01_Sparkify_Data_Exploration

March 22, 2022

1 Part 1: Data Exploration

1.1 Load libraries, create Spark session and import data

In [1]:
```
# import libraries
from pyspark.sql import SparkSession
from pyspark.sql import functions as F
from pyspark.sql.window import Window
from pyspark.sql.functions import countDistinct
from pyspark.sql.types import StringType, DoubleType, IntegerType

import datetime
import pandas as pd
%matplotlib inline
import matplotlib.pyplot as plt
import seaborn as sns
import re
import time
import numpy as np


from pyspark.ml import Pipeline
from pyspark.ml.feature import VectorAssembler, StandardScaler, StringIndexer
from sklearn.model_selection import train_test_split
from pyspark.ml.classification import LogisticRegression, RandomForestClassifier, GBTCla
from pyspark.ml.evaluation import BinaryClassificationEvaluator
from pyspark.ml.evaluation import  MulticlassClassificationEvaluator
from pyspark.ml.tuning import CrossValidator, ParamGridBuilder
```

In [2]:
```
# create a Spark session
spark = SparkSession \
    .builder \
    .appName("Sparkify Project Session Data Exploration") \
    .getOrCreate()
```

In [3]:
```
path = "data/mini_sparkify_event_data.json"
data = spark.read.json(path)
```

2 1. Business Understanding

The goal of this project is to predict which users will churn based on a dataset from a (fictional) music streaming service named 'Sparkify'. This business case is quite similar to Spotify. Similar as Spotify the Sparkify streaming service can be used on two levels: the free tier or the premium tier. Sparkify profits from both levels. The free tier is financed by adverts in between the songs and in the premium tier Sparkify earns money through a monthly subscription fee. If a user decides do pay for the premium service he can use the service without advertisment. The user can decide to downgrade or upgrade from free to premium or vice versa. It is also possible for a user to cancel the service completely anytime. So what exactly is the definition of churn? Customer churn ist the phenomenon where customers of a business no longer purchase or interact with the business. In our case a churned user is a user that canceled the service completly. The goal is to identify those users and predict which users will churn based on their past user activity.

3 2. Data Understanding

```
In [4]: data.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)
```

```
In [5]: #taking a look at the first entry
        data.head()
```

```
Out[5]: Row(artist='Martha Tilston', auth='Logged In', firstName='Colin', gender='M', itemInSess
```

```
In [6]: #number of total entries in the dataset
        data.count()
```

```
Out[6]: 286500
```

3.1 Exploration by Column

In the following we are going to look at the dataset column by column to get a better understanding of what information we get from each column ### Column 'artist'

```
In [7]: # counting how many unique artists are in the dataset
        data.select('artist').dropDuplicates().count()
```

```
Out[7]: 17656
```

```
In [8]: #what are the most listened to artists in the dataset
        data.select(['artist']).groupby('artist').count().orderBy('count', ascending=False).show
```

```
+--------------------+-----+
|              artist|count|
+--------------------+-----+
|                null|58392|
|       Kings Of Leon| 1841|
|            Coldplay| 1813|
|Florence + The Ma...| 1236|
|       Dwight Yoakam| 1135|
|            BjÃČÂűrk| 1133|
|      The Black Keys| 1125|
|                Muse| 1090|
|       Justin Bieber| 1044|
|        Jack Johnson| 1007|
|              Eminem|  953|
|           Radiohead|  884|
|     Alliance Ethnik|  876|
|               Train|  854|
|        Taylor Swift|  840|
|         OneRepublic|  828|
|         The Killers|  822|
|         Linkin Park|  787|
|         Evanescence|  781|
|            Harmonia|  729|
+--------------------+-----+
only showing top 20 rows
```

3.1.1 Column 'auth'

```
In [9]: #What values do we have in the auth column
        data.select('auth').groupby('auth').count().show()
```

```
+----------+------+
|      auth| count|
+----------+------+
```

```
|Logged Out|  8249|
| Cancelled|    52|
|     Guest|    97|
| Logged In|278102|
+----------+------+
```

The values "Cancelled" and "Guest" are not completly clear. Let's have a look into the data of how entries with those values look like to find out the meaning:

```
In [10]: # look into sample rows of authentication status guest
         data.filter('auth = "Guest"').select('artist', 'auth', 'firstName', 'length', 'level',
```

```
+------+-----+---------+------+-----+--------+------+------+---------+
|artist| auth|firstName|length|level|    page|status|userId|sessionId|
+------+-----+---------+------+-----+--------+------+------+---------+
|  null|Guest|     null|  null| free|   Error|   404|      |      151|
|  null|Guest|     null|  null| free|    Home|   200|      |      151|
|  null|Guest|     null|  null| free|Register|   200|      |      151|
|  null|Guest|     null|  null| free|    Help|   200|      |      151|
|  null|Guest|     null|  null| free|    Home|   200|      |      151|
+------+-----+---------+------+-----+--------+------+------+---------+
only showing top 5 rows
```

```
In [11]: #what pages are being visited with the auth = Guest
         data.select(['auth','page']).where(data.auth =='Guest').groupby('page').count().show()
```

```
+-------------------+-----+
|               page|count|
+-------------------+-----+
|               Home|   36|
|              About|   14|
|Submit Registration|    5|
|           Register|   18|
|               Help|   23|
|              Error|    1|
+-------------------+-----+
```

```
In [12]: #what do the userIds look like for the entries with auth = Guest
         data.select(['auth','userId']).where(data.auth =='Guest').groupby('userId').count().sho
```

```
+------+-----+
|userId|count|
+------+-----+
```

```
|      |   97|
+------+-----+
```

We see that it is possible to interact with Sparkify as a guest. For all entries the userId is empty therefore it is not possible to assign the user interactions of guest entries to a userId. These rows can be dropped.

```
In [13]: # look into sample rows of authentication status Cancelled
         data.filter('auth = "Cancelled"').select('artist', 'auth', 'firstName', 'length', 'leve
```

```
+------+---------+---------+------+-----+--------------------+------+------+---------+
|artist|     auth|firstName|length|level|                page|status|userId|sessionId|
+------+---------+---------+------+-----+--------------------+------+------+---------+
|  null|Cancelled|   Adriel|  null| paid|Cancellation Conf...|   200|    18|      514|
|  null|Cancelled|    Diego|  null| paid|Cancellation Conf...|   200|    32|      540|
|  null|Cancelled|    Mason|  null| free|Cancellation Conf...|   200|   125|      174|
|  null|Cancelled|Alexander|  null| paid|Cancellation Conf...|   200|   105|      508|
|  null|Cancelled|    Kayla|  null| paid|Cancellation Conf...|   200|    17|      797|
+------+---------+---------+------+-----+--------------------+------+------+---------+
only showing top 5 rows
```

```
In [14]: #what pages are being visited with the auth = Cancelled
         data.select(['auth','page']).where(data.auth =='Cancelled').groupby('page').count().sho
```

```
+--------------------+-----+
|                page|count|
+--------------------+-----+
|Cancellation Conf...|   52|
+--------------------+-----+
```

```
In [15]: #what do the userIds look like for the entries with auth = Cancelled
         data.select(['auth','userId']).where(data.auth =='Cancelled').groupby('userId').count()
```

```
+------+-----+
|userId|count|
+------+-----+
|   125|    1|
|    51|    1|
|    54|    1|
|100014|    1|
|   101|    1|
|    29|    1|
|100021|    1|
```

```
|    87|    1|
|    73|    1|
|     3|    1|
|    28|    1|
|100022|    1|
|100025|    1|
|300007|    1|
|100006|    1|
|    18|    1|
|    70|    1|
|100005|    1|
|    17|    1|
|100007|    1|
+------+-----+
only showing top 20 rows
```

On the other hand side the rows with "Cancelled" auth seem to be valid since they all have a userId. It seems that auth = "Cancelled" appears when a user clicks on the "Cancellation Confirmation" page. These entries will probably be important for our Churn prediction.

3.1.2 Columns 'userId' and 'gender'

```
In [16]: # counting how many unique userIds are in the dataset
         data.select('userId').dropDuplicates().count()

Out[16]: 226

In [17]: #how many of our unique users (userIds) are Female, Male and undefined
         data.select(['userId','gender']).dropDuplicates(['userId']).groupby('gender').count().s

+------+-----+
|gender|count|
+------+-----+
|     F|  104|
|  null|    1|
|     M|  121|
+------+-----+

In [18]: #see what userIds have the value null in gender column
         data.where(F.col('gender').isNull()).groupby('userId').count().show()

+------+-----+
|userId|count|
+------+-----+
|      | 8346|
```

```
+------+-----+
```

There are 226 unique userIds (225 valid, because when the gender = null, then we have an empty userId) in the dataset. Of those users 121 are male and 104 are female users. The gender is null when the userId is not known - when there is a userId given then there is also a gender given.

Conclusion: There is only a slight imbalance regarding the users genders.

3.1.3 Columns 'itemInSession', 'sessionId', 'song', 'ts' and 'length'

```
In [19]: # look at some example values to understand the columns better
         data.select(['itemInSession', 'userId', 'sessionId', 'ts', 'page', 'song', 'length']) \
             .filter(data.userId == 20) \
             .show(n=8)
```

```
+-------------+------+---------+-------------+---------+--------------------+---------+
|itemInSession|userId|sessionId|           ts|     page|                song|   length|
+-------------+------+---------+-------------+---------+--------------------+---------+
|            0|    20|      216|1538529207000|     Home|                null|     null|
|            1|    20|      216|1538529265000| NextSong|Trouble Dub - FAM...|210.33751|
|            2|    20|      216|1538529266000|Thumbs Up|                null|     null|
|            3|    20|      216|1538529475000| NextSong|            Float On|209.52771|
|            4|    20|      216|1538529684000| NextSong|Resposta Ao Tempo...|283.14077|
|            5|    20|      216|1538529685000|Thumbs Up|                null|     null|
|            6|    20|      216|1538529967000| NextSong|            One Time|214.67383|
|            7|    20|      216|1538530181000| NextSong|      True To Myself|225.59302|
+-------------+------+---------+-------------+---------+--------------------+---------+
only showing top 8 rows
```

- itemInSession: Count of the interactions(=events) which happened for one user during the same session (sessionId)
- sessionId: Id of the session for a user
- song: Song that is being played, has a value other than null when we are on the page 'NextSong'
- ts: Timestamp for the event
- length: duration of time a song was played, is null for other page-events than 'NextSong'

3.1.4 Column 'level'

```
In [20]: #Which levels do we have and what are their counts in the dataset
         data.select('level').groupby('level').count().show()
```

```
+-----+------+
|level| count|
+-----+------+
| free| 58338|
```

```
| paid|228162|
+-----+------+
```

In total there are 58338 entries with free level and 228162 entries with paid level.

3.1.5 Column 'location'

```
In [21]: # count of total unique locations
         data.select('location').dropDuplicates().count()
```

```
Out[21]: 115
```

```
In [22]: # see the highest count of unique locations per userId
         data.select(['userId','location'])\
             .groupby('userId').agg(F.countDistinct('location'))\
             .orderBy('count(DISTINCT location)', ascending=False).show(10)
```

```
+------+------------------------+
|userId|count(DISTINCT location)|
+------+------------------------+
|200002|                       1|
|100010|                       1|
|   125|                       1|
|    51|                       1|
|   124|                       1|
|     7|                       1|
|    54|                       1|
|    15|                       1|
|   155|                       1|
|100014|                       1|
+------+------------------------+
only showing top 10 rows
```

In total we have 115 different unique locations in the dataset. It seems like users always access the streaming service from the same location.

3.1.6 Column 'page'

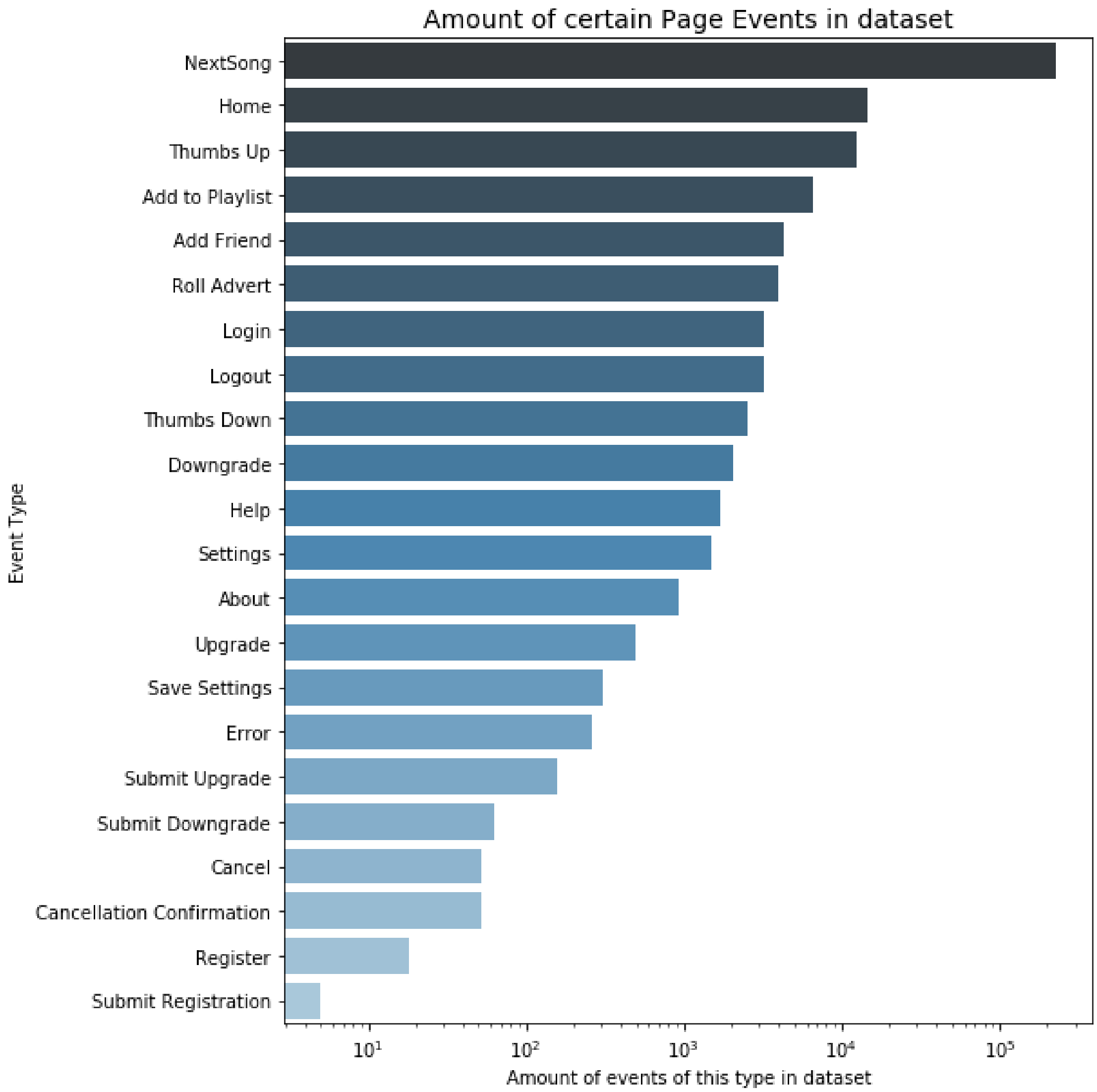
```
In [23]: page_events=data.select('page').groupby('page').count().orderBy('count', ascending = Fa
         page_events.show(22)
```

```
+--------------------+------+
|                page| count|
+--------------------+------+
|            NextSong|228108|
```

```
|                Home| 14457|
|           Thumbs Up| 12551|
|     Add to Playlist|  6526|
|          Add Friend|  4277|
|         Roll Advert|  3933|
|               Login|  3241|
|              Logout|  3226|
|         Thumbs Down|  2546|
|           Downgrade|  2055|
|                Help|  1726|
|            Settings|  1514|
|               About|   924|
|             Upgrade|   499|
|       Save Settings|   310|
|               Error|   258|
|      Submit Upgrade|   159|
|    Submit Downgrade|    63|
|              Cancel|    52|
|Cancellation Conf...|    52|
|            Register|    18|
| Submit Registration|     5|
+--------------------+------+
```

Lets look at the pages and how often they are clicked on also in graphical form:

```
In [24]: # convert pyspark df to pandas df
         page_events_pd = page_events.toPandas()

In [25]: #plot graph
         #configure the size
         plt.figure(figsize=(8,10))
         #barplot
         barplot=sns.barplot(x='count', y='page', data=page_events_pd,palette="Blues_d", log=Tru
         plt.title('Amount of certain Page Events in dataset', size=14)
         plt.xlabel('Amount of events of this type in dataset', size=10)
         plt.ylabel('Event Type', size=10)
         plt.show()

         page_events.unpersist(blocking = True)
```




```
Out[25]: DataFrame[page: string, count: bigint]
```

The most visited page is 'NextSong', which is also the main function of the streaming service. Second is the 'Home' page that the user enters when starting a streaming session. We can also see that Users visit 'Submit Upgrade' more than 'Submit Downgrade'. Also not everyone that visits the 'Downgrade' page also clicks on 'Submit Downgrade' to actually downgrade. But it seems like everyone who goes on 'Cancel' page also actually cancels (= churns) because the numbers of 'Cancel' page and 'Cancellation Confirmation' are equal.

3.1.7 Column 'registration'

```
In [26]: #how is the behaviour for registration column per userId
         data.select(['userId', 'registration']) \
             .groupby('userId') \
```

```
            .agg(F.countDistinct('registration')) \
            .orderBy('count(DISTINCT registration)', ascending=False) \
            .show(10)
```

```
+------+----------------------------+
|userId|count(DISTINCT registration)|
+------+----------------------------+
|100010|                           1|
|200002|                           1|
|   125|                           1|
|    51|                           1|
|   124|                           1|
|     7|                           1|
|    54|                           1|
|    15|                           1|
|   155|                           1|
|100014|                           1|
+------+----------------------------+
only showing top 10 rows
```

As expected a userId seems to be given when you register therefore also every userId only has one registration entry. The number in registration is the timestamp that the user registered.

3.1.8 Column 'userAgent'

```
In [27]: # show unique possible values of column 'userAgent'
         data.select('userAgent').dropDuplicates().show(5, truncate=False)
```

```
+-----------------------------------------------------------------------------------------------
|userAgent
+-----------------------------------------------------------------------------------------------
|"Mozilla/5.0 (Macintosh; Intel Mac OS X 10_8_5) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/3
|"Mozilla/5.0 (Windows NT 5.1) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/36.0.1985.143 Safar
|Mozilla/5.0 (X11; Ubuntu; Linux i686; rv:31.0) Gecko/20100101 Firefox/31.0
|"Mozilla/5.0 (Macintosh; Intel Mac OS X 10_9_4) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/3
|"Mozilla/5.0 (Macintosh; Intel Mac OS X 10_9_4) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/3
+-----------------------------------------------------------------------------------------------
only showing top 5 rows
```

The column user agent describes the software used to access the streaming service.

3.1.9 Column 'ts'

```
In [28]: #function to convert the timestamp in datetime format
         udf_convert_ts = F.udf(lambda timestamp: datetime.datetime.fromtimestamp(timestamp / 10
```

```
In [29]: #new dataframe with timestamp column in datetime form
         data_ts = data.withColumn('dt_timestamp', udf_convert_ts(data.ts))

In [30]: data_ts.head()

Out[30]: Row(artist='Martha Tilston', auth='Logged In', firstName='Colin', gender='M', itemInSes

In [31]: #Lets see from what timeframe we have data
         min_ts = data_ts.agg({'dt_timestamp' : 'min'}).collect()[0][0]
         max_ts = data_ts.agg({'dt_timestamp' : 'max'}).collect()[0][0]
         print('The dataset starts from {}'.format(min_ts))
         print('The dataset ends on {}'.format(max_ts))
         print('The dataset includes {}'.format(datetime.datetime.strptime(max_ts, "%Y-%m-%d %H:

The dataset starts from 2018-10-01 00:01:57
The dataset ends on 2018-12-03 01:11:16
The dataset includes 63 days, 1:09:19
```

The observed time frame is from 2018-10-01 to 2018-12-03, which is around 63 days.

4 3. Data Preparation and Cleaning

After getting a Data Understanding we will now prepare the dataset. The first step of the preparation is cleaning the data from invalid or missing data - for example records without userId or sessionId.

4.1 3.1 Checking for missing values

First we will check the dataset for missing values in certain columns

4.1.1 Checking column 'artist' for empty values

```
In [32]: data.where((F.col('artist').isNull()) | (data.artist=='')).show(5)

+------+---------+---------+------+-------------+--------+------+-----+--------------------+----
|artist|     auth|firstName|gender|itemInSession|lastName|length|level|            location|meth
+------+---------+---------+------+-------------+--------+------+-----+--------------------+----
|  null|Logged In|    Colin|     M|           54| Freeman|  null| paid|     Bakersfield, CA|   P
|  null|Logged In|    Micah|     M|           84|    Long|  null| free|Boston-Cambridge-...|   G
|  null|Logged In|    Micah|     M|           86|    Long|  null| free|Boston-Cambridge-...|   P
|  null|Logged In|    Alexi|     F|            4|  Warren|  null| paid|Spokane-Spokane V...|   G
|  null|Logged In|    Alexi|     F|            7|  Warren|  null| paid|Spokane-Spokane V...|   P
+------+---------+---------+------+-------------+--------+------+-----+--------------------+----
only showing top 5 rows
```

Conclusion: If there is a null or empty value in the artist column this is not invalid data. It regulary happens when the user takes other actions than listening to a song (for example Login, Logout, Home Page, etc.). Therefore these entries will stay in the dataset.

4.1.2 Checking column 'auth' for empty values

```
In [33]: data.where((F.col('auth').isNull()) | (data.auth=='')).show()
```

```
+------+----+---------+------+-------------+--------+------+-----+--------+------+----+--------
|artist|auth|firstName|gender|itemInSession|lastName|length|level|location|method|page|registrat
+------+----+---------+------+-------------+--------+------+-----+--------+------+----+--------
+------+----+---------+------+-------------+--------+------+-----+--------+------+----+--------
```

Conclusion: There are no null or empty values in the auth column

4.1.3 Checking column 'sessionId' for empty values

```
In [34]: data.where(F.col('sessionId').isNull()).show()
```

```
+------+----+---------+------+-------------+--------+------+-----+--------+------+----+--------
|artist|auth|firstName|gender|itemInSession|lastName|length|level|location|method|page|registrat
+------+----+---------+------+-------------+--------+------+-----+--------+------+----+--------
+------+----+---------+------+-------------+--------+------+-----+--------+------+----+--------
```

Conclusion: There are no null or empty values in the sessionId column.

4.1.4 Checking column 'userId' for empty values

```
In [35]: data.where((F.col('userId').isNull()) | (data.userId=='')).show()
```

```
+------+----------+---------+------+-------------+--------+------+-----+--------+------+-----+--
|artist|      auth|firstName|gender|itemInSession|lastName|length|level|location|method| page|re
+------+----------+---------+------+-------------+--------+------+-----+--------+------+-----+--
|  null|Logged Out|     null|  null|          100|    null|  null| free|    null|   GET| Home|
|  null|Logged Out|     null|  null|          101|    null|  null| free|    null|   GET| Help|
|  null|Logged Out|     null|  null|          102|    null|  null| free|    null|   GET| Home|
|  null|Logged Out|     null|  null|          103|    null|  null| free|    null|   PUT|Login|
|  null|Logged Out|     null|  null|            2|    null|  null| free|    null|   GET| Home|
|  null|Logged Out|     null|  null|            3|    null|  null| free|    null|   PUT|Login|
|  null|Logged Out|     null|  null|            0|    null|  null| free|    null|   PUT|Login|
|  null|Logged Out|     null|  null|            0|    null|  null| free|    null|   PUT|Login|
|  null|Logged Out|     null|  null|           14|    null|  null| free|    null|   GET| Home|
|  null|Logged Out|     null|  null|           15|    null|  null| free|    null|   PUT|Login|
|  null|Logged Out|     null|  null|           21|    null|  null| free|    null|   GET| Home|
|  null|Logged Out|     null|  null|           22|    null|  null| free|    null|   GET| Home|
|  null|Logged Out|     null|  null|           23|    null|  null| free|    null|   PUT|Login|
|  null|Logged Out|     null|  null|            0|    null|  null| free|    null|   GET| Home|
|  null|Logged Out|     null|  null|            1|    null|  null| free|    null|   GET|About|
|  null|Logged Out|     null|  null|            2|    null|  null| free|    null|   GET| Home|
```

```
|  null|Logged Out|     null|  null|           38|    null|  null| free|    null|   GET| Home|
|  null|Logged Out|     null|  null|           39|    null|  null| free|    null|   PUT|Login|
|  null|Logged Out|     null|  null|            0|    null|  null| free|    null|   GET| Home|
|  null|Logged Out|     null|  null|           47|    null|  null| free|    null|   GET| Home|
+------+----------+---------+------+-------------+--------+------+-----+--------+------+-----+--
only showing top 20 rows
```

There are some entrys where userId is empty. From the first look it looks like this always happens with Logged out users. Lets look at the values in auth column when userId is empty:

```
In [36]: data.where((F.col('userId').isNull()) | (data.userId=='')).groupby('auth').count().show
```

```
+----------+-----+
|      auth|count|
+----------+-----+
|Logged Out| 8249|
|     Guest|   97|
+----------+-----+
```

This shows that the userId is empty when the user is logged out or using the service as a Guest. These entrys we cannot assign to a certain userId. Lets have a look at what pages those Logged out or Guest users can visit:

```
In [37]: data.where((F.col('userId').isNull()) | (data.userId=='')).groupby('page').count().show
```

```
+-------------------+-----+
|               page|count|
+-------------------+-----+
|               Home| 4375|
|              About|  429|
|Submit Registration|    5|
|              Login| 3241|
|           Register|   18|
|               Help|  272|
|              Error|    6|
+-------------------+-----+
```

Conclusion: We cannot assign the entrys without a userId to an existing userId. Therefore we will only keep the rows that are not Null and not emptry in column userId. Additionally we also only want to keep the rows that have a valid sessionId (not null and not empty)

4.2 3.2 Cleaning dataset

We will now only keep the columns that do not have missing values in userId column

```
In [38]: data_clean = data_ts.where((F.col('userId').isNotNull()) & (data.userId!=''))
         data_ts.unpersist(blocking = True)
```

```
Out[38]: DataFrame[artist: string, auth: string, firstName: string, gender: string, itemInSessio
```

```
In [39]: print('The dataset contains {} rows before cleaning'.format(data_ts.count()))
```

```
The dataset contains 286500 rows before cleaning
```

```
In [40]: print('The dataset contains {} rows after dropping any null or empty value in userId an
```

```
The dataset contains 278154 rows after dropping any null or empty value in userId and sessionId
```

```
In [41]: data_clean.persist()
```

```
Out[41]: DataFrame[artist: string, auth: string, firstName: string, gender: string, itemInSessio
```

Lets see what authentification status the remaining entries from the clean dataset have:

```
In [42]: data_clean.groupby('auth').count().show()
```

```
+---------+------+
|     auth| count|
+---------+------+
|Cancelled|    52|
|Logged In|278102|
+---------+------+
```

```
In [43]: count_users_dropped = data_ts.count() - data_clean.count()
         count_logged_out = data_ts.select('auth').filter('auth = "Logged Out"').count()
         count_guest = data_ts.select('auth').filter('auth = "Guest"').count()
         count_rows_cleaned = data_clean.count()

         print('{} Rows without userId were dropped. From that were {} users with authentificati
         and {}"Guest". Resulting into {} remaining rows in the cleaned dataset.'.format(count_u
```

```
8346 Rows without userId were dropped. From that were 8249 users with authentification status "L
```

Conculsion: We only have entries with LoggedIn auth and Cancelled Status. Therefore entries with Guest or Logged Out have been removed. Now the question is what the Cancelled Status means. When running the cell below we can see an example from the data which shows that the Cancelled value is in auth when we have the page Cancellation Confirmation after Downgrading. The path is here: Downgrade -> Cancel -> Cancellation Confirmation (page column)

In [44]: data_clean.where(data_clean.auth == 'Cancelled').collect()[0]

Out[44]: Row(artist=None, auth='Cancelled', firstName='Adriel', gender='M', itemInSession=104, l

In [45]: data_clean.where((data_clean.userId==18)&(data_clean.sessionId==514)).show(110) #auth =

```
+--------------------+---------+---------+------+-------------+--------+---------+-----+--------
|              artist|     auth|firstName|gender|itemInSession|lastName|   length|level|
+--------------------+---------+---------+------+-------------+--------+---------+-----+--------
|         Alicia Keys|Logged In|   Adriel|     M|            0| Mendoza|268.06812| paid|Kansas C
|             Man Man|Logged In|   Adriel|     M|            1| Mendoza|171.88526| paid|Kansas C
|Two Door Cinema Club|Logged In|   Adriel|     M|            2| Mendoza|207.43791| paid|Kansas C
|                null|Logged In|   Adriel|     M|            3| Mendoza|     null| paid|Kansas C
|              Ayabie|Logged In|   Adriel|     M|            4| Mendoza|244.71465| paid|Kansas C
|                null|Logged In|   Adriel|     M|            5| Mendoza|     null| paid|Kansas C
|             Madonna|Logged In|   Adriel|     M|            6| Mendoza|260.75383| paid|Kansas C
|     matchbox twenty|Logged In|   Adriel|     M|            7| Mendoza|259.76118| paid|Kansas C
|        Mondo Marcio|Logged In|   Adriel|     M|            8| Mendoza|263.60118| paid|Kansas C
|               Musiq|Logged In|   Adriel|     M|            9| Mendoza|  294.922| paid|Kansas C
|           PANTyRAiD|Logged In|   Adriel|     M|           10| Mendoza|231.41832| paid|Kansas C
|      The Black Keys|Logged In|   Adriel|     M|           11| Mendoza|145.65832| paid|Kansas C
|              K.I.Z.|Logged In|   Adriel|     M|           12| Mendoza|219.66322| paid|Kansas C
|     Third Eye Blind|Logged In|   Adriel|     M|           13| Mendoza|277.65506| paid|Kansas C
|Marina And The Di...|Logged In|   Adriel|     M|           14| Mendoza|262.97424| paid|Kansas C
|      Cobra Starship|Logged In|   Adriel|     M|           15| Mendoza|158.04036| paid|Kansas C
|The All-American ...|Logged In|   Adriel|     M|           16| Mendoza|235.04934| paid|Kansas C
|                null|Logged In|   Adriel|     M|           17| Mendoza|     null| paid|Kansas C
|                null|Logged In|   Adriel|     M|           18| Mendoza|     null| paid|Kansas C
|                null|Logged In|   Adriel|     M|           19| Mendoza|     null| paid|Kansas C
|                null|Logged In|   Adriel|     M|           20| Mendoza|     null| paid|Kansas C
|                 Hem|Logged In|   Adriel|     M|           21| Mendoza|166.16444| paid|Kansas C
|            Deftones|Logged In|   Adriel|     M|           22| Mendoza|208.69179| paid|Kansas C
|           Sean Paul|Logged In|   Adriel|     M|           23| Mendoza|245.34159| paid|Kansas C
|           Nada Surf|Logged In|   Adriel|     M|           24| Mendoza|249.99138| paid|Kansas C
|Chingo Bling w/ F...|Logged In|   Adriel|     M|           25| Mendoza|225.64526| paid|Kansas C
|        Dar Williams|Logged In|   Adriel|     M|           26| Mendoza| 280.0322| paid|Kansas C
|                null|Logged In|   Adriel|     M|           27| Mendoza|     null| paid|Kansas C
|            The Cure|Logged In|   Adriel|     M|           28| Mendoza|345.46893| paid|Kansas C
|          The Smiths|Logged In|   Adriel|     M|           29| Mendoza|195.63057| paid|Kansas C
|            Godsmack|Logged In|   Adriel|     M|           30| Mendoza|243.85261| paid|Kansas C
|            Mastodon|Logged In|   Adriel|     M|           31| Mendoza|232.38485| paid|Kansas C
|              Xtreme|Logged In|   Adriel|     M|           32| Mendoza|213.96853| paid|Kansas C
|         Passion Pit|Logged In|   Adriel|     M|           33| Mendoza|174.75873| paid|Kansas C
|          Liane Foly|Logged In|   Adriel|     M|           34| Mendoza|300.12036| paid|Kansas C
|                  U2|Logged In|   Adriel|     M|           35| Mendoza|286.79791| paid|Kansas C
|                null|Logged In|   Adriel|     M|           36| Mendoza|     null| paid|Kansas C
|           Daft Punk|Logged In|   Adriel|     M|           37| Mendoza|239.80363| paid|Kansas C
|        Mayday Parade|Logged In|   Adriel|     M|           38| Mendoza|193.88036| paid|Kansas C
```

```
|          Boys Noize|Logged In|   Adriel|     M|           39| Mendoza|412.65587| paid|Kansas C
|  The Pussycat Dolls|Logged In|   Adriel|     M|           40| Mendoza|219.50649| paid|Kansas C
|         Third World|Logged In|   Adriel|     M|           41| Mendoza|236.40771| paid|Kansas C
|         Great White|Logged In|   Adriel|     M|           42| Mendoza|357.14567| paid|Kansas C
|Eminem / Dr. Dre ...|Logged In|   Adriel|     M|           43| Mendoza|  297.482| paid|Kansas C
|Hans Zimmer_ Jame...|Logged In|   Adriel|     M|           44| Mendoza|304.27383| paid|Kansas C
|              Hybrid|Logged In|   Adriel|     M|           45| Mendoza|577.07057| paid|Kansas C
|                null|Logged In|   Adriel|     M|           46| Mendoza|     null| paid|Kansas C
|       Elliott Smith|Logged In|   Adriel|     M|           47| Mendoza|149.31546| paid|Kansas C
|                null|Logged In|   Adriel|     M|           48| Mendoza|     null| paid|Kansas C
|     John Frusciante|Logged In|   Adriel|     M|           49| Mendoza|160.33914| paid|Kansas C
|          Basshunter|Logged In|   Adriel|     M|           50| Mendoza|332.69506| paid|Kansas C
|              Mariah|Logged In|   Adriel|     M|           51| Mendoza|432.19546| paid|Kansas C
|           The Kooks|Logged In|   Adriel|     M|           52| Mendoza|203.96363| paid|Kansas C
|             Genesis|Logged In|   Adriel|     M|           53| Mendoza|284.49914| paid|Kansas C
|       Radney Foster|Logged In|   Adriel|     M|           54| Mendoza|309.10649| paid|Kansas C
|      The Black Keys|Logged In|   Adriel|     M|           55| Mendoza|211.01669| paid|Kansas C
|               Yndio|Logged In|   Adriel|     M|           56| Mendoza|174.68036| paid|Kansas C
|          Immolation|Logged In|   Adriel|     M|           57| Mendoza|467.64363| paid|Kansas C
|                null|Logged In|   Adriel|     M|           58| Mendoza|     null| paid|Kansas C
|                null|Logged In|   Adriel|     M|           59| Mendoza|     null| paid|Kansas C
|                null|Logged In|   Adriel|     M|           63| Mendoza|     null| paid|Kansas C
|  Insane Clown Posse|Logged In|   Adriel|     M|           64| Mendoza|192.44363| paid|Kansas C
|           Sam Cooke|Logged In|   Adriel|     M|           65| Mendoza|122.04363| paid|Kansas C
|     Escape The Fate|Logged In|   Adriel|     M|           66| Mendoza|266.73587| paid|Kansas C
|           Aerosmith|Logged In|   Adriel|     M|           67| Mendoza|274.12853| paid|Kansas C
|                Live|Logged In|   Adriel|     M|           68| Mendoza|233.89995| paid|Kansas C
|Hans Zimmer_ Jame...|Logged In|   Adriel|     M|           69| Mendoza|304.27383| paid|Kansas C
|              Tarkan|Logged In|   Adriel|     M|           70| Mendoza|194.16771| paid|Kansas C
|  The Pussycat Dolls|Logged In|   Adriel|     M|           71| Mendoza|229.27628| paid|Kansas C
|                 Era|Logged In|   Adriel|     M|           72| Mendoza|200.56771| paid|Kansas C
|                null|Logged In|   Adriel|     M|           73| Mendoza|     null| paid|Kansas C
|The Brian Jonesto...|Logged In|   Adriel|     M|           74| Mendoza|194.61179| paid|Kansas C
|       R.L. Burnside|Logged In|   Adriel|     M|           75| Mendoza|205.92281| paid|Kansas C
|           Mangataot|Logged In|   Adriel|     M|           76| Mendoza|356.62322| paid|Kansas C
|     Nine Inch Nails|Logged In|   Adriel|     M|           77| Mendoza|275.43465| paid|Kansas C
|       Dwight Yoakam|Logged In|   Adriel|     M|           78| Mendoza| 239.3073| paid|Kansas C
|                null|Logged In|   Adriel|     M|           79| Mendoza|     null| paid|Kansas C
|      The Black Keys|Logged In|   Adriel|     M|           80| Mendoza|198.21669| paid|Kansas C
|            Bon Jovi|Logged In|   Adriel|     M|           81| Mendoza|222.61506| paid|Kansas C
|  Rihanna / J-Status|Logged In|   Adriel|     M|           82| Mendoza|251.16689| paid|Kansas C
|                null|Logged In|   Adriel|     M|           83| Mendoza|     null| paid|Kansas C
|Red Hot Chili Pep...|Logged In|   Adriel|     M|           84| Mendoza|219.76771| paid|Kansas C
|       The Offspring|Logged In|   Adriel|     M|           85| Mendoza|122.69669| paid|Kansas C
|      Alien Ant Farm|Logged In|   Adriel|     M|           86| Mendoza|253.04771| paid|Kansas C
|      Brisa RochÃČÂľ|Logged In|   Adriel|     M|           87| Mendoza| 48.19546| paid|Kansas C
|             OutKast|Logged In|   Adriel|     M|           88| Mendoza|239.35955| paid|Kansas C
|         Sheryl Crow|Logged In|   Adriel|     M|           89| Mendoza|298.78812| paid|Kansas C
```

```
|            Ane Brun|Logged In|   Adriel|     M|           90| Mendoza|269.58322| paid|Kansas C
|           FM Static|Logged In|   Adriel|     M|           91| Mendoza|212.61016| paid|Kansas C
|Martin O'Donnell ...|Logged In|   Adriel|     M|           92| Mendoza|185.80853| paid|Kansas C
|                 Air|Logged In|   Adriel|     M|           93| Mendoza|212.21832| paid|Kansas C
|         Miles Davis|Logged In|   Adriel|     M|           94| Mendoza|336.06485| paid|Kansas C
|Usher featuring w...|Logged In|   Adriel|     M|           95| Mendoza|395.72853| paid|Kansas C
|          Nickelback|Logged In|   Adriel|     M|           96| Mendoza|238.18404| paid|Kansas C
|               Tonic|Logged In|   Adriel|     M|           97| Mendoza|174.94159| paid|Kansas C
|          Arch Enemy|Logged In|   Adriel|     M|           98| Mendoza|280.86812| paid|Kansas C
|Les Ogres De Barback|Logged In|   Adriel|     M|           99| Mendoza|323.21261| paid|Kansas C
|The Notorious B.I.G.|Logged In|   Adriel|     M|          100| Mendoza| 286.1971| paid|Kansas C
|          Nickelback|Logged In|   Adriel|     M|          101| Mendoza|207.46404| paid|Kansas C
|                null|Logged In|   Adriel|     M|          102| Mendoza|     null| paid|Kansas C
|                null|Logged In|   Adriel|     M|          103| Mendoza|     null| paid|Kansas C
|                null|Cancelled|   Adriel|     M|          104| Mendoza|     null| paid|Kansas C
+--------------------+---------+---------+------+-------------+--------+---------+-----+--------
```

```
In [46]: data_clean.where(data_clean.auth == 'Cancelled').groupby('page').count().show()
```

```
+--------------------+-----+
|                page|count|
+--------------------+-----+
|Cancellation Conf...|   52|
+--------------------+-----+
```

4.3 3.3 Exploratory Analysis

The goal of the following parts is to find differences in the behaviour of customers who stayed and who churned. Therefore we will first define what churn means in our dataset. After that there will be some exploratory analysis to observe the behavior for users who stayed vs users who churned. For example by exploring aggregates on these two groups of users, observing how much if a specific action they experienced per a certain time unit or number of songs played.

4.3.1 Exploratory Data Analysis to define churn

In this section we will explore the data to get a better understanding. The following is done below: - First we will have a look again at what pages a user can visit - Then wil will find out which pages can be used to define when a user is churned - After that we will create two new Columns marking the Churning Event and another one marking the Churned User

```
In [47]: data_clean.groupby('page').count().sort('count').show()
```

```
+--------------------+------+
|                page| count|
+--------------------+------+
```

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

Just from the Webpage names the pages "Cancel", "Cancellation Confirmation", "Submit Downgrade", "Submit Upgrade" and "Downgrade" could be interesting for finding the events where users churn. Therefore we will have a closer look for some examples below:

```
In [48]: #example for user who was in free level and pressed then "Cancel" and then "Cancellatio
         data_clean.where(data_clean.userId == 125).show() #session item = 9
```

```
+--------------------+---------+---------+------+-------------+--------+---------+-----+--------
|              artist|     auth|firstName|gender|itemInSession|lastName|   length|level|
+--------------------+---------+---------+------+-------------+--------+---------+-----+--------
| Christopher O'Riley|Logged In|    Mason|     M|            0|    Hart|337.91955| free|Corpus C
|The Notorious B.I.G.|Logged In|    Mason|     M|            1|    Hart|230.03383| free|Corpus C
|           Betty Boo|Logged In|    Mason|     M|            2|    Hart| 203.2322| free|Corpus C
|          Nickelback|Logged In|    Mason|     M|            3|    Hart|210.83383| free|Corpus C
| Ready For The World|Logged In|    Mason|     M|            4|    Hart|391.26159| free|Corpus C
|   We Are The Fallen|Logged In|    Mason|     M|            5|    Hart|213.60281| free|Corpus C
|      Robert Johnson|Logged In|    Mason|     M|            6|    Hart|178.41587| free|Corpus C
|              Bonobo|Logged In|    Mason|     M|            7|    Hart|323.81342| free|Corpus C
|                null|Logged In|    Mason|     M|            8|    Hart|     null| free|Corpus C
|                null|Logged In|    Mason|     M|            9|    Hart|     null| free|Corpus C
|                null|Cancelled|    Mason|     M|           10|    Hart|     null| free|Corpus C
+--------------------+---------+---------+------+-------------+--------+---------+-----+--------
```

```
In [49]: #example for user who was in paid level and pressed then "Cancel" and then "Cancellatio
         data_clean.where((data_clean.userId == 122)&(data_clean.sessionId == 1029)).show(45) #s
```

```
+--------------------+---------+---------+------+-------------+---------+---------+-----+-------
|              artist|     auth|firstName|gender|itemInSession| lastName|   length|level|
+--------------------+---------+---------+------+-------------+---------+---------+-----+-------
|    Cage The Elephant|Logged In|    Molly|     F|            0|Patterson| 228.0224| free|Memphis
|                 null|Logged In|    Molly|     F|            1|Patterson|     null| free|Memphis
|            De-Phazz|Logged In|    Molly|     F|            2|Patterson|220.99546| free|Memphis
|    King Biscuit Time|Logged In|    Molly|     F|            3|Patterson|227.52608| free|Memphis
|                null|Logged In|    Molly|     F|            4|Patterson|     null| free|Memphis
|                null|Logged In|    Molly|     F|            5|Patterson|     null| free|Memphis
|                null|Logged In|    Molly|     F|            6|Patterson|     null| free|Memphis
|                null|Logged In|    Molly|     F|            7|Patterson|     null| paid|Memphis
|                null|Logged In|    Molly|     F|            8|Patterson|     null| paid|Memphis
|Afro-Cuban All Stars|Logged In|    Molly|     F|            9|Patterson| 288.1824| paid|Memphis
|      Spiritualized|Logged In|    Molly|     F|           10|Patterson|344.99873| paid|Memphis
|            Cyberfit|Logged In|    Molly|     F|           11|Patterson|303.15057| paid|Memphis
|Alison Krauss / U...|Logged In|    Molly|     F|           12|Patterson|171.04934| paid|Memphis
|             Cartola|Logged In|    Molly|     F|           13|Patterson|  127.242| paid|Memphis
|                null|Logged In|    Molly|     F|           14|Patterson|     null| paid|Memphis
|Thao with The Get...|Logged In|    Molly|     F|           15|Patterson|193.74975| paid|Memphis
|          Kanye West|Logged In|    Molly|     F|           16|Patterson|278.07302| paid|Memphis
|The Lonely Island...|Logged In|    Molly|     F|           17|Patterson| 192.9922| paid|Memphis
|Soulja Boy Tell'e...|Logged In|    Molly|     F|           18|Patterson| 194.2722| paid|Memphis
|      More Fire Crew|Logged In|    Molly|     F|           19|Patterson| 353.4624| paid|Memphis
|         Donna Lewis|Logged In|    Molly|     F|           20|Patterson|240.95302| paid|Memphis
|   Renegade Soundwave|Logged In|    Molly|     F|           21|Patterson|224.67873| paid|Memphis
|         Cat Stevens|Logged In|    Molly|     F|           22|Patterson|225.17506| paid|Memphis
|            La Renga|Logged In|    Molly|     F|           23|Patterson|307.35628| paid|Memphis
|              Eagles|Logged In|    Molly|     F|           24|Patterson|241.78893| paid|Memphis
|                null|Logged In|    Molly|     F|           25|Patterson|     null| paid|Memphis
|        Modest Mouse|Logged In|    Molly|     F|           26|Patterson|246.17751| paid|Memphis
|    59 Times the Pain|Logged In|    Molly|     F|           27|Patterson|144.95302| paid|Memphis
|         Suzi Quatro|Logged In|    Molly|     F|           28|Patterson|239.72526| paid|Memphis
|                null|Logged In|    Molly|     F|           29|Patterson|     null| paid|Memphis
|            No Doubt|Logged In|    Molly|     F|           30|Patterson|241.81506| paid|Memphis
|        Lisac Josipa|Logged In|    Molly|     F|           31|Patterson|  271.882| paid|Memphis
|         Kim Burrell|Logged In|    Molly|     F|           32|Patterson|249.67791| paid|Memphis
|              Xzibit|Logged In|    Molly|     F|           33|Patterson|262.29506| paid|Memphis
|        Brad Paisley|Logged In|    Molly|     F|           34|Patterson|266.91873| paid|Memphis
|        New Radicals|Logged In|    Molly|     F|           35|Patterson|219.19302| paid|Memphis
|               3OH!3|Logged In|    Molly|     F|           36|Patterson|  192.522| paid|Memphis
|      Robert Johnson|Logged In|    Molly|     F|           37|Patterson|154.09587| paid|Memphis
|                null|Logged In|    Molly|     F|           38|Patterson|     null| paid|Memphis
|                null|Logged In|    Molly|     F|           39|Patterson|     null| paid|Memphis
|         Bob Newhart|Logged In|    Molly|     F|           40|Patterson|367.98649| paid|Memphis
|                null|Logged In|    Molly|     F|           41|Patterson|     null| paid|Memphis
|                null|Logged In|    Molly|     F|           42|Patterson|     null| paid|Memphis
|                null|Cancelled|    Molly|     F|           43|Patterson|     null| paid|Memphis
+--------------------+---------+---------+------+-------------+---------+---------+-----+-------
```

It seems like users first visit 'Cancel' page and then 'Cancellation Confirmed' to churn. This is the same for free and paid level. Lets have a look at downgrade and submit downgrade below:

```
In [50]: data_clean.where(data_clean.page == "Downgrade").groupby("level").count().show() #Downg
         #therefore we only have Downgrading events for paid level users
```

```
+-----+-----+
|level|count|
+-----+-----+
| paid| 2055|
+-----+-----+
```

It seems that we only have Downgrading events for paid level users. Therefore the page Downgrade is for paid users who want to go from paid to free level. Below is an example for a user who downgrades to free level and goes from Downgrade to Submit Downgrade page before being in free level:

```
In [51]: data_clean.where((data_clean.userId == 54)&(data_clean.sessionId == 859)).show(50) #ses
```

```
+--------------------+---------+---------+------+-------------+--------+---------+-----+--------
|              artist|     auth|firstName|gender|itemInSession|lastName|   length|level|
+--------------------+---------+---------+------+-------------+--------+---------+-----+--------
|Israel & New Bree...|Logged In|    Alexi|     F|            0|  Warren|229.35465| paid|Spokane-
|             Peaches|Logged In|    Alexi|     F|            1|  Warren|268.90404| paid|Spokane-
|           The Verve|Logged In|    Alexi|     F|            2|  Warren|291.94404| paid|Spokane-
|                null|Logged In|    Alexi|     F|            3|  Warren|     null| paid|Spokane-
|                null|Logged In|    Alexi|     F|            4|  Warren|     null| paid|Spokane-
|              16Volt|Logged In|    Alexi|     F|            5|  Warren|187.19302| paid|Spokane-
|George Clinton an...|Logged In|    Alexi|     F|            6|  Warren|222.04036| paid|Spokane-
|             Bow Wow|Logged In|    Alexi|     F|            7|  Warren|204.79955| paid|Spokane-
|                El-P|Logged In|    Alexi|     F|            8|  Warren|266.70975| paid|Spokane-
|          All Saints|Logged In|    Alexi|     F|            9|  Warren|302.96771| paid|Spokane-
|    Enrique Iglesias|Logged In|    Alexi|     F|           10|  Warren|217.99138| paid|Spokane-
|      The Undertones|Logged In|    Alexi|     F|           11|  Warren|156.36853| paid|Spokane-
|             Lunapop|Logged In|    Alexi|     F|           12|  Warren|264.93342| paid|Spokane-
|                null|Logged In|    Alexi|     F|           13|  Warren|     null| paid|Spokane-
|        Eric Clapton|Logged In|    Alexi|     F|           14|  Warren|273.78893| paid|Spokane-
|                Ratt|Logged In|    Alexi|     F|           15|  Warren|205.97506| paid|Spokane-
|             Bauhaus|Logged In|    Alexi|     F|           16|  Warren|190.11873| paid|Spokane-
|                null|Logged In|    Alexi|     F|           17|  Warren|     null| paid|Spokane-
|    Carrie Underwood|Logged In|    Alexi|     F|           18|  Warren|257.88036| paid|Spokane-
|                null|Logged In|    Alexi|     F|           19|  Warren|     null| paid|Spokane-
|          Todd Barry|Logged In|    Alexi|     F|           20|  Warren|126.82404| paid|Spokane-
|        Joyce Cooling|Logged In|    Alexi|     F|           21|  Warren|248.11057| paid|Spokane-
```

```
|             Atreyu|Logged In|    Alexi|     F|            22|  Warren|279.35302| paid|Spokane-
|   Beyonce & Shakira|Logged In|    Alexi|     F|            23|  Warren|201.50812| paid|Spokane-
|     Franz Ferdinand|Logged In|    Alexi|     F|            24|  Warren|141.87057| paid|Spokane-
|     Vampire Weekend|Logged In|    Alexi|     F|            25|  Warren|138.29179| paid|Spokane-
|          Megh Stock|Logged In|    Alexi|     F|            26|  Warren|  202.762| paid|Spokane-
|        Kings Of Leon|Logged In|    Alexi|     F|            27|  Warren|201.79546| paid|Spokane-
|        Annie Lennox|Logged In|    Alexi|     F|            28|  Warren|291.34322| paid|Spokane-
|             Flyleaf|Logged In|    Alexi|     F|            29|  Warren|154.20036| paid|Spokane-
|             Cartola|Logged In|    Alexi|     F|            30|  Warren|  127.242| paid|Spokane-
|        Lole y Manuel|Logged In|    Alexi|     F|            31|  Warren|239.46404| paid|Spokane-
|        Dwight Yoakam|Logged In|    Alexi|     F|            32|  Warren| 239.3073| paid|Spokane-
|     Michael BublÃČÂľ|Logged In|    Alexi|     F|            33|  Warren|225.90649| paid|Spokane-
|Liquid Tension Ex...|Logged In|    Alexi|     F|            34|  Warren|535.35302| paid|Spokane-
|Bohren & Der Club...|Logged In|    Alexi|     F|            35|  Warren|491.38893| paid|Spokane-
|                Muse|Logged In|    Alexi|     F|            36|  Warren|258.06322| paid|Spokane-
|Los Fabulosos Cad...|Logged In|    Alexi|     F|            37|  Warren|185.80853| paid|Spokane-
|    Vivian Stanshall|Logged In|    Alexi|     F|            38|  Warren|181.21098| paid|Spokane-
|The Moon and the ...|Logged In|    Alexi|     F|            39|  Warren|  302.602| paid|Spokane-
|                null|Logged In|    Alexi|     F|            40|  Warren|     null| paid|Spokane-
|                null|Logged In|    Alexi|     F|            41|  Warren|     null| paid|Spokane-
|                null|Logged In|    Alexi|     F|            42|  Warren|     null| free|Spokane-
|   Bernadette Peters|Logged In|    Alexi|     F|            43|  Warren|197.79873| free|Spokane-
|        La Casa Azul|Logged In|    Alexi|     F|            44|  Warren| 63.32036| free|Spokane-
|Ryan Leslie / Cas...|Logged In|    Alexi|     F|            45|  Warren|240.01261| free|Spokane-
|                null|Logged In|    Alexi|     F|            46|  Warren|     null| free|Spokane-
|         Chris LeDoux|Logged In|    Alexi|     F|            47|  Warren|173.37424| free|Spokane-
| A Flock Of Seagulls|Logged In|    Alexi|     F|            48|  Warren|309.83791| free|Spokane-
|          Kanye West|Logged In|    Alexi|     F|            49|  Warren|311.84934| free|Spokane-
+--------------------+---------+---------+------+--------------+--------+---------+-----+--------
only showing top 50 rows
```

4.3.2 Definition of Churn:

A user is churning when he visits the pages "Cancel" and then "Cancellation Confirmation" -> Churned

Additionally we will want to mark it when a User Downgraded from paid to premium as it might be interesting to see if he will churn. This will happen in the Feature-Engineering section a bit later.

Now we will create a column that flags when a Cancellcation Event happened 'Churning_Event'. It should be 1 when the event includes page = 'Cancellation Confirmation' and 0 otheriwse:

```
In [52]: data_churn = data_clean.withColumn('Churning_Event', (F.when(F.col("page")=='Cancellati
         data_clean.unpersist(blocking = True)
```

```
Out[52]: DataFrame[artist: string, auth: string, firstName: string, gender: string, itemInSessio
```

Check the example below to see if it worked and the column was created correctly:

```
In [53]: data_churn.where((data_churn.page=='Cancellation Confirmation')|(data_churn.page=='Canc

+------+---------+---------+------+-------------+--------+------+-----+--------------------+----
|artist|     auth|firstName|gender|itemInSession|lastName|length|level|            location|meth
+------+---------+---------+------+-------------+--------+------+-----+--------------------+----
|  null|Logged In|   Adriel|     M|          103| Mendoza|  null| paid|  Kansas City, MO-KS|   P
|  null|Cancelled|   Adriel|     M|          104| Mendoza|  null| paid|  Kansas City, MO-KS|   G
|  null|Logged In|    Diego|     M|           55|   Mckee|  null| paid|Phoenix-Mesa-Scot...|   P
|  null|Cancelled|    Diego|     M|           56|   Mckee|  null| paid|Phoenix-Mesa-Scot...|   G
|  null|Logged In|    Mason|     M|            9|    Hart|  null| free|  Corpus Christi, TX|   P
|  null|Cancelled|    Mason|     M|           10|    Hart|  null| free|  Corpus Christi, TX|   G
+------+---------+---------+------+-------------+--------+------+-----+--------------------+----
only showing top 6 rows
```

```
In [54]: data_churn.where(data_churn.page=='Cancellation Confirmation').groupby('Churning_Event'

+--------------+-----+
|Churning_Event|count|
+--------------+-----+
|             1|   52|
+--------------+-----+
```

```
In [55]: data_churn.where(data_churn.page!='Cancellation Confirmation').groupby('Churning_Event'

+--------------+------+
|Churning_Event| count|
+--------------+------+
|             0|278102|
+--------------+------+
```

We can see that all entries with page 'Cancellation Confirmation' was correctly flagged with 1 and all other pages correctly with 0.

The new column 'Churning_Event' was created to mark the exact event of cancellation confirmation. But the goal of this project is to predict users who eventually churn. Therefore we need to find the features which describe differences in users who churned and who did not churn. Therefore we need to mark which users churned and which did not.

We will create a new column 'Churned_User' that is true if the user is churning in our dataset and false if not:

```
In [56]: #create list of churned users
         churned_users = data_churn.select('userId') \
```

```
            .filter(data_churn.Churning_Event == 1) \
            .dropDuplicates().collect()

        churned_userId = []
        for u in churned_users:
            churned_userId.append(u[0])
```

```
In [57]: data_churn = data_churn.withColumn('Churned_User', data_churn.userId.isin(churned_userI
```

```
In [58]: #check if churned users are actually also marked with true
         data_churn.where((data_churn.page=='Cancellation Confirmation')|(data_churn.page=='Canc
```

```
+------+---------+---------+------+-------------+--------+------+-----+--------------------+----
|artist|     auth|firstName|gender|itemInSession|lastName|length|level|            location|meth
+------+---------+---------+------+-------------+--------+------+-----+--------------------+----
|  null|Logged In|   Adriel|     M|          103| Mendoza|  null| paid|  Kansas City, MO-KS|   P
|  null|Cancelled|   Adriel|     M|          104| Mendoza|  null| paid|  Kansas City, MO-KS|   G
|  null|Logged In|    Diego|     M|           55|   Mckee|  null| paid|Phoenix-Mesa-Scot...|   P
|  null|Cancelled|    Diego|     M|           56|   Mckee|  null| paid|Phoenix-Mesa-Scot...|   G
|  null|Logged In|    Mason|     M|            9|    Hart|  null| free|  Corpus Christi, TX|   P
|  null|Cancelled|    Mason|     M|           10|    Hart|  null| free|  Corpus Christi, TX|   G
+------+---------+---------+------+-------------+--------+------+-----+--------------------+----
only showing top 6 rows
```

```
In [59]: data_churn.where((data_churn.page=='Cancellation Confirmation')).groupby('Churned_User'
```

```
+------------+-----+
|Churned_User|count|
+------------+-----+
|        true|   52|
+------------+-----+
```

With this new column we can now also check wether the dataset is imbalanced regarding how many users churn and how many do not churn. Therefore the churn rate is being calculated below:

```
In [60]: total_users = data_churn.select('userId').dropDuplicates().count()
         churned_users = data_churn.filter('Churned_User = true').select('userId').dropDuplicate
         stayed_users = data_churn.filter('Churned_User = false').select('userId').dropDuplicate

         print('Of total {} users, {} users stayed with the streaming service during the observe
         {} users eventually churned (churn rate: {:2.2f}% ).'.format(total_users, stayed_users,
```

```
Of total 225 users, 173 users stayed with the streaming service during the observed time and 52
```

We can see that there is an imbalance in users who churned and users who stayed in the dataset since the rate is 23.11%. This can influence the model as it won't have as much data to train on for churned users as on not-churned users. This also means that a model could score an accuracy of 76.89% if it just predicted "not-churned" on every user. We will work with this imbalanced data and concentrate on creating additional features for the ML models to predict on. Furthermore we will explicitly use evaluation metrics (F1 and AUC-ROC) that work best for imbalanced data. We will for example therefore not use the accuracy as a metric.

4.3.3 Exploratory Data Analysis to find out which features we want to create and use for our prediction

In this section we will explore the data to get a better understanding and find out what features we can use to find differences in users who churned and users who stayed for our prediction model.

We will have a look at the following: - Total page interactions/events per user - Days since registration - Per User (Number of Thumbs Up / Number of listened to Songs) - Per User (Number of Thumbs Down / Number of listened to Songs) - Per User (Number of Downgrades / Number total events) - Per User (Number of Uprades / Number total events) - Per User (Number of Add to Playlist / Number of listened to Songs) - Per User (Number of Add Friends / Number of listened to Songs) - Streaming time per active day - Gender - Average Amount of Songs played per Session - Exploration: Is churning more likely to happen for users on paid or on free level? - Per User (Number of Adverts/ Number of listened to Songs) - Percentage of days since registration where user was active

Total amount of interactions/events per user How many page events happened per userId in the observed time period? We will also calculate if there is any difference here between churned and non-churned users.

```
In [61]: #Count page visits per user
         data_churn.select(['page','userId']) \
             .groupby('userId').count() \
             .orderBy('count', ascending=False).show()
```

```
+------+-----+
|userId|count|
+------+-----+
|    39| 9632|
|    92| 7230|
|   140| 6880|
|300011| 5732|
|   124| 4825|
|300021| 4659|
|300017| 4428|
|    85| 4370|
|    42| 4257|
|200023| 3769|
|     6| 3761|
|    29| 3603|
|    54| 3437|
```

```
|   100| 3214|
|     9| 3191|
|   126| 3102|
|300015| 3051|
|    91| 3014|
|    98| 2891|
|    74| 2887|
+------+-----+
only showing top 20 rows
```

```
In [62]: #Counts page visits per userId with status of Churn or not Churn
         data_churn.select(['page','userId', 'Churned_User']) \
             .groupby('userId', 'Churned_User').count() \
             .orderBy('count', ascending=False).show()
```

```
+------+------------+-----+
|userId|Churned_User|count|
+------+------------+-----+
|    39|       false| 9632|
|    92|       false| 7230|
|   140|       false| 6880|
|300011|       false| 5732|
|   124|       false| 4825|
|300021|       false| 4659|
|300017|       false| 4428|
|    85|       false| 4370|
|    42|       false| 4257|
|200023|       false| 3769|
|     6|       false| 3761|
|    29|        true| 3603|
|    54|        true| 3437|
|   100|       false| 3214|
|     9|       false| 3191|
|   126|       false| 3102|
|300015|       false| 3051|
|    91|       false| 3014|
|    98|       false| 2891|
|    74|       false| 2887|
+------+------------+-----+
only showing top 20 rows
```

```
In [63]: #calculates the average of page interactions for churned and non churned users
         data_churn.select(['page','userId', 'Churned_User']) \
             .groupby('userId', 'Churned_User').count() \
             .groupby('Churned_User').mean().show()
```

```
+------------+------------------+
|Churned_User|        avg(count)|
+------------+------------------+
|        true| 862.7692307692307|
|       false|1348.4971098265896|
+------------+------------------+
```

On average it looks like users who stayed have more interactions as users who have churned. Therefore the number of interactions could be a good choice for a feature to make a better prediction.

Lets have a look at this graphically:

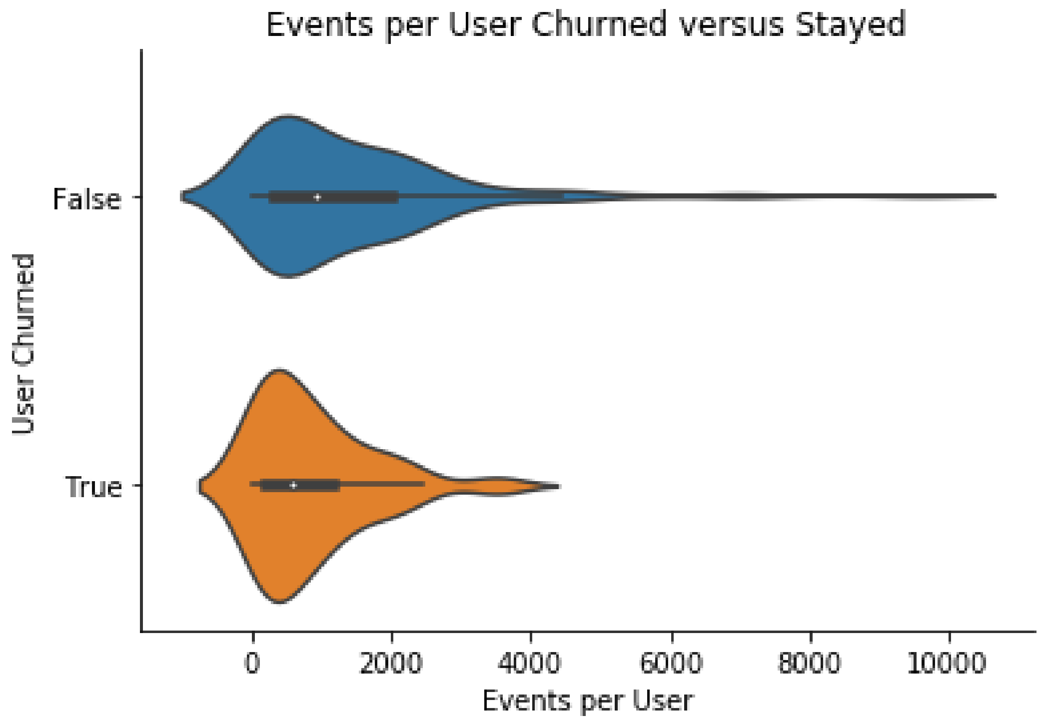
```
In [64]: page_event_per_user = data_churn.select(['page','userId', 'Churned_User']) \
             .groupby('userId', 'Churned_User').count()
```

```
In [65]: # convert spark object to pandas dataframe
         page_event_per_user_pd = page_event_per_user.select(['count', 'Churned_User']).toPandas
```

```
In [66]: page_event_per_user_pd.head()
         page_event_per_user.unpersist(blocking = True)
```

```
Out[66]: DataFrame[userId: string, Churned_User: boolean, count: bigint]
```

```
In [67]: # Plot
         ax = sns.violinplot(data=page_event_per_user_pd, y='Churned_User', x='count', orient='h
         plt.xlabel('Events per User')
         plt.ylabel('User Churned')
         plt.title('Events per User Churned versus Stayed')
         sns.despine(ax=ax);
```



Also graphically we can see that the distributions of counts per user are different for churned and non churned users. We can see that no churned user in the dataset has had more than 5000 events. It seems plausible in average since users who do not churn might stay longer with the service and therefore have more page events as well.

Therefore the metric of total events per userId will probably not be so helpful for our prediction model since also new users usually have less interactions even though they might not churn.

Therefore it would be more interesting to see if these user groups have more or less events per time period (for example a day or a session).

```
In [68]: data_churn.toPandas().head()
```

```
Out[68]:                artist        auth firstName gender  itemInSession lastName  \
         0     Martha Tilston  Logged In     Colin      M             50  Freeman
         1   Five Iron Frenzy  Logged In     Micah      M             79     Long
         2       Adam Lambert  Logged In     Colin      M             51  Freeman
         3             Enigma  Logged In     Micah      M             80     Long
         4          Daft Punk  Logged In     Colin      M             52  Freeman

               length level                        location method      ...         \
         0  277.89016  paid                 Bakersfield, CA    PUT      ...
         1  236.09424  free  Boston-Cambridge-Newton, MA-NH    PUT      ...
         2  282.82730  paid                 Bakersfield, CA    PUT      ...
         3  262.71302  free  Boston-Cambridge-Newton, MA-NH    PUT      ...
         4  223.60771  paid                 Bakersfield, CA    PUT      ...
```

```
    registration  sessionId                           song status  \
0  1538173362000         29                      Rockpools    200
1  1538331630000          8                         Canada    200
2  1538173362000         29              Time For Miracles    200
3  1538331630000          8   Knocking On Forbidden Doors    200
4  1538173362000         29  Harder Better Faster Stronger    200

               ts                                          userAgent userId  \
0  1538352117000  Mozilla/5.0 (Windows NT 6.1; WOW64; rv:31.0) G...     30
1  1538352180000  "Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebK...      9
2  1538352394000  Mozilla/5.0 (Windows NT 6.1; WOW64; rv:31.0) G...     30
3  1538352416000  "Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebK...      9
4  1538352676000  Mozilla/5.0 (Windows NT 6.1; WOW64; rv:31.0) G...     30

         dt_timestamp Churning_Event  Churned_User
0  2018-10-01 00:01:57              0         False
1  2018-10-01 00:03:00              0         False
2  2018-10-01 00:06:34              0         False
3  2018-10-01 00:06:56              0         False
4  2018-10-01 00:11:16              0         False

[5 rows x 21 columns]
```

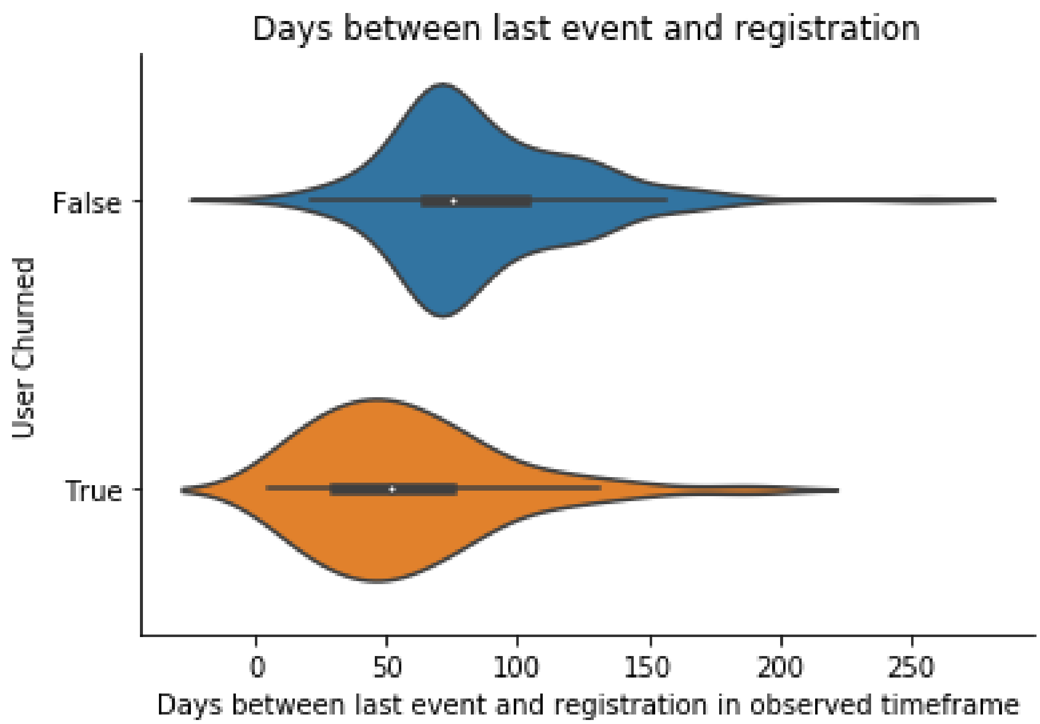
Metric: Days since registration This we will calculate as the timestamp from the last event of a user - the first event of a user

```
In [69]: #Create a str datetime column for registration timestamp
         data_churn = data_churn.withColumn('dt_registration', udf_convert_ts(data_churn.registr

In [70]: #Create an int column for timedifference from the event to the registration
         data_churn = data_churn.withColumn('milliseconds_since_registration', (data_churn.ts -
         data_churn = data_churn.withColumn('days_since_registration', (data_churn.ts - data_chu

In [71]: data_churn.head()

Out[71]: Row(artist='Martha Tilston', auth='Logged In', firstName='Colin', gender='M', itemInSes

In [72]: #Create Column that shows for each user the difference of days for the last event of th
         #for the observed timeframe
         window = Window.partitionBy('userId')
         data_churn = data_churn.withColumn('max_days_since_registration', F.max(data_churn.days

In [73]: data_churn.select(['userId', 'Churned_User', 'max_days_since_registration']).groupby('C

+------------+--------------------------------+
|Churned_User|avg(max_days_since_registration)|
+------------+--------------------------------+
|        true|               67.67311273092373|
|       false|               93.11115645812255|
```

```
+------------+--------------------------------+
```

```
In [74]: days_since_registration_pd = data_churn.select(['userId', 'Churned_User', 'max_days_sin
             .groupby('userId', 'Churned_User', 'max_days_since_registration').count() \
             .select(['userId', 'Churned_User', 'max_days_since_registration']).toPandas()
```

```
In [75]: #Plot
         ax = sns.violinplot(data=days_since_registration_pd, y='Churned_User', x='max_days_sinc
         plt.xlabel('Days between last event and registration in observed timeframe')
         plt.ylabel('User Churned')
         plt.title('Days between last event and registration')
         sns.despine(ax=ax);
```



We can a significant difference in the days between the last recorded event in the dataframe and the registration of the user for churned and non-churned users. The mean duration from registration to last interaction with the streaming service is 67.8 days for users who eventually churn and 93.1 days for users who stay with the service. But here we also have to keep in mind that this is kind of the behaviour we were expecting since a user who churns also has less events and therefore uses the service for a shorter time than a user who stays and whose records in the dataset go on for a longer time. Therefore this might also not be an ideal metrics for predicting churn for a new user.

Metric: per User (Number of Thumbs Up / Number of listened to Songs)

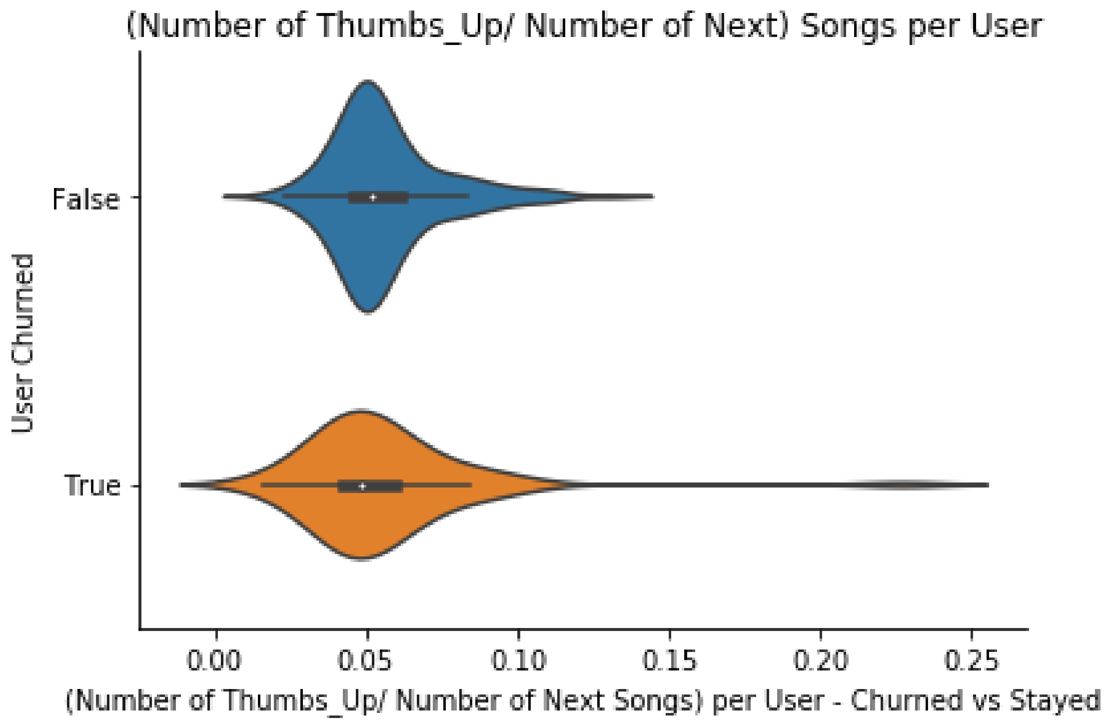
```
In [76]: def create_feature_column_page_event_vs_Songs_listened_per_userid(pyspark_df, page_even

             '''
             Function to add a column to a dataframe that shows the value of the following calcu
             The number of a certain page event vs. total Songs listetend to per userId.

             args:
                 pyspark_df - (pyspark dataframe) data_churn of Sparkify
                 page_event - (string) of page event (for example 'Thumbs Up')
             return:
                 feature_df - (pyspark dataframe) dataframe with feature

             '''

             page_name = page_event.replace(" ", "_")

             help_df = pyspark_df.select(['userId', 'page', 'Churned_User']) \
                 .where((pyspark_df.page == page_event) | (pyspark_df.page == 'NextSong')) \
                 .groupby('userId', 'page', 'Churned_User').count()

             join_df_NextSong = help_df.where(help_df.page == 'NextSong') \
                 .select(['userId', 'count']) \
                 .withColumnRenamed('count', 'Total_NextSong_perUser')

             renamed_Column = 'Total_' + page_name + '_perUser'

             join_df_event = help_df.where(help_df.page == page_event) \
                 .select(['userId', 'count']) \
                 .withColumnRenamed('count', renamed_Column)

             pyspark_df = pyspark_df.join(join_df_NextSong, on=['userId'], how='inner')
             pyspark_df = pyspark_df.join(join_df_event, on=['userId'], how='inner')

             feature_df = pyspark_df.withColumn(page_name + '_vs_NextSong', (F.col(renamed_Colum

             return feature_df


         def plot_feature_column_page_event_vs_Songs_listened_per_userid(feature_df, page_event)

             '''
             Function to create violinplot of pyspark dataframe created with
             'create_feature_column_page_event_vs_Songs_listened_per_userid' function.
             It also gives out the Mean value for Churned vs. Non-Churned users and
             the Mean value of the total page events per user in the groups churned vs. Non-chur
```

```
            args:
                pyspark_df - (pyspark dataframe) created with 'create_feature_column_page_event
                page_event - (string) of page event (for example 'Thumbs Up')

            return:
            '''

            page_name = page_event.replace(" ", "_")

            print("Mean value for Churned vs Non-Churned Users: ")
            feature_df.groupby('Churned_User').mean(page_name + '_vs_NextSong').show()

            help_pd = feature_df.select(['userId', 'Churned_User', page_name + '_vs_NextSong'])
                .groupby('userId', 'Churned_User', page_name + '_vs_NextSong').count() \
                .select(['userId', 'Churned_User', page_name + '_vs_NextSong']).toPandas()

            print("Mean value of Total " + page_name + " per User in groups churned vs. non-chu
            feature_df.groupby('Churned_User').mean('Total_' + page_name + '_perUser').show()

            print("Violinplot: ")
            ax = sns.violinplot(data=help_pd, y='Churned_User', x=page_name + '_vs_NextSong', o
            plt.xlabel('(Number of ' + page_name + '/ Number of Next Songs) per User - Churned
            plt.ylabel('User Churned')
            plt.title('(Number of ' + page_name + '/ Number of Next) Songs per User')
            sns.despine(ax=ax);
```

In [77]:
```
ThumbsUp_vs_Songs = create_feature_column_page_event_vs_Songs_listened_per_userid(data_
plot_feature_column_page_event_vs_Songs_listened_per_userid(ThumbsUp_vs_Songs, 'Thumbs
ThumbsUp_vs_Songs.unpersist(blocking = True)
```

```
Mean value for Churned vs Non-Churned Users:
+------------+-------------------------+
|Churned_User|avg(Thumbs_Up_vs_NextSong)|
+------------+-------------------------+
|        true|       0.0511905532458333|
|       false|      0.05587908156155405|
+------------+-------------------------+

Mean value of Total Thumbs_Up per User in groups churned vs. non-churned:
+------------+---------------------------+
|Churned_User|avg(Total_Thumbs_Up_perUser)|
+------------+---------------------------+
|        true|           72.9306491188936|
|       false|          131.4456573469449|
+------------+---------------------------+

Violinplot:
```

Out[77]: DataFrame[userId: string, artist: string, auth: string, firstName: string, gender: stri



Graphically we can see a different distribution of Total_Thumbs_Up/Total_Songs_Played per user for churned vs. non-churned. The mean doesn't show a huge difference though. But the mean of Number of Thumbs Up per User for churned vs. non-churned shows a bigger difference. This is also logical though because Users who did not churn are also more likely to have more events and therefore more Thumbs Up.

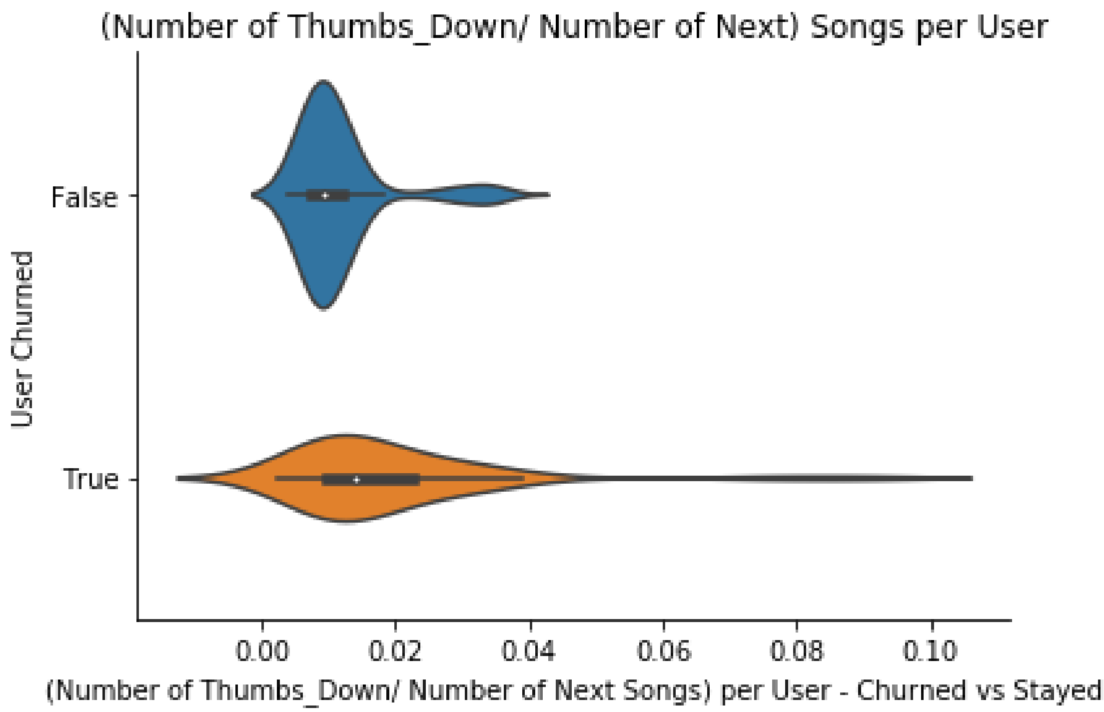
Metric: per User (Number of Thumbs Down / Number of listened to Songs)

In [78]:
```
ThumbsDown_vs_Songs = create_feature_column_page_event_vs_Songs_listened_per_userid(dat
plot_feature_column_page_event_vs_Songs_listened_per_userid(ThumbsDown_vs_Songs, 'Thumb
ThumbsDown_vs_Songs.unpersist(blocking = True)
```

```
Mean value for Churned vs Non-Churned Users:
+------------+--------------------------+
|Churned_User|avg(Thumbs_Down_vs_NextSong)|
+------------+--------------------------+
|        true|       0.013941022728977119|
|       false|       0.010831479965426652|
+------------+--------------------------+

Mean value of Total Thumbs_Down per User in groups churned vs. non-churned:
+------------+-----------------------------+
|Churned_User|avg(Total_Thumbs_Down_perUser)|
```

```
+------------+----------------------------+
|        true|           16.35016819914779|
|       false|           24.86821323869107|
+------------+----------------------------+

Violinplot:
```

Out[78]: DataFrame[userId: string, artist: string, auth: string, firstName: string, gender: stri



Graphically we can see a different distribution of Total_Thumbs_Down/Total_Songs_Played per user for churned vs. non-churned. The mean doesn't show a huge difference though. But the mean of Number of Thumbs Down per User for churned vs. non-churned shows a bigger difference. This is also logical though because Users who did not churn are also more likely to have more events and therefore more Thumbs Down.

Metric: per User (Number of Downgrades / Number total events) For the Events: - Submit Downgrade - Downgrade

In [79]:
```
def create_feature_column_page_event_vs_total_events_per_userid(pyspark_df, page_event)

    '''
    Function to add a column to a dataframe that shows the value of the following calcu
    The number of a certain page event vs. total events per userId.
```

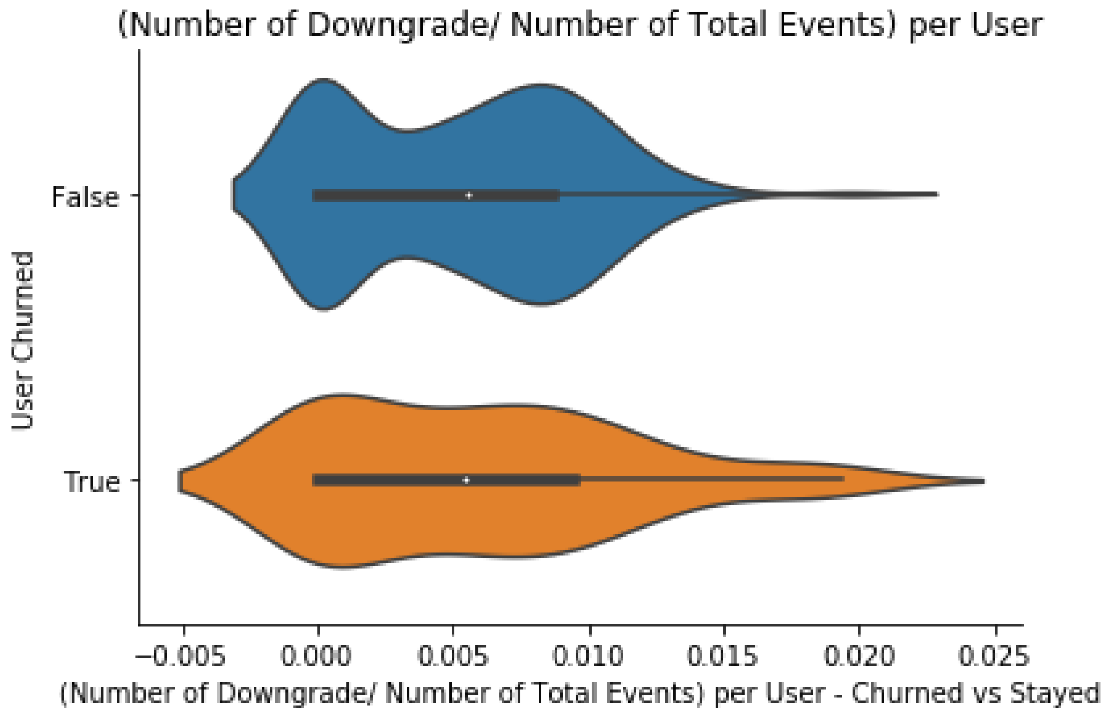
```
    args:
        pyspark_df - (pyspark dataframe) data_churn of Sparkify
        page_event - (string) of page event (for example 'Thumbs Up')
    return:
        feature_df - (pyspark dataframe) dataframe with feature

    '''

    page_name = page_event.replace(" ", "_")

    help_df = pyspark_df.select(['userId', 'page', 'Churned_User']) \
        .where((pyspark_df.page == page_event)) \
        .groupby('userId', 'page', 'Churned_User').count()

    join_df_allevents = data_churn.groupby('userId').count() \
        .withColumnRenamed('count', 'Total_User_Events')

    renamed_Column = 'Total_' + page_name + '_perUser'

    join_df_event = help_df.select(['userId', 'count']) \
        .withColumnRenamed('count', renamed_Column)

    pyspark_df = pyspark_df.join(join_df_allevents, on=['userId'], how='left')
    pyspark_df = pyspark_df.join(join_df_event, on=['userId'], how='left') \
        .fillna({'Total_' + page_name + '_perUser':0})

    feature_df = pyspark_df.withColumn(page_name + '_vs_Total_User_Events', (F.col(rena

    return feature_df


def plot_feature_column_page_event_vs_total_events_per_userid(feature_df, page_event):

    '''
    Function to create violinplot of pyspark dataframe created with
    'create_feature_column_page_event_vs_Songs_listened_per_userid' function.
    It also gives out the Mean value for Churned vs. Non-Churned users and
    the Mean value of the total page events per user in the groups churned vs. Non-chur

    args:
        pyspark_df - (pyspark dataframe) created with 'create_feature_column_page_event
        page_event - (string) of page event (for example 'Thumbs Up')

    return:
    '''

    page_name = page_event.replace(" ", "_")
```

```
            print("Mean value of (" + page_name + "/Total Events) for Churned vs Non-Churned Us
            feature_df.groupby('Churned_User').mean(page_name + '_vs_Total_User_Events').show(t

            help_pd = feature_df.select(['userId', 'Churned_User', page_name + '_vs_Total_User_
                .groupby('userId', 'Churned_User', page_name + '_vs_Total_User_Events').count()
                .select(['userId', 'Churned_User', page_name + '_vs_Total_User_Events']).toPand

            print("Mean value of Total " + page_name + " per User in groups churned vs. non-chu
            feature_df.groupby('Churned_User').mean('Total_' + page_name + '_perUser').show(tru

            print("Violinplot: ")
            ax = sns.violinplot(data=help_pd, y='Churned_User', x=page_name + '_vs_Total_User_E
            plt.xlabel('(Number of ' + page_name + '/ Number of Total Events) per User - Churne
            plt.ylabel('User Churned')
            plt.title('(Number of ' + page_name + '/ Number of Total Events) per User')
            sns.despine(ax=ax);
```

In [80]:
```
DG_df=create_feature_column_page_event_vs_total_events_per_userid(data_churn, 'Downgrad
plot_feature_column_page_event_vs_total_events_per_userid(DG_df, 'Downgrade')
DG_df.unpersist(blocking = True)
```

```
Mean value of (Downgrade/Total Events) for Churned vs Non-Churned Users:
+------------+----------------------------------+
|Churned_User|avg(Downgrade_vs_Total_User_Events)|
+------------+----------------------------------+
|true        |0.007511590584878605              |
|false       |0.007364224784602821              |
+------------+----------------------------------+

Mean value of Total Downgrade per User in groups churned vs. non-churned:
+------------+---------------------------+
|Churned_User|avg(Total_Downgrade_perUser)|
+------------+---------------------------+
|true        |13.299304564907276         |
|false       |22.22561618586309          |
+------------+---------------------------+

Violinplot:
```

Out[80]:
```
DataFrame[userId: string, artist: string, auth: string, firstName: string, gender: stri
```



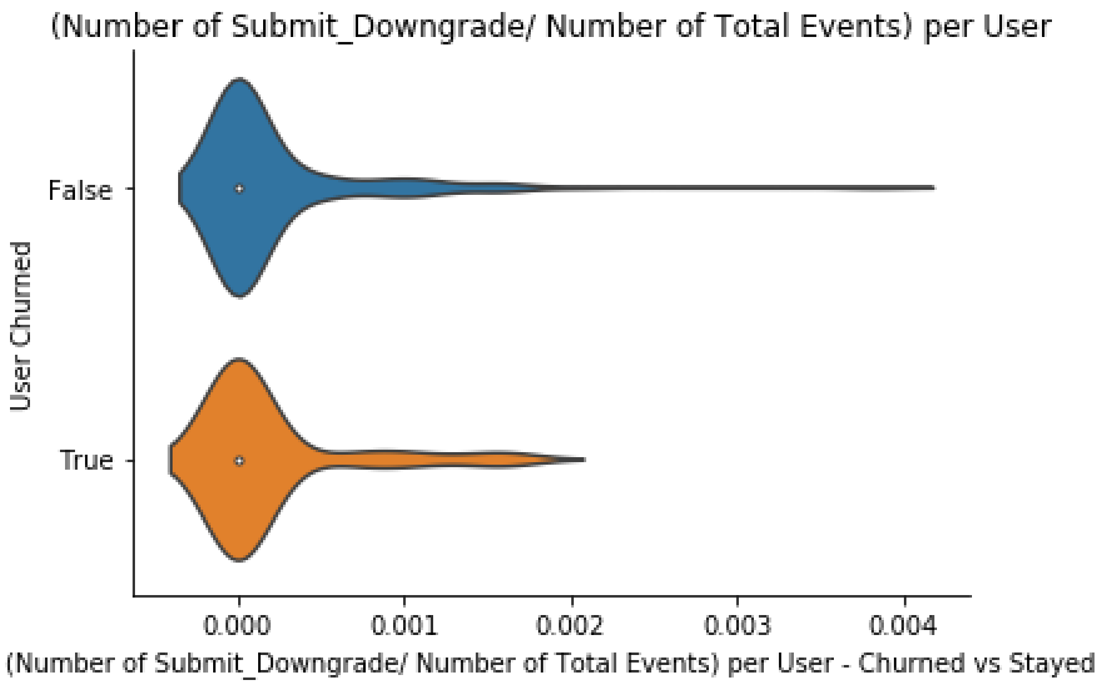
Graphically we can see a difference in the distribution. The mean doesn't show a huge difference though. But the mean of Number of Downgrade events per User for churned vs. non-churned shows a bigger difference. This is also logical though because Users who did not churn are also more likely to have more events and therefore more Downgrade events.

```
In [81]: DG_Submit_df=create_feature_column_page_event_vs_total_events_per_userid(data_churn, 'S
         plot_feature_column_page_event_vs_total_events_per_userid(DG_Submit_df, 'Submit Downgra
         DG_Submit_df.unpersist(blocking = True)
```

```
Mean value of (Submit_Downgrade/Total Events) for Churned vs Non-Churned Users:
+------------+-----------------------------------------+
|Churned_User|avg(Submit_Downgrade_vs_Total_User_Events)|
+------------+-----------------------------------------+
|true        |2.0060627674750337E-4                    |
|false       |2.3147155900381282E-4                    |
+------------+-----------------------------------------+

Mean value of Total Submit_Downgrade per User in groups churned vs. non-churned:
+------------+----------------------------------+
|Churned_User|avg(Total_Submit_Downgrade_perUser)|
+------------+----------------------------------+
|true        |0.247436697574893                 |
|false       |0.6426207724291654                |
+------------+----------------------------------+
```

```
Violinplot:
```

```
Out[81]: DataFrame[userId: string, artist: string, auth: string, firstName: string, gender: stri
```



Graphically we can see only a very slight difference in the distribution. The mean doesn't show a huge difference also. But the mean of Number of Submit Downgrade events per User for churned vs. non-churned shows a bigger difference. Churned Users seem to have a smaller amount of Submit Downgrading Events than the ones that stayed. This is interesting but also logical because Users who did not churn are also more likely to have more events and therefore more Submit Downgrade events.

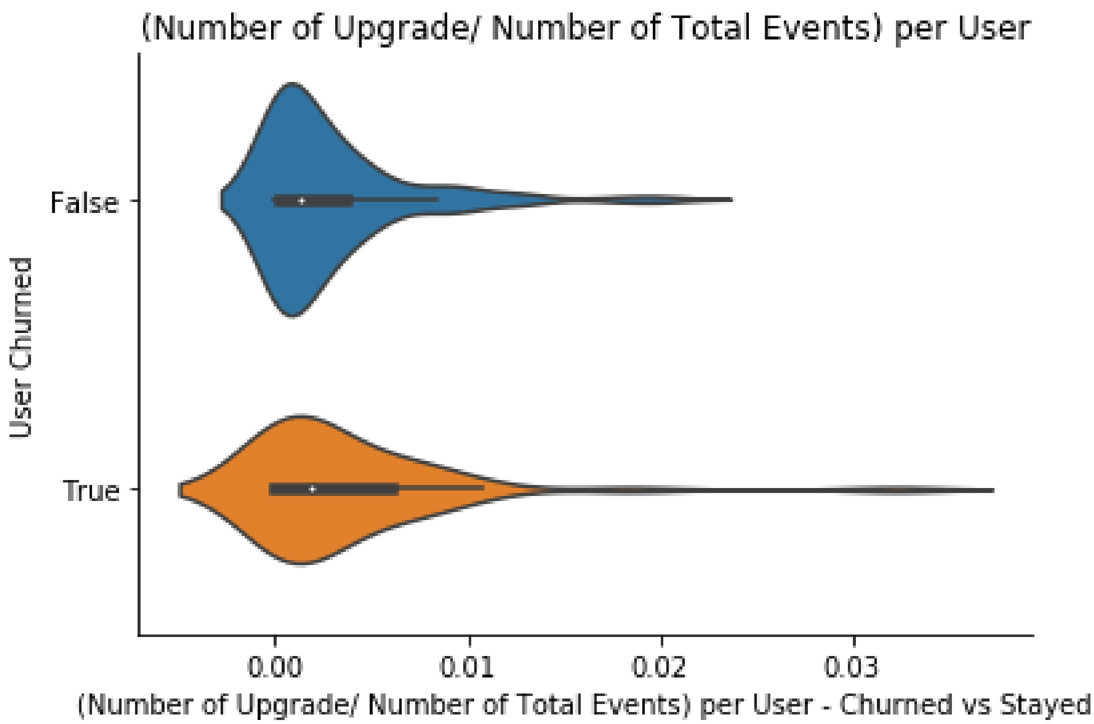
Metric: per User (Number of Upgrades / Number total events) For the Events: - Submit Upgrade - Upgrade

```
In [82]: UG_df=create_feature_column_page_event_vs_total_events_per_userid(data_churn, 'Upgrade'
         plot_feature_column_page_event_vs_total_events_per_userid(UG_df, 'Upgrade')
         UG_df.unpersist(blocking = True)
```

```
Mean value of (Upgrade/Total Events) for Churned vs Non-Churned Users:
+------------+---------------------------------+
|Churned_User|avg(Upgrade_vs_Total_User_Events)|
+------------+---------------------------------+
|true        |0.0024964336661911584            |
|false       |0.0016588795061939872            |
+------------+---------------------------------+
```

```
Mean value of Total Upgrade per User in groups churned vs. non-churned:
+------------+-------------------------+
|Churned_User|avg(Total_Upgrade_perUser)|
+------------+-------------------------+
|true        |2.9054029957203995       |
|false       |3.4733636246731536       |
+------------+-------------------------+

Violinplot:
```

Out[82]: DataFrame[userId: string, artist: string, auth: string, firstName: string, gender: stri

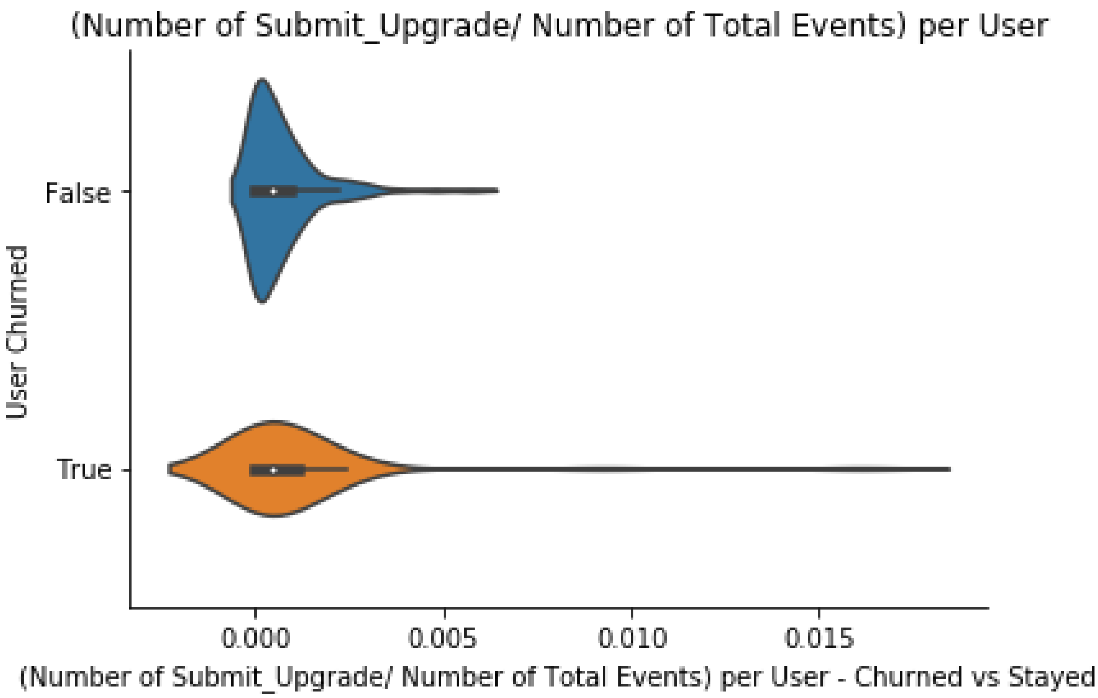


In [83]:
```python
Submit_UG_df=create_feature_column_page_event_vs_total_events_per_userid(data_churn, 'S
plot_feature_column_page_event_vs_total_events_per_userid(Submit_UG_df, 'Submit Upgrade
Submit_UG_df.unpersist(blocking = True)
```

```
Mean value of (Submit_Upgrade/Total Events) for Churned vs Non-Churned Users:
+------------+--------------------------------------+
|Churned_User|avg(Submit_Upgrade_vs_Total_User_Events)|
+------------+--------------------------------------+
|true        |7.132667617688996E-4                  |
|false       |5.443868146941574E-4                  |
```

```
+------------+--------------------------------------+

Mean value of Total Submit_Upgrade per User in groups churned vs. non-churned:
+------------+--------------------------------+
|Churned_User|avg(Total_Submit_Upgrade_perUser)|
+------------+--------------------------------+
|true        |0.8919623751783167              |
|false       |1.2159072399159845              |
+------------+--------------------------------+

Violinplot:
```

Out[83]: DataFrame[userId: string, artist: string, auth: string, firstName: string, gender: stri



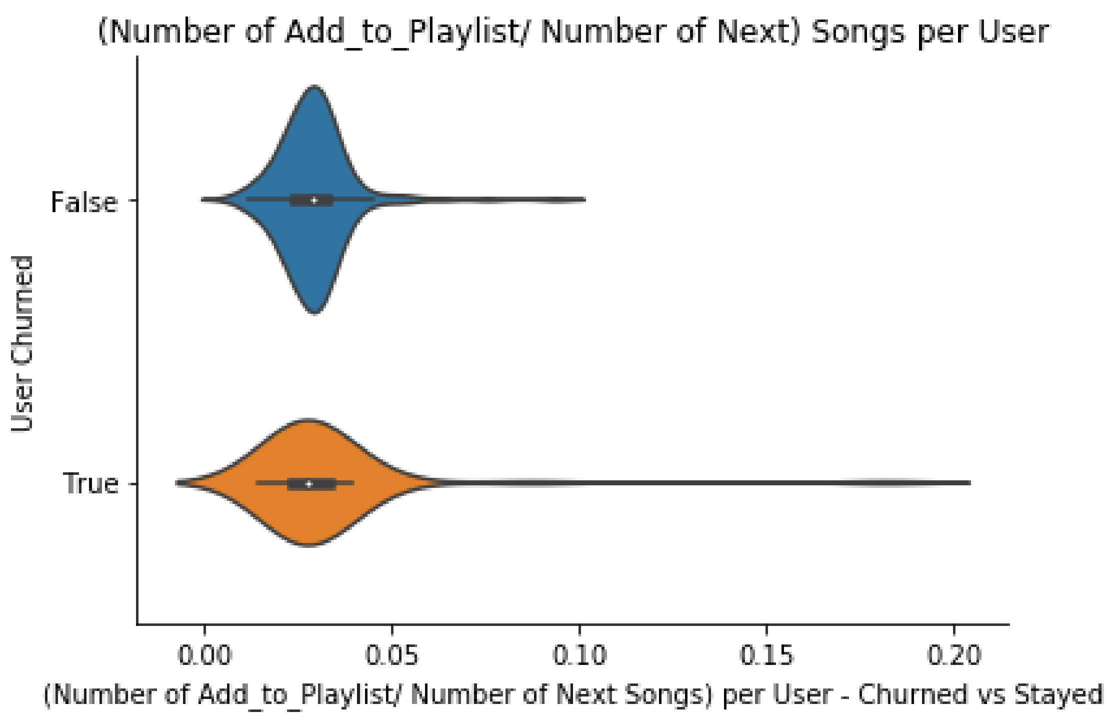
Metric: per User (Number of Add to Playlist / Number of listened to Songs)

In [84]:
```
Playlist_vs_Songs = create_feature_column_page_event_vs_Songs_listened_per_userid(data_
plot_feature_column_page_event_vs_Songs_listened_per_userid(Playlist_vs_Songs, 'Add to
Playlist_vs_Songs.unpersist(blocking = True)
```

```
Mean value for Churned vs Non-Churned Users:
+------------+-------------------------------+
|Churned_User|avg(Add_to_Playlist_vs_NextSong)|
+------------+-------------------------------+
```

```
|        true|            0.028570195266199073|
|       false|            0.028686228895917598|
+------------+--------------------------------+

Mean value of Total Add_to_Playlist per User in groups churned vs. non-churned:
+------------+----------------------------------+
|Churned_User|avg(Total_Add_to_Playlist_perUser)|
+------------+----------------------------------+
|        true|                 39.95666822857398|
|       false|                 68.23724089954611|
+------------+----------------------------------+

Violinplot:
```

```
Out[84]: DataFrame[userId: string, artist: string, auth: string, firstName: string, gender: stri
```



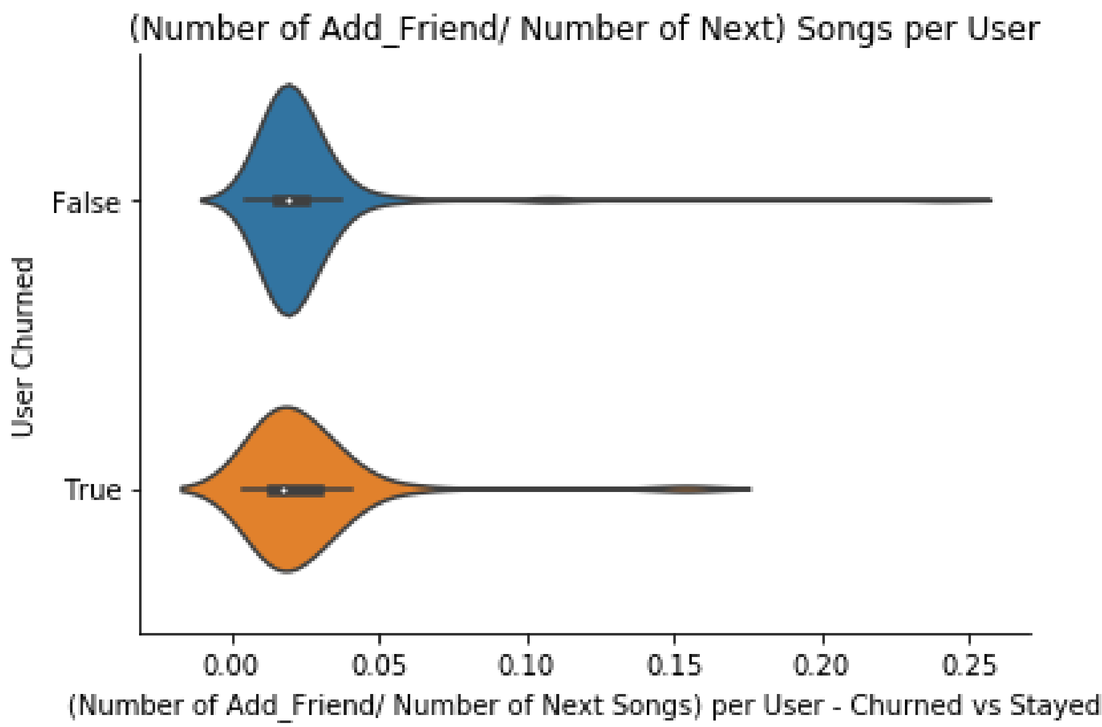
Metric: per User (Number of Add Friend / Number of listened to Songs)

```
In [85]: Friend_vs_Songs = create_feature_column_page_event_vs_Songs_listened_per_userid(data_ch
         plot_feature_column_page_event_vs_Songs_listened_per_userid(Friend_vs_Songs, 'Add Frien
         Friend_vs_Songs.unpersist(blocking = True)
```

```
Mean value for Churned vs Non-Churned Users:
+------------+--------------------------+
```

```
|Churned_User|avg(Add_Friend_vs_NextSong)|
+------------+--------------------------+
|        true|       0.01786939670579943|
|       false|      0.019144506440527658|
+------------+--------------------------+

Mean value of Total Add_Friend per User in groups churned vs. non-churned:
+------------+----------------------------+
|Churned_User|avg(Total_Add_Friend_perUser)|
+------------+----------------------------+
|        true|           22.207563425306642|
|       false|            43.58670535160824|
+------------+----------------------------+

Violinplot:
```

Out[85]: DataFrame[userId: string, artist: string, auth: string, firstName: string, gender: stri



Metric: Streaming time per active day

In [86]:
```
def function_streaming(data_churn):

    streaming_time_df = data_churn.filter(data_churn.page == 'NextSong') \
```

```
            .groupBy('userId', 'Churned_User') \
            .agg((F.sum('length')/3600).alias('streamingTime_h'), #'length' column is in se
             F.countDistinct('date').alias('days_active'))

        streaming_time_df = streaming_time_df \
            .withColumn('streaming_per_active_day', streaming_time_df.streamingTime_h/strea

        data_churn = data_churn.join(streamin_time_df, on=['userId'], how='left')

        return data_churn
```

```
In [87]: # change format of timestamp to date
         udf_convert_ts_to_date = F.udf(lambda timestamp: datetime.datetime.fromtimestamp(
             timestamp / 1000.0).strftime('%Y-%m-%d'))

         data_churn = data_churn.withColumn("date", udf_convert_ts_to_date(data_churn.ts))
```

```
In [88]: streaming_time_df = data_churn.filter(data_churn.page == 'NextSong') \
             .groupBy('userId', 'Churned_User') \
             .agg((F.sum('length')/3600).alias('streamingTime_h'), #'length' column is in second
                  F.countDistinct('date').alias('days_active'))
```

```
In [89]: streaming_time_df = streaming_time_df \
             .withColumn('streaming_per_active_day', streaming_time_df.streamingTime_h/streaming
```

```
In [90]: streaming_time_df.show()
```

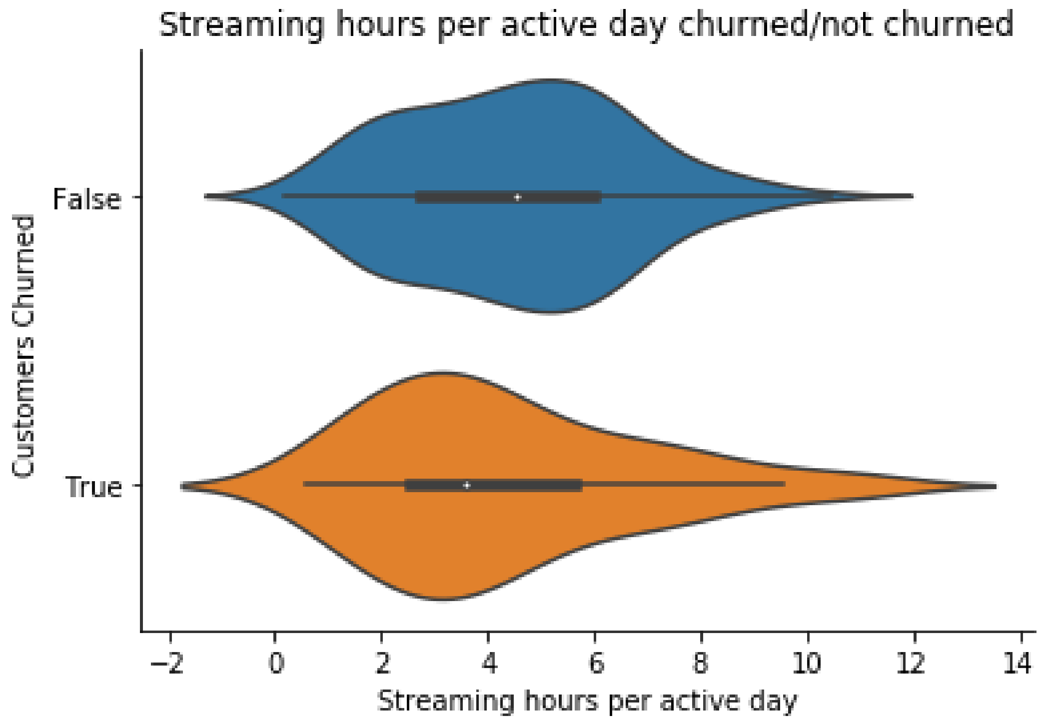
```
+------+------------+------------------+-----------+------------------------+
|userId|Churned_User|   streamingTime_h|days_active|streaming_per_active_day|
+------+------------+------------------+-----------+------------------------+
|    19|       false| 15.13359274166667|          2|       7.566796370833335|
|300007|        true| 7.785752911111113|          2|      3.8928764555555566|
|100005|        true|10.288089447222225|          4|       2.572022361805556|
|200007|       false| 4.372185919444444|          2|       2.186092959722222|
|200002|       false| 26.11357664999999|          7|       3.730510949999999|
|    50|       false| 34.09757655833333|          8|       4.262197069791666|
|    30|       false| 99.94673490000002|         23|        4.34551021304348|
|100012|        true| 32.08426475277778|          7|      4.5834663932539685|
|     8|       false| 17.60931441944444|          7|      2.5156163456349203|
|100011|        true|0.7893609722222222|          1|      0.7893609722222222|
|     4|       false|140.59445593888893|         26|       5.407479074572651|
|    65|       false| 147.0438626166667|         24|       6.126827609027779|
|   101|        true|124.29555963055556|         13|        9.56119689465812|
|    13|       false| 87.12195035833334|         26|      3.3508442445512823|
|   153|       false| 64.85991957777777|         13|       4.989224582905982|
|    42|       false|244.94249059166665|         41|       5.974207087601625|
|100014|        true|18.806520022222223|          6|       3.134420003703704|
|    94|       false|10.391310405555554|          6|      1.7318850675925923|
|    62|       false|109.76057915000004|         12|        9.14671492916667|
```

```
|   114|       false| 91.06569847777779|        17|       5.356805792810458|
+------+------------+------------------+-----------+------------------------+
only showing top 20 rows
```

In [91]:
```python
# compare difference in mean
print("Difference in mean of streaming time per active day for churned vs. not-churned"
streaming_time_df.groupBy('Churned_User').mean().show()

# convert to pandas df to plot
streaming_time_pd = streaming_time_df.toPandas()

# plot
ax = sns.violinplot(data=streaming_time_pd, y='Churned_User', x='streaming_per_active_d
plt.xlabel('Streaming hours per active day')
plt.ylabel('Customers Churned')
plt.title('Streaming hours per active day churned/not churned')
sns.despine(ax=ax);

streaming_time_df.unpersist(blocking = True)
```

```
Difference in mean of streaming time per active day for churned vs. not-churned
+------------+--------------------+------------------+-----------------------------+
|Churned_User|avg(streamingTime_h)|  avg(days_active)|avg(streaming_per_active_day)|
+------------+--------------------+------------------+-----------------------------+
|        true|   48.33729681981836| 9.596153846153847|            4.320860089568765|
|       false|   76.71303818550095|14.791907514450868|            4.495159812330317|
+------------+--------------------+------------------+-----------------------------+
```

Out[91]:
```
DataFrame[userId: string, Churned_User: boolean, streamingTime_h: double, days_active:
```



We see there is a difference in the metrics between churned and non-churned. Churned Users seem to have on average less streaming time, less active days but only very little less streaming-time per active day than non-churned user. This makes sense since non-churned users have more events in the dataset therefore are also more likely to have more streaming time and active days.

Metric: gender

```
In [92]: Total_Churned_User = data_churn.select('userId', 'Churned_User').dropDuplicates() \
             .where(data_churn.Churned_User=='true').count()

         Total_Stayed_User = data_churn.select('userId', 'Churned_User').dropDuplicates() \
             .where(data_churn.Churned_User=='false').count()

In [93]: help_df_churned = data_churn.select('userId', 'Churned_User', 'gender').dropDuplicates(
             .where(data_churn.Churned_User=='true') \
             .groupby('gender', 'Churned_User').count() \
             .withColumnRenamed('count', 'count_churned_gender')

         help_df_churned.withColumn('count/total_churned_users', help_df_churned.count_churned_g
```

```
+------+------------+--------------------+-------------------------+
|gender|Churned_User|count_churned_gender|count/total_churned_users|
+------+------------+--------------------+-------------------------+
|     M|        true|                  32|       0.6153846153846154|
|     F|        true|                  20|      0.38461538461538464|
```

```
+------+------------+-------------------+------------------------+
```

```
In [94]: help_df_churned = data_churn.select('userId', 'Churned_User', 'gender').dropDuplicates(
             .where(data_churn.Churned_User=='false') \
             .groupby('gender', 'Churned_User').count() \
             .withColumnRenamed('count', 'count_stayed_gender')

         help_df_churned.withColumn('count/total_stayed_users', help_df_churned.count_stayed_gen

         help_df_churned.unpersist(blocking = True)
```

```
+------+------------+-------------------+-----------------------+
|gender|Churned_User|count_stayed_gender|count/total_stayed_users|
+------+------------+-------------------+-----------------------+
|     F|       false|                 84|     0.48554913294797686|
|     M|       false|                 89|      0.5144508670520231|
+------+------------+-------------------+-----------------------+
```

```
Out[94]: DataFrame[gender: string, Churned_User: boolean, count_stayed_gender: bigint]
```

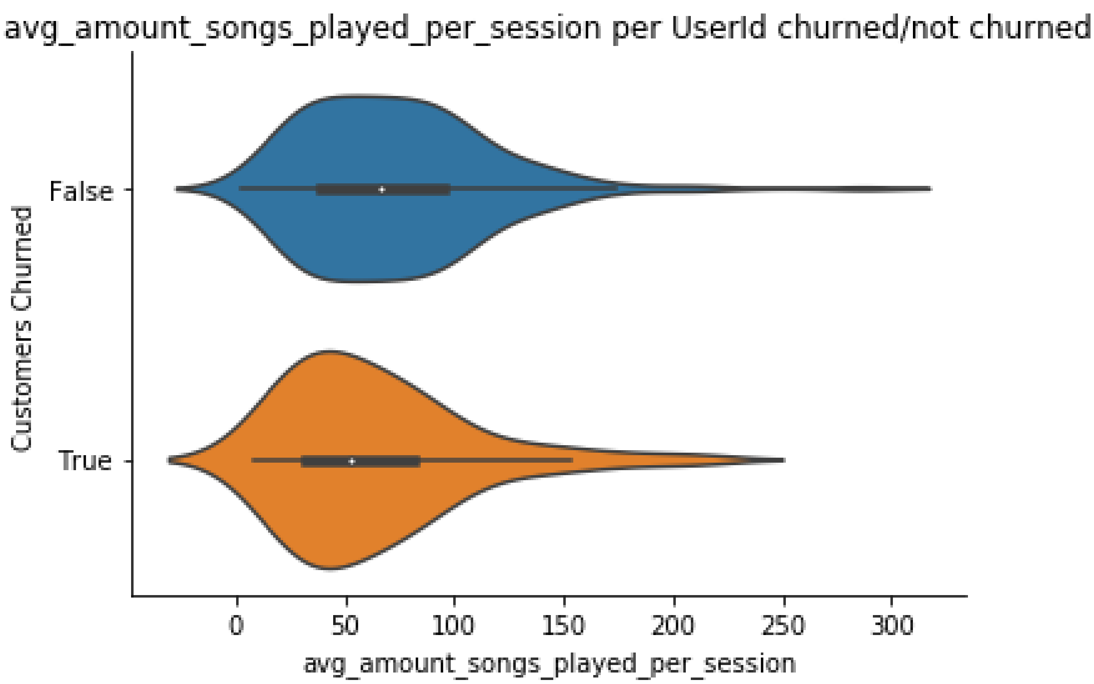
In our dataset it seems like it is more likely for males to churn but this could also just be a convincidence as we only have 225 users in the dataset.

Metric: Average Amount of Songs played per Session

```
In [95]: avg_amount_songs_played_per_session = data_churn.select('userId', 'sessionId', 'Churned
             .filter('page = "NextSong"') \
             .groupby('userId', 'sessionId', 'Churned_User').count() \
             .select('userId', 'Churned_User', 'count') \
             .groupby('userId', 'Churned_User').mean() \
             .withColumnRenamed('avg(count)', 'avg_amount_songs_played_per_session')

In [96]: print("Average amount of songs played per session Churned vs. Non-Churned")
         avg_amount_songs_played_per_session.groupBy('Churned_User').mean().show()
```

```
Average amount of songs played per session Churned vs. Non-Churned
+------------+----------------------------------------+
|Churned_User|avg(avg_amount_songs_played_per_session)|
+------------+----------------------------------------+
|        true|                       63.537152578095224|
|       false|                        72.96967249911357|
+------------+----------------------------------------+
```

```
In [97]: df_pd = avg_amount_songs_played_per_session.toPandas()

         ax = sns.violinplot(data=df_pd, y='Churned_User', x='avg_amount_songs_played_per_sessio
         plt.xlabel('avg_amount_songs_played_per_session')
         plt.ylabel('Customers Churned')
         plt.title('avg_amount_songs_played_per_session per UserId churned/not churned')
         sns.despine(ax=ax);

         avg_amount_songs_played_per_session.unpersist(blocking = True)

Out[97]: DataFrame[userId: string, Churned_User: boolean, avg_amount_songs_played_per_session: d
```



In our dataset it seems like user who did not churn played slightly more songs per session than users who churned.

Exploration: Is churning more likely to happen for users on paid or on free level?

```
In [98]: df_help= data_churn.select(['userId','Churning_Event','level', 'Churned_User']) \
             .where(data_churn.Churning_Event == 1) \
             .groupby('level','Churned_User').count()

         df_help.show()
         #this is the amount of users who have churned in free and paid level

         df_help.unpersist(blocking = True)
```

```
+-----+------------+-----+
|level|Churned_User|count|
+-----+------------+-----+
| paid|        true|   31|
| free|        true|   21|
+-----+------------+-----+
```

```
Out[98]: DataFrame[level: string, Churned_User: boolean, count: bigint]
```

In our dataset it seems like the users who churned did it more often when they were in the paid-level.

Metric: Number of Adverts / Number of Songs listented to per User We usually get Adverts after playing a song. We get Adverts in free level but also in paid level (see below).

```
In [99]: data_churn.where(data_churn.page == 'Roll Advert').groupby('level').count().show()
```

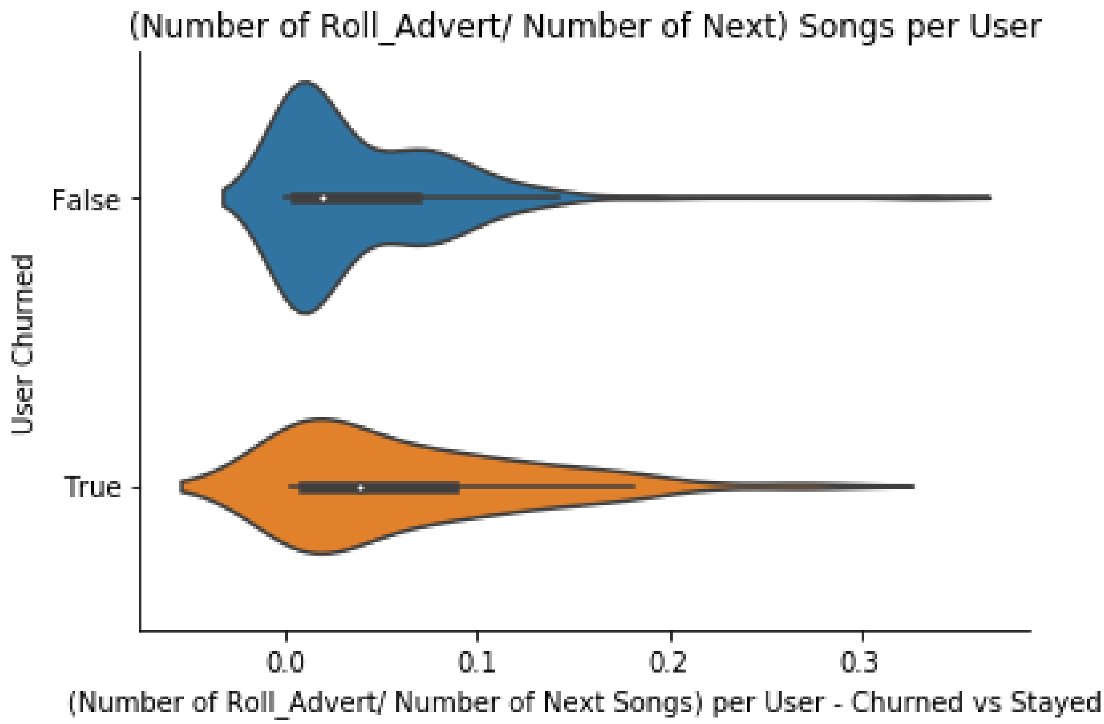
```
+-----+-----+
|level|count|
+-----+-----+
| free| 3687|
| paid|  246|
+-----+-----+
```

```
In [100]: Adverts_vs_Songs = create_feature_column_page_event_vs_Songs_listened_per_userid(data_
          plot_feature_column_page_event_vs_Songs_listened_per_userid(Adverts_vs_Songs, 'Roll Ad

          Adverts_vs_Songs.unpersist(blocking = True)
```

```
Mean value for Churned vs Non-Churned Users:
+------------+---------------------------+
|Churned_User|avg(Roll_Advert_vs_NextSong)|
+------------+---------------------------+
|        true|       0.030179766545188547|
|       false|         0.0167053058017698|
+------------+---------------------------+

Mean value of Total Roll_Advert per User in groups churned vs. non-churned:
+------------+-----------------------------+
|Churned_User|avg(Total_Roll_Advert_perUser)|
+------------+-----------------------------+
|        true|            25.155107790006447|
|       false|             27.91491327931427|
+------------+-----------------------------+
```

```
Violinplot:
```

```
Out[100]: DataFrame[userId: string, artist: string, auth: string, firstName: string, gender: str
```



In our dataset it seems like the users who churned had on average more Adverts/number of total listened songs than users who stayed.

Metric: Percentage of days since registration where user was active

```
In [101]: active_df = data_churn.groupBy('userId', 'Churned_User') \
                  .agg(F.max('days_since_registration').alias('days_since_registration'),
                      F.countDistinct('date').alias('days_user_active'))

In [102]: active_df = active_df.filter('days_since_registration >= 1') #only get the entries whe

In [103]: active_df = active_df \
                  .withColumn('percentage_active_days', active_df.days_user_active/active_df.days_si

In [104]: active_df.show(5)
```

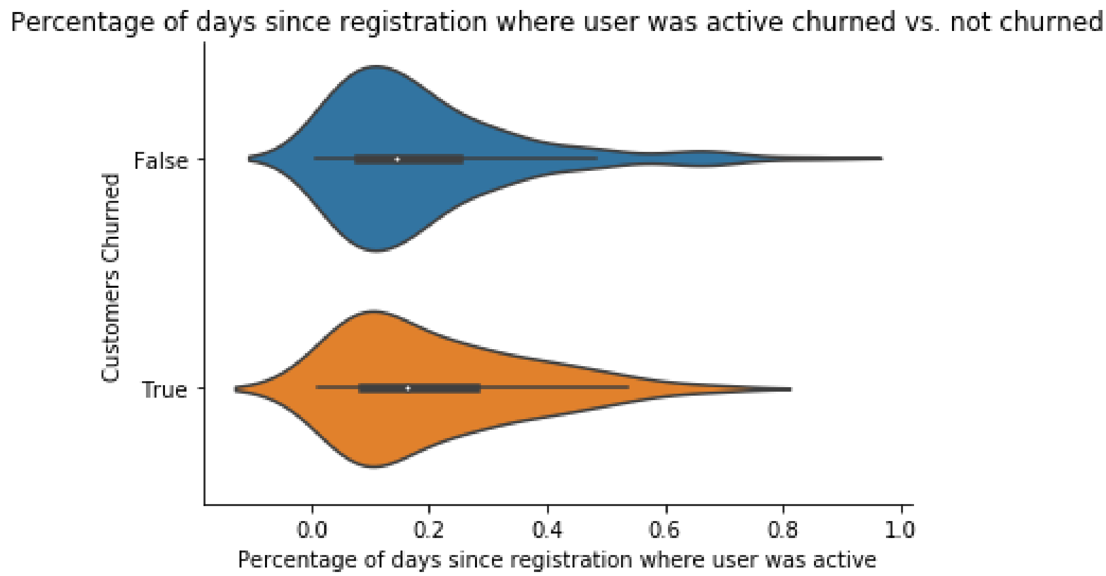
```
+------+------------+-----------------------+----------------+----------------------+
|userId|Churned_User|days_since_registration|days_user_active|percentage_active_days|
+------+------------+-----------------------+----------------+----------------------+
|    19|       false|     21.451435185185186|               2|   0.09323385511199932|
```

```
|300007|        true|     11.553703703703704|               2|   0.17310466420900786|
|200007|       false|      53.39774305555556|               3|    0.0561821498125636|
|100005|        true|      85.19559027777778|               4|   0.04695078685361666|
|200002|       false|      70.07462962962963|               7|   0.09989349978726385|
+------+------------+-----------------------+----------------+----------------------+
only showing top 5 rows
```

In [105]:
```python
# compare difference in mean
active_df.groupBy('Churned_User').mean().show()

# For plotting:
active_pd = active_df.toPandas()

# plot
ax = sns.violinplot(data=active_pd, y='Churned_User', x='percentage_active_days', orie
plt.xlabel('Percentage of days since registration where user was active')
plt.ylabel('Customers Churned')
plt.title('Percentage of days since registration where user was active churned vs. not
sns.despine(ax=ax);

active_df.unpersist(blocking = True)
```

```
+------------+---------------------------+---------------------+---------------------------+
|Churned_User|avg(days_since_registration)|avg(days_user_active)|avg(percentage_active_days)|
+------------+---------------------------+---------------------+---------------------------+
|        true|          57.30599292200854|    9.826923076923077|        0.20854583851910954|
|       false|          87.12240363910423|   15.186046511627907|        0.19036440426128262|
+------------+---------------------------+---------------------+---------------------------+
```

Out[105]:
```
DataFrame[userId: string, Churned_User: boolean, days_since_registration: double, days
```



In the next part we will creat Features for the dataframe based on the Exploration in this notebook. Please see 02_Sparkify_Feature_Engineering.