Sparkify

September 9, 2022

1 Sparkify Churn: Capstone Project

1.0.1 by Pengfei Hu

1.1 Introduction

In this project I will load and manipulate a music app dataset similar to Spotify with Spark to engineer relevant features for predicting churn. Where Churn is cancelling their service altogether. By identifying these customers before they churn, the business can offer discounts and incentives to stay thereby potentially saving the business revenue. This workspace contains a tiny subset (128MB) of the full dataset available (12GB).

```
In [1]: # import libraries
        import pyspark
        from pyspark import SparkConf
        from pyspark.sql import SparkSession
        from pyspark.sql.functions import udf
        from pyspark.sql.types import StringType
        from pyspark.sql.types import IntegerType
        from pyspark.sql.functions import isnan, count, when, col, desc, udf, col, sort_array, a
        from pyspark.sql.functions import sum as Fsum
        from pyspark.sql.window import Window
        from pyspark.sql import Row
        from pyspark.sql import functions as F
        from pyspark.sql.functions import *

        from pyspark.ml import Pipeline
        from pyspark.ml.classification import LogisticRegression, RandomForestClassifier, GBTCla
        from pyspark.ml.evaluation import MulticlassClassificationEvaluator
        from pyspark.ml.feature import CountVectorizer, IDF, PCA, RegexTokenizer, VectorAssemble
        from pyspark.ml.regression import LinearRegression
        from pyspark.ml.tuning import CrossValidator, ParamGridBuilder

        import datetime
        import time

        import pandas as pd
        import numpy as np
```

```
import re
%matplotlib inline
import matplotlib.pyplot as plt
import seaborn as sns
sns.set()
```

We can now create a Spark Session.

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

```
In [3]: spark.sparkContext.getConf().getAll()
```

```
Out[3]: [('spark.rdd.compress', 'True'),
         ('spark.driver.port', '34953'),
         ('spark.app.id', 'local-1662674415282'),
         ('spark.serializer.objectStreamReset', '100'),
         ('spark.master', 'local[*]'),
         ('spark.executor.id', 'driver'),
         ('spark.submit.deployMode', 'client'),
         ('spark.ui.showConsoleProgress', 'true'),
         ('spark.driver.host', '6dfc594f9881'),
         ('spark.app.name', 'Sparkify Project')]
```

2 Load and Clean Dataset

In this workspace, the mini-dataset file is `mini_sparkify_event_data.json`. Load and clean the dataset, checking for invalid or missing data - for example, records without userids or sessionids.

```
In [4]: # load in the dataset
        df = spark.read.json("mini_sparkify_event_data.json")
```

```
In [5]: # print the schema
        df.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 [6]:
```
df.describe()
```

Out[6]:
```
DataFrame[summary: string, artist: string, auth: string, firstName: string, gender: stri
```

The datatypes make sense at this stage so we don't need to make any changes.

In [7]:
```
df.take(2)
```

Out[7]:
```
[Row(artist='Martha Tilston', auth='Logged In', firstName='Colin', gender='M', itemInSes
 Row(artist='Five Iron Frenzy', auth='Logged In', firstName='Micah', gender='M', itemInS
```

In [8]:
```
# get the count of the dataset before we do any cleaning - this is 286500
df.count()
```

Out[8]:
```
286500
```

The dataset before any cleaning is performed has 286500 rows.

2.0.1 Drop Rows with Missing Values

First we will drop any rows with missing values in the userid or sessionid.

In [9]:
```
# drop rows with missing values in userid and/or sessionid
df = df.dropna(how = 'any', subset = ["userId", "sessionId"])
```

In [10]:
```
df.count()
```

Out[10]:
```
286500
```

As we can see from the above, the row count is still the same at 286500. Let's take a closer look.

In [11]:
```
# drop userid duplicates
df.select("userId").dropDuplicates().sort("userId").show()
```

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

```
|    10|
|   100|
|100001|
|100002|
|100003|
|100004|
|100005|
|100006|
|100007|
|100008|
|100009|
|100010|
|100011|
|100012|
|100013|
|100014|
|100015|
|100016|
|100017|
+------+
only showing top 20 rows
```

From the above, we can see that there are empty strings being used for a userId. We will drop these after we further investigate the sessionid.

```
In [12]: df.select("sessionId").dropDuplicates().sort("sessionId").show()
```

```
+---------+
|sessionId|
+---------+
|        1|
|        2|
|        3|
|        4|
|        5|
|        6|
|        7|
|        8|
|        9|
|       10|
|       11|
|       12|
|       13|
|       15|
|       16|
|       17|
```

```
|       18|
|       19|
|       20|
|       21|
+---------+
only showing top 20 rows
```

The sessionId looks as expected. However we saw from above that there are entries with an empty string for the userId. These should now be removed.

```
In [13]: # remove those with an empty string userId
         df = df.filter(df["userId"] != "")
```

```
In [14]: df.count()
```

```
Out[14]: 278154
```

We have dropped (286500-278154) = **8346 rows** in this cleaning step.

```
In [15]: df_pandas = df.toPandas()
         df_pandas
```

```
Out[15]:                                      artist       auth firstName gender  \
         0                           Martha Tilston  Logged In     Colin      M
         1                         Five Iron Frenzy  Logged In     Micah      M
         2                             Adam Lambert  Logged In     Colin      M
         3                                   Enigma  Logged In     Micah      M
         4                                Daft Punk  Logged In     Colin      M
         5                 The All-American Rejects  Logged In     Micah      M
         6             The Velvet Underground / Nico  Logged In     Micah      M
         7                             Starflyer 59  Logged In     Colin      M
         8                                     None  Logged In     Colin      M
         9                                 Frumpies  Logged In     Colin      M
         10                            Britt Nicole  Logged In     Micah      M
         11                                    None  Logged In     Micah      M
         12         Edward Sharpe & The Magnetic Zeros  Logged In     Colin      M
         13                                   Tesla  Logged In     Micah      M
         14                                    None  Logged In     Micah      M
         15                             Stan Mosley  Logged In     Colin      M
         16                  Florence + The Machine  Logged In     Micah      M
         17                       Tokyo Police Club  Logged In   Ashlynn      F
         18                                 Orishas  Logged In     Colin      M
         19                                 Ratatat  Logged In     Micah      M
         20                            Manolo Garcia  Logged In   Ashlynn      F
         21                                Downhere  Logged In     Colin      M
         22                                   Modjo  Logged In     Alexi      F
         23                        MÃČÂűtley CrÃČÂije  Logged In     Micah      M
```

```
24                                 David Bowie  Logged In   Ashlynn      F
25                                     Skillet  Logged In     Colin      M
26                               Edwyn Collins  Logged In     Alexi      F
27                                Telepopmusik  Logged In     Micah      M
28                               Kings Of Leon  Logged In   Ashlynn      F
29                     Florence + The Machine  Logged In     Colin      M
...                                        ...        ...       ...    ...
278124                                    Muse  Logged In    Emilia      F
278125                     Natural Born Deejays  Logged In    Emilia      F
278126                                    None  Logged In    Emilia      F
278127                                  Shaggy  Logged In    Emilia      F
278128                                 Natiruts  Logged In    Emilia      F
278129                                In Flames  Logged In    Emilia      F
278130                                  Eminem  Logged In    Emilia      F
278131                                Blink-182  Logged In    Emilia      F
278132                                    None  Logged In    Emilia      F
278133                               Jason Mraz  Logged In    Emilia      F
278134                     Jessica Lea Mayfield  Logged In    Emilia      F
278135                             Jimi Hendrix  Logged In    Emilia      F
278136                                    None  Logged In    Emilia      F
278137             Jamie Foxx featuring Twista  Logged In    Emilia      F
278138                                  Aaliyah  Logged In    Emilia      F
278139                                    None  Logged In    Emilia      F
278140                                  Hermano  Logged In    Emilia      F
278141                                   Saliva  Logged In    Emilia      F
278142                              Lupe Fiasco  Logged In    Emilia      F
278143                                 Harmonia  Logged In    Emilia      F
278144                       The Rolling Stones  Logged In    Emilia      F
278145                           Alejandro Sanz  Logged In    Emilia      F
278146                     Valley of the Giants  Logged In    Emilia      F
278147                                    None  Logged In    Emilia      F
278148                                    Olive  Logged In    Emilia      F
278149                              Iron Maiden  Logged In    Emilia      F
278150                                    None  Logged In    Emilia      F
278151                                    None  Logged In    Emilia      F
278152                                    None  Logged In    Emilia      F
278153                           Camera Obscura  Logged In    Emilia      F

        itemInSession  lastName     length level  \
0                  50   Freeman  277.89016  paid
1                  79      Long  236.09424  free
2                  51   Freeman  282.82730  paid
3                  80      Long  262.71302  free
4                  52   Freeman  223.60771  paid
5                  81      Long  208.29995  free
6                  82      Long  260.46649  free
7                  53   Freeman  185.44281  paid
8                  54   Freeman        NaN  paid
```

9	55	Freeman	134.47791	paid
10	83	Long	229.87710	free
11	84	Long	NaN	free
12	56	Freeman	223.58159	paid
13	85	Long	201.06404	free
14	86	Long	NaN	free
15	57	Freeman	246.69995	paid
16	87	Long	168.64608	free
17	0	Williams	166.11220	free
18	58	Freeman	222.22322	paid
19	88	Long	229.77261	free
20	1	Williams	283.74159	free
21	59	Freeman	223.92118	paid
22	0	Warren	250.93179	paid
23	89	Long	231.26159	free
24	2	Williams	174.41914	free
25	60	Freeman	233.32526	paid
26	1	Warren	216.84200	paid
27	90	Long	241.60608	free
28	3	Williams	307.46077	free
29	61	Freeman	219.66322	paid
...	...	...	...	...
278124	13	House	228.93669	paid
278125	14	House	203.85914	paid
278126	15	House	NaN	paid
278127	16	House	201.37751	paid
278128	17	House	210.88608	paid
278129	18	House	217.65179	paid
278130	19	House	250.82730	paid
278131	20	House	173.76608	paid
278132	21	House	NaN	paid
278133	22	House	218.40934	paid
278134	23	House	180.74077	paid
278135	24	House	380.68200	paid
278136	25	House	NaN	paid
278137	26	House	258.79465	paid
278138	27	House	228.28363	paid
278139	28	House	NaN	paid
278140	29	House	115.90485	paid
278141	30	House	222.56281	paid
278142	31	House	273.94567	paid
278143	32	House	655.77751	paid
278144	33	House	225.30567	paid
278145	34	House	241.52771	paid
278146	35	House	420.46649	paid
278147	36	House	NaN	paid
278148	37	House	264.12363	paid
278149	38	House	258.66404	paid

```
278150             39     House        NaN  paid
278151             43     House        NaN  paid
278152             44     House        NaN  paid
278153             45     House  170.89261  paid

                                   location method             page  \
0                             Bakersfield, CA    PUT         NextSong
1              Boston-Cambridge-Newton, MA-NH    PUT         NextSong
2                             Bakersfield, CA    PUT         NextSong
3              Boston-Cambridge-Newton, MA-NH    PUT         NextSong
4                             Bakersfield, CA    PUT         NextSong
5              Boston-Cambridge-Newton, MA-NH    PUT         NextSong
6              Boston-Cambridge-Newton, MA-NH    PUT         NextSong
7                             Bakersfield, CA    PUT         NextSong
8                             Bakersfield, CA    PUT  Add to Playlist
9                             Bakersfield, CA    PUT         NextSong
10             Boston-Cambridge-Newton, MA-NH    PUT         NextSong
11             Boston-Cambridge-Newton, MA-NH    GET      Roll Advert
12                            Bakersfield, CA    PUT         NextSong
13             Boston-Cambridge-Newton, MA-NH    PUT         NextSong
14             Boston-Cambridge-Newton, MA-NH    PUT        Thumbs Up
15                            Bakersfield, CA    PUT         NextSong
16             Boston-Cambridge-Newton, MA-NH    PUT         NextSong
17                            Tallahassee, FL    PUT         NextSong
18                            Bakersfield, CA    PUT         NextSong
19             Boston-Cambridge-Newton, MA-NH    PUT         NextSong
20                            Tallahassee, FL    PUT         NextSong
21                            Bakersfield, CA    PUT         NextSong
22                Spokane-Spokane Valley, WA    PUT         NextSong
23             Boston-Cambridge-Newton, MA-NH    PUT         NextSong
24                            Tallahassee, FL    PUT         NextSong
25                            Bakersfield, CA    PUT         NextSong
26                Spokane-Spokane Valley, WA    PUT         NextSong
27             Boston-Cambridge-Newton, MA-NH    PUT         NextSong
28                            Tallahassee, FL    PUT         NextSong
29                            Bakersfield, CA    PUT         NextSong
...                                       ...    ...              ...
278124  New York-Newark-Jersey City, NY-NJ-PA    PUT         NextSong
278125  New York-Newark-Jersey City, NY-NJ-PA    PUT         NextSong
278126  New York-Newark-Jersey City, NY-NJ-PA    GET             Home
278127  New York-Newark-Jersey City, NY-NJ-PA    PUT         NextSong
278128  New York-Newark-Jersey City, NY-NJ-PA    PUT         NextSong
278129  New York-Newark-Jersey City, NY-NJ-PA    PUT         NextSong
278130  New York-Newark-Jersey City, NY-NJ-PA    PUT         NextSong
278131  New York-Newark-Jersey City, NY-NJ-PA    PUT         NextSong
278132  New York-Newark-Jersey City, NY-NJ-PA    PUT        Thumbs Up
278133  New York-Newark-Jersey City, NY-NJ-PA    PUT         NextSong
278134  New York-Newark-Jersey City, NY-NJ-PA    PUT         NextSong
```

```
278135  New York-Newark-Jersey City, NY-NJ-PA    PUT          NextSong
278136  New York-Newark-Jersey City, NY-NJ-PA    PUT        Add Friend
278137  New York-Newark-Jersey City, NY-NJ-PA    PUT          NextSong
278138  New York-Newark-Jersey City, NY-NJ-PA    PUT          NextSong
278139  New York-Newark-Jersey City, NY-NJ-PA    GET              Home
278140  New York-Newark-Jersey City, NY-NJ-PA    PUT          NextSong
278141  New York-Newark-Jersey City, NY-NJ-PA    PUT          NextSong
278142  New York-Newark-Jersey City, NY-NJ-PA    PUT          NextSong
278143  New York-Newark-Jersey City, NY-NJ-PA    PUT          NextSong
278144  New York-Newark-Jersey City, NY-NJ-PA    PUT          NextSong
278145  New York-Newark-Jersey City, NY-NJ-PA    PUT          NextSong
278146  New York-Newark-Jersey City, NY-NJ-PA    PUT          NextSong
278147  New York-Newark-Jersey City, NY-NJ-PA    GET              Home
278148  New York-Newark-Jersey City, NY-NJ-PA    PUT          NextSong
278149  New York-Newark-Jersey City, NY-NJ-PA    PUT          NextSong
278150  New York-Newark-Jersey City, NY-NJ-PA    PUT            Logout
278151  New York-Newark-Jersey City, NY-NJ-PA    GET              Home
278152  New York-Newark-Jersey City, NY-NJ-PA    GET             About
278153  New York-Newark-Jersey City, NY-NJ-PA    PUT          NextSong

         registration  sessionId  \
0       1538173362000         29
1       1538331630000          8
2       1538173362000         29
3       1538331630000          8
4       1538173362000         29
5       1538331630000          8
6       1538331630000          8
7       1538173362000         29
8       1538173362000         29
9       1538173362000         29
10      1538331630000          8
11      1538331630000          8
12      1538173362000         29
13      1538331630000          8
14      1538331630000          8
15      1538173362000         29
16      1538331630000          8
17      1537365219000        217
18      1538173362000         29
19      1538331630000          8
20      1537365219000        217
21      1538173362000         29
22      1532482662000         53
23      1538331630000          8
24      1537365219000        217
25      1538173362000         29
26      1532482662000         53
```

```
27      1538331630000          8
28      1537365219000        217
29      1538173362000         29
...               ...        ...
278124  1538336771000        500
278125  1538336771000        500
278126  1538336771000        500
278127  1538336771000        500
278128  1538336771000        500
278129  1538336771000        500
278130  1538336771000        500
278131  1538336771000        500
278132  1538336771000        500
278133  1538336771000        500
278134  1538336771000        500
278135  1538336771000        500
278136  1538336771000        500
278137  1538336771000        500
278138  1538336771000        500
278139  1538336771000        500
278140  1538336771000        500
278141  1538336771000        500
278142  1538336771000        500
278143  1538336771000        500
278144  1538336771000        500
278145  1538336771000        500
278146  1538336771000        500
278147  1538336771000        500
278148  1538336771000        500
278149  1538336771000        500
278150  1538336771000        500
278151  1538336771000        500
278152  1538336771000        500
278153  1538336771000        500

                                                        song  status  \
0                                                  Rockpools     200
1                                                     Canada     200
2                                          Time For Miracles     200
3                                Knocking On Forbidden Doors     200
4                              Harder Better Faster Stronger     200
5                                              Don't Leave Me     200
6                                                 Run Run Run     200
7                             Passengers (Old Album Version)     200
8                                                       None     200
9                                                 Fuck Kitty     200
10                                         Walk On The Water     200
11                                                      None     200
```

12	Jade	200
13	Gettin' Better	200
14	None	307
15	So-Called Friends	200
16	You've Got The Love	200
17	Citizens Of Tomorrow	200
18	Represent	200
19	Swisha	200
20	Carbon Y Ramas Secas	200
21	Here I Am	200
22	What I Mean	200
23	Sticky Sweet	200
24	Sorrow (1997 Digital Remaster)	200
25	Rebirthing (Album Version)	200
26	You'll Never Know (My Love) (Bovellian 07 Mix)	200
27	Smile	200
28	I Want You	200
29	Dog Days Are Over (Radio Edit)	200
...	...	...
278124	Endlessly	200
278125	Breathe	200
278126	None	200
278127	Lucky Day	200
278128	Jamaica Roots II(Agora E Sempre)	200
278129	Embody the invisible	200
278130	Mockingbird	200
278131	Feeling This	200
278132	None	307
278133	Geek In The Pink [Phil Tan Remix]	200
278134	Greater Heights	200
278135	Bleeding Heart	200
278136	None	307
278137	DJ Play A Love Song	200
278138	More Than A Woman	200
278139	None	200
278140	Letters From Madrid	200
278141	King Of The Stereo	200
278142	Shining Down [feat. Matthew Santos] (Amended A...	200
278143	Sehr kosmisch	200
278144	Tops	200
278145	Tu no tienes alma	200
278146	Bala Bay Inn	200
278147	None	200
278148	You're Not Alone	200
278149	Murders In The Rue Morgue (1998 Digital Remaster)	200
278150	None	307
278151	None	200
278152	None	200

```
278153                             The Sun On His Back     200

                   ts                                           userAgent  \
0       1538352117000  Mozilla/5.0 (Windows NT 6.1; WOW64; rv:31.0) G...
1       1538352180000  "Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebK...
2       1538352394000  Mozilla/5.0 (Windows NT 6.1; WOW64; rv:31.0) G...
3       1538352416000  "Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebK...
4       1538352676000  Mozilla/5.0 (Windows NT 6.1; WOW64; rv:31.0) G...
5       1538352678000  "Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebK...
6       1538352886000  "Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebK...
7       1538352899000  Mozilla/5.0 (Windows NT 6.1; WOW64; rv:31.0) G...
8       1538352905000  Mozilla/5.0 (Windows NT 6.1; WOW64; rv:31.0) G...
9       1538353084000  Mozilla/5.0 (Windows NT 6.1; WOW64; rv:31.0) G...
10      1538353146000  "Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebK...
11      1538353150000  "Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebK...
12      1538353218000  Mozilla/5.0 (Windows NT 6.1; WOW64; rv:31.0) G...
13      1538353375000  "Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebK...
14      1538353376000  "Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebK...
15      1538353441000  Mozilla/5.0 (Windows NT 6.1; WOW64; rv:31.0) G...
16      1538353576000  "Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebK...
17      1538353668000  "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_9_4...
18      1538353687000  Mozilla/5.0 (Windows NT 6.1; WOW64; rv:31.0) G...
19      1538353744000  "Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebK...
20      1538353834000  "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_9_4...
21      1538353909000  Mozilla/5.0 (Windows NT 6.1; WOW64; rv:31.0) G...
22      1538353930000  Mozilla/5.0 (Windows NT 6.1; WOW64; rv:32.0) G...
23      1538353973000  "Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebK...
24      1538354117000  "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_9_4...
25      1538354132000  Mozilla/5.0 (Windows NT 6.1; WOW64; rv:31.0) G...
26      1538354180000  Mozilla/5.0 (Windows NT 6.1; WOW64; rv:32.0) G...
27      1538354204000  "Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebK...
28      1538354291000  "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_9_4...
29      1538354365000  Mozilla/5.0 (Windows NT 6.1; WOW64; rv:31.0) G...
...               ...                                                 ...
278124  1543616883000  Mozilla/5.0 (compatible; MSIE 9.0; Windows NT ...
278125  1543617111000  Mozilla/5.0 (compatible; MSIE 9.0; Windows NT ...
278126  1543617317000  Mozilla/5.0 (compatible; MSIE 9.0; Windows NT ...
278127  1543617391000  Mozilla/5.0 (compatible; MSIE 9.0; Windows NT ...
278128  1543617592000  Mozilla/5.0 (compatible; MSIE 9.0; Windows NT ...
278129  1543617802000  Mozilla/5.0 (compatible; MSIE 9.0; Windows NT ...
278130  1543618019000  Mozilla/5.0 (compatible; MSIE 9.0; Windows NT ...
278131  1543618269000  Mozilla/5.0 (compatible; MSIE 9.0; Windows NT ...
278132  1543618270000  Mozilla/5.0 (compatible; MSIE 9.0; Windows NT ...
278133  1543618442000  Mozilla/5.0 (compatible; MSIE 9.0; Windows NT ...
278134  1543618660000  Mozilla/5.0 (compatible; MSIE 9.0; Windows NT ...
278135  1543618840000  Mozilla/5.0 (compatible; MSIE 9.0; Windows NT ...
278136  1543618841000  Mozilla/5.0 (compatible; MSIE 9.0; Windows NT ...
278137  1543619220000  Mozilla/5.0 (compatible; MSIE 9.0; Windows NT ...
```

```
278138  1543619478000  Mozilla/5.0 (compatible; MSIE 9.0; Windows NT ...
278139  1543619556000  Mozilla/5.0 (compatible; MSIE 9.0; Windows NT ...
278140  1543619706000  Mozilla/5.0 (compatible; MSIE 9.0; Windows NT ...
278141  1543619821000  Mozilla/5.0 (compatible; MSIE 9.0; Windows NT ...
278142  1543620043000  Mozilla/5.0 (compatible; MSIE 9.0; Windows NT ...
278143  1543620316000  Mozilla/5.0 (compatible; MSIE 9.0; Windows NT ...
278144  1543620971000  Mozilla/5.0 (compatible; MSIE 9.0; Windows NT ...
278145  1543621196000  Mozilla/5.0 (compatible; MSIE 9.0; Windows NT ...
278146  1543621437000  Mozilla/5.0 (compatible; MSIE 9.0; Windows NT ...
278147  1543621485000  Mozilla/5.0 (compatible; MSIE 9.0; Windows NT ...
278148  1543621857000  Mozilla/5.0 (compatible; MSIE 9.0; Windows NT ...
278149  1543622121000  Mozilla/5.0 (compatible; MSIE 9.0; Windows NT ...
278150  1543622122000  Mozilla/5.0 (compatible; MSIE 9.0; Windows NT ...
278151  1543622248000  Mozilla/5.0 (compatible; MSIE 9.0; Windows NT ...
278152  1543622398000  Mozilla/5.0 (compatible; MSIE 9.0; Windows NT ...
278153  1543622411000  Mozilla/5.0 (compatible; MSIE 9.0; Windows NT ...

        userId
0           30
1            9
2           30
3            9
4           30
5            9
6            9
7           30
8           30
9           30
10           9
11           9
12          30
13           9
14           9
15          30
16           9
17          74
18          30
19           9
20          74
21          30
22          54
23           9
24          74
25          30
26          54
27           9
28          74
29          30
```

```
...        ...
278124  300011
278125  300011
278126  300011
278127  300011
278128  300011
278129  300011
278130  300011
278131  300011
278132  300011
278133  300011
278134  300011
278135  300011
278136  300011
278137  300011
278138  300011
278139  300011
278140  300011
278141  300011
278142  300011
278143  300011
278144  300011
278145  300011
278146  300011
278147  300011
278148  300011
278149  300011
278150  300011
278151  300011
278152  300011
278153  300011

[278154 rows x 18 columns]
```

2.1 Exploratory Data Analysis

2.1.1 Define Churn

A column `Churn` will be created to use as the label for our model. `Cancellation Confirmation` events is used to define churn, which happen for both paid and free users. We will assign a 1 where a user has churned and a 0 where they have not churned.

2.1.2 Explore Data

Exploratory data analysis will be performed to observe the behavior for users who stayed vs users who churned. Starting by exploring aggregates on these two groups of users, observing how much of a specific action they experienced per a certain time unit or number of songs played.

2.1.3 Identify users who have churned

First, we will identify the users who have churned using the `Cancellation Confirmation` event under the `page` column.

```
In [16]: # check Cancellation Confirmation page
         df.select("page").dropDuplicates().show()
```

```
+--------------------+
|                page|
+--------------------+
|              Cancel|
|    Submit Downgrade|
|         Thumbs Down|
|                Home|
|           Downgrade|
|         Roll Advert|
|              Logout|
|       Save Settings|
|Cancellation Conf...|
|               About|
|            Settings|
|     Add to Playlist|
|          Add Friend|
|            NextSong|
|           Thumbs Up|
|                Help|
|             Upgrade|
|               Error|
|      Submit Upgrade|
+--------------------+
```

From the above we can see that Cancellation Confirmation is the page that a user is taken to once they have confirmed that they would like to cancel their service. Again, this is how we are identifying churn.

```
In [17]: # number of users who churned
         df.select(["userId", "page"]).where(df.page == "Cancellation Confirmation").count()
```

```
Out[17]: 52
```

From the above cell we can see that there are 52 users in our dataset that have churned. We can take a closer look at the userids that churned.

```
In [18]: df.select(["userId", "page"]).where(df.page == "Cancellation Confirmation").show()
```

```
+------+--------------------+
|userId|                page|
```

```
+------+--------------------+
|    18|Cancellation Conf...|
|    32|Cancellation Conf...|
|   125|Cancellation Conf...|
|   105|Cancellation Conf...|
|    17|Cancellation Conf...|
|   143|Cancellation Conf...|
|   101|Cancellation Conf...|
|   129|Cancellation Conf...|
|   121|Cancellation Conf...|
|    51|Cancellation Conf...|
|    87|Cancellation Conf...|
|   122|Cancellation Conf...|
|    12|Cancellation Conf...|
|    58|Cancellation Conf...|
|    73|Cancellation Conf...|
|     3|Cancellation Conf...|
|   106|Cancellation Conf...|
|   103|Cancellation Conf...|
|    28|Cancellation Conf...|
|    54|Cancellation Conf...|
+------+--------------------+
only showing top 20 rows
```

We will now create the flag for churned users who will be assigned a 1 if churned and a 0 where they have not churned. This flag will be added to the dataset as a column named "churn".

```
In [19]: # flag the records where Cancellation Confirmation page is reached - 1 if it is and 0 i
         churn_event = udf(lambda x: 1 if x == "Cancellation Confirmation" else 0, IntegerType()
         #creating churn column
         df = df.withColumn("churn", churn_event("page"))
         df.head()
```

```
Out[19]: Row(artist='Martha Tilston', auth='Logged In', firstName='Colin', gender='M', itemInSes
```

From the above example we can see that the churn column has been successfully added to the dataframe and a 0 has been assigned for this particular userId. Now we can sort our records for a userId in reverse time order and add up the values in the churn column.

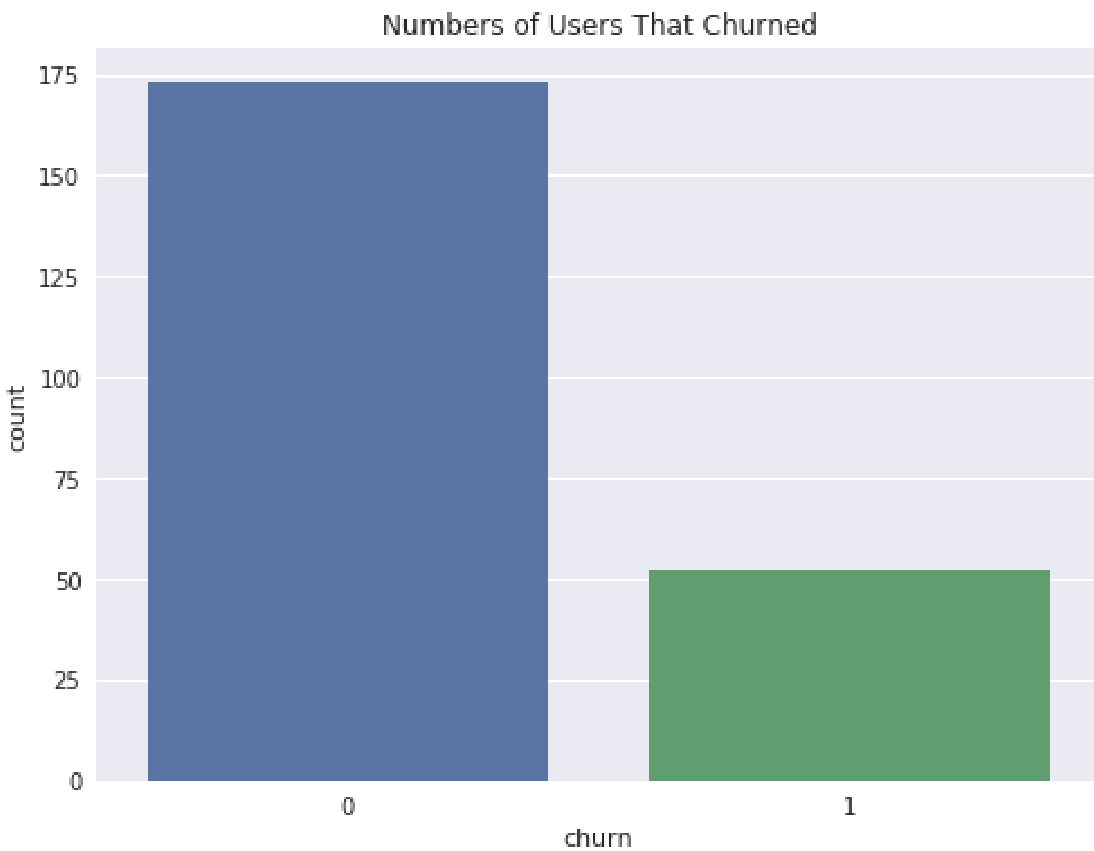
```
In [20]: # sort records for a user in reverse time order so we can add up vals in churn column
         windowval = Window.partitionBy("userId").orderBy(desc("ts")).rangeBetween(Window.unboun
         # create column churn which contains sum of churn 1s over records
         df = df.withColumn("churn", Fsum("churn").over(windowval))
         # groupby churn to get counts
         df_churn = df.select(['userId', 'churn']).dropDuplicates().groupBy('churn').count()
         df_churn.show()
```

```
+-----+-----+
|churn|count|
+-----+-----+
|    0|  173|
|    1|   52|
+-----+-----+
```

2.1.4 EDA for Users that Stayed vs Users that Churned

Now we can examine behaviour of those who churned vs those who did not churn. First we will visualise those who churned vs those who stayed.

In [21]:
```
# convert to pandas for visualisation
df_churn = df_churn.toPandas()
# plot the number of users that churned
plt.figure(figsize = [8,6])
ax = sns.barplot(data = df_churn, x = 'churn', y='count')
plt.title("Numbers of Users That Churned");
```



In [22]:
```
# calculate churn rate
52/(173+52) * 100
```

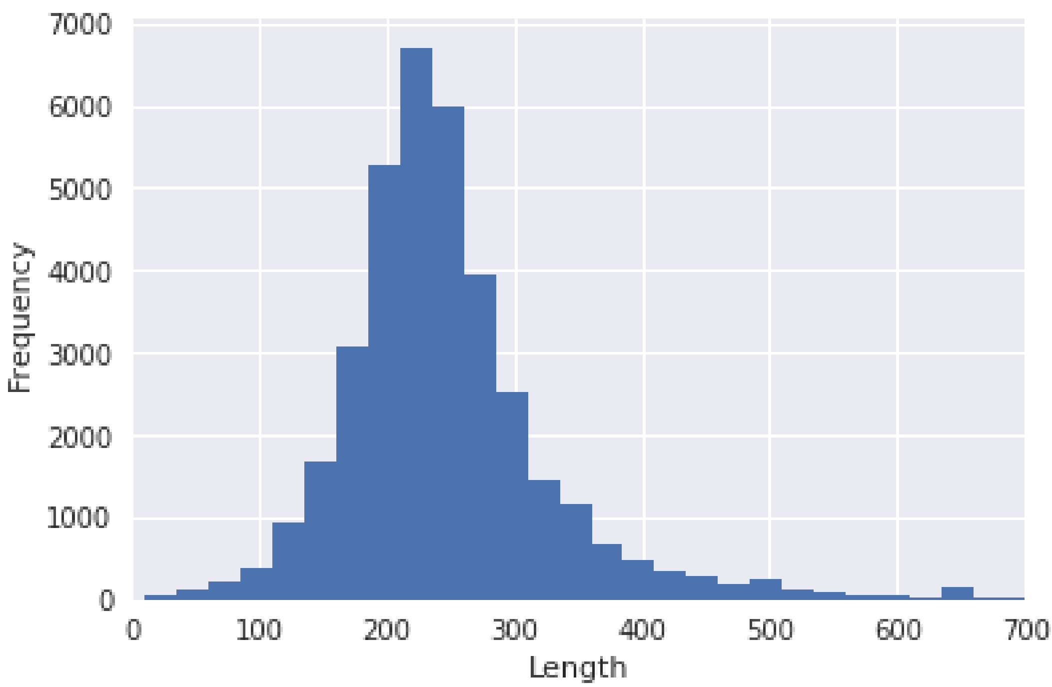
Out[22]:
```
23.11111111111111
```

From the above, we can see that 173 users stayed while 52 users churned. Therefore this means that 23% of our users churned. It is important to note moving forward that this is an imbalance.

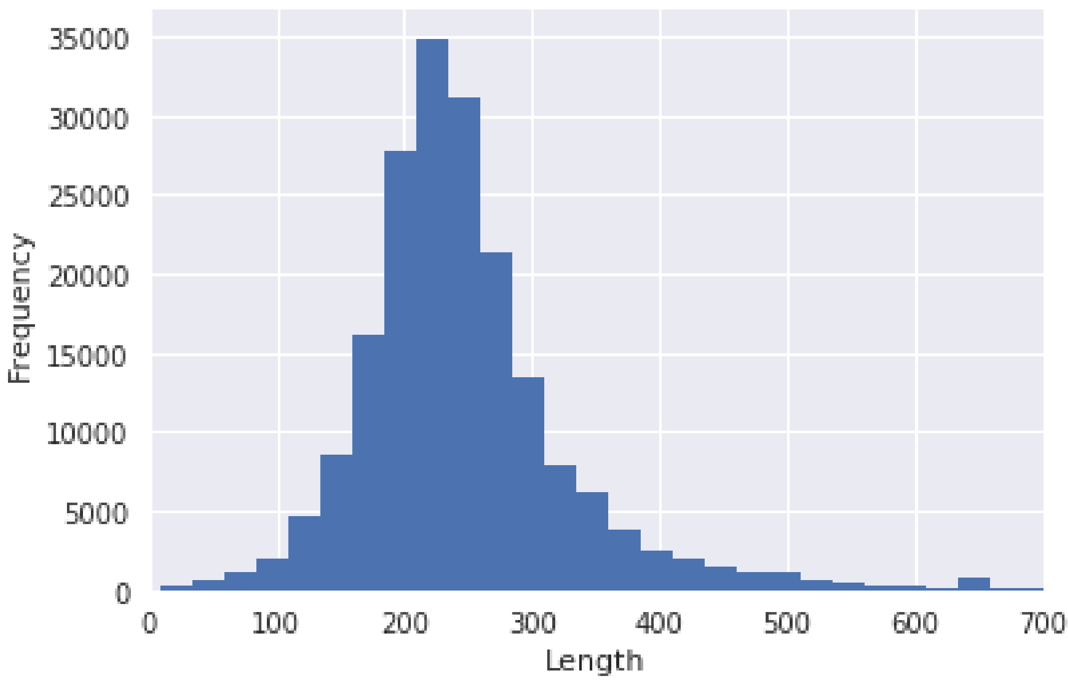
2.1.5 Length of time: Users that Churned vs. Users that Stayed

We can now look at the length distribution for customers who stayed and those which churned.

In [23]:
```
# get those customers who churned
df_len = df.filter(df.churn ==1)
# convert to pandas
df_pd = df_len.toPandas()
# drop the nulls
df_pd.length.dropna(inplace=True)
# plot the distribution
bin_edges = np.arange (10, df_pd['length'].max()+25, 25)
plt.hist(data = df_pd, x = 'length', bins = bin_edges)
plt.xlim(0,700)
plt.xlabel('Length')
plt.ylabel('Frequency');
```



```
In [24]: # users who stayed
         df_len_stay = df.filter(df.churn ==0)
         # convert to pandas
         df_pd = df_len_stay.toPandas()
         # drop nulls
         df_pd.length.dropna(inplace=True)
         # plot distribution
         bin_edges = np.arange (10, df_pd['length'].max()+25, 25)
         plt.hist(data = df_pd, x = 'length', bins = bin_edges)
         plt.xlim(0,700)
         plt.xlabel('Length')
         plt.ylabel('Frequency');
```



We can see from the above plots that length distribution is very similar for users that churned and those who stayed. This won't be very useful for predicting customer churn. Let's try a categorical feature: gender.

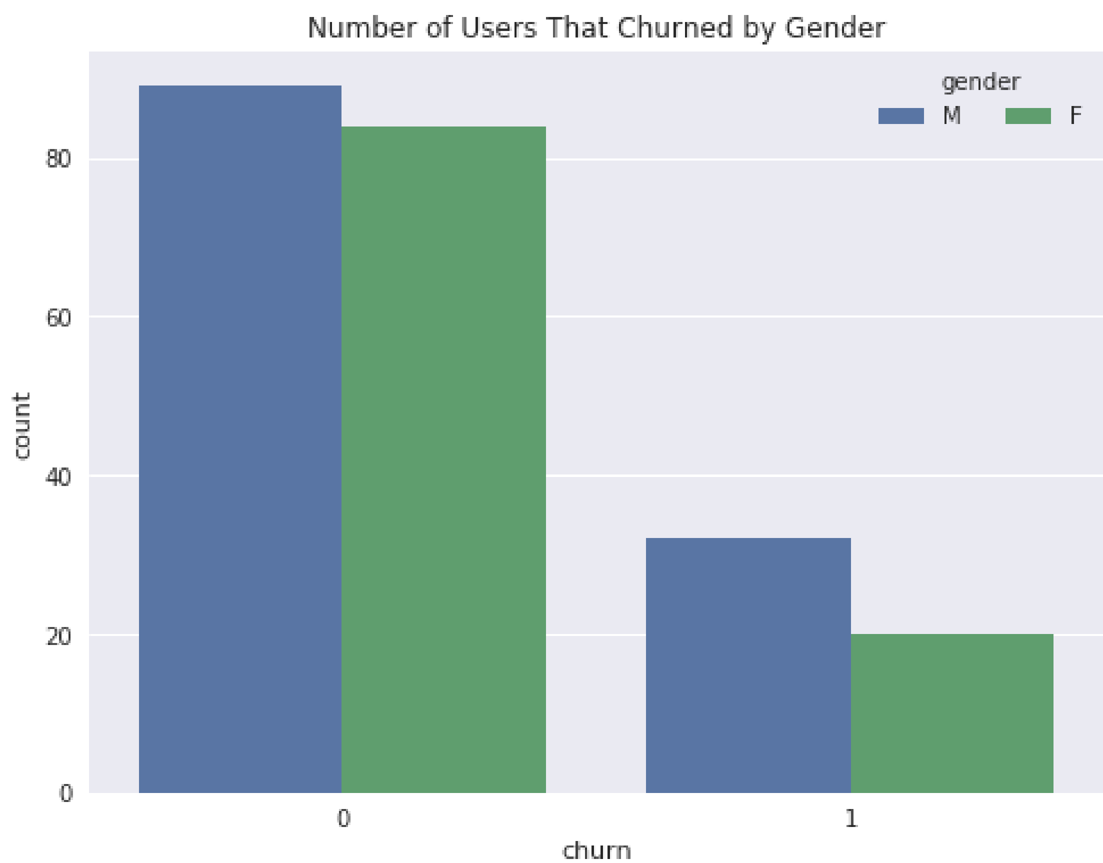
2.1.6 Gender - Users who Churned vs Users who Stayed

Now we can examine if gender had an effect on users that churned vs. those that stayed.

```
In [25]: # create gender df grouped by churn and gender
         df_gender = df.select(['userId', 'churn', 'gender']).dropDuplicates().groupBy('gender',
         # show gender df
         df_gender.show()
```

```
+------+-----+-----+
|gender|churn|count|
+------+-----+-----+
|     F|    0|   84|
|     F|    1|   20|
|     M|    0|   89|
|     M|    1|   32|
+------+-----+-----+
```

In [26]:
```python
# convert to pandas for visualisation
df_gender = df_gender.toPandas()
# order for the visualisation
df_gender = df_gender.sort_values('count', ascending = False)
# seaborn barplot
plt.figure(figsize = [8,6])
ax = sns.barplot(data = df_gender, x = 'churn', y='count', hue = 'gender')
ax.legend(loc = 1, ncol = 2, framealpha =1, title = 'gender')
plt.title("Number of Users That Churned by Gender");
```



In [27]:
```
# male churn rate
32/(89+32)
```

Out[27]: 0.2644628099173554

In [28]:
```
# female churn rate
20/(20+84)
```

Out[28]: 0.19230769230769232

From the above chart, we can see that more male users churned(rate of 0.264) compared to female users (rate of 0.192).

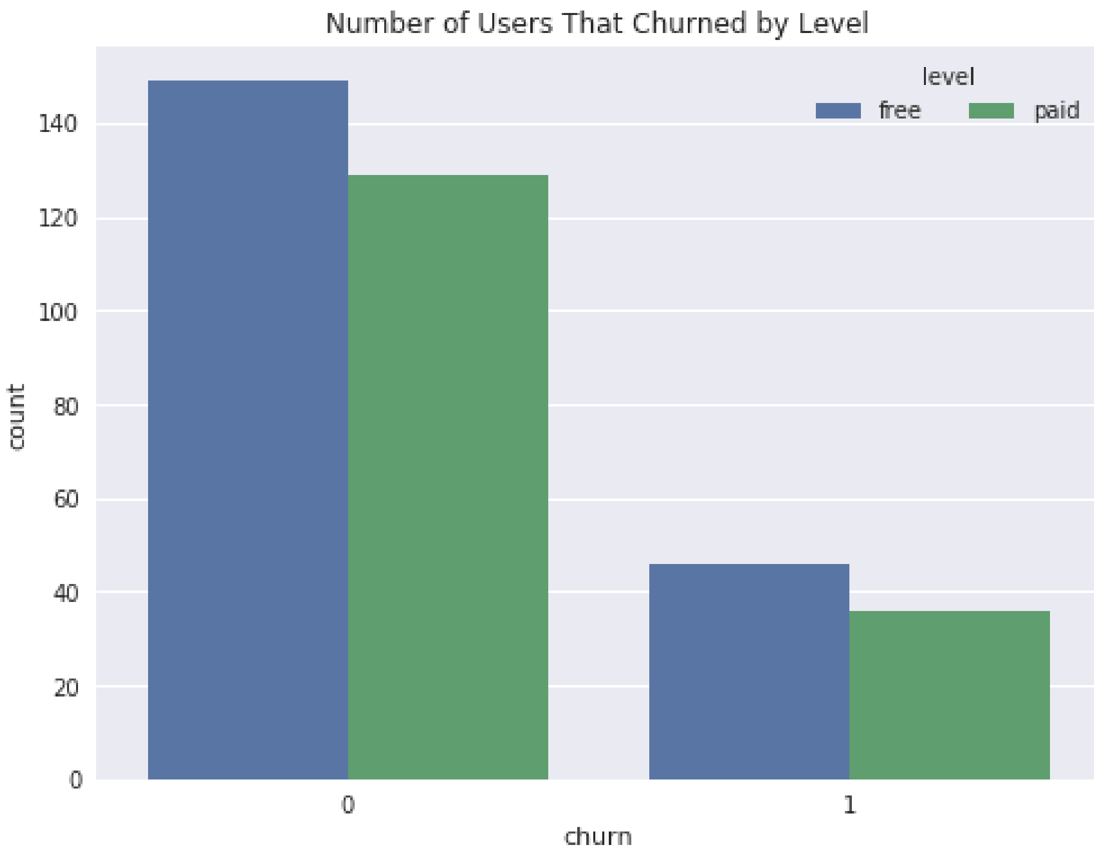
2.1.7 Users who Churned vs Stayed by Level

Next we can examine if level has an effect on whether a user will churn or not. By level here we mean if the user paid for the app or if they used it for free with ads.

In [29]:
```
# create the level dataframe
df_level = df.select(['userId', 'churn', 'level']).dropDuplicates().groupBy('level','ch
df_level.show()
```

```
+-----+-----+-----+
|level|churn|count|
+-----+-----+-----+
| free|    0|  149|
| paid|    0|  129|
| free|    1|   46|
| paid|    1|   36|
+-----+-----+-----+
```

In [30]:
```
# convert to pandas for visualisation
df_level = df_level.toPandas()
# plot the barplot using seaborn
plt.figure(figsize = [8,6])
ax = sns.barplot(data = df_level, x = 'churn', y='count', hue = 'level')
ax.legend(loc = 1, ncol = 2, framealpha =1, title = 'level')
plt.title("Number of Users That Churned by Level");
```




```
In [31]: # free churn rate
         46/(46+149)

Out[31]: 0.2358974358974359

In [32]: # paid churn rate
         36/(129+36)

Out[32]: 0.21818181818181817
```

We can see from the above chart that more users who used the service for free were slightly more likely to churn (rate of 0.236) compared to those who paid for the app (0.218).

2.1.8 Pages Visited by Those that Churned vs. Those That Stayed

Next we can examine if there were different pages visited by users that churned compared to those that remained.

```
In [33]: df_page = df.select(['userId', 'churn', 'page']).groupBy('page','churn').count()
         df_page.show(40)
```

```
+--------------------+-----+------+
|                page|churn| count|
+--------------------+-----+------+
|            Settings|    0|  1244|
|         Thumbs Down|    1|   496|
|           Thumbs Up|    1|  1859|
|     Add to Playlist|    1|  1038|
|               Error|    1|    32|
|               About|    1|    56|
|         Thumbs Down|    0|  2050|
|         Roll Advert|    1|   967|
|                Home|    0|  8410|
|Cancellation Conf...|    1|    52|
|               Error|    0|   220|
|              Cancel|    1|    52|
|            Settings|    1|   270|
|          Add Friend|    1|   636|
|             Upgrade|    0|   387|
|           Downgrade|    1|   337|
|              Logout|    1|   553|
|    Submit Downgrade|    1|     9|
|       Save Settings|    0|   252|
|           Thumbs Up|    0| 10692|
|           Downgrade|    0|  1718|
|      Submit Upgrade|    0|   127|
|         Roll Advert|    0|  2966|
|    Submit Downgrade|    0|    54|
|              Logout|    0|  2673|
|                Home|    1|  1672|
|          Add Friend|    0|  3641|
|             Upgrade|    1|   112|
|      Submit Upgrade|    1|    32|
|               About|    0|   439|
|     Add to Playlist|    0|  5488|
|       Save Settings|    1|    58|
|                Help|    1|   239|
|            NextSong|    1| 36394|
|            NextSong|    0|191714|
|                Help|    0|  1215|
+--------------------+-----+------+
```

In [34]:
```
# convert to pandas
df_page = df_page.toPandas()
# create counts for those who churned and those who stayed
churn_count =  df_page[df_page['churn'] == 1].sum()
stay_count =  df_page[df_page['churn'] == 0].sum()
```

Now that we have a count of the number of customers who churned and those that stayed we can calculate the rate and create this as a column on our dataFrame.

```
In [35]: # calculate the rate of pages visited by those who churned vs. those who stayed
         df_page['rate'] = np.where(
             df_page['churn'] == 0, df_page['count']/stay_count['count'], np.where(
             df_page['churn'] == 1, df_page['count']/churn_count['count'],df_page['count']/churn
         df_page.head()
```

Out[35]:

	page	churn	count	rate
0	Settings	0	1244	0.005332
1	Thumbs Down	1	496	0.011056
2	Thumbs Up	1	1859	0.041436
3	Add to Playlist	1	1038	0.023137
4	Error	1	32	0.000713

```
In [36]: # plot the pages by churn
         plt.figure(figsize=[20,16])
         sns.barplot(data = df_page, x = 'rate', y = 'page', hue = 'churn')
         plt.title('Rate of Pages Navigated to by Users that Churned vs. Users that Stayed');
```

From the above chart, we can see that the most popular action for both users that stayed and those that churned was to skip to the next song. We can also see that churned users rolled the ad and thumbs down songs more. Those who were more likely to stay performed more thumbs up actions, added friends and also added songs to playlist.

2.1.9 Calculating Songs per Hour

We can now turn our attention to calculating the number of songs listened to by churn and non churn users per hour.

```
In [37]: # get hour from the timestamp
         get_hour = udf(lambda x: datetime.datetime.fromtimestamp(x / 1000.0). hour)
         # create hour column
         df = df.withColumn("hour", get_hour(df.ts))
         df.head()
```

```
Out[37]: Row(artist=None, auth='Logged In', firstName='Darianna', gender='F', itemInSession=34,
```

First we can look at those who didn't churn.

```
In [38]: # create a df with those who didnt churn and which counts when user goes to next song p
         songs_in_hour_stay = df.filter((df.page == "NextSong") & (df.churn == 0)).groupby(df.ho
         songs_in_hour_stay.show(20)
```

```
+----+-----+
|hour|count|
+----+-----+
|   0| 7527|
|   1| 7035|
|   2| 7014|
|   3| 7063|
|   4| 6914|
|   5| 6960|
|   6| 6836|
|   7| 6873|
|   8| 7023|
|   9| 7268|
|  10| 7502|
|  11| 7440|
|  12| 7918|
|  13| 8073|
|  14| 8792|
|  15| 9462|
|  16| 9721|
|  17| 9464|
|  18| 9146|
|  19| 9112|
+----+-----+
only showing top 20 rows
```

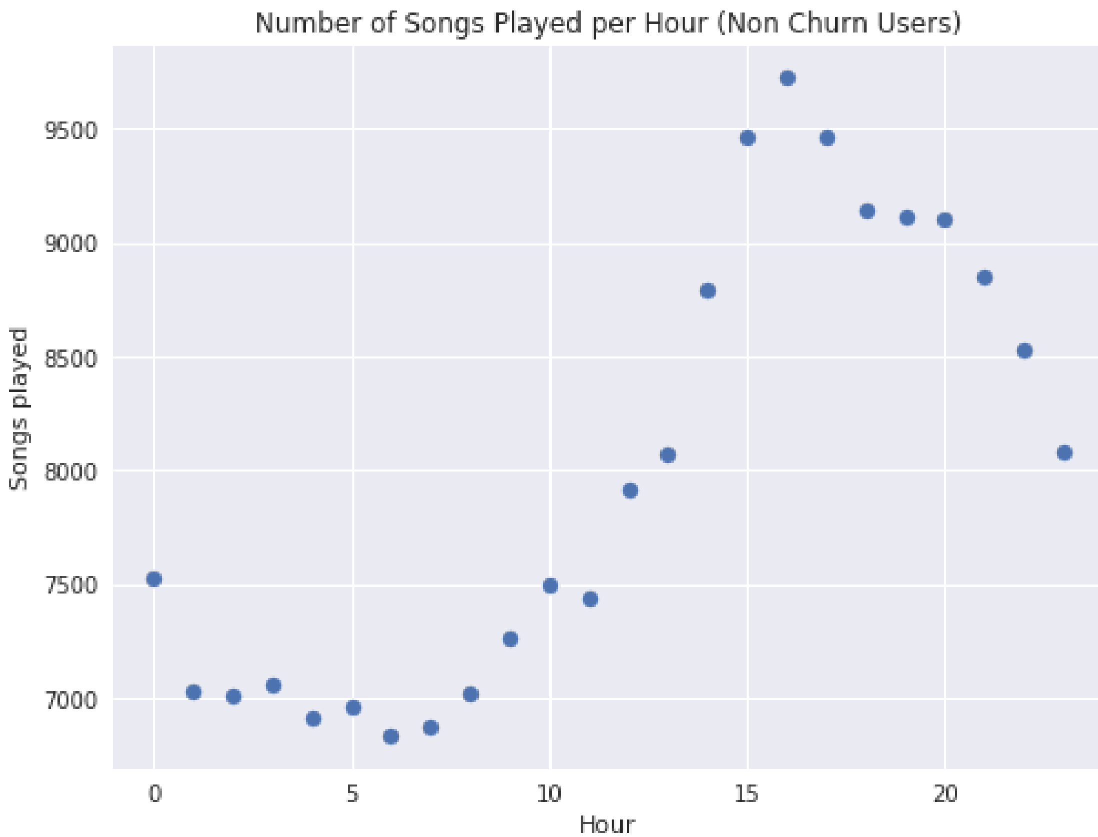
In [39]:
```python
# convert to pandas and then to numeric
songs_in_hour_stay_pd = songs_in_hour_stay.toPandas()
songs_in_hour_stay_pd.hour = pd.to_numeric(songs_in_hour_stay_pd.hour)
```

In [40]:
```python
songs_in_hour_stay_pd
```

Out[40]:

	hour	count
0	0	7527
1	1	7035
2	2	7014
3	3	7063
4	4	6914
5	5	6960
6	6	6836
7	7	6873
8	8	7023
9	9	7268
10	10	7502
11	11	7440
12	12	7918
13	13	8073
14	14	8792
15	15	9462
16	16	9721
17	17	9464
18	18	9146
19	19	9112
20	20	9107
21	21	8853
22	22	8526
23	23	8085

In [41]:
```python
#plot the distribution
plt.figure(figsize = [8,6])
plt.scatter(songs_in_hour_stay_pd["hour"], songs_in_hour_stay_pd["count"])
plt.xlim(-1, 24)
plt.xlabel("Hour")
plt.ylabel("Songs played")
plt.title("Number of Songs Played per Hour (Non Churn Users)");
```

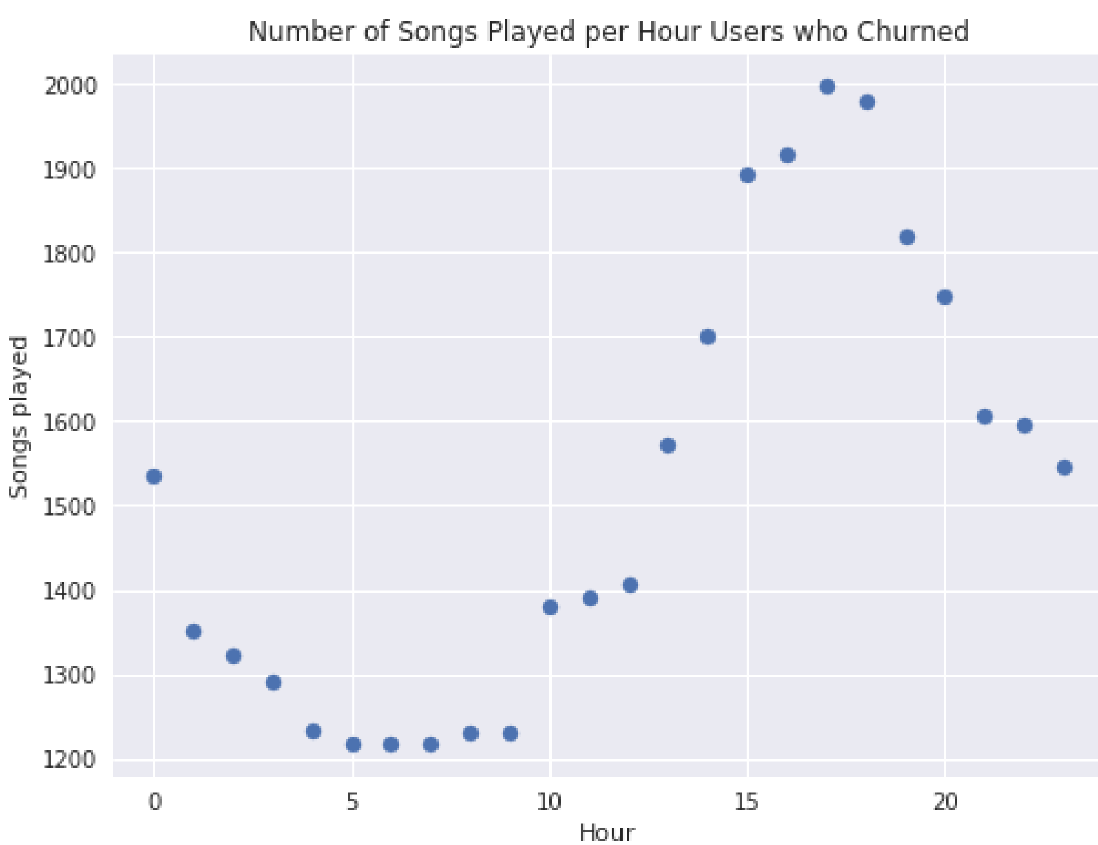


From above we can see that there is a peak of songs played between 3pm and 8pm. Next we will examine users who churned by using the same process.

```
In [42]: # dataframe with customers who churned and count next song page
         songs_in_hour_churned = df.filter((df.page == "NextSong") & (df.churn == 1)).groupby(df

In [43]: songs_in_hour_churned.show()

+----+-----+
|hour|count|
+----+-----+
|   0| 1535|
|   1| 1353|
|   2| 1322|
|   3| 1292|
|   4| 1233|
|   5| 1218|
|   6| 1218|
|   7| 1218|
|   8| 1230|
|   9| 1230|
|  10| 1380|
```

```
|  11| 1390|
|  12| 1408|
|  13| 1571|
|  14| 1702|
|  15| 1892|
|  16| 1915|
|  17| 1996|
|  18| 1978|
|  19| 1818|
+----+-----+
only showing top 20 rows
```

In [44]:
```python
# convert to pandas and to numeric
songs_in_hour_churned = songs_in_hour_churned.toPandas()
songs_in_hour_churned.hour = pd.to_numeric(songs_in_hour_churned.hour)
#plot distribution of songs per hour for churned
plt.figure(figsize = [8,6])
plt.scatter(songs_in_hour_churned["hour"], songs_in_hour_churned["count"])
plt.xlim(-1, 24)
plt.xlabel("Hour")
plt.ylabel("Songs played")
plt.title("Number of Songs Played per Hour Users who Churned");
```



We can see users that churned had a similar distribution, however they listened to fewer songs per hour than users that stayed.

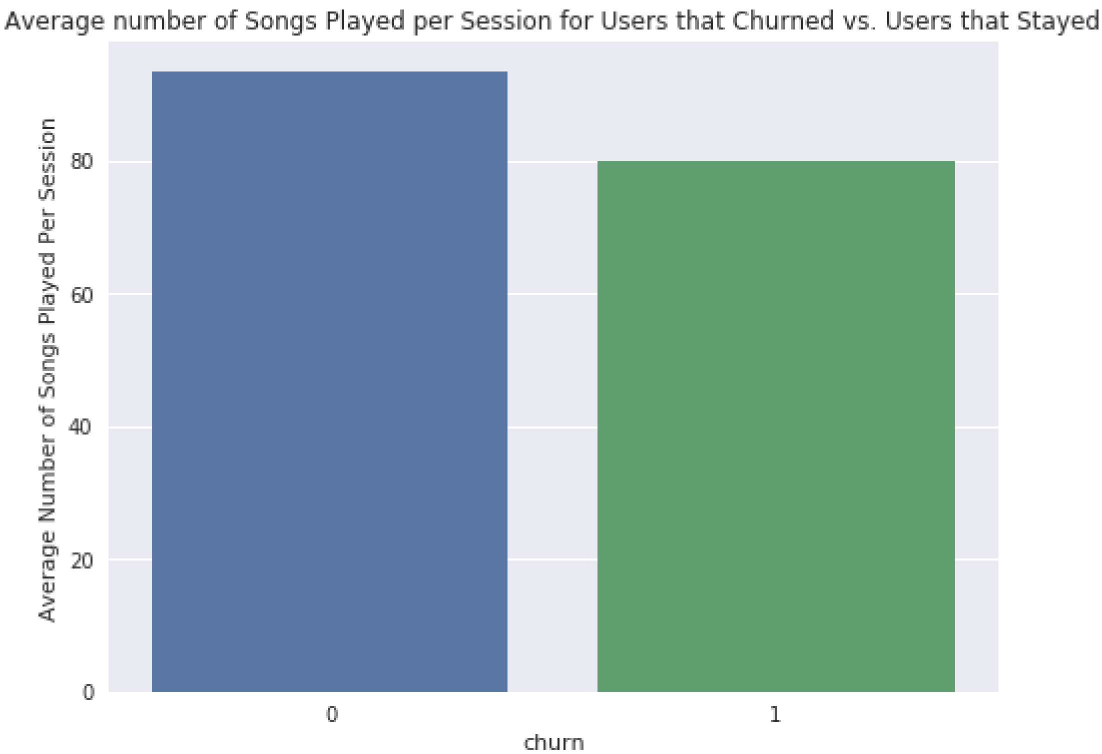
2.1.10 Songs Per Session for Users who Churned vs. Those who Stayed

We can plot this in a simple way which will allow us to compare those who churned and those who stayed in a bar chart by getting the averages for both groups.

```
In [45]: df_songs = df.filter(df.page == "NextSong").dropDuplicates().groupBy('sessionId','churn
         # get average grouped by churn
         df_songs.groupby('churn').agg({"count":"avg"}).show()
```

```
+-----+-----------------+
|churn|       avg(count)|
+-----+-----------------+
|    0| 93.3369036027264|
|    1|79.81140350877193|
+-----+-----------------+
```

```
In [46]: df_songs = df_songs.groupby('churn').agg({"count":"avg"})
         # convert this to pandas df
         df_songs = df_songs.toPandas()
         #plot
         plt.figure(figsize = [8,6])
         ax = sns.barplot(data = df_songs, x = 'churn', y='avg(count)')
         plt.title("Average number of Songs Played per Session for Users that Churned vs. Users
         plt.ylabel("Average Number of Songs Played Per Session");
```



From the chart we can see that those churned from Sparkify actually listening to fewer songs on average per session.

2.1.11 Number of Unique Artists Listened to

We can create a similar chart for the number of artists that users listened to.

```
In [47]: df_artists = df.select("artist", "userId", "churn").dropDuplicates().groupby("userId",
         # get averages
         df_artists.groupby('churn').agg({"count":"avg"}).show()
```

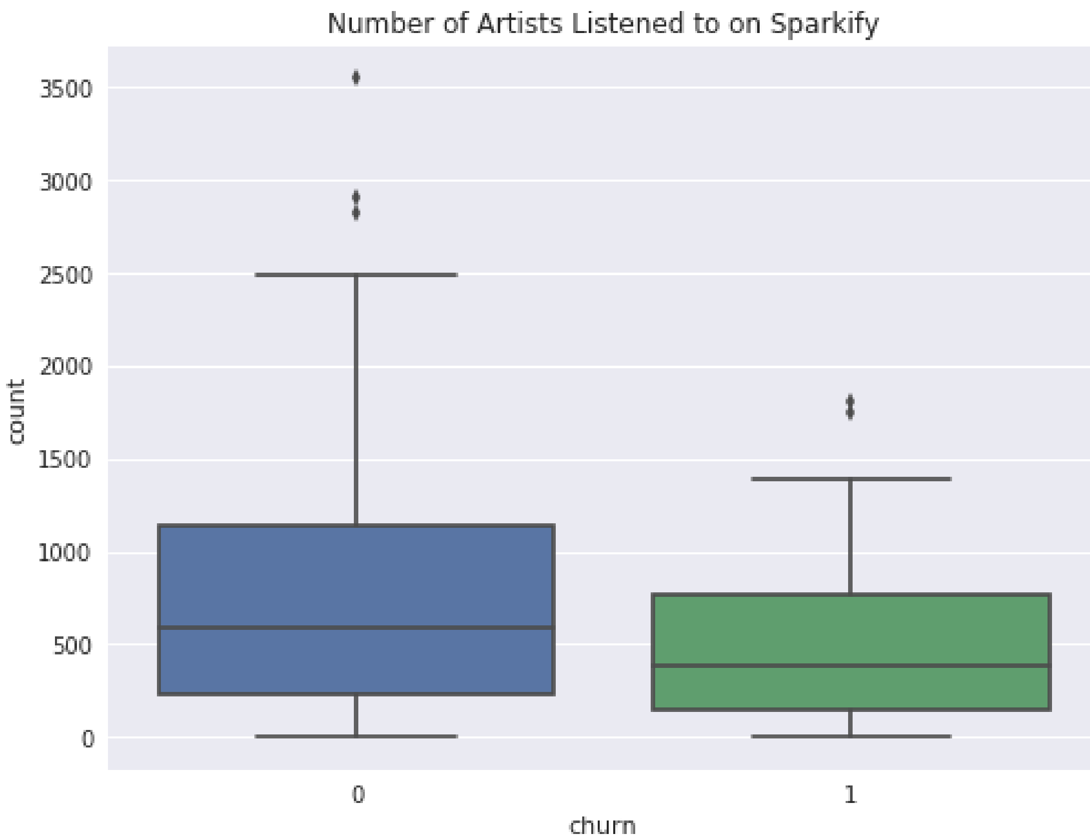
```
+-----+-----------------+
|churn|       avg(count)|
+-----+-----------------+
|    0|750.7803468208092|
|    1|519.6923076923077|
+-----+-----------------+
```

We can plot this as a boxplot to see the max and medians for both groups.

```
In [48]: # convert to pandas
         df_artists = df_artists.toPandas()
```

```
#plot boxplot
plt.figure(figsize = [8,6])
ax = sns.boxplot(data = df_artists, x = 'churn', y='count')
plt.title("Number of Artists Listened to on Sparkify")
```

```
Out[48]: Text(0.5,1,'Number of Artists Listened to on Sparkify')
```



From the above we can see that those who didn't churn listened to a larger number of different artists compared to those who churned.

2.1.12 Location

We can now examine if location had an effect on churn.

```
In [49]: df.select("location", "userId", "churn").groupby("location").count().show()
```

```
+--------------------+-----+
|            location|count|
+--------------------+-----+
|     Gainesville, FL| 1229|
|Atlantic City-Ham...| 2176|
|Deltona-Daytona B...|   73|
```

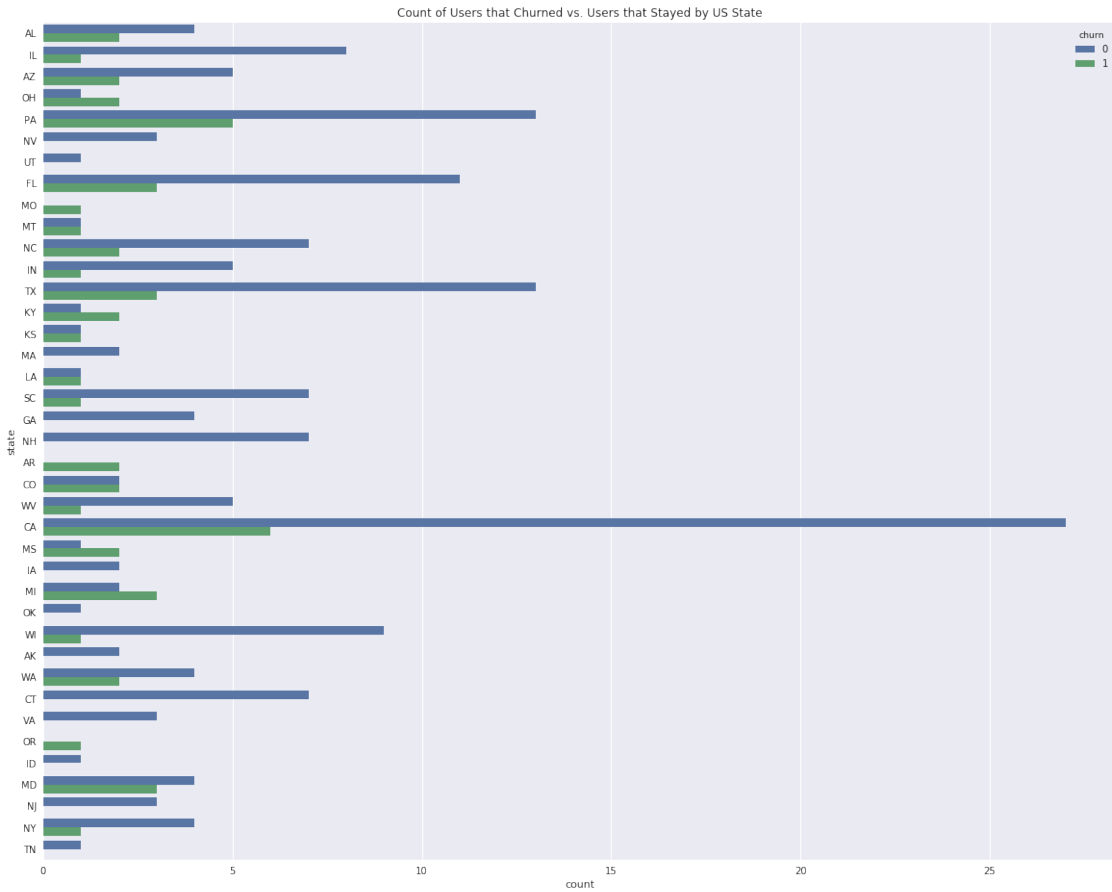
```
|San Diego-Carlsba...|  754|
|Cleveland-Elyria, OH| 1392|
|Kingsport-Bristol...| 1863|
|New Haven-Milford...| 4007|
|Birmingham-Hoover...|   75|
|  Corpus Christi, TX|   11|
|         Dubuque, IA|  651|
|Las Vegas-Henders...| 2042|
|Indianapolis-Carm...|  970|
|Seattle-Tacoma-Be...|  246|
|          Albany, OR|   23|
|   Winston-Salem, NC|  819|
|     Bakersfield, CA| 1775|
|Los Angeles-Long ...|30131|
|Minneapolis-St. P...| 2134|
|San Francisco-Oak...| 2647|
|Phoenix-Mesa-Scot...| 4846|
+--------------------+-----+
only showing top 20 rows
```

Let's just extract the state from the location by taking the last two characters in the location string.

In [50]:
```
# get last two characters
get_state = udf(lambda x: x[-2:])
# create state column
df_state = df.withColumn("state", get_state(df.location))
# check that create state column worked
df_state.take(2)
```

Out[50]:
```
[Row(artist=None, auth='Logged In', firstName='Darianna', gender='F', itemInSession=34,
 Row(artist='Lily Allen', auth='Logged In', firstName='Darianna', gender='F', itemInSes
```

In [51]:
```
df_state = df_state.select("state", "userId", "churn").dropDuplicates().groupby("state"
# convert to pandas
df_state_pd = df_state.toPandas()
# plot
plt.figure(figsize=[20,16])
sns.barplot(data = df_state_pd, x = 'count', y = 'state', hue = 'churn')
plt.title('Count of Users that Churned vs. Users that Stayed by US State');
```



Most users were based in CA. More users in MI, KY, and OH states churned than stayed. This may be difficult to engineer a useful feature for when it comes to modelling. Let's leave this for now and move onto another column from our dataset; operating systems and browsers.

2.1.13 UserAgent: Operating System and Browsers

Now we can extract the Operating System a user is on to understand if this has an effect on churn.

```
In [52]: df_opsys = df.select("userId", "userAgent", "churn").dropDuplicates(['userId'])
         df_opsys.show()
```

```
+------+--------------------+-----+
|userId|           userAgent|churn|
+------+--------------------+-----+
|100010|"Mozilla/5.0 (iPh...|    0|
|200002|"Mozilla/5.0 (iPh...|    0|
|   125|"Mozilla/5.0 (Mac...|    1|
|   124|"Mozilla/5.0 (Mac...|    0|
|    51|"Mozilla/5.0 (Win...|    1|
|     7|Mozilla/5.0 (Wind...|    0|
|    15|"Mozilla/5.0 (Win...|    0|
```

```
|    54|Mozilla/5.0 (Wind...|    1|
|   155|"Mozilla/5.0 (Win...|    0|
|100014|"Mozilla/5.0 (Win...|    1|
|   132|"Mozilla/5.0 (Mac...|    0|
|   154|"Mozilla/5.0 (Win...|    0|
|   101|Mozilla/5.0 (Wind...|    1|
|    11|Mozilla/5.0 (Wind...|    0|
|   138|"Mozilla/5.0 (iPa...|    0|
|300017|"Mozilla/5.0 (Mac...|    0|
|100021|"Mozilla/5.0 (Mac...|    1|
|    29|"Mozilla/5.0 (Mac...|    1|
|    69|"Mozilla/5.0 (Win...|    0|
|   112|Mozilla/5.0 (Wind...|    0|
+------+--------------------+-----+
only showing top 20 rows
```

In [53]:
```python
# convert to pandas
df_opsys = df_opsys.toPandas()
# get the possible list of operating systems
df_opsys.userAgent.value_counts()
```

Out[53]:
```
"Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/36.0
Mozilla/5.0 (Windows NT 6.1; WOW64; 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)
"Mozilla/5.0 (Macintosh; Intel Mac OS X 10_9_4) AppleWebKit/537.77.4 (KHTML, like Gecko
"Mozilla/5.0 (Macintosh; Intel Mac OS X 10_9_4) AppleWebKit/537.36 (KHTML, like Gecko)
"Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/36.0
"Mozilla/5.0 (Macintosh; Intel Mac OS X 10_9_4) AppleWebKit/537.78.2 (KHTML, like Gecko
Mozilla/5.0 (Macintosh; Intel Mac OS X 10.9; rv:31.0) Gecko/20100101 Firefox/31.0
"Mozilla/5.0 (iPhone; CPU iPhone OS 7_1_2 like Mac OS X) AppleWebKit/537.51.2 (KHTML, l
"Mozilla/5.0 (Macintosh; Intel Mac OS X 10_9_4) AppleWebKit/537.36 (KHTML, like Gecko)
"Mozilla/5.0 (Windows NT 6.3; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/36.0
Mozilla/5.0 (Windows NT 6.1; WOW64; Trident/7.0; rv:11.0) like Gecko
"Mozilla/5.0 (Windows NT 6.3; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/36.0
Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:31.0) Gecko/20100101 Firefox/31.0
"Mozilla/5.0 (iPhone; CPU iPhone OS 7_1 like Mac OS X) AppleWebKit/537.51.2 (KHTML, lik
"Mozilla/5.0 (Windows NT 5.1) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/36.0.1985.1
Mozilla/5.0 (Windows NT 6.3; WOW64; rv:31.0) Gecko/20100101 Firefox/31.0
"Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/36.0.198
"Mozilla/5.0 (Windows NT 6.1) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/36.0.1985.1
"Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/37.0
"Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/37.0
Mozilla/5.0 (Windows NT 6.1; rv:31.0) Gecko/20100101 Firefox/31.0
Mozilla/5.0 (compatible; MSIE 9.0; Windows NT 6.1; WOW64; Trident/5.0)
"Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/36.0.198
"Mozilla/5.0 (Windows NT 6.2; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/36.0
```

```
"Mozilla/5.0 (Macintosh; Intel Mac OS X 10_7_5) AppleWebKit/537.77.4 (KHTML, like Gecko
"Mozilla/5.0 (Macintosh; Intel Mac OS X 10_10) AppleWebKit/600.1.8 (KHTML, like Gecko)
"Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/35.0
"Mozilla/5.0 (Macintosh; Intel Mac OS X 10_9_4) AppleWebKit/537.36 (KHTML, like Gecko)
"Mozilla/5.0 (Macintosh; Intel Mac OS X 10_9_2) AppleWebKit/537.36 (KHTML, like Gecko)
"Mozilla/5.0 (iPad; CPU OS 7_1_2 like Mac OS X) AppleWebKit/537.51.2 (KHTML, like Gecko
Mozilla/5.0 (Macintosh; Intel Mac OS X 10.7; rv:31.0) Gecko/20100101 Firefox/31.0
Mozilla/5.0 (Windows NT 6.1; WOW64; rv:32.0) Gecko/20100101 Firefox/32.0
Mozilla/5.0 (Windows NT 6.2; WOW64; rv:31.0) Gecko/20100101 Firefox/31.0
"Mozilla/5.0 (Macintosh; Intel Mac OS X 10_9_2) AppleWebKit/537.75.14 (KHTML, like Geck
Mozilla/5.0 (compatible; MSIE 9.0; Windows NT 6.1; Trident/5.0)
Mozilla/5.0 (compatible; MSIE 10.0; Windows NT 6.1; WOW64; Trident/6.0)
"Mozilla/5.0 (Macintosh; Intel Mac OS X 10_8_5) AppleWebKit/537.36 (KHTML, like Gecko)
Mozilla/5.0 (X11; Ubuntu; Linux i686; rv:31.0) Gecko/20100101 Firefox/31.0
Mozilla/5.0 (X11; Linux x86_64; rv:31.0) Gecko/20100101 Firefox/31.0
"Mozilla/5.0 (Macintosh; Intel Mac OS X 10_10) AppleWebKit/600.1.3 (KHTML, like Gecko)
Mozilla/5.0 (Macintosh; Intel Mac OS X 10.6; rv:31.0) Gecko/20100101 Firefox/31.0
"Mozilla/5.0 (Windows NT 5.1) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/36.0.1985.1
Mozilla/5.0 (Macintosh; Intel Mac OS X 10.8; rv:31.0) Gecko/20100101 Firefox/31.0
"Mozilla/5.0 (Macintosh; Intel Mac OS X 10_7_5) AppleWebKit/537.36 (KHTML, like Gecko)
"Mozilla/5.0 (Macintosh; Intel Mac OS X 10_9_2) AppleWebKit/537.74.9 (KHTML, like Gecko
"Mozilla/5.0 (Macintosh; Intel Mac OS X 10_8_5) AppleWebKit/537.36 (KHTML, like Gecko)
Mozilla/5.0 (Windows NT 6.1; WOW64; rv:24.0) Gecko/20100101 Firefox/24.0
Mozilla/5.0 (Windows NT 6.1; WOW64; rv:30.0) Gecko/20100101 Firefox/30.0
"Mozilla/5.0 (Windows NT 6.1) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/36.0.1985.1
"Mozilla/5.0 (iPad; CPU OS 7_1_1 like Mac OS X) AppleWebKit/537.51.2 (KHTML, like Gecko
"Mozilla/5.0 (Macintosh; Intel Mac OS X 10_6_8) AppleWebKit/537.36 (KHTML, like Gecko)
"Mozilla/5.0 (Macintosh; Intel Mac OS X 10_9_3) AppleWebKit/537.76.4 (KHTML, like Gecko
"Mozilla/5.0 (iPhone; CPU iPhone OS 7_1_1 like Mac OS X) AppleWebKit/537.51.2 (KHTML, l
"Mozilla/5.0 (Windows NT 6.2; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/36.0
Mozilla/5.0 (Windows NT 6.0; rv:31.0) Gecko/20100101 Firefox/31.0
Name: userAgent, dtype: int64
```

In [54]:
```python
# create list of operating systems
os_list = ["Windows", "Mac", "Linux", "iPhone", "iPad"]
# create os column and extract strings that match our os_list and add to column
df_opsys['os'] = df_opsys.userAgent.str.extract('(?i)({0})'.format('|'.join(os_list)))
# check that worked
df_opsys
```

Out[54]:

	userId	userAgent	churn	os
0	100010	"Mozilla/5.0 (iPhone; CPU iPhone OS 7_1_2 like...	0	iPhone
1	200002	"Mozilla/5.0 (iPhone; CPU iPhone OS 7_1 like M...	0	iPhone
2	125	"Mozilla/5.0 (Macintosh; Intel Mac OS X 10_9_4...	1	Mac
3	124	"Mozilla/5.0 (Macintosh; Intel Mac OS X 10_9_4...	0	Mac
4	51	"Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebK...	1	Windows
5	7	Mozilla/5.0 (Windows NT 6.1; rv:31.0) Gecko/20...	0	Windows
6	15	"Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebK...	0	Windows

7	54	Mozilla/5.0 (Windows NT 6.1; WOW64; rv:32.0) G...	1	Windows
8	155	"Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebK...	0	Windows
9	100014	"Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebK...	1	Windows
10	132	"Mozilla/5.0 (Macintosh; Intel Mac OS X 10_9_4...	0	Mac
11	154	"Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebK...	0	Windows
12	101	Mozilla/5.0 (Windows NT 6.2; WOW64; rv:31.0) G...	1	Windows
13	11	Mozilla/5.0 (Windows NT 6.1; WOW64; Trident/7...	0	Windows
14	138	"Mozilla/5.0 (iPad; CPU OS 7_1_1 like Mac OS X...	0	iPad
15	300017	"Mozilla/5.0 (Macintosh; Intel Mac OS X 10_10)...	0	Mac
16	100021	"Mozilla/5.0 (Macintosh; Intel Mac OS X 10_9_4...	1	Mac
17	29	"Mozilla/5.0 (Macintosh; Intel Mac OS X 10_9_4...	1	Mac
18	69	"Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebK...	0	Windows
19	112	Mozilla/5.0 (Windows NT 6.1; WOW64; rv:31.0) G...	0	Windows
20	42	"Mozilla/5.0 (Windows NT 6.1) AppleWebKit/537...	0	Windows
21	73	Mozilla/5.0 (Windows NT 6.1; WOW64; rv:31.0) G...	1	Windows
22	87	"Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebK...	1	Windows
23	200010	Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:31...	0	Linux
24	64	"Mozilla/5.0 (Macintosh; Intel Mac OS X 10_9_4...	0	Mac
25	3	"Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebK...	1	Windows
26	113	"Mozilla/5.0 (Macintosh; Intel Mac OS X 10_9_4...	0	Mac
27	30	Mozilla/5.0 (Windows NT 6.1; WOW64; rv:31.0) G...	0	Windows
28	34	"Mozilla/5.0 (Macintosh; Intel Mac OS X 10_9_4...	0	Mac
29	133	"Mozilla/5.0 (Windows NT 6.3; WOW64) AppleWebK...	0	Windows
..	...	...	...	...
195	83	"Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/5...	0	Linux
196	109	Mozilla/5.0 (Windows NT 6.1; WOW64; rv:31.0) G...	0	Windows
197	123	"Mozilla/5.0 (iPhone; CPU iPhone OS 7_1 like M...	0	iPhone
198	200022	"Mozilla/5.0 (Macintosh; Intel Mac OS X 10_7_5...	0	Mac
199	13	"Mozilla/5.0 (iPhone; CPU iPhone OS 7_1_2 like...	0	iPhone
200	200019	"Mozilla/5.0 (Macintosh; Intel Mac OS X 10_9_4...	0	Mac
201	14	Mozilla/5.0 (Macintosh; Intel Mac OS X 10.9; r...	0	Mac
202	21	Mozilla/5.0 (Windows NT 6.3; WOW64; rv:31.0) G...	0	Windows
203	66	Mozilla/5.0 (Windows NT 6.1; WOW64; rv:31.0) G...	0	Windows
204	91	Mozilla/5.0 (Windows NT 6.3; WOW64; rv:31.0) G...	0	Windows
205	94	"Mozilla/5.0 (Windows NT 6.2; WOW64) AppleWebK...	0	Windows
206	137	"Mozilla/5.0 (Macintosh; Intel Mac OS X 10_9_4...	0	Mac
207	72	"Mozilla/5.0 (Macintosh; Intel Mac OS X 10_7_5...	0	Mac
208	74	"Mozilla/5.0 (Macintosh; Intel Mac OS X 10_9_4...	0	Mac
209	300016	Mozilla/5.0 (Windows NT 6.1; rv:31.0) Gecko/20...	0	Windows
210	151	"Mozilla/5.0 (Macintosh; Intel Mac OS X 10_9_4...	0	Mac
211	200015	"Mozilla/5.0 (iPhone; CPU iPhone OS 7_1_2 like...	1	iPhone
212	129	"Mozilla/5.0 (Windows NT 6.3; WOW64) AppleWebK...	1	Windows
213	76	"Mozilla/5.0 (Macintosh; Intel Mac OS X 10_9_4...	0	Mac
214	2	"Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebK...	0	Windows
215	100002	Mozilla/5.0 (Windows NT 6.1; WOW64; rv:32.0) G...	0	Windows
216	100018	"Mozilla/5.0 (Macintosh; Intel Mac OS X 10_10)...	0	Mac
217	80	"Mozilla/5.0 (Macintosh; Intel Mac OS X 10_9_4...	0	Mac
218	145	"Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/5...	0	Linux

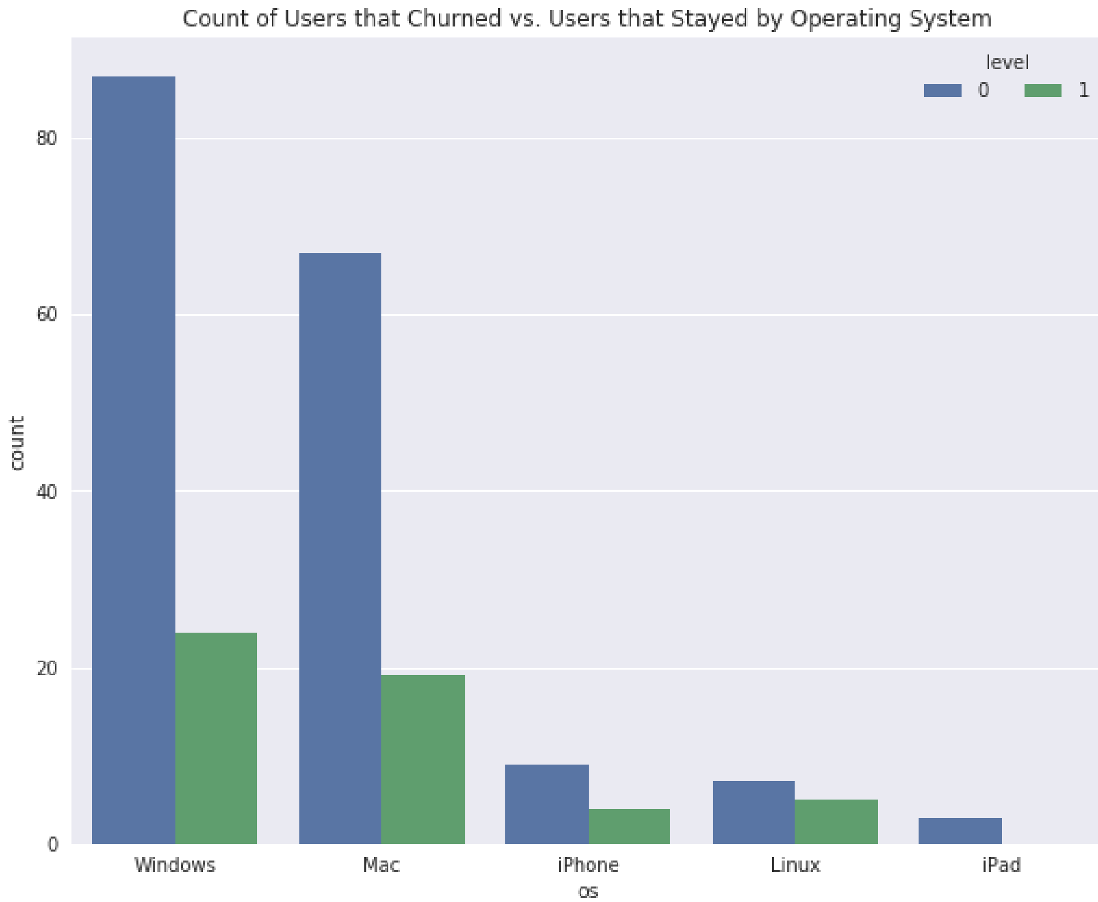
```
219      50  "Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebK...      0  Windows
220      45  "Mozilla/5.0 (Windows NT 6.3; WOW64) AppleWebK...      0  Windows
221      57  "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_9_4...      0      Mac
222  200021  Mozilla/5.0 (Macintosh; Intel Mac OS X 10.7; r...      1      Mac
223     119  "Mozilla/5.0 (Windows NT 6.2; WOW64) AppleWebK...      0  Windows
224  100001  "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_6_8...      1      Mac

[225 rows x 4 columns]
```

In [55]:
```
df_opsys.os.value_counts()
```

Out[55]:
```
Windows    111
Mac         86
iPhone      13
Linux       12
iPad         3
Name: os, dtype: int64
```

In [56]:
```
# order for the plot
os_order = df_opsys.os.value_counts().index
```

In [57]:
```
# plot count for churn and non churn users
plt.figure(figsize=[10,8])
sns.countplot(data = df_opsys, x = 'os', hue ='churn', order = os_order)
plt.title('Count of Users that Churned vs. Users that Stayed by Operating System')
plt.legend(loc = 1, ncol = 2, framealpha =1, title = 'level');
```



Windows was the most used. Linux users have the highest rate of churn. It is very few customers that this has affected therefore this won't be used in our model.

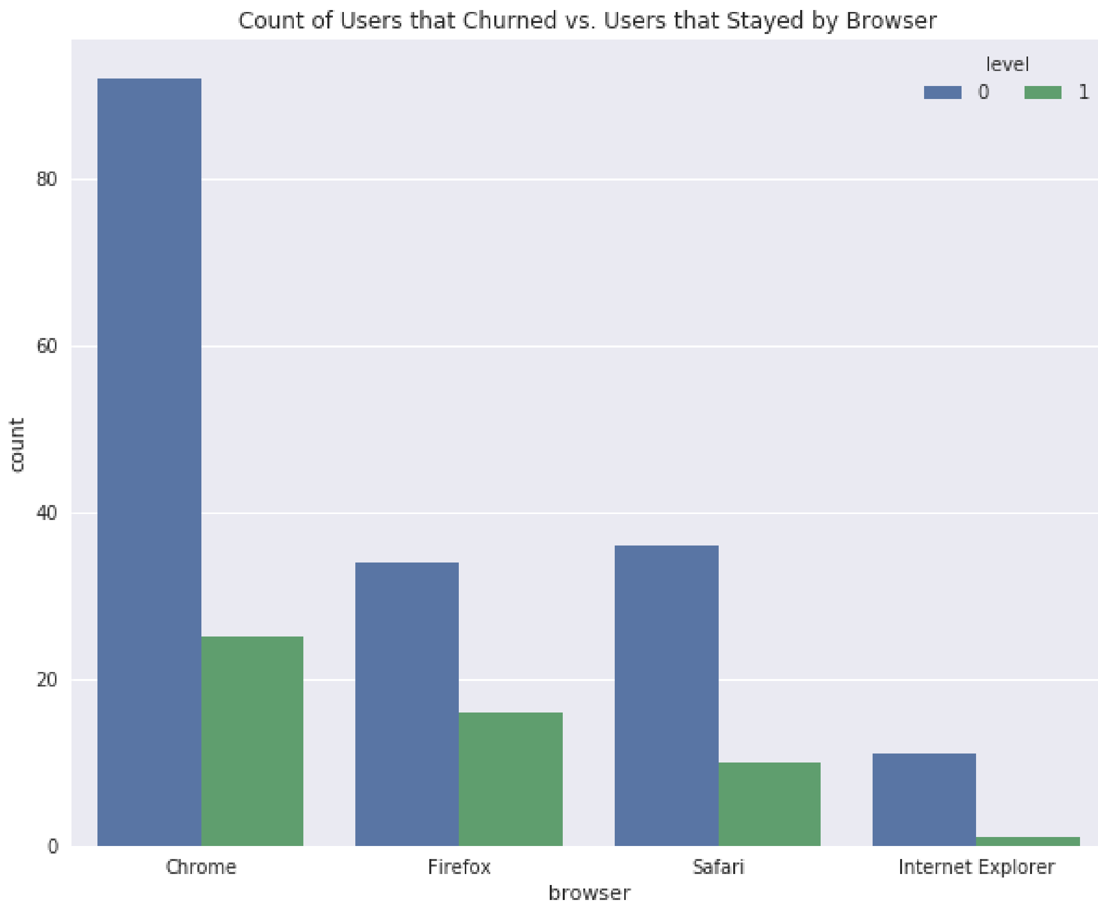
We can also look if browsers had an effect on churn using the same process.

```
In [58]: browser_list = ["Chrome", "Firefox", "Safari", "Trident"]
         df_opsys['browser'] = df_opsys.userAgent.str.extract('(?i)({0})'.format('|'.join(browse
         df_opsys.browser.value_counts()

Out[58]: Chrome     117
         Firefox     50
         Safari      46
         Trident     12
         Name: browser, dtype: int64
```

Here Trident is Internet Explorer software. Let's change Trident to 'Internet Explorer' as it is better known.

```
In [59]: df_opsys['browser'].replace({"Trident":"Internet Explorer"}, inplace = True)
         # order for the plot
         browser_order = df_opsys.browser.value_counts().index
```

```
plt.figure(figsize=[10,8])
sns.countplot(data = df_opsys, x = 'browser', hue ='churn', order = browser_order)
plt.title('Count of Users that Churned vs. Users that Stayed by Browser')
plt.legend(loc = 1, ncol = 2, framealpha =1, title = 'level');
```



Chrome was the most popular browser. Firefox users were most likely to churn. Internet Explorer had the fewest number of users that churned. There is no clear issue with browsers which is making users churn. Therefore this won't be used in our model.

2.1.14 Days Since Registration for Sparkify

Finally, we can look at the number of days since a user had registered.

```
In [60]: df_days = df.select(['userId', 'registration', 'ts', 'churn']).dropDuplicates().sort('u
         # order by last timestamp
         w = Window.partitionBy("userId").orderBy(desc("ts"))
         # create a rank with the most recent timestamp as rank number 1
         df_days = df_days.withColumn("Rank", dense_rank().over(w))
         df_days.show()
```

```
+------+-------------+-------------+-----+----+
|userId| registration|           ts|churn|Rank|
+------+-------------+-------------+-----+----+
|    10|1538159495000|1542631788000|    0|   1|
|    10|1538159495000|1542631753000|    0|   2|
|    10|1538159495000|1542631690000|    0|   3|
|    10|1538159495000|1542631518000|    0|   4|
|    10|1538159495000|1542631517000|    0|   5|
|    10|1538159495000|1542631090000|    0|   6|
|    10|1538159495000|1542630866000|    0|   7|
|    10|1538159495000|1542630637000|    0|   8|
|    10|1538159495000|1542630407000|    0|   9|
|    10|1538159495000|1542630394000|    0|  10|
|    10|1538159495000|1542630248000|    0|  11|
|    10|1538159495000|1542630247000|    0|  12|
|    10|1538159495000|1542630029000|    0|  13|
|    10|1538159495000|1542629861000|    0|  14|
|    10|1538159495000|1542629636000|    0|  15|
|    10|1538159495000|1542629464000|    0|  16|
|    10|1538159495000|1542629238000|    0|  17|
|    10|1538159495000|1542629029000|    0|  18|
|    10|1538159495000|1542629028000|    0|  19|
|    10|1538159495000|1542628798000|    0|  20|
+------+-------------+-------------+-----+----+
only showing top 20 rows
```

In [61]:
```
# just get those with a rank of 1 i.e the first rows
df_days = df_days.filter(df_days.Rank == 1).drop(df_days.Rank)
df_days.show()
```

```
+------+-------------+-------------+-----+
|userId| registration|           ts|churn|
+------+-------------+-------------+-----+
|    10|1538159495000|1542631788000|    0|
|   100|1537982255000|1543587349000|    0|
|100001|1534627466000|1538498205000|    1|
|100002|1529934689000|1543799476000|    0|
|100003|1537309344000|1539274781000|    1|
|100004|1528560242000|1543459065000|    0|
|100005|1532610926000|1539971825000|    1|
|100006|1537964483000|1538753070000|    1|
|100007|1533522419000|1543491909000|    1|
|100008|1537440271000|1543335219000|    0|
|100009|1537376437000|1540611104000|    1|
|100010|1538016340000|1542823952000|    0|
|100011|1537970819000|1538417085000|    1|
```

```
|100012|1537381154000|1541100900000|    1|
|100013|1537367773000|1541184816000|    1|
|100014|1535389443000|1542740649000|    1|
|100015|1537208989000|1543073753000|    1|
|100016|1536854322000|1543335647000|    0|
|100017|1533247234000|1540062847000|    1|
|100018|1533812833000|1543378360000|    0|
+------+-------------+-------------+-----+
only showing top 20 rows
```

In [62]:
```python
# Now need to minus these and work that out in days.
# need to minus the registration from ts
df_days = df_days.withColumn("delta_days", (df_days['ts']) - (df_days['registration']))
df_days.show()
```

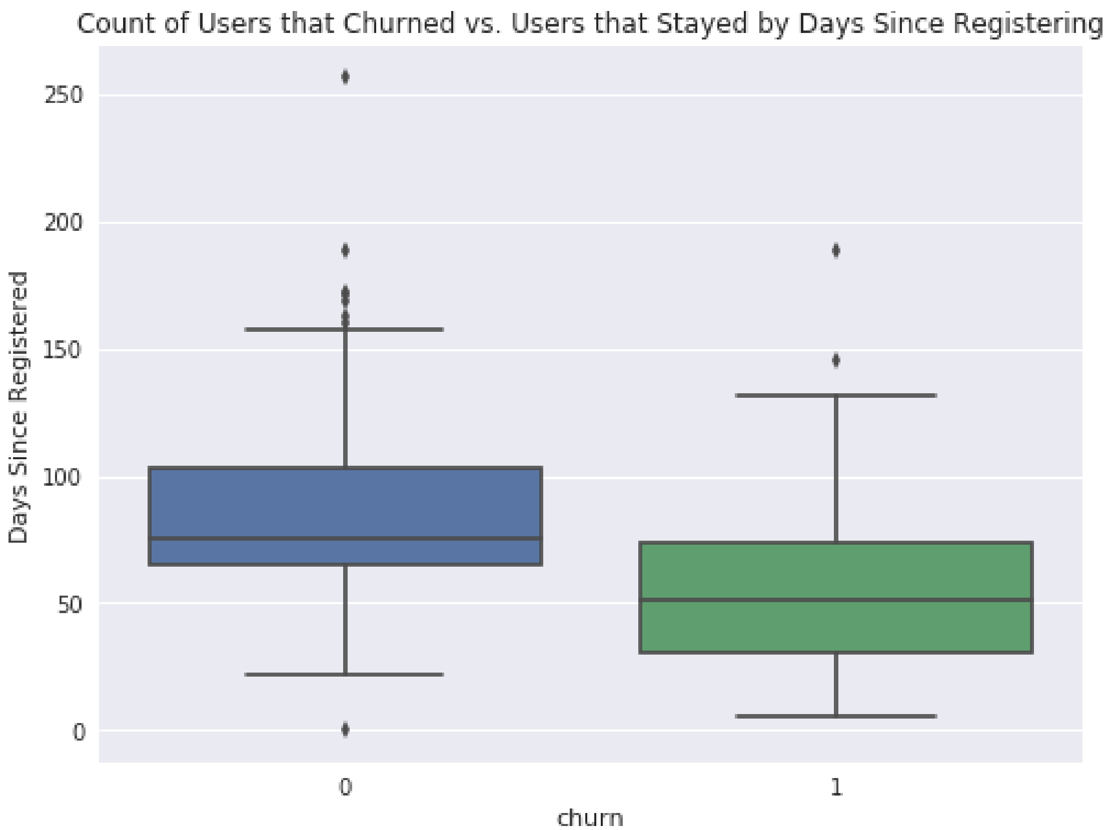
```
+------+-------------+-------------+-----+-----------+
|userId| registration|           ts|churn| delta_days|
+------+-------------+-------------+-----+-----------+
|    10|1538159495000|1542631788000|    0| 4472293000|
|   100|1537982255000|1543587349000|    0| 5605094000|
|100001|1534627466000|1538498205000|    1| 3870739000|
|100002|1529934689000|1543799476000|    0|13864787000|
|100003|1537309344000|1539274781000|    1| 1965437000|
|100004|1528560242000|1543459065000|    0|14898823000|
|100005|1532610926000|1539971825000|    1| 7360899000|
|100006|1537964483000|1538753070000|    1|  788587000|
|100007|1533522419000|1543491909000|    1| 9969490000|
|100008|1537440271000|1543335219000|    0| 5894948000|
|100009|1537376437000|1540611104000|    1| 3234667000|
|100010|1538016340000|1542823952000|    0| 4807612000|
|100011|1537970819000|1538417085000|    1|  446266000|
|100012|1537381154000|1541100900000|    1| 3719746000|
|100013|1537367773000|1541184816000|    1| 3817043000|
|100014|1535389443000|1542740649000|    1| 7351206000|
|100015|1537208989000|1543073753000|    1| 5864764000|
|100016|1536854322000|1543335647000|    0| 6481325000|
|100017|1533247234000|1540062847000|    1| 6815613000|
|100018|1533812833000|1543378360000|    0| 9565527000|
+------+-------------+-------------+-----+-----------+
only showing top 20 rows
```

In [63]:
```python
df_days = df_days.withColumn('days',(df_days['delta_days']/1000/3600/24))
df_days.show()
```

```
+------+-------------+-------------+-----+-----------+------------------+
|userId| registration|           ts|churn| delta_days|              days|
```

```
+------+-------------+-------------+-----+-----------+------------------+
|    10|1538159495000|1542631788000|    0| 4472293000| 51.76265046296297|
|   100|1537982255000|1543587349000|    0| 5605094000| 64.87377314814815|
|100001|1534627466000|1538498205000|    1| 3870739000| 44.80021990740741|
|100002|1529934689000|1543799476000|    0|13864787000|160.47207175925925|
|100003|1537309344000|1539274781000|    1| 1965437000|22.748113425925926|
|100004|1528560242000|1543459065000|    0|14898823000|172.44008101851853|
|100005|1532610926000|1539971825000|    1| 7360899000| 85.19559027777778|
|100006|1537964483000|1538753070000|    1|  788587000| 9.127164351851851|
|100007|1533522419000|1543491909000|    1| 9969490000|115.38761574074074|
|100008|1537440271000|1543335219000|    0| 5894948000| 68.22856481481482|
|100009|1537376437000|1540611104000|    1| 3234667000| 37.43827546296296|
|100010|1538016340000|1542823952000|    0| 4807612000|  55.6436574074074|
|100011|1537970819000|1538417085000|    1|  446266000| 5.165115740740741|
|100012|1537381154000|1541100900000|    1| 3719746000| 43.05261574074074|
|100013|1537367773000|1541184816000|    1| 3817043000| 44.17873842592593|
|100014|1535389443000|1542740649000|    1| 7351206000| 85.08340277777778|
|100015|1537208989000|1543073753000|    1| 5864764000| 67.87921296296297|
|100016|1536854322000|1543335647000|    0| 6481325000| 75.01533564814815|
|100017|1533247234000|1540062847000|    1| 6815613000| 78.88440972222223|
|100018|1533812833000|1543378360000|    0| 9565527000|110.71211805555555|
+------+-------------+-------------+-----+-----------+------------------+
only showing top 20 rows
```

In [64]:
```
# to Pandas for the plot
df_days_pd = df_days.toPandas()
# plot boxplot
plt.figure(figsize=[8,6])
sns.boxplot(data = df_days_pd, x = 'churn', y ='days')
plt.title('Count of Users that Churned vs. Users that Stayed by Days Since Registering'
plt.ylabel("Days Since Registered");
```



On average those who had been registered with Sparkify for longer were more likely to stay. Users who had registered more recently were more likely to churn.

3 Feature Engineering

Now that EDA has been performed, we can build out the features that seem most promising to train our model on.

The features we will build out are: - Categorical: - gender - level

- Numerical:
- number of songs per session
- number of rollads actions
- number of thumb down actions
- number of thumbs up actions
- number of friends added
- number of songs added to playlist
- number of different artists listened to on Sparkify
- number of days since registering

We will also then add a churn label and join these all together. This will create a dataFrame where each row represents information pertaining to each individual user. Once we drop the userId, this dataframe can be vectorised, standarised and fed into our different machine learning algorithms.

First we will take our categorical variables and convert these into numeric variables, ready for our model.

3.0.1 Gender

```
In [65]: # Our first feature is gender which is a categorical one. We will assign a 1 for 'femal
         gender_f1 = df.select(['userId', 'gender']).dropDuplicates()
         # create gender column
         gender_f1 = gender_f1.withColumn('gender', when(col('gender') == 'F', 1).otherwise(0))
         print(gender_f1.count())
         # check
         gender_f1.show(20)
```

```
225
+------+------+
|userId|gender|
+------+------+
|    44|     1|
|    46|     1|
|    41|     1|
|    72|     1|
|300023|     1|
|    39|     1|
|100010|     1|
|    40|     1|
|    94|     1|
|    35|     1|
|    75|     1|
|   116|     1|
|200001|     0|
|200020|     0|
|100008|     1|
|200015|     0|
|   100|     0|
|100006|     1|
|300005|     1|
|    25|     1|
+------+------+
only showing top 20 rows
```

3.0.2 Level

```
In [66]: # The next feature we will take is level. The level can change so we need to only take
         df2 = df.select(['userId', 'level', 'ts']).dropDuplicates().sort('userId')
```

```
w = Window.partitionBy("userId").orderBy(desc("ts"))
df2 = df2.withColumn("Rank", dense_rank().over(w))
df2.show()
```

```
+------+-----+-------------+----+
|userId|level|           ts|Rank|
+------+-----+-------------+----+
|    10| paid|1542631788000|   1|
|    10| paid|1542631753000|   2|
|    10| paid|1542631690000|   3|
|    10| paid|1542631518000|   4|
|    10| paid|1542631517000|   5|
|    10| paid|1542631090000|   6|
|    10| paid|1542630866000|   7|
|    10| paid|1542630637000|   8|
|    10| paid|1542630407000|   9|
|    10| paid|1542630394000|  10|
|    10| paid|1542630248000|  11|
|    10| paid|1542630247000|  12|
|    10| paid|1542630029000|  13|
|    10| paid|1542629861000|  14|
|    10| paid|1542629636000|  15|
|    10| paid|1542629464000|  16|
|    10| paid|1542629238000|  17|
|    10| paid|1542629029000|  18|
|    10| paid|1542629028000|  19|
|    10| paid|1542628798000|  20|
+------+-----+-------------+----+
only showing top 20 rows
```

```
In [67]: level_f2 = df2.filter(df2.Rank == 1).drop(df2.Rank)
         level_f2 = level_f2.drop('ts')
         level_f2 = level_f2.withColumn('level', when(col('level') == 'paid', 1).otherwise(0))
         print(level_f2.count())
         level_f2.show(20)
```

```
225
+------+-----+
|userId|level|
+------+-----+
|    10|    1|
|   100|    1|
|100001|    0|
|100002|    1|
|100003|    0|
|100004|    1|
```

```
|100005|    0|
|100006|    0|
|100007|    1|
|100008|    0|
|100009|    0|
|100010|    0|
|100011|    0|
|100012|    0|
|100013|    1|
|100014|    1|
|100015|    1|
|100016|    0|
|100017|    0|
|100018|    0|
+------+-----+
only showing top 20 rows
```

3.0.3 Average Number of songs per session

In [68]:
```
# Our third feature is average number of songs per session for each user.
song_f3 = df.filter(df.page == "NextSong").groupBy('userId','sessionId').count()
df.filter(df.page == "NextSong").groupBy('userId','sessionId').count().show(2)
```

```
+------+---------+-----+
|userId|sessionId|count|
+------+---------+-----+
|    92|      358|   57|
|    42|      433|   16|
+------+---------+-----+
only showing top 2 rows
```

In [69]:
```
song_f3 = song_f3.groupby('userId').agg({"count":"avg"})
song_f3 = song_f3.withColumnRenamed("avg(count)", "avg_song")
print(song_f3.count())
song_f3.show(2)
```

```
225
+------+------------------+
|userId|          avg_song|
+------+------------------+
|100010|39.285714285714285|
|200002|              64.5|
+------+------------------+
only showing top 2 rows
```

3.0.4 Number of rollads actions

Next feature we can consider is number of roll advert actions. This had a higher number of roll ad count for those who churned since those who use the app for free are shown ads whereas paid subscribers aren't shown ads.

```
In [70]: rollad_f4 = df.select(["userId", "page"])
         rollad_event = udf(lambda x: 1 if x == "Roll Advert" else 0, IntegerType())
         #creating rollad column
         rollad_f4 = rollad_f4.withColumn("rollad", rollad_event("page"))
         rollad_f4 = rollad_f4.groupby('userId').sum("rollad")
         rollad_f4 = rollad_f4.withColumnRenamed("sum(rollad)", "roll_ad")
         rollad_f4.show(2)
```

```
+------+-------+
|userId|roll_ad|
+------+-------+
|100010|     52|
|200002|      7|
+------+-------+
only showing top 2 rows
```

3.0.5 Number of thumb down actions

The fifth feature we can add to our feature dataframe is thumbs down. Users who had churned in the past had performed more thumbs down actions than those who stayed with the service.

```
In [71]: thumbdown_f5 = df.select(["userId", "page"])
         thumddown_event = udf(lambda x: 1 if x == "Thumbs Down" else 0, IntegerType())
         thumbdown_f5 = thumbdown_f5.withColumn("Thumbs Down", thumddown_event("page"))
         thumbdown_f5 = thumbdown_f5.groupby('userId').sum("Thumbs Down")
         thumbdown_f5 = thumbdown_f5.withColumnRenamed("sum(Thumbs Down)", "thumbs_down")
         thumbdown_f5.show(2)
```

```
+------+-----------+
|userId|thumbs_down|
+------+-----------+
|100010|          5|
|200002|          6|
+------+-----------+
only showing top 2 rows
```

3.0.6 Number of thumbs up actions

We can do the same for thumb up actions. Users who stayed with the service had performed more thumbs up actions in the past.

```
In [72]: thumbup_f6 = df.select(["userId", "page"])
         thumbup_event = udf(lambda x: 1 if x == "Thumbs Up" else 0, IntegerType())
         thumbup_f6 = thumbup_f6.withColumn("Thumbs Up", thumbup_event("page"))
         thumbup_f6 = thumbup_f6.groupby('userId').sum("Thumbs Up")
         thumbup_f6 = thumbup_f6.withColumnRenamed("sum(Thumbs Up)", "thumbs_up")
         thumbup_f6.show(2)
```

```
+------+---------+
|userId|thumbs_up|
+------+---------+
|100010|       17|
|200002|       21|
+------+---------+
only showing top 2 rows
```

3.0.7 Number of friends added

Similarly, number of friends added can indicate if a user is likely to churn or not. In the past, those who added more friends stayed with the app.

```
In [73]: friend_f7 = df.select(["userId", "page"])
         add_friend = udf(lambda x: 1 if x == "Add Friend" else 0, IntegerType())
         friend_f7 = friend_f7.withColumn("add_friend", add_friend("page"))
         friend_f7 = friend_f7.groupby('userId').sum("add_friend")
         friend_f7 = friend_f7.withColumnRenamed("sum(add_friend)", "add_friend")
         friend_f7.show(2)
```

```
+------+----------+
|userId|add_friend|
+------+----------+
|100010|         4|
|200002|         4|
+------+----------+
only showing top 2 rows
```

3.0.8 Number of songs added to playlist

Again, those who added more songs to their playlists had stayed with the service so this can provide an indication of whether a user is likely to churn.

```
In [74]: playlist_f8 = df.select(["userId", "page"])
         add_playlist = udf(lambda x: 1 if x == "Add to Playlist" else 0, IntegerType())
         playlist_f8 = playlist_f8.withColumn("Playlist", add_playlist("page"))
         playlist_f8 = playlist_f8.groupby('userId').sum("Playlist")
         playlist_f8 = playlist_f8.withColumnRenamed("sum(Playlist)", "playlist")
         playlist_f8.show(2)
```

```
+------+--------+
|userId|playlist|
+------+--------+
|100010|       7|
|200002|       8|
+------+--------+
only showing top 2 rows
```

3.0.9 Number of different Artists Listened to on Sparkify

As we discovered in EDA, users that listened to more diverse artists were less likely to churn.

```
In [75]: artists_f9 = df.select("userId", "artist").dropDuplicates().groupby("userId").count()
         artists_f9 = artists_f9.withColumnRenamed("count", "num_artists")
         artists_f9.show(2)
```

```
+------+-----------+
|userId|num_artists|
+------+-----------+
|100010|        253|
|200002|        340|
+------+-----------+
only showing top 2 rows
```

3.0.10 Number of Days Since Registering

Number of days since registering also looked useful from our EDA. We saw that users who had a shorter number of days since registering churned more than those who had used the service for a longer time.

```
In [76]: days_f10 = df_days.drop('registration', 'ts', 'churn', 'delta_days')
         days_f10.count()
         days_f10.show()
```

```
+------+------------------+
|userId|              days|
+------+------------------+
|    10| 51.76265046296297|
```

```
|   100| 64.87377314814815|
|100001| 44.80021990740741|
|100002|160.47207175925925|
|100003|22.748113425925926|
|100004|172.44008101851853|
|100005| 85.19559027777778|
|100006| 9.127164351851851|
|100007|115.38761574074074|
|100008| 68.22856481481482|
|100009| 37.43827546296296|
|100010|  55.6436574074074|
|100011| 5.165115740740741|
|100012| 43.05261574074074|
|100013| 44.17873842592593|
|100014| 85.08340277777778|
|100015| 67.87921296296297|
|100016| 75.01533564814815|
|100017| 78.88440972222223|
|100018|110.71211805555555|
+------+------------------+
only showing top 20 rows
```

3.0.11 Label

Now we can create our label column indicating if the user churned (1) or not (0).

```
In [77]: label = df.select("userId", "churn").dropDuplicates().groupby("userId", "churn").count(
         label = label.drop('count')
         label = label.withColumnRenamed("churn", "label")
         label.show()
```

```
+------+-----+
|userId|label|
+------+-----+
|100010|    0|
|200002|    0|
|   125|    1|
|   124|    0|
|    51|    1|
|     7|    0|
|    15|    0|
|    54|    1|
|   155|    0|
|100014|    1|
|   132|    0|
|   154|    0|
```

```
|   101|    1|
|    11|    0|
|   138|    0|
|300017|    0|
|100021|    1|
|    29|    1|
|    69|    0|
|   112|    0|
+------+-----+
only showing top 20 rows
```

3.0.12 Create Features Dataset

Now that we have our features we need to join these together on `userId`.

```
In [78]: feature_df = gender_f1.join(level_f2, ["userId"]).join(song_f3, ["userId"]).join(rollad
         feature_df.show()
```

```
+------+------+-----+------------------+-------+-----------+---------+----------+--------+------
|userId|gender|level|          avg_song|roll_ad|thumbs_down|thumbs_up|add_friend|playlist|num_ar
+------+------+-----+------------------+-------+-----------+---------+----------+--------+------
|100010|     1|    0|39.285714285714285|     52|          5|       17|         4|       7|
|200002|     0|    1|              64.5|      7|          6|       21|         4|       8|
|   125|     0|    0|               8.0|      1|          0|        0|         0|       0|
|   124|     1|    1|145.67857142857142|      4|         41|      171|        74|     118|
|    51|     0|    1|             211.1|      0|         21|      100|        28|      52|
|     7|     0|    0|21.428571428571427|     16|          1|        7|         1|       5|
|    15|     0|    1|136.71428571428572|      1|         14|       81|        31|      59|
|    54|     1|    1| 81.17142857142858|     47|         29|      163|        33|      72|
|   155|     1|    1|136.66666666666666|      8|          3|       58|        11|      24|
|100014|     0|    1|42.833333333333336|      2|          3|       17|         6|       7|
|   132|     1|    1|             120.5|      2|         17|       96|        41|      38|
|   154|     1|    0|              28.0|     10|          0|       11|         3|       1|
|   101|     0|    1|             179.7|      8|         16|       86|        29|      61|
|    11|     1|    1|           40.4375|     39|          9|       40|         6|      20|
|   138|     0|    1|             138.0|     17|         24|       95|        41|      67|
|300017|     1|    1|59.540983606557376|     11|         28|      303|        63|     113|
|100021|     0|    0|              46.0|     30|          5|       11|         7|       7|
|    29|     0|    1| 89.05882352941177|     22|         22|      154|        47|      89|
|    69|     1|    1|             125.0|      3|          9|       72|        12|      33|
|   112|     0|    0| 23.88888888888889|     21|          3|        9|         7|       7|
+------+------+-----+------------------+-------+-----------+---------+----------+--------+------
only showing top 20 rows
```

```
In [79]: # drop the user_id
         feature_df = feature_df.drop('userId')
         feature_df.show()
```

```
+------+-----+-----------------+-------+-----------+---------+----------+--------+-----------+-
|gender|level|         avg_song|roll_ad|thumbs_down|thumbs_up|add_friend|playlist|num_artists|
+------+-----+-----------------+-------+-----------+---------+----------+--------+-----------+-
|     1|    0|39.285714285714285|     52|          5|       17|         4|       7|        253|
|     0|    1|             64.5|      7|          6|       21|         4|       8|        340|
|     0|    0|              8.0|      1|          0|        0|         0|       0|          9|
|     1|    1|145.67857142857142|      4|         41|      171|        74|     118|       2233|1
|     0|    1|            211.1|      0|         21|      100|        28|      52|       1386|1
|     0|    0|21.428571428571427|     16|          1|        7|         1|       5|        143|
|     0|    1|136.71428571428572|      1|         14|       81|        31|      59|       1303|5
|     1|    1| 81.17142857142858|     47|         29|      163|        33|      72|       1745|1
|     1|    1|136.66666666666666|      8|          3|       58|        11|      24|        644|2
|     0|    1|42.833333333333336|      2|          3|       17|         6|       7|        234|
|     1|    1|            120.5|      2|         17|       96|        41|      38|       1300|
|     1|    0|             28.0|     10|          0|       11|         3|       1|         79|2
|     0|    1|            179.7|      8|         16|       86|        29|      61|       1242|5
|     1|    1|          40.4375|     39|          9|       40|         6|      20|        535|1
|     0|    1|            138.0|     17|         24|       95|        41|      67|       1333|
|     1|    1|59.540983606557376|     11|         28|      303|        63|     113|       2071|
|     0|    0|             46.0|     30|          5|       11|         7|       7|        208|
|     0|    1| 89.05882352941177|     22|         22|      154|        47|      89|       1805|6
|     1|    1|            125.0|      3|          9|       72|        12|      33|        866|
|     0|    0| 23.88888888888889|     21|          3|        9|         7|       7|        196|
+------+-----+-----------------+-------+-----------+---------+----------+--------+-----------+-
only showing top 20 rows
```

Now we have a dataframe with all the features we can into our model where each row represents a user.However first we need to do some preprocessing.

3.1 Preprocessing

```
In [80]: # print schema
         feature_df.printSchema()
```

```
root
 |-- gender: integer (nullable = false)
 |-- level: integer (nullable = false)
 |-- avg_song: double (nullable = true)
 |-- roll_ad: long (nullable = true)
 |-- thumbs_down: long (nullable = true)
 |-- thumbs_up: long (nullable = true)
 |-- add_friend: long (nullable = true)
 |-- playlist: long (nullable = true)
```

```
 |-- num_artists: long (nullable = false)
 |-- days: double (nullable = true)
 |-- label: long (nullable = true)
```

Now we need to take these columns and convert into the numerical datatypes that will be used in our model: integers and floats. We can use write a function to adhere to DRY principles.

```
In [81]: for feature in feature_df.columns:
             feature_df = feature_df.withColumn(feature, feature_df[feature].cast('float'))
         #check this works
         feature_df.printSchema()
```

```
root
 |-- gender: float (nullable = false)
 |-- level: float (nullable = false)
 |-- avg_song: float (nullable = true)
 |-- roll_ad: float (nullable = true)
 |-- thumbs_down: float (nullable = true)
 |-- thumbs_up: float (nullable = true)
 |-- add_friend: float (nullable = true)
 |-- playlist: float (nullable = true)
 |-- num_artists: float (nullable = false)
 |-- days: float (nullable = true)
 |-- label: float (nullable = true)
```

3.1.1 Vector Assembler

The purpose of vector assembler is to tranform our features into a vector. The vector can then be standardised and fed into our chosen algorithms.

```
In [82]: assembler = VectorAssembler(inputCols = ["gender", "level", "avg_song", "roll_ad", "thu
         feature_df = assembler.transform(feature_df)
         feature_df.show()
```

```
+------+-----+---------+-------+-----------+---------+----------+--------+-----------+----------
|gender|level| avg_song|roll_ad|thumbs_down|thumbs_up|add_friend|playlist|num_artists|      days
+------+-----+---------+-------+-----------+---------+----------+--------+-----------+----------
|   1.0|  0.0|39.285713|   52.0|        5.0|     17.0|       4.0|     7.0|      253.0| 55.643658
|   0.0|  1.0|     64.5|    7.0|        6.0|     21.0|       4.0|     8.0|      340.0|  70.07463
|   0.0|  0.0|      8.0|    1.0|        0.0|      0.0|       0.0|     0.0|        9.0|  71.31689
|   1.0|  1.0|145.67857|    4.0|       41.0|    171.0|      74.0|   118.0|     2233.0| 131.55591
|   0.0|  1.0|    211.1|    0.0|       21.0|    100.0|      28.0|    52.0|     1386.0| 19.455845
|   0.0|  0.0|21.428572|   16.0|        1.0|      7.0|       1.0|     5.0|      143.0|  72.77818
|   0.0|  1.0|136.71428|    1.0|       14.0|     81.0|      31.0|    59.0|     1303.0| 56.513577
|   1.0|  1.0|81.171425|   47.0|       29.0|    163.0|      33.0|    72.0|     1745.0|110.751686
```

```
|   1.0|  1.0|136.66667|    8.0|         3.0|       58.0|        11.0|      24.0|        644.0| 23.556019
|   0.0|  1.0|42.833332|    2.0|         3.0|       17.0|         6.0|       7.0|        234.0| 85.083405
|   1.0|  1.0|    120.5|    2.0|        17.0|       96.0|        41.0|      38.0|       1300.0|  66.88911
|   1.0|  0.0|     28.0|   10.0|         0.0|       11.0|         3.0|       1.0|         79.0| 23.872038
|   0.0|  1.0|    179.7|    8.0|        16.0|       86.0|        29.0|      61.0|       1242.0|  53.96594
|   1.0|  1.0|  40.4375|   39.0|         9.0|       40.0|         6.0|      20.0|        535.0| 124.47825
|   0.0|  1.0|    138.0|   17.0|        24.0|       95.0|        41.0|      67.0|       1333.0| 66.626686
|   1.0|  1.0|59.540985|   11.0|        28.0|      303.0|        63.0|     113.0|       2071.0|  74.35852
|   0.0|  0.0|     46.0|   30.0|         5.0|       11.0|         7.0|       7.0|        208.0|  64.73887
|   0.0|  1.0| 89.05882|   22.0|        22.0|      154.0|        47.0|      89.0|       1805.0|  60.10405
|   1.0|  1.0|    125.0|    3.0|         9.0|       72.0|        12.0|      33.0|        866.0| 71.424446
|   0.0|  0.0| 23.88889|   21.0|         3.0|        9.0|         7.0|       7.0|        196.0|  87.46262
+------+-----+---------+-------+-----------+---------+----------+--------+-----------+----------
only showing top 20 rows
```

3.1.2 Standardisation

Now that we have our vectors we can standardise our values. This is important for our machine learning model so that those features with the highest values don't dominate the results and so that we can make the individual features look like standard normally distributed data.

```
In [83]: scaler = StandardScaler(inputCol="vec_features", outputCol="features", withStd=True)
         scaler_model = scaler.fit(feature_df)
         feature_df = scaler_model.transform(feature_df)
         feature_df.head(2)

Out[83]: [Row(gender=1.0, level=0.0, avg_song=39.28571319580078, roll_ad=52.0, thumbs_down=5.0,
          Row(gender=0.0, level=1.0, avg_song=64.5, roll_ad=7.0, thumbs_down=6.0, thumbs_up=21.0
```

We can see from above that standardisation has worked by comparing vec_features=DenseVector([0.0, 1.0, 64.5, 7.0, 6.0, 21.0, 4.0, 8.0, 340.0, 70.0746]), to features=DenseVector([0.0, 2.0844, 1.5135, 0.3248, 0.4588, 0.3207, 0.1943, 0.2445, 0.563, 1.8606]).

3.2 Train / Test / Validation Split

Let's check how many records we have in total is 225 as it should be.

```
In [84]: feature_df.groupby('label').count().show()

+-----+-----+
|label|count|
+-----+-----+
|  1.0|   52|
|  0.0|  173|
+-----+-----+
```

This count is what we would expect, now we can split our data into train, test and validation sets. Here we will do a 60:20:20 split and include a seed so we can reproduce the result. I've included the same seed for the different machine learning models so that my results can be reproduced.

```
In [85]: train, test, valid = feature_df.randomSplit([0.6, 0.2, 0.2], seed = 1996)
         print("Training Dataset:" + str(train.count()))
         print("Test Dataset:" + str(test.count()))
         print("Validation Dataset:" + str(valid.count()))
```

```
Training Dataset:142
Test Dataset:47
Validation Dataset:36
```

4 Modelling

Now we have created our features dataFrame with only numeric variables, we can split the full dataset into train, test, and validation sets. We will test out different machine learning classification algorithms including: - Logistic Regression - Random Forest Classifier - Gradient-Boosted Tree Classifier - Linear Support Vector Machine - Naive Bayes

We will use these classification algorithms since churn prediction is a binary classification problem, meaning that customers will either churn (1) or they will stay (0) in a certain period of time.

4.0.1 Metrics

We will evaluate the accuracy of the various models, tuning parameters as necessary. We will finally determine our winning model based on test accuracy and report results on the validation set. Since the churned users are a fairly small subset, I will use F1 score as the metric to optimize. F1 is a measure of the model's accuracy on a dataset and is used to evaluate binary classification systems like we have here. F1-score is a way of combining the precision and recall of the model and gives a better measure of the incorrectly classified cases than accuracy metric. F1 is also better for dealing with imbalanced classes like we have here.

Now we can start modelling. When we identify the model with the best F1 score, accuracy and time we will then tune the model.

The models I have selected are below with the reasons why these have been chosen. Each model that has been chosen is suitable for our binary classification problem of predicting churn.

- **Logistic Regression:** Logistic regression is the first machine learning algorithm we can try. Logistic regression is a reliable machine learning algorithm to try since this is a binary classification problem and logistic regression provides a model with good explainability. Logistic regression is also easy to implement, interpret and is efficient to train. It is also less inclined to overfitting.

- **Random Forest:** Random Forest is a powerful supervised learning algorithm that can be used for classification. RF is an ensemble method that creates multiple decision trees to make predictions and takes a majority vote of decisions reached. This can help avoid overfitting. RF is also robust and has good performance on imbalanced datasets like we have here.

- **Gradient Boosted Tree Classifier:** GBT provides good predictive accuracy. This works by building one tree at a time where each new tree helps correct errors made by the previous tree compared to RF which builds trees independently. There is a risk of overfitting with GBT so this needs to be considered. However GBT performs well with unbalanced data which we have here.

- **Linear SVC:** SVC is another supervised learning binary classification algorithm. It works well with clear margins of separations between classes and is memory efficient.

- **Naive Bayes:** Finally, we will try Naive Bayes. This is another classifier algorithm that is easy to implement and is fast.

4.0.2 Training the Models & Evaluating the Model Performance

Steps: - Instantiate - Fit Models on Train - Predicting - Evaluating

In [86]:
```
# instantiate all of our models and include a seed for reproducibility where possible
lr = LogisticRegression(featuresCol = 'features', labelCol = 'label', maxIter=10)
rf = RandomForestClassifier(featuresCol = 'features', labelCol = 'label', seed=1996)
gbt = GBTClassifier(featuresCol = 'features', labelCol = 'label', maxIter=10, seed=1996
lsvc = LinearSVC(featuresCol = 'features', labelCol = 'label')
nb = NaiveBayes(featuresCol = 'features', labelCol = 'label')
#list of models
model_list = [lr,rf,gbt,lsvc,nb]
# evaluator we are using is multiclassclassificationevaluator to get the F1 scores
evaluator = MulticlassClassificationEvaluator(labelCol = 'label', predictionCol='predic
```

In [87]:
```
# for loop to go through all our models
for model in model_list:
    # get model name
    model_name = model.__class__.__name__

    # print training started
    print(model_name, 'training started')

    # start time
    start = time.time()
    # fit the models on train dataset
    model = model.fit(train)
    # end time
    end = time.time()

    # print training ended
    print(model_name, 'training ended')
    # print time taken
```

```python
            print('Time taken for {} is:'.format(model_name),(end-start),'seconds')

            # predict
            print(model_name, 'predicting started')
            predictions = model.transform(valid)
            print(model_name, 'predicting ended')

            # get metrics to evaluate
            # f1
            print('F1 for {} is:'.format(model_name), evaluator.evaluate(predictions, {evaluato
            # accuracy
            accuracy = predictions.filter(predictions.label == predictions.prediction).count()
            print("The accuracy of the {} model is:".format(model_name), accuracy)
```

```
LogisticRegression training started
LogisticRegression training ended
Time taken for LogisticRegression is: 118.57629203796387 seconds
LogisticRegression predicting started
LogisticRegression predicting ended
F1 for LogisticRegression is: 0.6523297491039427
The accuracy of the LogisticRegression model is: 0.7222222222222222
RandomForestClassifier training started
RandomForestClassifier training ended
Time taken for RandomForestClassifier is: 177.9151096343994 seconds
RandomForestClassifier predicting started
RandomForestClassifier predicting ended
F1 for RandomForestClassifier is: 0.874074074074074
The accuracy of the RandomForestClassifier model is: 0.8888888888888888
GBTClassifier training started
GBTClassifier training ended
Time taken for GBTClassifier is: 248.3633008003235 seconds
GBTClassifier predicting started
GBTClassifier predicting ended
F1 for GBTClassifier is: 0.8888888888888888
The accuracy of the GBTClassifier model is: 0.8888888888888888
LinearSVC training started
LinearSVC training ended
Time taken for LinearSVC is: 3572.962161540985 seconds
LinearSVC predicting started
LinearSVC predicting ended
F1 for LinearSVC is: 0.6805555555555557
The accuracy of the LinearSVC model is: 0.7777777777777778
NaiveBayes training started
NaiveBayes training ended
Time taken for NaiveBayes is: 123.66462445259094 seconds
NaiveBayes predicting started
NaiveBayes predicting ended
F1 for NaiveBayes is: 0.6805555555555557
```

```
The accuracy of the NaiveBayes model is: 0.7777777777777778
```

Now that we have our results we can choose our best model. Random Forest and Gradient Boosted Trees performed well but random forest was faster so I will choose this one to tune.

4.1 Model Tuning for Best Models:

Now we can tune our model using paramGridbuilder and CrossValidator. I am going to select Random Forest since this is the best compromise for F1 score, accuracy, and time to run. Random Forrest had a F1 score of 0.87 and accuracy of 0.88 and took 2 min 57s compared to GTB which achieved a similar score of 0.88 for both F1 score and accuracy but took 3 min 51s.

4.1.1 Random Forest

```
In [88]: #Let's see what parameters we can tune.
         print(rf.explainParams())
```

```
cacheNodeIds: If false, the algorithm will pass trees to executors to match instances with nodes
checkpointInterval: set checkpoint interval (>= 1) or disable checkpoint (-1). E.g. 10 means tha
featureSubsetStrategy: The number of features to consider for splits at each tree node. Supporte
featuresCol: features column name. (default: features, current: features)
impurity: Criterion used for information gain calculation (case-insensitive). Supported options:
labelCol: label column name. (default: label, current: label)
maxBins: Max number of bins for discretizing continuous features.  Must be >=2 and >= number of
maxDepth: Maximum depth of the tree. (>= 0) E.g., depth 0 means 1 leaf node; depth 1 means 1 int
maxMemoryInMB: Maximum memory in MB allocated to histogram aggregation. If too small, then 1 nod
minInfoGain: Minimum information gain for a split to be considered at a tree node. (default: 0.0
minInstancesPerNode: Minimum number of instances each child must have after split. If a split ca
numTrees: Number of trees to train (>= 1). (default: 20)
predictionCol: prediction column name. (default: prediction)
probabilityCol: Column name for predicted class conditional probabilities. Note: Not all models
rawPredictionCol: raw prediction (a.k.a. confidence) column name. (default: rawPrediction)
seed: random seed. (default: 7130716980068400954, current: 1996)
subsamplingRate: Fraction of the training data used for learning each decision tree, in range (0
```

4.2 Parameters

I will select numTrees and maxDepth for our RF model tuning. - **NumTrees**: I have chosen to go up to 100 trees to improve performance. Since these trees are individual randomised models in an ensemble there is not a great risk of overfitting with this numTrees parameter. - **Maxdepth**: I have chosen a max of 15 to reduce the possibility of overfitting. Anything over 15 would increase the risk of overfitting greatly. - **Numfolds**: I originally had numFolds = 5 but had to change to 3 to speed up the process.

```
In [89]: paramGrid = ParamGridBuilder() \
             .addGrid(rf.numTrees,[20, 50, 100]) \
             .addGrid(rf.maxDepth,[5, 10, 15]) \
```