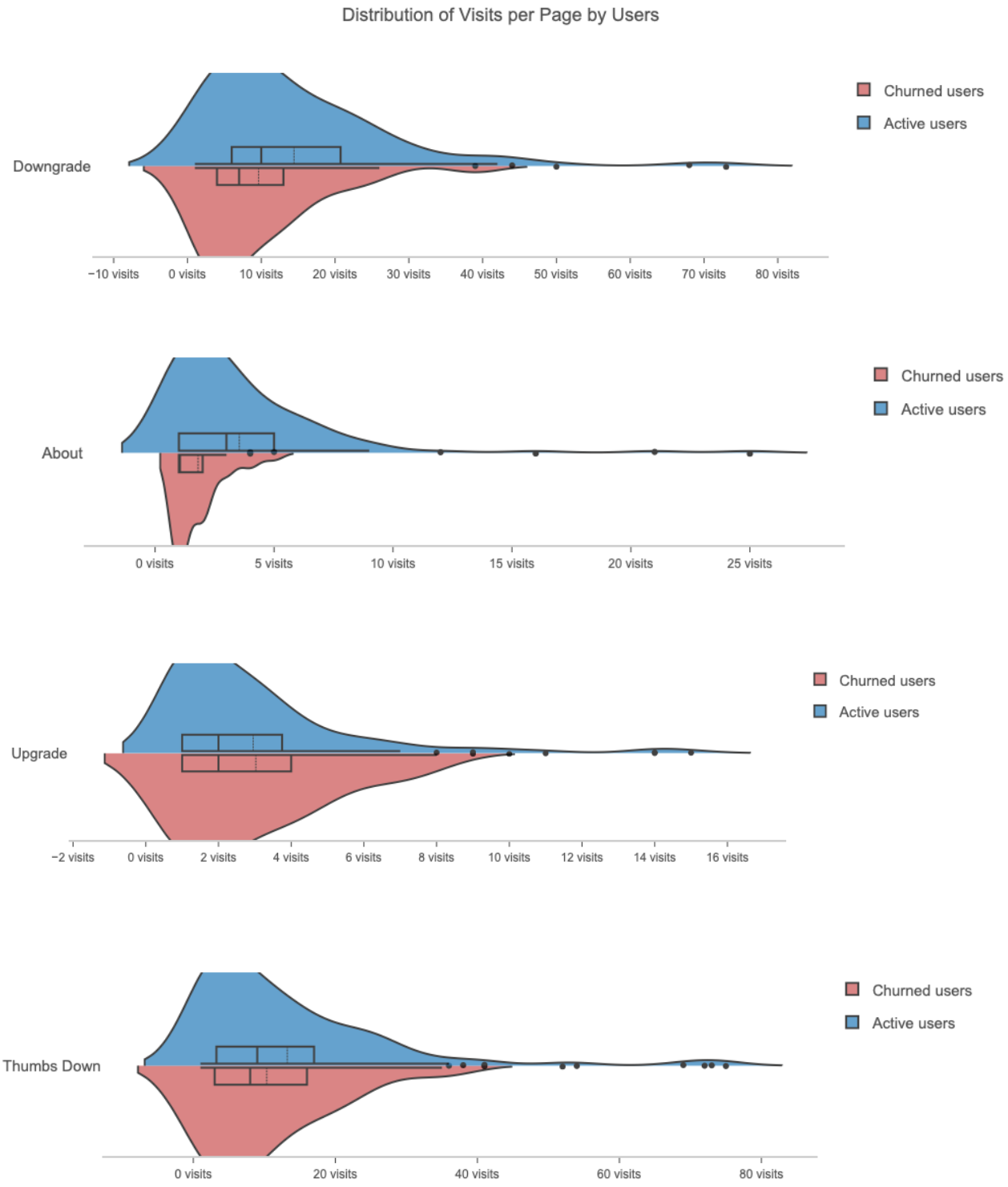
NextSong	228108
Thumbs Up	12551
Home	10082
Add to Playlist	6526
Add Friend	4277
Roll Advert	3933
Logout	3226
Thumbs Down	2546
Downgrade	2055
Settings	1514
Help	1454
Upgrade	499
About	495
Save Settings	310
Error	252
Submit Upgrade	159
Submit Downgrade	63
Cancellation Confirmation	52
Cancel	52

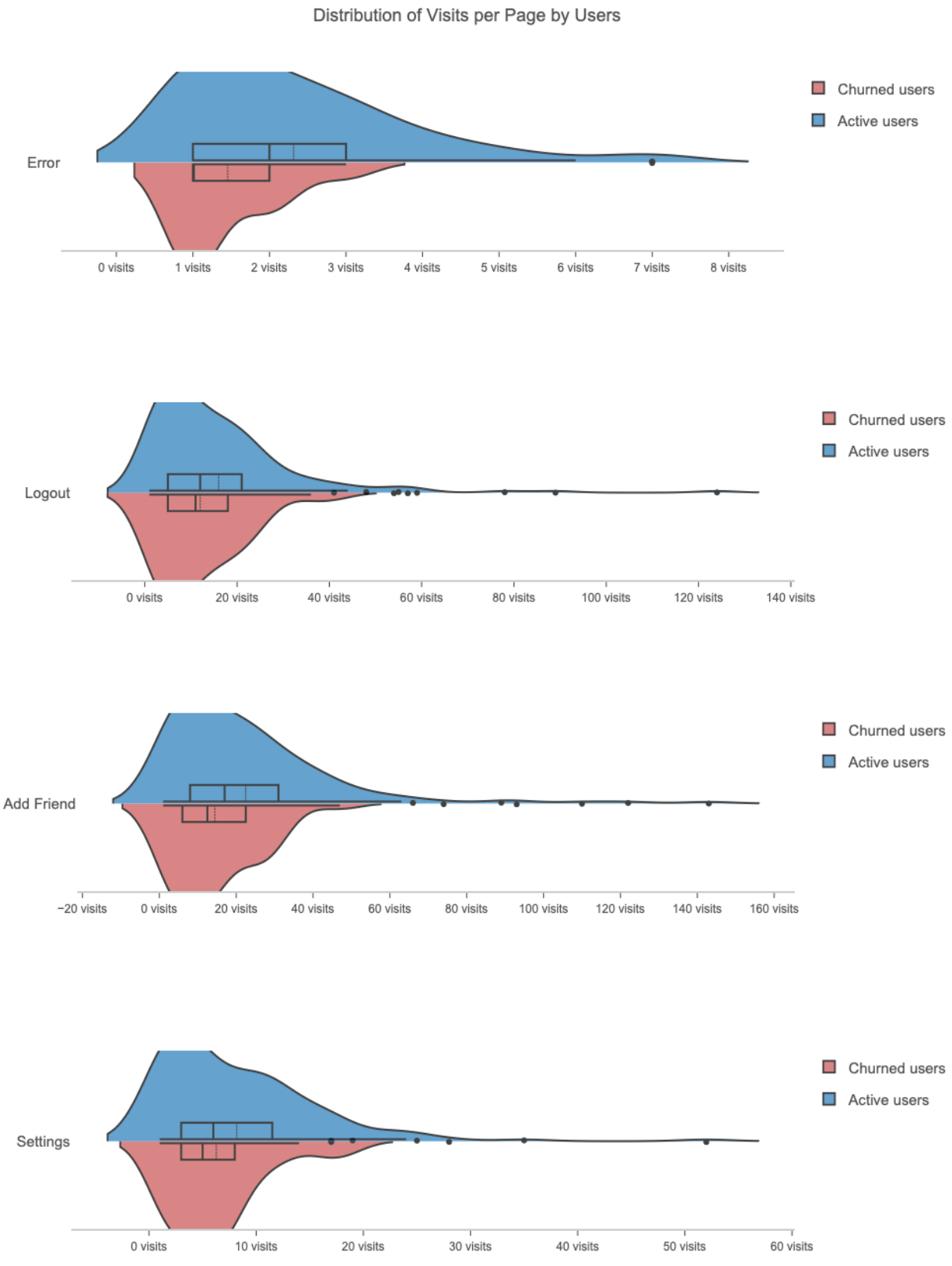
Name: page, dtype: int64

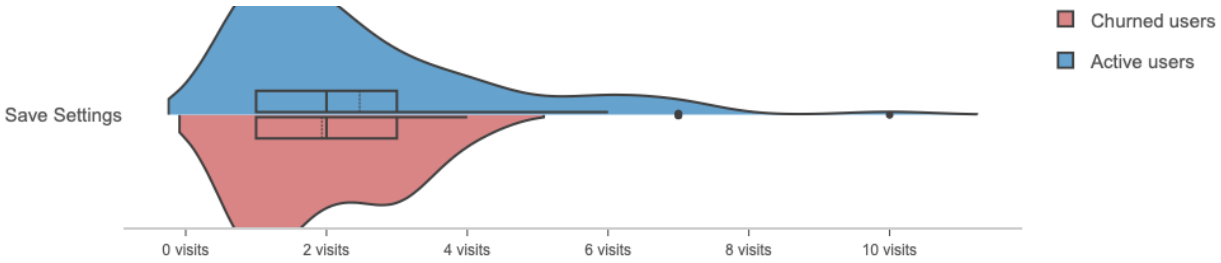


Source: author's elaboration.


Distribution of Visits per Page by Users
Add to Playlist
0 visits
50 visits
100 visits
150 visits
200 visits
250 visits
Churned users
Active users
Thumbs Up
0 visits
100 visits
200 visits
300 visits
400 visits
500 visits
Churned users
Active users
Roll Advert
−20 visits
0 visits
20 visits
40 visits
60 visits
80 visits
100 visits
120 visits
140 visits
Churned users
Active users
Help
0 visits
10 visits
20 visits
30 visits
40 visits
50 visits
Churned users
Active users


Distribution of Visits per Page by Users
Churned users
Active users
Downgrade
−10 visits
0 visits
10 visits
20 visits
30 visits
40 visits
50 visits
60 visits
70 visits
80 visits
Churned users
Active users
About
0 visits
5 visits
10 visits
15 visits
20 visits
25 visits
Churned users
Active users
Upgrade
−2 visits
0 visits
2 visits
4 visits
6 visits
8 visits
10 visits
12 visits
14 visits
16 visits
Churned users
Active users
Thumbs Down
0 visits
20 visits
40 visits
60 visits
80 visits

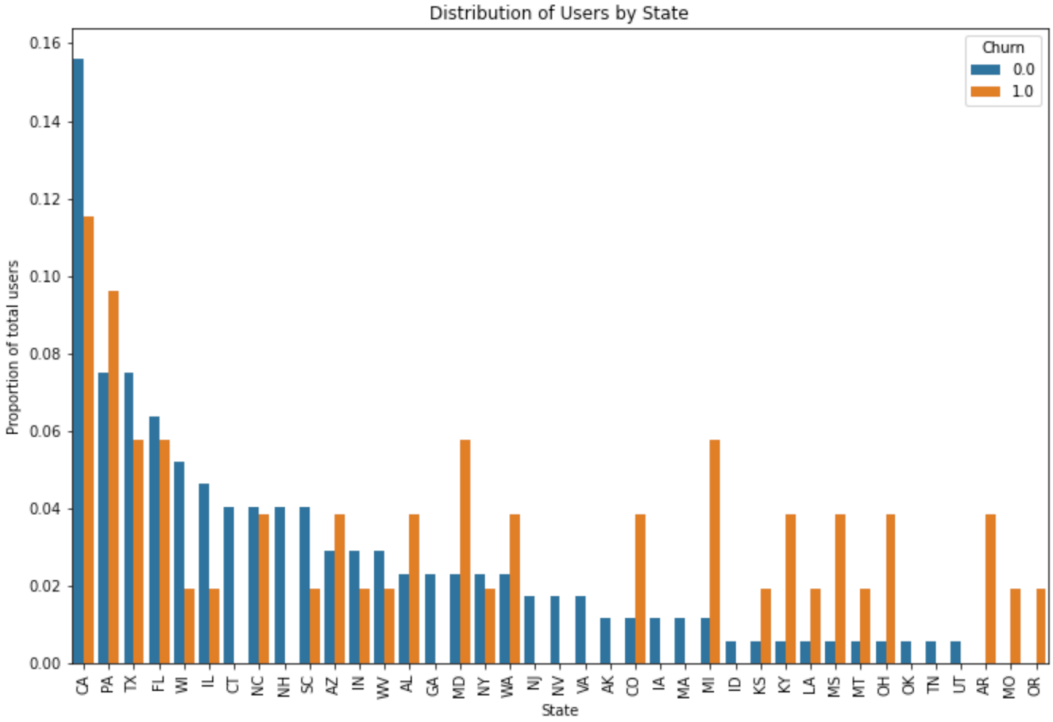

Distribution of Visits per Page by Users
Churned users
Active users
Error
0 visits
1 visits
2 visits
3 visits
4 visits
5 visits
6 visits
7 visits
8 visits
Churned users
Active users
Logout
0 visits
20 visits
40 visits
60 visits
80 visits
100 visits
120 visits
140 visits
Churned users
Active users
Add Friend
−20 visits
0 visits
20 visits
40 visits
60 visits
80 visits
100 visits
120 visits
140 visits
160 visits
Churned users
Active users
Settings
0 visits
10 visits
20 visits
30 visits
40 visits
50 visits
60 visits



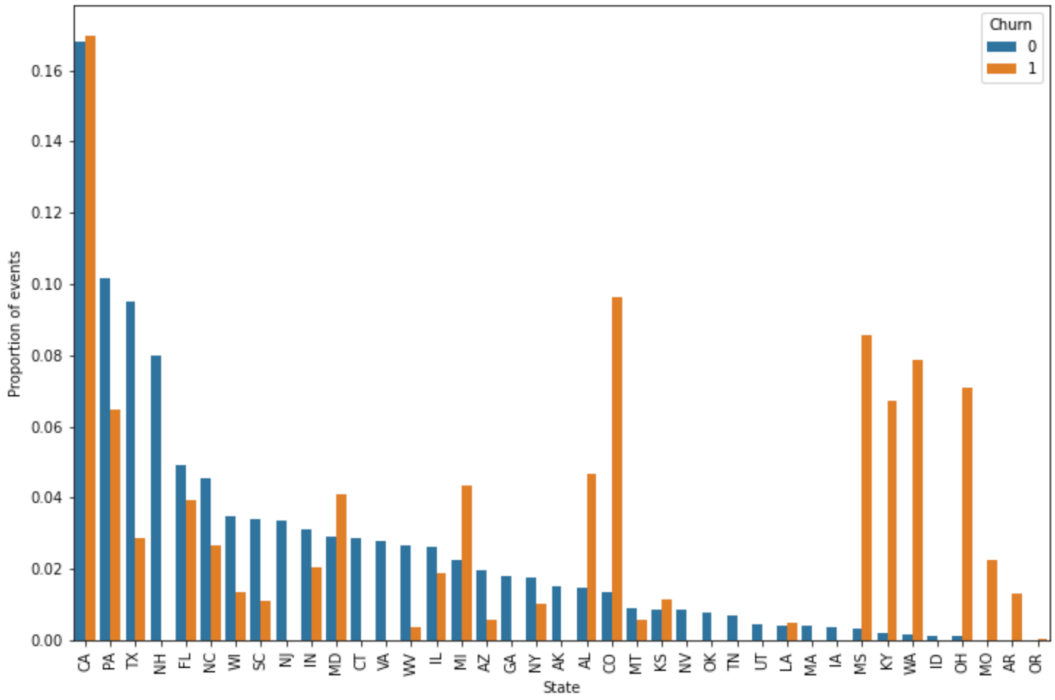
Variable	Churned (average)	Not churned (average)
Thumbs up	35.75	61.08
Thumbs down	9.54	11.85
Add to playlist	19.96	31.72
Add a friend	12.23	21.05

On average, **active users give more thumbs up and thumbs down, add more songs to the playlist, and add more friends** to the platform than users who cancelled, confirming their **higher engagement with the platform**.

Does user location affect churn status?



Source: author's elaboration.



Source: author's elaboration

- Overall, **there appear to be differences in the distribution of active and churned users across states**. In addition, there are some states that have no churned users (e.g., Connecticut and New Hampshire), and some states that have only churned users (e.g., Arkansas, Missouri, Oregon).
- **California, Pennsylvania, Texas, Florida have the most users and events** for active and churned users.

Methodology

Data Preprocessing

Before proceeding with the implementation of machine leaning models, we preprocessed the raw dataset following these steps:

1. **Remove rows where the user was not logged in** corresponding to missing userId, firstName, etc., and duplicates values.
2. **Define churn as 1 if the user cancel her subscription** as per '***Cancellation Confirmation***' page event and 0 otherwise.
3. Create a new dataset which contains the features extracted for each user.

To predict whether the user is going to churn, we extracted the following features:

- **Gender**: EDA showed that male and female users may have a different behavior.
- **Days since registration**: from EDA it appears that active users have been registered onto the platform for a longer time than churned users.
- **Last state**: state in which the most recent session is recorded.
- **Average songs played per day**: to see how much the user is active on the platform.
- **Last level**: the most recent user subscription plan.
- **Thumbs up proportion** (ratio thumbs up over thumbs down): to see the level of appreciation.
- **Number of add a friend events**: to see the level of engagement.
- **Average roll adverts per day**

Extract of the dataset preprocessed

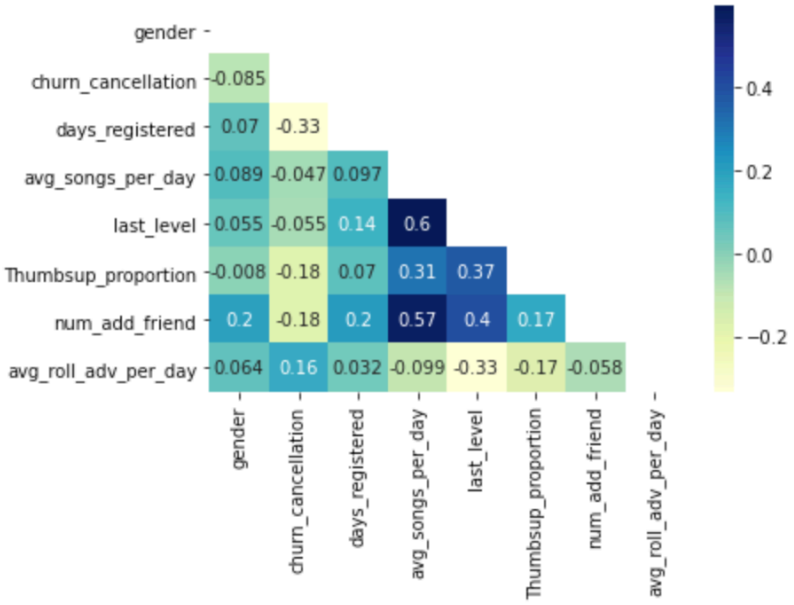
	userId	gender	churn_cancellation	days_registered	last_state	avg_songs_per_day	last_level	Thumbsup_proportion	num_add_friend
0	100010	1	0	56.0	CT	34.38	0	3.40	4
1	107	1	0	74.0	IL	49.20	0	7.00	2
2	110	0	0	69.0	WI	29.67	0	7.00	7
3	12	1	1	73.0	CA	78.82	1	4.67	13
4	137	0	0	123.0	PA	51.33	1	8.00	8

We also explored the **correlation with the target variable** and the **correlation among features** to check for **multicollinearity**.

Correlation with target variable
'churn_cancellation'

```
churn_cancellation      1.000000
days_registered         0.328647
num_add_friend          0.180956
Thumbsup_proportion     0.180752
avg_roll_adv_per_day    0.163344
gender                  0.085340
last_level              0.055309
avg_songs_per_day       0.046930
Name: churn_cancellation, dtype: float64
```

Correlation among features



Implementation

The **full dataset available contains a large amount of data**, which cannot be processed on a single machine. This is why we **first performed data analysis and model building with `Spark` (Pyspark and SparkML libraries) on the small subset** and then, we extended the analysis to the **entire dataset** using **AWS EMR clusters**.

The implementation consist of two steps:

1. Apply **several machine learning models** to predict user churn along with creating **machine learning pipelines**, **evaluating and tuning parameters**.
2. Develop a **web app** to demonstrate the classification of users based on the underlying model chosen to make the predictions.

Machine learning pipelines

After preprocessing the raw dataset and before proceeding with step 1, we **split the dataset into train and test sets** (80% - 20%), and the we built an **ML pipeline to predict user churn** that consists of:

- **String indexer** and **encoder** for categorical features.
- **Features assembler**.
- **Scaler**: we use *MinMaxScaler()* which transforms each column value into the range [0,1], preserving the shape of the data.
- **Classifiers: Logistic Regression, Random Forest, Gradient-Boosting Tree**.

Evaluation metrics

As mentioned earlier, **to check the performance of the models**, we used the **F1 score** as the metric. The reason is that even if we create a model with high accuracy, this may be due to the fact that the model is biased towards the majority class (‘not churned’, 0). In that case, the model does not serve our problem at all, which is to identify users who are likely to churn.

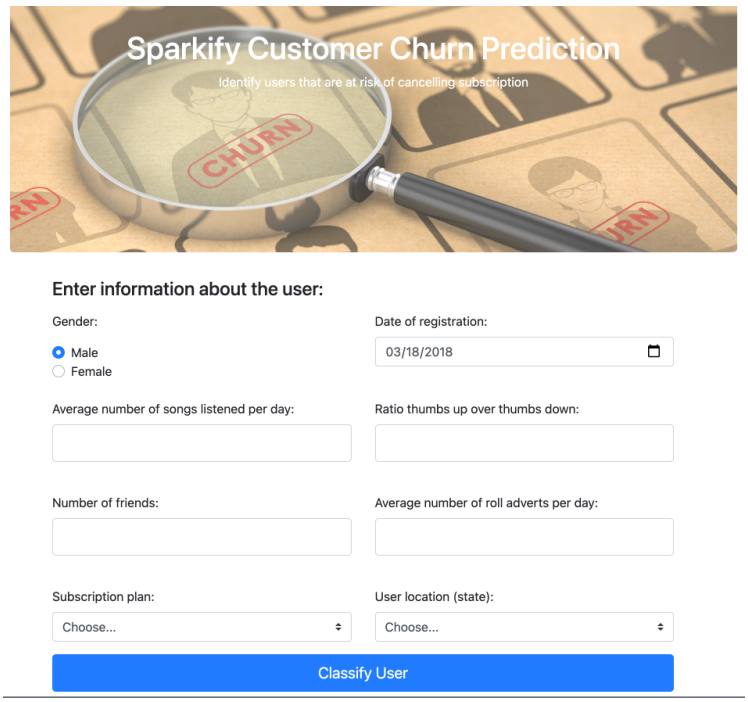
We created a **user defined function to compute evaluation metrics** (accuracy, precision, recall, F1) and the **confusion matrix**.

Web application

The we app has been implemented using **Flask** on the back-end and **Bootstrap** on the front-end.

To construct the web app we used the following scripts:

- ***run.py*** that contains the logic of the application. It handles the queries and display results on the web app.
- Folder **templates**:
  - ***master.html*** controls the main page
  - ***go.html*** controls classification result page
- Folder **static:**
  - **githublogo2.png** is the github logo used in the main page
  - **churn_image.png** is the background image downloaded from (https://www.shutterstock.com)
- Folder **cvModel_gbt_weighted.mdl** contains all the files of the model that we chose to make churn predictions.



The web application allows to **enter the information about a user**, **classify her** and **get whether she is at risk of cancelling her subscription or not**.

Refinement

Improving class imbalance using class weight

As mentioned earlier, we have an **unbalanced dataset**: **the number of active users is more than three times the number of unsubscribed users**.
Class imbalance is a problem that often occurs in machine learning classification problems. It means that the frequency of the target class is highly imbalanced, i.e., the occurrence of one of the classes is very high compared to the other classes present. In other words, there is a bias or skewness towards the majority class present in the target variable.
Most machine learning models assume that the data is evenly distributed within the classes, so in the case of imbalanced data, **the algorithm is more biased towards predicting the majority class**, in our case 'not churned' (0), because it does not have enough data to learn the pattern in the minority class ('churned', 1).

To try to address this problem, we followed the **approach of giving different weights to both the majority class** ('not churned', 0) **and the minority class** ('churned', 1). In particular, we reduced the weight of the majority class and at the same time we increased the weight of the minority class. This difference in weights affects the classification during the training of the model.

To **compute the weights** we used the following formula and **created a new column in the train set**.

$$w_j = \frac{s}{c \times s_j}$$

Where:

$j \in [0,1]$ corresponds to label equal to 1 when the user churned and 0 otherwise

w_j is the weight for class j

c is the number of classes (2 in our case)

s_j is the total number of rows in class j

s is the total number of rows

We ran all models with and without weights on the label and compared the results as shown in the next section.

Results

Model Evaluation and Validation

Results obtained running the ML models using the mini-dataset (128MB)

- **Models without label weights**

Classifier	Accuracy - train	Accuracy - test	F1 - train	F1 - test
Logistic regression	0.76	0.81	-	-
Random Forest	0.87	0.75	0.63	0.27
Gradient-boosted tree	0.97	0.75	0.95	0.35

- **Models with cross-validation and parameter tuning**

Classifier (best parameters)	Accuracy - train	Accuracy - test	F1 - train	F1 - test
Logistic regression	0.76	0.82	-	-
Random Forest	0.89	0.81	0.71	0.33
Gradient-boosted tree	0.84	0.75	0.74	0.26

To tune parameters, we applied cross-validation. F1 scores for different parameter combinations are reported below.

Logistic regression parameter combinations and F1 score

	f1	maxIter	regParam	elasticNetParam	family
0	0.649527	10	0.1	0.8	auto
1	0.652730	10	0.3	0.8	auto
2	0.649527	30	0.1	0.8	auto
3	0.652730	30	0.3	0.8	auto
4	0.649527	50	0.1	0.8	auto
5	0.652730	50	0.3	0.8	auto

Random Forest parameter combinations and F1 score

	f1	numTrees	maxDepth	impurity
0	0.661881	10	3	entropy
1	0.674118	10	3	gini
2	0.712410	10	4	entropy
3	0.750892	10	4	gini
4	0.733926	10	5	entropy
5	0.768095	10	5	gini
6	0.665404	20	3	entropy
7	0.699887	20	3	gini
8	0.697862	20	4	entropy
9	0.720352	20	4	gini
10	0.718065	20	5	entropy
11	0.730014	20	5	gini
12	0.696962	30	3	entropy
13	0.665175	30	3	gini
14	0.723107	30	4	entropy

Gradient-boosted tree parameter combinations and F1 score

	f1	maxIter	maxDepth
0	0.761805	10	2
1	0.760065	10	3
2	0.727771	10	5
3	0.783531	20	2
4	0.760065	20	3
5	0.732105	20	5
6	0.773224	40	2
7	0.766424	40	3
8	0.737940	40	5

◆ **Models with label weights, cross-validation and parameter tuning**

Classifier (best parameters)	Accuracy - train	Accuracy - test	F1 - train	F1 - test
Logistic regression	0.68	0.52	0.52	0.28
Random Forest	0.80	0.70	0.63	0.24
Gradient-boosted tree	0.93	0.70	0.86	0.32

Results obtained running the ML models using the full available dataset (12GB)

The **raw full dataset** contains **26,259,199 events**, **18 features** and **22,278 unique users**. We ran the analysis on **AWS EMR**. See Jupiter notebook '**Sparkify_AWS-110321.ipynb**' for all the details and cluster configuration.

◆ **Models with cross-validation and parameter tuning**

Classifier (best parameters)	Accuracy - train	Accuracy - test	F1 - train	F1 - test
Logistic regression	0.78	0.77	-	-
Random Forest	0.79	0.79	0.18	0.18
Gradient-boosted tree	0.84	0.84	0.57	0.56

◆ **Models with label weights, cross-validation and parameter tuning**

Classifier (best parameters)	Accuracy - train	Accuracy - test	F1 - train	F1 - test
Logistic regression	0.78	0.77	-	-
Random Forest	0.79	0.79	0.59	0.56
Gradient-boosted tree	0.79	0.80	0.58	0.58

Justification

Comparing the performance of different classifiers we ran in both the mini and the full dataset, we observe that:

- **Logistic regression performed poorly especially in predicting true positives** (users who churned). Performance improves by adding label weights to account for the highly imbalanced dataset but the robustness of this result is not confirmed when we use the full dataset. Perhaps more tuning of the "class_weight" parameter can provide better and more stable results.
- **Random Forest also seems to be very sensitive to class imbalance**. We see a significant improvement in the F1 score on the test test when we add 'class_weight' on the model we ran on the full dataset, even if the opposite is observed using the mini-dataset.
- **Gradient-boosted tree classifier with weights is the best, especially when we look at the F1 score on the test set (58%) of the full dataset**. It seems that the model is **overall more stable** and resistant to overfitting when we use a large amount of data. We chose to use this model to make churn prediction on the web app.

Conclusion

Reflection

The goal of the project is to help Sparkify to **identify in advance the customers that are likely to abandon the service** by canceling their subscription.

The solution we propose comes from a detailed analysis that consist of the following steps:

- ☑ Data cleaning and characterization.
- ☑ Exploratory data analysis to understand the data at hand, and possibly extract useful information on the customer base in terms of demographic and interactions with the service.
- ☑ Feature engineering to identify features that are useful in predicting churn: gender, days since registration, state, average songs played per day, level of subscription, ratio thumbs up over thumbs down, number of friends, average roll adverts per day.
- ☑ Binary classification modeling with parameter tuning and cross-validation: Logistic regression, Random forest, Gradient-boosted tree.

- ☑ Model evaluation considering the highly imbalance dataset at hand to select the model that best predicts user churn.
- ☑ Web app where by entering information about the user we can immediately get an idea if the customer is likely to churn.

The most challenging part of the work on this project was the feature engineering, where it took a lot of time to select features that help predict churn, making sure they were relevant and not multicollinear. Also, running the analysis using the big dataset required some effort especially in setting the proper cluster on AWS to successfully execute all the models.

Improvement

To improve the performance of the most promising models we can:

- ☐ Spend more time engineering features by trying to think of more meaningful features (e.g., sequence of events that precede user's churn).
- ☐ Do more tuning of hyperparameters.
- ☐ Perform stacking of models.
- ☐ Find a better approach to dealing with strongly unbalanced data (oversampling, under sampling, SMOTE).

File Description

- **WebApp**
  - **templates**
    - *master.html* # main page of web app
    - *go.html* # classification result page of web app
  - **static**
    - *githublogo2.png* # github logo used in the main page
    - *churn_image.png* # downloaded from (https://www.shutterstock.com)
  - *run.py* # Flask script that runs app
  - **cvModel_gbt_weighted.mdl** # ML model chosen to make prediction
    - bestModel
    - estimator
    - evaluator
    - metadata
  - **demo**
    - *demo_webapp.gif* # animation with prediction of churn
- *Sparkify.ipynb* # Jupyter notebook that contains detailed data analysis and model building with Spark (Pyspark and SparkML libraries) run on the small subset
- *Sparkify_AWS-110321.ipynb* # Jupyter notebook that contains the analysis run using the full available dataset on AWS EMR