In [1]:
```
# Starter code
from pyspark.sql import SparkSession
```

```
Starting Spark application
```

ID	YARN Application ID	Kind	State	Spark UI	
0	application_1569178844032_0001	pyspark	idle	Link (http://ip-172-31-44-119.us-east-2.compute.internal:20888/proxy/application_1569178844032_0001/)	2.compute.internal:8042/nod

```
SparkSession available as 'spark'.
```

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

In [3]:
```
# Read in full sparkify dataset
event_data = "s3n://udacity-dsnd/sparkify/sparkify_event_data.json"
df = spark.read.json(event_data)
df.head()
```

▸ Spark Job Progress

```
Row(artist='Popol Vuh', auth='Logged In', firstName='Shlok', gender='M', itemInSession=278, lastName
='Johnson', length=524.32934, level='paid', location='Dallas-Fort Worth-Arlington, TX', method='PUT',
page='NextSong', registration=1533734541000, sessionId=22683, song='Ich mache einen Spiegel - Dream Pa
rt 4', status=200, ts=1538352001000, userAgent='"Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.3
6 (KHTML, like Gecko) Chrome/36.0.1985.143 Safari/537.36"', userId='1749042')
```

In [7]:
```
# import libraries

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, asc, avg
from pyspark.sql.functions import sum as Fsum

from pyspark.sql.window import Window

import datetime

import numpy as np
#import pandas as pd
#%matplotlib inline
#import matplotlib.pyplot as plt



import re
#import seaborn as sns

from pyspark.sql.functions import explode, lit, min, max, split, isnull

from pyspark.ml import Pipeline
from pyspark.ml.classification import LogisticRegression, RandomForestClassifier, GBTClassifier, Decisi
from pyspark.ml.evaluation import MulticlassClassificationEvaluator
from pyspark.ml.feature import CountVectorizer, IDF, Normalizer, PCA, RegexTokenizer, StandardScaler, St
from pyspark.ml.regression import LinearRegression
from pyspark.ml.tuning import CrossValidator, ParamGridBuilder
from pyspark.ml.linalg import DenseVector, SparseVector
```

In [8]:
```
userlog=df
```

In [9]:
```
print('&&&&&&&')
print('detailed data demo')
print(userlog.head(n=3))   ## or use take
print('&&&&&&&')
print('type')
print(userlog.describe())
print('&&&&&&&')
print('detailed data demo 2')
print(userlog.show(n=3))
print('&&&&&&&')
print('Schema')
print(userlog.printSchema())
```

▸ Spark Job Progress

```
&&&&&&&
detailed data demo
[Row(artist='Popol Vuh', auth='Logged In', firstName='Shlok', gender='M', itemInSession=278, lastNam
e='Johnson', length=524.32934, level='paid', location='Dallas-Fort Worth-Arlington, TX', method='PU
T', page='NextSong', registration=1533734541000, sessionId=22683, song='Ich mache einen Spiegel - Dr
eam Part 4', status=200, ts=1538352001000, userAgent='"Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebK
it/537.36 (KHTML, like Gecko) Chrome/36.0.1985.143 Safari/537.36"', userId='1749042'), Row(artist='L
os Bunkers', auth='Logged In', firstName='Vianney', gender='F', itemInSession=9, lastName='Miller',
length=238.39302, level='paid', location='San Francisco-Oakland-Hayward, CA', method='PUT', page='Ne
xtSong', registration=1537500318000, sessionId=20836, song='MiÃ\x83Â©ntele', status=200, ts=15383520
02000, userAgent='"Mozilla/5.0 (Macintosh; Intel Mac OS X 10_9_4) AppleWebKit/537.36 (KHTML, like Ge
cko) Chrome/36.0.1985.143 Safari/537.36"', userId='1563081'), Row(artist='Lush', auth='Logged In', f
irstName='Vina', gender='F', itemInSession=109, lastName='Bailey', length=140.35546, level='paid', l
ocation='Hilo, HI', method='PUT', page='NextSong', registration=1536414505000, sessionId=4593, song
='Baby Talk', status=200, ts=1538352002000, userAgent='Mozilla/5.0 (Macintosh; Intel Mac OS X 10.9;
rv:31.0) Gecko/20100101 Firefox/31.0', userId='1697168')]
&&&&&&&
type
DataFrame[summary: string, artist: string, auth: string, firstName: string, gender: string, itemInSe
ssion: string, lastName: string, length: string, level: string, location: string, method: string, pa
ge: string, registration: string, sessionId: string, song: string, status: string, ts: string, userA
gent: string, userId: string]
&&&&&&&
detailed data demo 2
+-----------+---------+---------+------+-------------+--------+---------+-----+--------------------+
```

```
------+--------+-------------+---------+--------------------+------+-------------+-----------------
--+-------+
|     artist|     auth|firstName|gender|itemInSession|lastName|   length|level|            location|
method|    page| registration|sessionId|                song|status|           ts|           userAge
nt| userId|
+-----------+---------+---------+------+-------------+--------+---------+-----+--------------------+
------+--------+-------------+---------+--------------------+------+-------------+-----------------
--+-------+
|  Popol Vuh|Logged In|    Shlok|     M|          278| Johnson|524.32934| paid|Dallas-Fort Worth...|
PUT|NextSong|1533734541000|    22683|Ich mache einen S...|   200|1538352001000|"Mozilla/5.0 (Win...|
1749042|
|Los Bunkers|Logged In|  Vianney|     F|            9|  Miller|238.39302| paid|San Francisco-Oak...|
PUT|NextSong|1537500318000|    20836|         MiÃƒÂ©ntele|   200|1538352002000|"Mozilla/5.0 (Mac...|
1563081|
|       Lush|Logged In|     Vina|     F|          109|  Bailey|140.35546| paid|            Hilo, HI|
PUT|NextSong|1536414505000|     4593|           Baby Talk|   200|1538352002000|Mozilla/5.0 (Maci...|
1697168|
+-----------+---------+---------+------+-------------+--------+---------+-----+--------------------+
------+--------+-------------+---------+--------------------+------+-------------+-----------------
--+-------+
only showing top 3 rows

None
&&&&&&&
Schema
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)

None
```

Missing Value

In [10]:
```
print(userlog.count())
userlog_valid = userlog.dropna(how = "any", subset = ["userId", "sessionId"])
print(userlog_valid.count())
```

▸ Spark Job Progress

```
26259199
26259199
```

In [11]:
```
userlog_valid = userlog_valid.filter(userlog_valid["userId"] != "")
userlog_valid.count()
```

▸ Spark Job Progress

```
26259199
```

Define Churn

In [12]:
```
# add time to see the time clear
get_time = udf(lambda x: datetime.datetime.fromtimestamp(x / 1000.0).strftime("%Y-%m-%d %H:%M:%S"))
userlog_valid = userlog_valid.withColumn("time", get_time(userlog_valid.ts))
```

In [13]:
```
churnuser = userlog_valid.filter(userlog_valid.page=="Cancellation Confirmation").select("userId").dropl
churnuserlist = [(row['userId']) for row in churnuser.collect()]
userlog_valid = userlog_valid.withColumn("churn", userlog_valid.userId.isin(churnuserlist))
```

▸ Spark Job Progress

In [14]:
```
userlog_valid.show(n=3)
```

▸ Spark Job Progress

```
+-----------+---------+---------+------+-------------+--------+---------+-----+--------------------+--
----+--------+-------------+---------+--------------------+------+-------------+--------------------+-
------+------------------+-----+
|     artist|     auth|firstName|gender|itemInSession|lastName|   length|level|            location|me
thod|    page| registration|sessionId|                song|status|           ts|           userAgent|
userId|               time|churn|
+-----------+---------+---------+------+-------------+--------+---------+-----+--------------------+--
----+--------+-------------+---------+--------------------+------+-------------+--------------------+-
------+------------------+-----+
|  Popol Vuh|Logged In|    Shlok|     M|          278| Johnson|524.32934| paid|Dallas-Fort Worth...|
PUT|NextSong|1533734541000|    22683|Ich mache einen S...|   200|1538352001000|"Mozilla/5.0 (Win...|17
49042|2018-10-01 00:00:01| true|
|Los Bunkers|Logged In|  Vianney|     F|            9|  Miller|238.39302| paid|San Francisco-Oak...|
PUT|NextSong|1537500318000|    20836|         MiÃƒÂ©ntele|   200|1538352002000|"Mozilla/5.0 (Mac...|15
63081|2018-10-01 00:00:02|false|
|       Lush|Logged In|     Vina|     F|          109|  Bailey|140.35546| paid|            Hilo, HI|
PUT|NextSong|1536414505000|     4593|           Baby Talk|   200|1538352002000|Mozilla/5.0 (Maci...|16
97168|2018-10-01 00:00:02|false|
+-----------+---------+---------+------+-------------+--------+---------+-----+--------------------+--
----+--------+-------------+---------+--------------------+------+-------------+--------------------+-
------+------------------+-----+
only showing top 3 rows
```

Feature Engineering

In [15]:
```
## Feature 1: number of days since registration
user_max_ts = userlog_valid.groupby("userId").max("ts").sort("userId")
user_reg_ts = userlog_valid.select("userId", "registration").dropDuplicates().sort("userId")
user_reg_days = user_reg_ts.join(user_max_ts, user_reg_ts.userId == user_max_ts.userId).select(user_reg
```

In [16]:
```
## Feature 2: avg & min & max time per session
user_session_time = userlog_valid.groupby("userId", "sessionId").agg(((max(userlog_valid.ts)-min(userlo
user_session_time_stat = user_session_time.groupby("userId").agg(avg(user_session_time.sessionTime).ali
```

In [17]:
```
## Feature 3: number of songs per session
user_session_songs = userlog_valid.filter(userlog_valid.page=="NextSong").groupby("userId", "sessionId"
user_session_songs_avg = user_session_songs.groupby("userId").agg(avg(user_session_songs["count"]).alia
```

In [18]:
```
## Feature 4: number of sessions
user_session_count = userlog_valid.select("userId", "sessionId").dropDuplicates().groupby("userId").cou
user_session_count = user_session_count.withColumnRenamed("count", "sessionCount")
```

In [19]:
```
## Feature 5: gender
user_gender = userlog_valid.select("userId", "gender").dropDuplicates()
user_gender = user_gender.replace(["M", "F"], ["0", "1"], "gender")
user_gender = user_gender.select("userId", user_gender.gender.cast("int"))
```

In [20]:
```
## Feature 6: whether the user is currently a paid user
user_paid = userlog_valid.groupby("userId", "level").agg(max(userlog_valid.ts).alias("finalTime")).sort
user_recent_level_time = user_paid.groupby("userId").agg(max(user_paid.finalTime).alias("recent"))
user_recent_level = user_recent_level_time.join(user_paid, [user_paid.userId == user_recent_level_time.
user_recent_level = user_recent_level.replace(["free", "paid"], ["0", "1"], "level")
user_recent_level = user_recent_level.select("userId", user_recent_level.level.cast("int"))
```

In [21]:
```
## Feature 7: frequency of use of pages
# get all the type of page
page_list = [(row['page']) for row in userlog_valid.select("page").dropDuplicates().collect()]

# must remove the column which will cause data leakage
page_list.remove("Cancel")
page_list.remove("Cancellation Confirmation")

# caculate the total page each user view
user_page_view_count = userlog_valid.groupby("userId").count()
user_page_view_count = user_page_view_count.withColumnRenamed("count", "pageCount")


for page in page_list:
    col_name = "count" + page.replace(" ", "")
    view_count = userlog_valid.filter(userlog_valid.page==page).groupby("userId").count()
    view_count = view_count.withColumnRenamed("count", col_name).withColumnRenamed("userId", "userIdTemp
    user_page_view_count = user_page_view_count.join(view_count, user_page_view_count.userId==view_count
user_page_view_count = user_page_view_count.sort("userId")
user_page_view_count = user_page_view_count.fillna(0)
```

▸ Spark Job Progress

In [23]:
```
col_list = user_page_view_count.columns
col_list.remove("userId")
col_list.remove("pageCount")
freq_sql = "select userId"
for col in col_list:
    col_name = col.replace("count", "freq")
    sql_str = ", (" + col + "/(pageCount/100)) as " + col_name
    freq_sql = freq_sql + sql_str
freq_sql = freq_sql + " from user_page_view_count"
```

In [24]:
```
user_page_view_count.createOrReplaceTempView("user_page_view_count")
col_list = user_page_view_count.columns
col_list.remove("userId")
col_list.remove("pageCount")
freq_sql = "select userId"
```

In [25]:
```
for col in col_list:
    col_name = col.replace("count", "freq")
    sql_str = ", (" + col + "/(pageCount/100)) as " + col_name
    freq_sql = freq_sql + sql_str
freq_sql = freq_sql + " from user_page_view_count"
```

In [27]:
```
#user_page_view_freq = spark.sql(freq_sql) ## need to check
```

In [28]:
```
## Feature 8: how many singers have the user heard
user_artist_count = userlog_valid.filter(userlog_valid.page=="NextSong").select("userId", "artist").drop
user_artist_count = user_artist_count.withColumnRenamed("count", "aritstCount")
```

In [29]:
```
## churn
user_churn = userlog_valid.select("userId", "churn").dropDuplicates()
user_churn = user_churn.select("userId", user_churn.churn.cast("int"))
```

Preparation for the modeling

In [30]:
```
# put all the features dataframe into a list
features_list = []
features_list.append(user_reg_days)
features_list.append(user_session_time_stat)
features_list.append(user_session_songs_avg)
features_list.append(user_session_count)
features_list.append(user_gender)
features_list.append(user_recent_level)
#features_list.append(user_page_view_freq)
features_list.append(user_artist_count)
features_list.append(user_churn)
```

In [31]:
```
# prepare the final dataframe to join all the other features
df_final = userlog_valid.select("userId").dropDuplicates()
```

In [32]:
```
def features_merge(df1, df2):
    """
    This function is used to merge the feature using left join
    input: two data frame to be merged
    output: merged dataframe
    """
    df2 = df2.withColumnRenamed("userId", "userIdTemp")
    df = df1.join(df2, df1.userId == df2.userIdTemp, "left").drop("userIdTemp")
    return df
```

In [33]:
```
# use function to merge the features in the list
for feature in features_list:
    df_final = features_merge(df_final, feature)
```

In [34]:
```
# sort and view the final dataframe
df_final = df_final.sort("userId")
df_final.persist()
df_final.show(5)
```

▸ Spark Job Progress

```
+-------+------------------+------------------+------------------+------------------+-----------------
+------------+------+-----+-----------+-----+
| userId|            regDay|    avgSessionTime|    minSessionTime|    maxSessionTime|  avgSessionSongs
|sessionCount|gender|level|aritstCount|churn|
+-------+------------------+------------------+------------------+------------------+-----------------
+------------+------+-----+-----------+-----+
|1000025|100.46038194444445| 404.7931372549019| 7.433333333333334|1639.3666666666666|97.76470588235294
|          17|     0|    1|       1162|    1|
|1000035| 68.57350694444445|235.93636363636352|               0.0| 785.2166666666667|            64.05
|          22|     1|    1|        926|    0|
|1000083| 34.66885416666667|186.10454545454547| 4.866666666666666|            536.45|45.54545454545455
|          11|     0|    1|        427|    1|
|1000103| 59.81548611111111| 68.93333333333334|13.316666666666666|             162.4|             18.0
|           4|     1|    1|         69|    0|
|1000164|110.30630787037038|218.88981481481483|               0.0| 704.1333333333333| 56.8235294117647
|          18|     1|    1|        739|    0|
+-------+------------------+------------------+------------------+------------------+-----------------
+------------+------+-----+-----------+-----+
only showing top 5 rows
```

Modeling

Convert all the features to numeric.

In [35]:
```
num_features_list = df_final.columns[1:]
for f in num_features_list:
    f_name = f + "Num"
    df_final = df_final.withColumn(f_name, df_final[f].cast("float"))
    df_final = df_final.drop(f)
```

Put the features to be trained into a vector

In [36]:
```
assembler = VectorAssembler(inputCols=df_final.columns[1:-1], outputCol="NumFeatures")
data = assembler.transform(df_final)
```

In [40]:
```
#scaler = StandardScaler(inputCol="NumFeatures", outputCol="ScaledNumFeatures", withStd=True)
#scalerModel = scaler.fit(data)
#data = scalerModel.transform(data)
```

In [41]:
```
data = data.select(data.churnNum.alias("label"), data.NumFeatures.alias("features"))
```

In [42]:
```
train, test = data.randomSplit([0.9, 0.1], seed=42)
train = train.cache()
```

In [43]:
```
def model_fit(train, test, model):
    '''
    INPUTS:
    train (Spark df): training data
    test (Spark df): testing data
    model (string): 'LogisticRegression', 'DecisionTree', 'RandomForest', 'GradientBoosting'

    OUTPUT:
    None, prints out accuracy and f1 score for the model

    '''

    if model == 'LogisticRegression':
        ml = LogisticRegression()

    elif model == 'DecisionTree':
        ml = DecisionTreeClassifier()

    elif model == 'RandomForest':
        ml = RandomForestClassifier()

    elif model == 'GradientBoosting':
        ml = GBTClassifier()

    else:
        return "Please choose an appropriate model"


    classification = ml.fit(train)
    results = classification.transform(test)


    accuracy_evaluator = MulticlassClassificationEvaluator(metricName='accuracy')
    accuracy = accuracy_evaluator.evaluate(results.select('label', 'prediction'))

    f1_score_evaluator = MulticlassClassificationEvaluator(metricName='f1')
    f1_score = f1_score_evaluator.evaluate(results.select('label', 'prediction'))

    print('For {}, the accuracy is {:.2%} and the F-1 score is {}'\
    .format(model, accuracy, f1_score))
```

In [ ]:
```
for model in ['LogisticRegression', 'DecisionTree', 'RandomForest', 'GradientBoosting']:
    model_fit(train, test, model)
```