Sparkify Project Workspace

In [1]:
```
import ibmos2spark, os
# @hidden_cell
metadata = {
    'endpoint': 'https://s3.private.eu-de.cloud-object-storage.appdomain.cloud',
    'service_id': 'iam-ServiceId-45f608af-e504-4a7b-bace-37aec030b482',
    'iam_service_endpoint': 'https://iam.cloud.ibm.com/oidc/token',
    'api_key': '9iYHmxmI7JXEtlA5kxALJOOTDysO7Ifv4edkkcpDc2ym'
}

configuration_name = 'os_0b007a47f9e648fa8220dacec0258f51_configs'
cos = ibmos2spark.CloudObjectStorage(sc, metadata, configuration_name, 'bluemix_cos')

from pyspark.sql import SparkSession
spark = SparkSession.builder.getOrCreate()
# Since JSON data can be semi-structured and contain additional metadata, it is possibl
# Please read the documentation of 'SparkSession.read()' to learn more about the possib
# PySpark documentation: http://spark.apache.org/docs/2.0.2/api/python/pyspark.sql.html

# df_data_1 = spark.read.json(cos.url('medium-sparkify-event-data.json', 'sparkify-dono
# df_data_1.take(5)
```

In [2]:
```
# import libraries
from pyspark.sql import SparkSession
from pyspark.sql.functions import avg, col, concat, desc, explode, lit, min, max, split
from pyspark.sql.types import IntegerType
from pyspark.ml import Pipeline
from pyspark.ml.classification import LogisticRegression
from pyspark.ml.evaluation import MulticlassClassificationEvaluator
from pyspark.ml.feature import CountVectorizer, IDF, Normalizer, PCA, RegexTokenizer, S
from pyspark.ml.regression import LinearRegression
from pyspark.ml.tuning import CrossValidator, ParamGridBuilder
import pandas as pd
from IPython.display import display, HTML
import re
import datetime
from pyspark.sql.functions import col, from_unixtime
import numpy as np
%matplotlib inline
import matplotlib.pyplot as plt
import seaborn as sns
from pyspark.sql import functions as F
from pyspark.sql.window import Window
from pyspark.sql.functions import when
from pyspark.sql.functions import min as Fmin, max as Fmax, sum as Fsum, round as Froun
from pyspark.sql import Window
```

In [3]:
```
# create a Spark session
spark = SparkSession.builder \
    .master("local") \
    .appName("Creating Features") \
    .config("spark.sql.repl.eagerEval.enabled", True) \
    .getOrCreate()
```

Import and View Data

Sub Part - 1: Import Data

In [4]:
```
stack_overflow_data = 'medium-sparkify-event-data.json'
```

In [5]:
```
user_log = spark.read.json(cos.url(stack_overflow_data, 'sparkify-donotdelete-pr-r0ihlk
user_log.persist()
```

Out[5]:

artist	auth	firstName	gender	itemInSession	lastName	length	level	location	metho
Martin Orford	Logged In	Joseph	M	20	Morales	597.55057	free	Corpus Christi, TX	PU
John Brown's Body	Logged In	Sawyer	M	74	Larson	380.21179	free	Houston-The Woodl...	PU
Afroman	Logged In	Maverick	M	184	Santiago	202.37016	paid	Orlando-Kissimmee...	PU
null	Logged In	Maverick	M	185	Santiago	null	paid	Orlando-Kissimmee...	PU
Lily Allen	Logged In	Gianna	F	22	Campos	194.53342	paid	Mobile, AL	PU
Carter USM	Logged In	Sofia	F	266	Gordon	138.29179	paid	Rochester, MN	PU
null	Logged Out	null	null	186	null	null	paid	null	GE
null	Logged Out	null	null	187	null	null	paid	null	GE
null	Logged Out	null	null	188	null	null	paid	null	GE
null	Logged Out	null	null	189	null	null	paid	null	PU
null	Logged In	Maverick	M	190	Santiago	null	paid	Orlando-Kissimmee...	GE
Aerosmith	Logged In	Lacey	F	107	Castaneda	220.39465	free	El Campo, TX	PU
null	Logged In	Colin	M	0	Larson	null	free	Dallas-Fort Worth...	GE
Amy Winehouse	Logged In	Colin	M	1	Larson	201.50812	free	Dallas-Fort Worth...	PU
Drake / Kanye Wes...	Logged In	Sofia	F	267	Gordon	357.66812	paid	Rochester, MN	PU
Bob Dylan	Logged In	Gianna	F	23	Campos	256.96608	paid	Mobile, AL	PU
The Bar-Kays	Logged In	Maverick	M	191	Santiago	164.23138	paid	Orlando-Kissimmee...	PU
Black Kids	Logged In	Payton	F	83	Campbell	251.48036	free	Los Angeles-Long ...	PU
null	Logged In	Sofia	F	268	Gordon	null	paid	Rochester, MN	GE
null	Logged In	Maverick	M	192	Santiago	null	paid	Orlando-Kissimmee...	PU

only showing top 20 rows

Sub Part - 2: View Data

In [6]:
```
print("Number of Entry in the dataset is {}.".format(user_log.count()))
```

```
Number of Entry in the dataset is 543705.
```

In [7]:
```
user_log.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 [8]:
```
#print the first row as example
user_log.take(1)
```

Out[8]:
```
[Row(artist='Martin Orford', auth='Logged In', firstName='Joseph', gender='M', itemInSession=20, lastName='Morales', length=597.55057, level='free', location='Corpus Christi, TX', method='PUT', page='NextSong', registration=1532063507000, sessionId=292, song='Grand Designs', status=200, ts=1538352011000, userAgent='"Mozilla/5.0 (Macintosh; Intel Mac OS X 10_9_4) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/36.0.1985.125 Safari/537.36"', userId='293')]
```

In [9]:
```
user_log_pandas = user_log.toPandas()
```

First General Analysis

In [10]:
```
#Statistics of Numeric Columns
user_log_pandas.describe()
```

Out[10]:

	itemInSession	length	registration	sessionId	status	ts
count	543705.000000	432877.000000	5.280050e+05	543705.000000	543705.000000	5.437050e+05
mean	107.306291	248.664593	1.535523e+12	2040.814353	210.018291	1.540965e+12
std	116.723508	98.412670	3.078725e+09	1434.338931	31.471919	1.482057e+09
min	0.000000	0.783220	1.509854e+12	1.000000	200.000000	1.538352e+12
25%	26.000000	199.392200	1.534368e+12	630.000000	200.000000	1.539720e+12
50%	68.000000	234.004440	1.536556e+12	1968.000000	200.000000	1.541005e+12
75%	147.000000	276.793020	1.537612e+12	3307.000000	200.000000	1.542177e+12
max	1005.000000	3024.665670	1.543074e+12	4808.000000	404.000000	1.543622e+12

In [11]:
```
#This part is just investigation. It shouldn't be on the original code.
summary_table = pd.DataFrame(columns=['Column Name', 'Unique Values','Number of Unique
for column in user_log_pandas.columns:
    unique_values = user_log_pandas[column].unique()
    unique_count = len(unique_values)
    isnull_values = user_log_pandas[column].isnull().sum()
    summary_table = summary_table.append({'Column Name': column, 'Unique Values': uniqu
summary_table
```

```
/tmp/ipykernel_9203/1628209480.py:7: FutureWarning: The frame.append method is deprecat
ed and will be removed from pandas in a future version. Use pandas.concat instead.
  summary_table = summary_table.append({'Column Name': column, 'Unique Values': unique_
values, 'Number of Unique Value': unique_count, 'Number of Null': isnull_values}, ignor
e_index=True)
/tmp/ipykernel_9203/1628209480.py:7: FutureWarning: The frame.append method is deprecat
ed and will be removed from pandas in a future version. Use pandas.concat instead.
  summary_table = summary_table.append({'Column Name': column, 'Unique Values': unique_
values, 'Number of Unique Value': unique_count, 'Number of Null': isnull_values}, ignor
e_index=True)
/tmp/ipykernel_9203/1628209480.py:7: FutureWarning: The frame.append method is deprecat
ed and will be removed from pandas in a future version. Use pandas.concat instead.
  summary_table = summary_table.append({'Column Name': column, 'Unique Values': unique_
values, 'Number of Unique Value': unique_count, 'Number of Null': isnull_values}, ignor
e_index=True)
/tmp/ipykernel_9203/1628209480.py:7: FutureWarning: The frame.append method is deprecat
ed and will be removed from pandas in a future version. Use pandas.concat instead.
  summary_table = summary_table.append({'Column Name': column, 'Unique Values': unique_
values, 'Number of Unique Value': unique_count, 'Number of Null': isnull_values}, ignor
e_index=True)
/tmp/ipykernel_9203/1628209480.py:7: FutureWarning: The frame.append method is deprecat
ed and will be removed from pandas in a future version. Use pandas.concat instead.
  summary_table = summary_table.append({'Column Name': column, 'Unique Values': unique_
values, 'Number of Unique Value': unique_count, 'Number of Null': isnull_values}, ignor
e_index=True)
/tmp/ipykernel_9203/1628209480.py:7: FutureWarning: The frame.append method is deprecat
ed and will be removed from pandas in a future version. Use pandas.concat instead.
  summary_table = summary_table.append({'Column Name': column, 'Unique Values': unique_
values, 'Number of Unique Value': unique_count, 'Number of Null': isnull_values}, ignor
e_index=True)
/tmp/ipykernel_9203/1628209480.py:7: FutureWarning: The frame.append method is deprecat
ed and will be removed from pandas in a future version. Use pandas.concat instead.
  summary_table = summary_table.append({'Column Name': column, 'Unique Values': unique_
values, 'Number of Unique Value': unique_count, 'Number of Null': isnull_values}, ignor
e_index=True)
/tmp/ipykernel_9203/1628209480.py:7: FutureWarning: The frame.append method is deprecat
ed and will be removed from pandas in a future version. Use pandas.concat instead.
  summary_table = summary_table.append({'Column Name': column, 'Unique Values': unique_
values, 'Number of Unique Value': unique_count, 'Number of Null': isnull_values}, ignor
e_index=True)
/tmp/ipykernel_9203/1628209480.py:7: FutureWarning: The frame.append method is deprecat
ed and will be removed from pandas in a future version. Use pandas.concat instead.
  summary_table = summary_table.append({'Column Name': column, 'Unique Values': unique_
values, 'Number of Unique Value': unique_count, 'Number of Null': isnull_values}, ignor
e_index=True)
/tmp/ipykernel_9203/1628209480.py:7: FutureWarning: The frame.append method is deprecat
ed and will be removed from pandas in a future version. Use pandas.concat instead.
  summary_table = summary_table.append({'Column Name': column, 'Unique Values': unique_
values, 'Number of Unique Value': unique_count, 'Number of Null': isnull_values}, ignor
e_index=True)
/tmp/ipykernel_9203/1628209480.py:7: FutureWarning: The frame.append method is deprecat
ed and will be removed from pandas in a future version. Use pandas.concat instead.
  summary_table = summary_table.append({'Column Name': column, 'Unique Values': unique_
values, 'Number of Unique Value': unique_count, 'Number of Null': isnull_values}, ignor
e_index=True)
/tmp/ipykernel_9203/1628209480.py:7: FutureWarning: The frame.append method is deprecat
ed and will be removed from pandas in a future version. Use pandas.concat instead.
  summary_table = summary_table.append({'Column Name': column, 'Unique Values': unique_
values, 'Number of Unique Value': unique_count, 'Number of Null': isnull_values}, ignor
e_index=True)
/tmp/ipykernel_9203/1628209480.py:7: FutureWarning: The frame.append method is deprecat
```

```
ed and will be removed from pandas in a future version. Use pandas.concat instead.
  summary_table = summary_table.append({'Column Name': column, 'Unique Values': unique_
values, 'Number of Unique Value': unique_count, 'Number of Null': isnull_values}, ignor
e_index=True)
/tmp/ipykernel_9203/1628209480.py:7: FutureWarning: The frame.append method is deprecat
ed and will be removed from pandas in a future version. Use pandas.concat instead.
  summary_table = summary_table.append({'Column Name': column, 'Unique Values': unique_
values, 'Number of Unique Value': unique_count, 'Number of Null': isnull_values}, ignor
e_index=True)
/tmp/ipykernel_9203/1628209480.py:7: FutureWarning: The frame.append method is deprecat
ed and will be removed from pandas in a future version. Use pandas.concat instead.
  summary_table = summary_table.append({'Column Name': column, 'Unique Values': unique_
values, 'Number of Unique Value': unique_count, 'Number of Null': isnull_values}, ignor
e_index=True)
/tmp/ipykernel_9203/1628209480.py:7: FutureWarning: The frame.append method is deprecat
ed and will be removed from pandas in a future version. Use pandas.concat instead.
  summary_table = summary_table.append({'Column Name': column, 'Unique Values': unique_
values, 'Number of Unique Value': unique_count, 'Number of Null': isnull_values}, ignor
e_index=True)
/tmp/ipykernel_9203/1628209480.py:7: FutureWarning: The frame.append method is deprecat
ed and will be removed from pandas in a future version. Use pandas.concat instead.
  summary_table = summary_table.append({'Column Name': column, 'Unique Values': unique_
values, 'Number of Unique Value': unique_count, 'Number of Null': isnull_values}, ignor
e_index=True)
/tmp/ipykernel_9203/1628209480.py:7: FutureWarning: The frame.append method is deprecat
ed and will be removed from pandas in a future version. Use pandas.concat instead.
  summary_table = summary_table.append({'Column Name': column, 'Unique Values': unique_
values, 'Number of Unique Value': unique_count, 'Number of Null': isnull_values}, ignor
e_index=True)
```

Out[11]:

	Column Name	Unique Values	Number of Unique Value	Number of Null
0	artist	[Martin Orford, John Brown's Body, Afroman, No...	21248	110828
1	auth	[Logged In, Logged Out, Cancelled, Guest]	4	0
2	firstName	[Joseph, Sawyer, Maverick, Gianna, Sofia, None...	346	15700
3	gender	[M, F, None]	3	15700
4	itemInSession	[20, 74, 184, 185, 22, 266, 186, 187, 188, 189...	1006	0
5	lastName	[Morales, Larson, Santiago, Campos, Gordon, No...	276	15700
6	length	[597.55057, 380.21179, 202.37016, nan, 194.533...	16680	110828
7	level	[free, paid]	2	0
8	location	[Corpus Christi, TX, Houston-The Woodlands-Sug...	193	15700
9	method	[PUT, GET]	2	0
10	page	[NextSong, Logout, Home, Login, Downgrade, Add...	22	0
11	registration	[1532063507000.0, 1538069638000.0, 15359534550...	449	15700
12	sessionId	[292, 97, 178, 245, 162, 442, 497, 38, 440, 19...	4590	0
13	song	[Grand Designs, Bulls, Because I Got High, Non...	80293	110828
14	status	[200, 307, 404]	3	0
15	ts	[1538352011000, 1538352025000, 1538352118000, ...	513108	0
16	userAgent	["Mozilla/5.0 (Macintosh; Intel Mac OS X 10_9_...	72	15700
17	userId	[293, 98, 179, 246, 163, , 175, 100, 39, 147, ...	449	0

Data Analysis and Modification of User-log Dataframe

Analysis 1: Gender Distribution

In [12]:
```
numof_women = user_log.filter(user_log.gender == 'F') \
    .select('userId', 'gender') \
    .dropDuplicates() \
    .count()
numof_men = user_log.filter(user_log.gender == 'M') \
    .select('userId', 'gender') \
    .dropDuplicates() \
    .count()
print("Number of women: {}\nnumber of men: {}".format(numof_women,numof_men))
```

```
Number of women: 198
number of men: 250
```

Analysis 2: Unique Number of Users

In [13]:
```
numof_unique_users = user_log.select('userId') \
    .dropDuplicates() \
    .count()
print("Number of unique user is {}.".format(numof_unique_users))
```

```
Number of unique user is 449.
```

Analysis 3: Users who downgrade their subscription.

In [14]:
```
numof_canceler = user_log.filter("page = 'Submit Downgrade'").count()
print("Number of people who cancelled their account is {}.".format(numof_canceler))
```

```
Number of people who cancelled their account is 117.
```

Analysis 4: How many user has no userId?

In [15]:
```
numof_rec_with_noId = user_log.groupBy("userId").count().orderBy("userId")
print("Number of record without userId is:\n")
numof_rec_with_noId.show(1)
```

```
Number of record without userId is:

+------+-----+
|userId|count|
+------+-----+
|      |15700|
+------+-----+
only showing top 1 row
```

Analysis 5: Most Popular Pages

In [16]:
```
# Example: Most visited pages
popular_pages = user_log.groupBy("page").count().orderBy(col("count").desc())
popular_pages.show(50)
```

```
+--------------------+------+
|                page| count|
+--------------------+------+
|            NextSong|432877|
|                Home| 27412|
|           Thumbs Up| 23826|
|     Add to Playlist| 12349|
|          Add Friend|  8087|
|         Roll Advert|  7773|
|               Login|  6011|
|              Logout|  5990|
|         Thumbs Down|  4911|
|           Downgrade|  3811|
|                Help|  3150|
|            Settings|  2964|
|               About|  1855|
|             Upgrade|   968|
|       Save Settings|   585|
|               Error|   519|
|      Submit Upgrade|   287|
|    Submit Downgrade|   117|
|              Cancel|    99|
|Cancellation Conf...|    99|
|            Register|    11|
| Submit Registration|     4|
+--------------------+------+
```

Analysis 6:Number of Users vs. States

In [17]:
```
# Example: Distribution of users by state
user_location_distribution = user_log.groupBy("location").count().orderBy(col("count").
user_location_distribution.show(10)
```

```
+--------------------+-----+
|            location|count|
+--------------------+-----+
|New York-Newark-J...|40156|
|Los Angeles-Long ...|34278|
|Boston-Cambridge-...|17574|
|                null|15700|
|Chicago-Napervill...|15194|
|San Francisco-Oak...|11428|
|Atlanta-Sandy Spr...|11211|
|Phoenix-Mesa-Scot...|11184|
|Dallas-Fort Worth...|11061|
|Denver-Aurora-Lak...| 9808|
+--------------------+-----+
only showing top 10 rows
```

Analysis 7: Most Popular Artists

In [18]:
```
# Example: Most listened-to artists
popular_artists = user_log.groupBy("artist").count().orderBy(col("count").desc())
popular_artists.show(10)
```

```
+--------------------+------+
|              artist| count|
+--------------------+------+
|                null|110828|
|       Kings Of Leon|  3497|
|            Coldplay|  3439|
|Florence + The Ma...|  2314|
|                Muse|  2194|
|       Dwight Yoakam|  2187|
|      The Black Keys|  2160|
|            BjÃÂ¶rk|  2150|
|       Justin Bieber|  2096|
|        Jack Johnson|  2049|
+--------------------+------+
only showing top 10 rows
```

Analysis 8: Most Popular Browsers

In [19]:
```
# Example: Distribution of user agents
user_agent_distribution = user_log.groupBy("userAgent").count().orderBy(col("count").de
user_agent_distribution.show(10)
```

```
+--------------------+-----+
|           userAgent|count|
+--------------------+-----+
|"Mozilla/5.0 (Mac...|46082|
|Mozilla/5.0 (Wind...|39456|
|"Mozilla/5.0 (Win...|38551|
|"Mozilla/5.0 (Win...|31702|
|"Mozilla/5.0 (Mac...|31653|
|"Mozilla/5.0 (Mac...|24357|
|"Mozilla/5.0 (Mac...|18384|
|"Mozilla/5.0 (Mac...|17518|
|Mozilla/5.0 (Maci...|16137|
|                null|15700|
+--------------------+-----+
only showing top 10 rows
```

Looks like Mozilla has domination on the app.

Analysis 9: Most Popular Songs

In [20]:
```
# Example: Most listened-to artists
popular_artists = user_log.groupBy("song").count().orderBy(col("count").desc())
popular_artists.show(10)
```

```
+--------------------+------+
|                song| count|
+--------------------+------+
|                null|110828|
|      You're The One|  2219|
|                Undo|  1938|
|             Revelry|  1613|
|       Sehr kosmisch|  1341|
|Horn Concerto No....|  1236|
|Dog Days Are Over...|  1048|
|             Secrets|   916|
|        Use Somebody|   894|
|              Canada|   836|
+--------------------+------+
only showing top 10 rows
```

Analysis 10: Hourly Average Listened Music

In [21]:
```
get_hour = udf(lambda x: datetime.datetime.fromtimestamp(x / 1000.0). hour)
```

In [22]:
```
user_log = user_log.withColumn("hour", get_hour(user_log.ts))
```

In [23]:
```
user_log.head()
```

Out[23]:
```
Row(artist='Martin Orford', auth='Logged In', firstName='Joseph', gender='M', itemInSes
sion=20, lastName='Morales', length=597.55057, level='free', location='Corpus Christi,
TX', method='PUT', page='NextSong', registration=1532063507000, sessionId=292, song='Gr
and Designs', status=200, ts=1538352011000, userAgent='"Mozilla/5.0 (Macintosh; Intel M
ac OS X 10_9_4) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/36.0.1985.125 Safari/537.
36"', userId='293', hour='0')
```

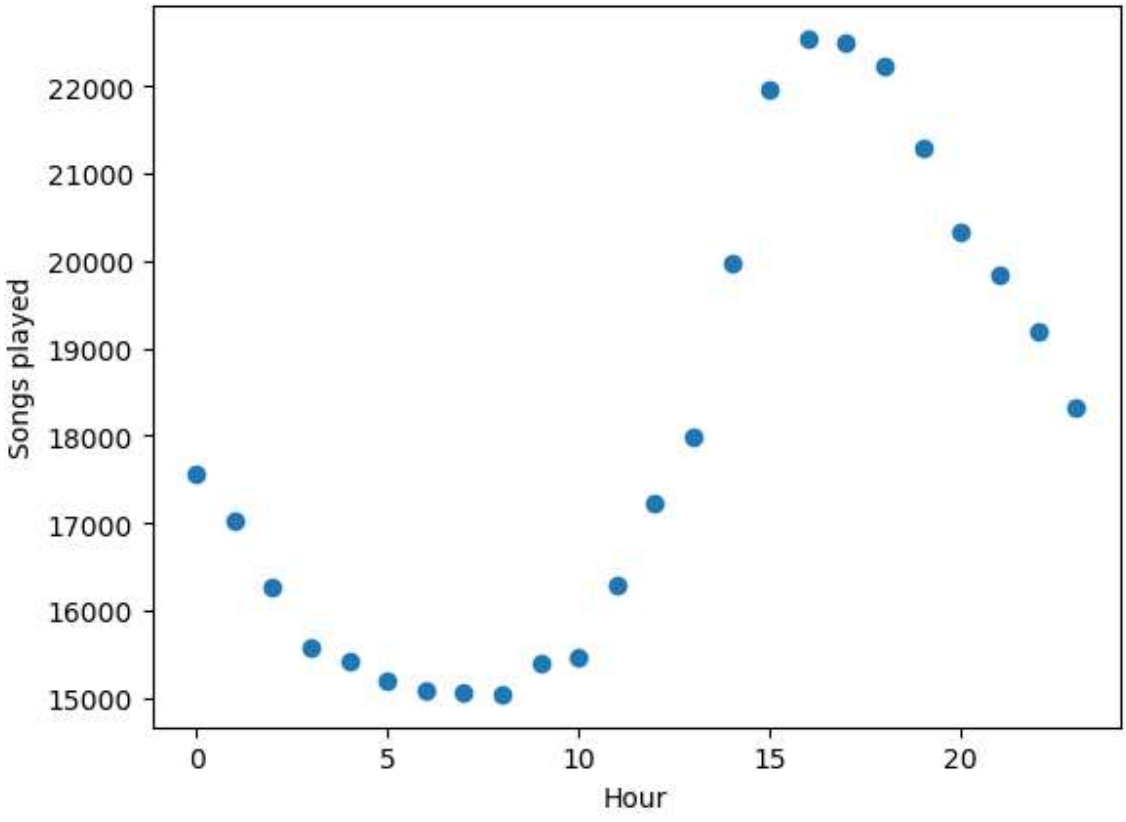
In [24]:
```
songs_in_hour = user_log.filter(user_log.page == "NextSong").groupby(user_log.hour).cou
```

In [25]:
```
songs_in_hour.show()
```

```
+----+-----+
|hour|count|
+----+-----+
|   0|17555|
|   1|17038|
|   2|16265|
|   3|15581|
|   4|15430|
|   5|15203|
|   6|15094|
|   7|15069|
|   8|15045|
|   9|15410|
|  10|15476|
|  11|16290|
|  12|17230|
|  13|17995|
|  14|19978|
|  15|21953|
|  16|22542|
|  17|22500|
|  18|22223|
|  19|21296|
+----+-----+
only showing top 20 rows
```

In [26]:
```
songs_in_hour_pd = songs_in_hour.toPandas()
songs_in_hour_pd.hour = pd.to_numeric(songs_in_hour_pd.hour)
```

In [27]:
```
plt.scatter(songs_in_hour_pd["hour"], songs_in_hour_pd["count"])
plt.xlabel("Hour")
plt.ylabel("Songs played");
```



Data Wrangling 1: Transformating Registration dates

In [28]:
```
user_log = user_log.withColumn("registration_timestamp", from_unixtime(col("registratio
user_log.select("registration", "registration_timestamp").show(10)
```

```
+-------------+---------------------+
| registration|registration_timestamp|
+-------------+---------------------+
|1532063507000|   2018-07-20 05:11:47|
|1538069638000|   2018-09-27 17:33:58|
|1535953455000|   2018-09-03 05:44:15|
|1535953455000|   2018-09-03 05:44:15|
|1535931018000|   2018-09-02 23:30:18|
|1533175710000|   2018-08-02 02:08:30|
|         null|                 null|
|         null|                 null|
|         null|                 null|
|         null|                 null|
+-------------+---------------------+
only showing top 10 rows
```

Data Wrangling 2: Transforming timestamps of Logs

In [29]:
```
user_log = user_log.withColumn("ts_timestamp", from_unixtime(col("ts") / 1000))
user_log.select("ts", "ts_timestamp").show(10)
```

```
+-------------+-------------------+
|           ts|       ts_timestamp|
+-------------+-------------------+
|1538352011000|2018-10-01 00:00:11|
|1538352025000|2018-10-01 00:00:25|
|1538352118000|2018-10-01 00:01:58|
|1538352119000|2018-10-01 00:01:59|
|1538352124000|2018-10-01 00:02:04|
|1538352125000|2018-10-01 00:02:05|
|1538352148000|2018-10-01 00:02:28|
|1538352151000|2018-10-01 00:02:31|
|1538352168000|2018-10-01 00:02:48|
|1538352169000|2018-10-01 00:02:49|
+-------------+-------------------+
only showing top 10 rows
```

Data Wrangling 3: Remove rows with no userId

In [30]:
```
user_log_valid = user_log.dropna(how = "any", subset = ["userId", "sessionId"])
```

In [31]:
```
user_log_valid.count()
```

Out[31]:
```
543705
```

In [32]:
```
user_log.select("userId").dropDuplicates().sort("userId").show(10)
```

```
+------+
|userId|
+------+
|      |
|    10|
|   100|
|100001|
|100002|
|100003|
|100004|
|100005|
|100006|
|100007|
+------+
only showing top 10 rows
```

In [33]:
```
user_log_valid = user_log_valid.filter(user_log_valid["userId"] != "")
```

In [34]:
```
user_log_valid.count()
```

Out[34]: 528005

In [35]:
```
user_log_valid.select("userId").dropDuplicates().sort("userId").count()
```

Out[35]: 448

Data Wrangling 4: Create User Dataframe for Churn Activity

In [36]:
```
### Burada iki adet sütun oluşturuluyor. Biri page sütununda Submit downgrade olan kull
### İkincisi si page sütununda Cancellation Confirmation olan kullanıcıların tüm kayıtl

# add downgrade and churn

user_log_valid = user_log_valid.withColumn('downgrade', when(user_log_valid.page == 'Su
user_log_valid = user_log_valid.withColumn('churn', when(user_log_valid.page == 'Cancel
user_log_valid = user_log_valid.withColumn('user_downgrade', Fmax('downgrade').over(Win
user_log_valid = user_log_valid.withColumn('user_churn', Fmax('churn').over(Window.part
user_log_valid =user_log_valid.drop(col("downgrade"))
user_log_valid = user_log_valid.drop(col("churn"))
user_log_valid.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)
 |-- hour: string (nullable = true)
 |-- registration_timestamp: string (nullable = true)
 |-- ts_timestamp: string (nullable = true)
 |-- user_downgrade: integer (nullable = true)
 |-- user_churn: integer (nullable = true)
```

In [37]:
```
user_log_valid = user_log_valid.withColumn('churn', when((user_log_valid.user_churn ==
user_log_valid = user_log_valid.drop(col("user_churn"))
user_log_simplified = user_log_valid.drop(col("user_downgrade"))
```

In [38]:
```
churn_matrice = user_log_simplified.dropDuplicates(["userId"])
churn_matrice.toPandas().head(5)
```

Out[38]:

	artist	auth	firstName	gender	itemInSession	lastName	length	level	location	meth
0	Charttraxx Karaoke	Logged In	Colin	M	0	Larson	225.17506	paid	Dallas-Fort Worth-Arlington, TX	P
1	None	Logged In	Emily	F	0	Fisher	NaN	free	Syracuse, NY	G
2	None	Logged In	Cason	M	0	Smith	NaN	free	Monroe, LA	G
3	None	Logged In	Leyla	F	0	Barnes	NaN	free	Flint, MI	G
4	OneRepublic	Logged In	Piper	F	0	Cook	224.67873	free	Little Rock-North Little Rock-Conway, AR	P

5 rows × 22 columns

In [39]:
```
user_log_valid.show(10)
```

```
+---------------+---------+---------+------+-------------+--------+---------+-----+---
----------------+------+----------+-------------+---------+--------------------+-----
-+-------------+--------------------+------+----+---------------------+--------------
----+-------------+-----+
|          artist|     auth|firstName|gender|itemInSession|lastName|   length|level|
location|method|       page| registration|sessionId|                song|status|
ts|           userAgent|userId|hour|registration_timestamp|       ts_timestamp|user_dow
ngrade|churn|
+---------------+---------+---------+------+-------------+--------+---------+-----+---
----------------+------+----------+-------------+---------+--------------------+-----
-+-------------+--------------------+------+----+---------------------+--------------
----+-------------+-----+
|            null|Logged In|    Colin|     M|            0|  Larson|     null| free|Dal
las-Fort Worth...|   GET|       Home|1537982255000|      497|                null|   20
0|1538352241000|Mozilla/5.0 (Wind...|   100|   0|   2018-09-26 17:17:35|2018-10-01 00:0
4:01|             0|    0|
|   Amy Winehouse|Logged In|    Colin|     M|            1|  Larson|201.50812| free|Dal
las-Fort Worth...|   PUT|   NextSong|1537982255000|      497|    Teach Me Tonight|   20
0|1538352259000|Mozilla/5.0 (Wind...|   100|   0|   2018-09-26 17:17:35|2018-10-01 00:0
4:19|             0|    0|
|     The Strokes|Logged In|    Colin|     M|            2|  Larson|187.34975| free|Dal
las-Fort Worth...|   PUT|   NextSong|1537982255000|      497|  You Only Live Once|   20
0|1538352460000|Mozilla/5.0 (Wind...|   100|   0|   2018-09-26 17:17:35|2018-10-01 00:0
7:40|             0|    0|
|            null|Logged In|    Colin|     M|            3|  Larson|     null| free|Dal
las-Fort Worth...|   GET|Roll Advert|1537982255000|      497|                null|   20
0|1538352460000|Mozilla/5.0 (Wind...|   100|   0|   2018-09-26 17:17:35|2018-10-01 00:0
7:40|             0|    0|
|Charly GarcÃ‍Âa|Logged In|    Colin|     M|            4|  Larson|223.05914| free|Dall
as-Fort Worth...|   PUT|   NextSong|1537982255000|      497|        AsesÃ‍Âname|   200|
1538352647000|Mozilla/5.0 (Wind...|   100|   0|   2018-09-26 17:17:35|2018-10-01 00:10:
47|             0|    0|
|            null|Logged In|    Colin|     M|            5|  Larson|     null| free|Dal
las-Fort Worth...|   GET|Roll Advert|1537982255000|      497|                null|   20
0|1538352650000|Mozilla/5.0 (Wind...|   100|   0|   2018-09-26 17:17:35|2018-10-01 00:1
0:50|             0|    0|
|   Dwight Yoakam|Logged In|    Colin|     M|            6|  Larson| 239.3073| free|Dal
las-Fort Worth...|   PUT|   NextSong|1537982255000|      497|      You're The One|   20
0|1538352870000|Mozilla/5.0 (Wind...|   100|   0|   2018-09-26 17:17:35|2018-10-01 00:1
4:30|             0|    0|
|            null|Logged In|    Colin|     M|            7|  Larson|     null| free|Dal
las-Fort Worth...|   GET|Roll Advert|1537982255000|      497|                null|   20
0|1538352902000|Mozilla/5.0 (Wind...|   100|   0|   2018-09-26 17:17:35|2018-10-01 00:1
5:02|             0|    0|
| Bobby Valentino|Logged In|    Colin|     M|            8|  Larson|258.63791| free|Dal
las-Fort Worth...|   PUT|   NextSong|1537982255000|      497|           Slow Down|   20
0|1538353109000|Mozilla/5.0 (Wind...|   100|   0|   2018-09-26 17:17:35|2018-10-01 00:1
8:29|             0|    0|
|John Butler Trio|Logged In|    Colin|     M|            9|  Larson|  227.082| free|Dal
las-Fort Worth...|   PUT|   NextSong|1537982255000|      497|Losing You (Album...|   20
0|1538353367000|Mozilla/5.0 (Wind...|   100|   0|   2018-09-26 17:17:35|2018-10-01 00:2
2:47|             0|    0|
+---------------+---------+---------+------+-------------+--------+---------+-----+---
----------------+------+----------+-------------+---------+--------------------+-----
-+-------------+--------------------+------+----+---------------------+--------------
----+-------------+-----+
only showing top 10 rows
```

Feautre Engineering

Custom Function for Page Statistics

In [40]:
```
def calculate_correlation(comparison_matrice, main_matrice):
    main_df = main_matrice.join(comparison_matrice, on='userId')
    correlation = main_df.stat.corr("churn", "count")
    return "Correlation: {}".format(correlation)
```

Effect Analysis 1: Number of Sessions vs. Churn

In [41]:
```
numof_ses_byuser = user_log_simplified.groupby('userId').count().sort('count')

calculate_correlation(numof_ses_byuser,churn_matrice)
```

Out[41]:
```
'Correlation: 0.17380290006145288'
```

Effect Analysis 2: Average Session Time vs. Churn

In [42]:
```
from pyspark.sql import functions as F
from pyspark.sql.functions import sum
from pyspark.sql.functions import count, col
#from pyspark.sql.functions import col, max as spark_max, min as spark_min
# Define a window specification based on userId and sessionId
windowSpec = Window.partitionBy("userId", "sessionId")

# Create a new column "songs_in_session" counting the number of songs in each session
session_valid = user_log_valid.withColumn("sessionmax", F.max("ts").over(windowSpec))
session_valid = session_valid.withColumn("sessionmin", F.min("ts").over(windowSpec))

session_valid = session_valid.withColumn("session_time", col("sessionmax") - col("sessi


session_valid_matrice = session_valid.dropDuplicates(["userId"])
session_valid_matrice = session_valid_matrice['userId','session_time'].groupBy('userId'

# Show the result
calculate_correlation(session_valid_matrice,churn_matrice)
```

Out[42]:
```
'Correlation: 0.04041638235878587'
```

Effect Analysis 3: Total Spent Time on Platform vs. Churn

In [43]:
```
# Add a new column 'current_timestamp' with the current timestamp
time_spent = user_log_valid.withColumn("current_timestamp", F.current_timestamp())

# Calculate the time difference in days between registration and today
time_spent = time_spent.withColumn(
    "time_spent_since_registration",
    F.datediff("current_timestamp", F.from_unixtime(col("registration") / 1000))  # Ass
)
max_time_spent_per_user = time_spent.groupBy("userId").agg(F.max("time_spent_since_regi
```

```
# Show the updated DataFrame
max_time_spent_per_user = max_time_spent_per_user.select("userId", "count")
calculate_correlation(max_time_spent_per_user,churn_matrice)
```

Out[43]: 'Correlation: 0.09536557400977332'

Effect Analysis 4: Number of Songs Played vs. Churn

In [44]:
```
# Group by userId and count the occurrences of "NextSong " for each user
next_song_by_user = user_log_valid.groupBy("userId").agg(count(when(col("page") == "Nex

calculate_correlation(next_song_by_user,churn_matrice)
```

Out[44]: 'Correlation: 0.16859515722502344'

Effect Analysis 5: Number of Thumbs Up vs. Churn

In [45]:
```
from pyspark.sql.functions import count, col
# Filter rows where the page is "Thumbs Up"

thumbs_up_by_user = user_log_valid.groupBy("userId").agg(count(when(col("page") == "Thu

calculate_correlation(thumbs_up_by_user,churn_matrice)
```

Out[45]: 'Correlation: 0.13467096155675198'

In [46]:
```
thumbs_up_by_user.sort('userId').show(5)
```

```
+------+-----+
|userId|count|
+------+-----+
|    10|   17|
|   100|  143|
|100001|    5|
|100002|    8|
|100003|   16|
+------+-----+
only showing top 5 rows
```

Effect Analysis 6: Number of Thumbs Down vs. Churn

In [47]:
```
# Filter rows where the page is "Thumbs Down"
thumbs_down_by_user = user_log_valid.groupBy("userId").agg(count(when(col("page") == "T

calculate_correlation(thumbs_down_by_user,churn_matrice)
```

Out[47]: 'Correlation: 0.19568222201099328'

In [48]:
```
thumbs_down_by_user.sort('userId').show(5)
```

```
+------+-----+
|userId|count|
+------+-----+
|    10|    1|
|   100|   37|
|100001|    1|
|100002|    2|
|100003|   10|
+------+-----+
only showing top 5 rows
```

Effect Analysis 7: Ratio of Thumbs-up and Thumbs-down vs. Churn

In [49]:
```
thumbs_up_by_user2 = user_log_valid.groupBy("userId").agg(count(when(col("page") == "Th

thumbs_down_by_user3 = user_log_valid.groupBy("userId").agg(count(when(col("page") == "

merged_df = thumbs_down_by_user3.join(thumbs_up_by_user2, on='userId', how="outer")
merged_df = merged_df.withColumn("count",when((col("count3") == 0) | (col("count2") ==
merged_df =merged_df.drop(col("count2"))
merged_df =merged_df.drop(col("count3"))

#merged_df.show()

calculate_correlation(merged_df,churn_matrice)
```

Out[49]:
```
'Correlation: 0.034357839449410225'
```

In [50]:
```
merged_df.sort('userId').show(5)
```

```
+------+-----------------+
|userId|            count|
+------+-----------------+
|    10|             17.0|
|   100|3.864864864864865|
|100001|              5.0|
|100002|              4.0|
|100003|              1.6|
+------+-----------------+
only showing top 5 rows
```

Effect Analysis 8: Number of Downgrade Visit vs. Churn

In [51]:
```
# Group by userId and count the occurrences of "Downgrade" for each user
downgrade_by_user = user_log_valid.groupBy("userId").agg(count(when(col("page") == "Dow

calculate_correlation(downgrade_by_user,churn_matrice)
```

Out[51]:
```
'Correlation: 0.1897378989219716'
```

Effect Analysis 9: Error on Website vs. Churn

In [52]:
```
#
error_page_by_user = user_log_valid.groupBy("userId").agg(count(when((col("page") == "E
```

```
#
calculate_correlation(error_page_by_user,churn_matrice)
```

Out[52]: 'Correlation: 0.11851348984507289'

Effect Analysis 10: Average Songs Played in Session vs. Churn

In [53]:
```
from pyspark.sql import functions as F
from pyspark.sql.functions import sum

# Define a window specification based on userId and sessionId
windowSpec = Window.partitionBy("userId", "sessionId")

# Create a new column "songs_in_session" counting the number of songs in each session
song_valid = user_log_valid.withColumn("songs_in_session", F.count("song").over(windowS

# Calculate the average number of songs per session for each user
song_valid = song_valid.groupBy("userId", "sessionId").agg(F.avg("songs_in_session").al
song_valid = song_valid.groupBy("userId").agg(sum("count2").alias("count"))

# Show the result
calculate_correlation(song_valid,churn_matrice)
```

Out[53]: 'Correlation: 0.16859515722502344'

Effect Analysis 11: Number of Friends added vs. Churn

In [54]:
```
# Group by userId and count the occurrences of "Add Friend" for each user
addfriend_by_user = user_log_valid.groupBy("userId").agg(count(when(col("page") == "Add

calculate_correlation(addfriend_by_user,churn_matrice)
```

Out[54]: 'Correlation: 0.14852737080612036'

Effect Analysis 12: Help Page Visits vs. Churn

In [55]:
```
help_by_user = user_log_valid.groupBy("userId").agg(count(when(col("page") == "Help", T


calculate_correlation(help_by_user,churn_matrice)
```

Out[55]: 'Correlation: 0.16919942387953402'

Effect Analysis 13: Number of Songs added to Playlist vs. Churn

In [56]:
```
playlist_by_user = user_log_valid.groupBy("userId").agg(count(when(col("page") == "Add

calculate_correlation(playlist_by_user,churn_matrice)
```

Out[56]: 'Correlation: 0.15207017575648235'

Effect Analysis 14: Number of Advertisement Seen per Session vs. Churn

In [57]:
```
advert_by_user = user_log_valid.groupBy("userId").agg(count(when(col("page") == "Roll A

calculate_correlation(advert_by_user,churn_matrice)
```

Out[57]:
```
'Correlation: 0.29078291552497315'
```

In [58]:
```
advert_by_user.show()
```

```
+------+-----+
|userId|count|
+------+-----+
|   296|    9|
|   125|    6|
|    51|    1|
|   124|    1|
|     7|   35|
|   169|   13|
|   205|    0|
|   272|   12|
|    54|    0|
|   282|   37|
|   232|   38|
|   234|   30|
|    15|    0|
|   155|    0|
|   154|    9|
|   132|    5|
|   200|    0|
|   101|    6|
|    11|   10|
|   279|   11|
+------+-----+
only showing top 20 rows
```

Merging Features

In [59]:
```
churn_data = churn_matrice['userId','churn']
```

In [60]:
```
# Number of Sessions vs. Churn                                    numof_ses_byuser
# Average Session Time vs. Churn                                  session_valid_matric
# Total Spent Time on Platform vs. Churn                          max_time_spent_per_u
# Number of Songs Played vs. Churn                                next_song_by_user
# Number of Thumbs Up vs. Churn                                   thumbs_up_by_user
# Number of Thumbs Down vs. Churn                                 thumbs_down_by_user
# Ratio of Thumbs-up and Thumbs-down vs. Churn                    merged_df
# Number of Downgrade Visit vs. Churn                             downgrade_by_user
# Error on Website vs. Churn                                      error_page_by_user
# Average Songs Played in Session vs. Churn                       song_valid
# Number of Friends added vs. Churn                               addfriend_by_user
# Help Page Visits vs. Churn                                      help_by_user
# Number of Songs added to Playlist vs. Churn                     playlist_by_user
# Number of Advertisement Seen per Session vs. Churn              advert_by_user


# Define the array of dataframe names
dataframe_names = [
    "churn_data",
    "numof_ses_byuser",
    "session_valid_matrice",
```

```
    "max_time_spent_per_user",
    "next_song_by_user",
    "thumbs_up_by_user",
    "thumbs_down_by_user",
    "merged_df",
    "downgrade_by_user",
    "error_page_by_user",
    "song_valid",
    "addfriend_by_user",
    "help_by_user",
    "playlist_by_user",
    "advert_by_user"
]

# Function to join dataframes based on "userId"
def join_dataframes(df_names):
    # Initial dataframe
    result_df = globals()[df_names[0]]

    # Join the rest of the dataframes
    for df_name in df_names[1:]:
        result_df = result_df.join(globals()[df_name].withColumnRenamed("count", df_nam

    return result_df

# Call the function with the dataframe names array
result_df = join_dataframes(dataframe_names)
```

In [61]:
```
# Show the null counts
null_counts = result_df.select([sum(col(c).isNull().cast("int")).alias(c) for c in resu
null_counts_pandas = null_counts.toPandas()
null_counts_pandas
```

Out[61]:

	userId	churn	numof_ses_byuser	session_valid_matrice	max_time_spent_per_user	next_song_by_user
0	0	0	0	0	0	0

So, we confirmed that there is no missing value in the merged dataframe. We are ready for ML Pipeline build-up.

General View of Final Dataframe

In [62]:
```
result_df_pandas = result_df.toPandas()
```

In [63]:
```
result_df_pandas.head()
```

Out[63]:

	userId	churn	numof_ses_byuser	session_valid_matrice	max_time_spent_per_user	next_song_by_user
0	10	1	423	90144000	1890	360
1	100	0	3999	75377000	1892	3382
2	100001	1	134	12128000	1931	96
3	100002	1	177	3158000	1985	137
4	100003	1	827	14389000	1900	661

In [64]:
```
result_df_pandas.shape
```

Out[64]: (448, 16)

Modelling

In [65]:
```
from pyspark.ml import Pipeline
from pyspark.ml.feature import VectorAssembler, StandardScaler
from pyspark.ml.classification import RandomForestClassifier
from pyspark.ml.evaluation import MulticlassClassificationEvaluator

# Define input columns (excluding "userId" and "churn")
input_columns = ["numof_ses_byuser", "session_valid_matrice", "max_time_spent_per_user"
                 "next_song_by_user", "thumbs_up_by_user", "thumbs_down_by_user",
                 "merged_df", "downgrade_by_user", "error_page_by_user",
                 "song_valid", "addfriend_by_user", "help_by_user",
                 "playlist_by_user", "advert_by_user"]

# Create a VectorAssembler to assemble features into a single vector
vector_assembler = VectorAssembler(inputCols=input_columns, outputCol="features")

# Create a StandardScaler to normalize the input features
scaler = StandardScaler(inputCol="features", outputCol="scaled_features", withMean=True

# Create a RandomForestClassifier model
rf_classifier = RandomForestClassifier(labelCol="churn", featuresCol="scaled_features")

# Create a Pipeline with stages
pipeline = Pipeline(stages=[vector_assembler, scaler, rf_classifier])

# Split the data into training and testing sets (80% train, 20% test)
train_data, test_data = result_df.randomSplit([0.8, 0.2], seed=35)

# Fit the pipeline on the training data
model = pipeline.fit(train_data)

# Make predictions on the test data
predictions = model.transform(test_data)

# Evaluate the model using F1 score
evaluator = MulticlassClassificationEvaluator(labelCol="churn", metricName="f1")
f1_score = evaluator.evaluate(predictions)

# Print the F1 score
print(f"F1 Score: {f1_score}")
```

```
F1 Score: 0.6970649155411272
```

Last Steps: Grid Search with Cross Validation

In [ ]:

```
from pyspark.ml import Pipeline
from pyspark.ml.feature import VectorAssembler, StandardScaler
from pyspark.ml.classification import LogisticRegression, RandomForestClassifier, GBTCl
from pyspark.ml.evaluation import BinaryClassificationEvaluator
from pyspark.ml.tuning import ParamGridBuilder, CrossValidator
from pyspark.sql import SparkSession


input_cols = [
    'numof_ses_byuser', 'session_valid_matrice', 'max_time_spent_per_user',
    'next_song_by_user', 'thumbs_up_by_user', 'thumbs_down_by_user', 'merged_df',
    'downgrade_by_user', 'error_page_by_user', 'song_valid', 'addfriend_by_user',
    'help_by_user', 'playlist_by_user', 'advert_by_user'
]

output_col = 'churn'

# Assemble features into a vector
assembler = VectorAssembler(inputCols=input_cols, outputCol="features")

# Standardize features
scaler = StandardScaler(inputCol="features", outputCol="scaled_features")

# Define machine learning algorithms
lr = LogisticRegression(featuresCol="scaled_features", labelCol=output_col)
rf = RandomForestClassifier(featuresCol="scaled_features", labelCol=output_col)
gbt = GBTClassifier(featuresCol="scaled_features", labelCol=output_col)

# Create a pipeline for each algorithm
lr_pipeline = Pipeline(stages=[assembler, scaler, lr])
rf_pipeline = Pipeline(stages=[assembler, scaler, rf])
gbt_pipeline = Pipeline(stages=[assembler, scaler, gbt])

# Define parameter grids for each algorithm
lr_param_grid = ParamGridBuilder().addGrid(lr.regParam, [0.1, 0.01]).build()
rf_param_grid = ParamGridBuilder().addGrid(rf.maxDepth, [5, 10]).build()
gbt_param_grid = ParamGridBuilder().addGrid(gbt.maxDepth, [5, 10]).build()

# Create evaluators
evaluator = BinaryClassificationEvaluator(labelCol=output_col)

# Perform train-test split
train_data, test_data = result_df.randomSplit([0.8, 0.2], seed=35)

# Define cross-validator for each algorithm
lr_cv = CrossValidator(estimator=lr_pipeline, estimatorParamMaps=lr_param_grid, evaluat
rf_cv = CrossValidator(estimator=rf_pipeline, estimatorParamMaps=rf_param_grid, evaluat
gbt_cv = CrossValidator(estimator=gbt_pipeline, estimatorParamMaps=gbt_param_grid, eval

# Fit models
lr_model = lr_cv.fit(train_data)
rf_model = rf_cv.fit(train_data)
gbt_model = gbt_cv.fit(train_data)

# Make predictions
lr_predictions = lr_model.transform(test_data)
rf_predictions = rf_model.transform(test_data)
```

```
gbt_predictions = gbt_model.transform(test_data)

# Evaluate F1 Score
lr_f1 = evaluator.evaluate(lr_predictions)
rf_f1 = evaluator.evaluate(rf_predictions)
gbt_f1 = evaluator.evaluate(gbt_predictions)

# Display results
print("Logistic Regression F1 Score:", lr_f1)
print("Random Forest F1 Score:", rf_f1)
print("GBT F1 Score:", gbt_f1)
```

```
Logistic Regression F1 Score: 0.6724137931034483
Random Forest F1 Score: 0.7552565180824223
GBT F1 Score: 0.6833473507148865
```

In [69]:
```
# Access best hyperparameters
best_lr_params = lr_model.bestModel.stages[-1].extractParamMap()
best_rf_params = rf_model.bestModel.stages[-1].extractParamMap()
best_gbt_params = gbt_model.bestModel.stages[-1].extractParamMap()
```

In [70]:
```
# Display best hyperparameters
print("\nBest Logistic Regression Hyperparameters:")
for param, value in best_lr_params.items():
    print(f"{param.name}: {value}")

print("\nBest Random Forest Hyperparameters:")
for param, value in best_rf_params.items():
    print(f"{param.name}: {value}")

print("\nBest GBT Hyperparameters:")
for param, value in best_gbt_params.items():
    print(f"{param.name}: {value}")
```

```
Best Logistic Regression Hyperparameters:
aggregationDepth: 2
elasticNetParam: 0.0
family: auto
featuresCol: scaled_features
fitIntercept: True
labelCol: churn
maxBlockSizeInMB: 0.0
maxIter: 100
predictionCol: prediction
probabilityCol: probability
rawPredictionCol: rawPrediction
regParam: 0.01
standardization: True
threshold: 0.5
tol: 1e-06

Best Random Forest Hyperparameters:
bootstrap: True
cacheNodeIds: False
checkpointInterval: 10
featureSubsetStrategy: auto
featuresCol: scaled_features
impurity: gini
labelCol: churn
leafCol:
maxBins: 32
maxDepth: 5
maxMemoryInMB: 256
minInfoGain: 0.0
minInstancesPerNode: 1
minWeightFractionPerNode: 0.0
numTrees: 20
predictionCol: prediction
probabilityCol: probability
rawPredictionCol: rawPrediction
seed: -5387697053847413545
subsamplingRate: 1.0

Best GBT Hyperparameters:
cacheNodeIds: False
checkpointInterval: 10
featureSubsetStrategy: all
featuresCol: scaled_features
impurity: variance
labelCol: churn
leafCol:
lossType: logistic
maxBins: 32
maxDepth: 5
maxIter: 20
maxMemoryInMB: 256
minInfoGain: 0.0
minInstancesPerNode: 1
minWeightFractionPerNode: 0.0
predictionCol: prediction
probabilityCol: probability
rawPredictionCol: rawPrediction
seed: 3504127614838123891
stepSize: 0.1
subsamplingRate: 1.0
validationTol: 0.01
```

In [71]:
```
# Close the Spark session
spark.stop()
```

In [ ]: