Team Members

Name :	Himanshu Kumar	Akhilesh Bangalore Chandrashekar	Aakash Shetty
NetID:	hk3427	ab10138	aks9108
University ID :	N18516342	N11937821	N17739166
Date of Submission :	17th May 2022		

Abstract

Many businesses now use a subscription-based business model for their businesses. Attrition analysis provides such companies with a better understanding of their customer retention and ways of keeping their customers. Attrition analysis reduces customer churn rates and helps a business understand the steps necessary for preventing voluminous loss of revenue due to customer churn. Churn rate is the rate at which customers stop doing business with a company over a given period of time. Churn may also apply to the number of subscribers who cancel or don't renew a subscription. The higher your churn rate, the more customers stop buying from your business. Understanding your customer churn is essential to evaluating the effectiveness of your marketing efforts and the overall satisfaction of your customers. Due to the popularity of subscription business models, it's critical for many businesses to understand where, how, and why their customers may be churning. Identifying users at risk of downgrading from free to premium subscription or canceling the service, or users who are likely to churn, is crucial to the success of a music streaming service.

Background

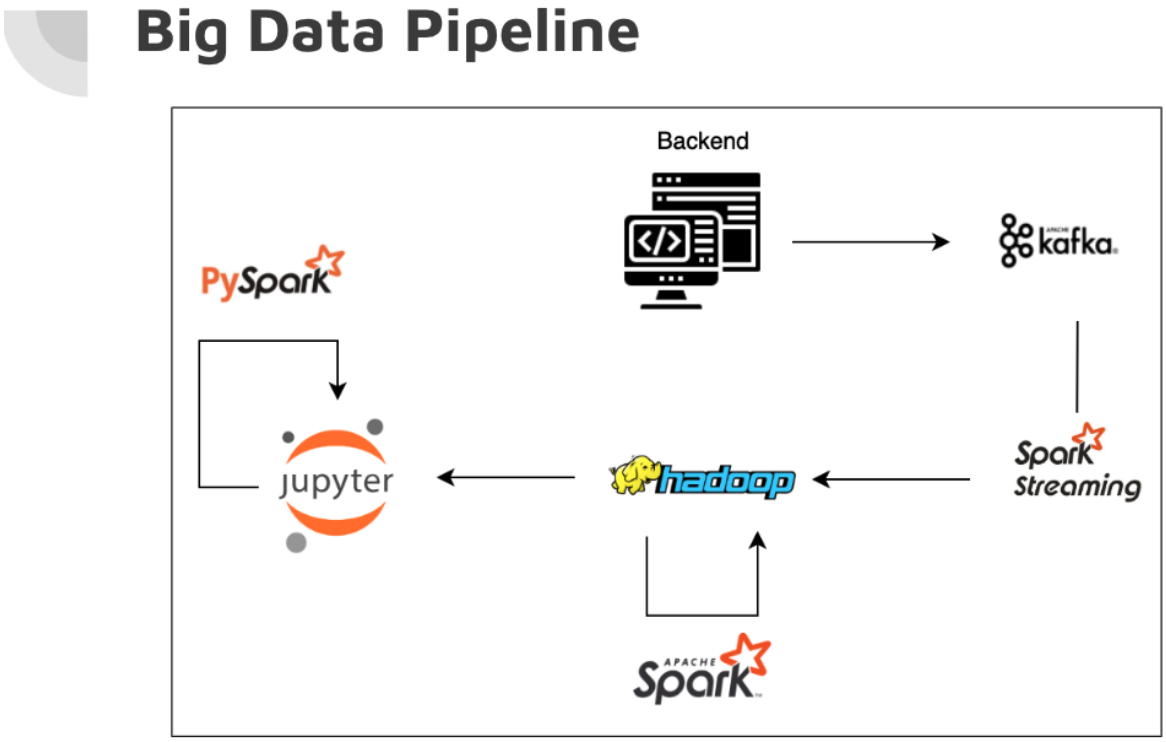
Sparkify is a music streaming service (like Spotify, Pandora, etc.). In Sparkify users can either listen to music for free or buy a subscription. The free users will listen to ads while the subscription users (or paid users) will listen to songs ad-free. Users need a login to listen to a song on the service.

From a subscription perspective at any point, users can do any of the following:

1. Upgrade from the free tier to the paid tier
2. Downgrade from the paid tier to the free tier
3. Cancel their account and leave the service

We want to identify the users that could potentially cancel their account and leave the service. By identifying this population upfront, we could incentivize them by giving them discounts or other rewards. This could make them stick, giving us a loyal customer base which is key for a company's growth.

System Design Architecture



We have built an end to end data pipeline starting with data streaming from a log file emulating the application backend to the Kafka cluster. There will be a dedicated Spark streaming application to consume the stream message and ingest to local file storage emulating Hadoop, stored as json files, and retrieved with Spark Dataframe API via Jupyter to perform Churn Prediction analysis. The experiment focuses on analyzing and predicting the factors impacting churn rate.

Kafka Streaming

The kafka producer server parses the json log file and prepares the message to be sent over as a stream of data. The app.cfg lists out parameters used in the application (specific for Kafka and Spark). The kafka producer notebook contains the logic to send the kafka message. A sample kafka consumer notebook is used for testing the message sent by the kafka producer. In the Spark streaming notebook, we consume the message from Kafka, transform and store on local file storage.

Dataset(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)
```

- artist: Artist name (ex. Daft Punk)
- auth: User authentication status (ex. Logged)
- firstName: User first name (ex. Colin)
- gender: Gender (ex. F or M)
- itemInSession: Item count in a session (ex. 52)
- lastName: User last name (ex. Freeman)
- length: Length of song (ex. 223.60771)
- level: User plan (ex. paid)
- location: User's location (ex. Bakersfield)
- method: HTTP method (ex. PUT)
- page: Page name (ex. NextSong)
- registration: Registration timestamp (unix timestamp) (ex. 1538173362000)
- sessionId: Session ID (ex. 29)
- song: Song (ex. Harder Better Faster Stronger)
- status: HTTP status (ex. 200)
- ts: Event timestamp(unix timestamp) (ex. 1538352676000)
- userAgent: User's browser agent (ex. Mozilla/5.0 (Windows NT 6.1; WOW64; rv:31.0) Gecko/20100101 Firefox/31.0)

- userId: User ID (ex. 30)

Defining churn

Churn refers to the action of canceling a user's account on the platforms. Another term for a churned user is one that has canceled their account and left the platform.

Why Big Data?

Data that is so large, fast or complex that it's difficult or impossible to process using traditional methods. The act of accessing and storing large amounts of information for analytics has been around for a long time. Organizations collect data from a variety of sources, data streams into businesses at an unprecedented speed and must be handled in a timely manner. We will make use of big data technology to handle music streaming data from customers over the period of their app use. While the app deals with millions of users on a daily basis and provides subscriptions to most of the users it will be really helpful to use big data to help grow and monitor their business. BigData analytics with Machine Learning were found to be an efficient way for identifying churn.

Data Investigation

1. Load Data (Spark)

Pyspark Query

```
def spark_read(spark):
    return spark.read.option('inferSchema', 'true').option('header',
'true').option('encoding', 'utf-8')
```

Pyspark Query

```
def load_full_json_file(spark):
    df = spark_read(spark).json('sparkify_event_data.json')
    return df
```

2. Total data count - 26 million records

3. Considering records for logged in user sessions. Thus, removing guest or logged out users as we would not be able to obtain future sessions for these user IDs

   Pyspark Query - df.select('auth').distinct().collect()
   Pyspark Query - df.where(df.auth.isin(['Guest', 'Logged Out'])).count()

4. Checking if records have user IDs present or not ?
   Pyspark Query - df.where(col('userId').isNull()).count()

5. Creating new user columns based on the page analysis and its timestamp. This is done by extracting the window and creating the feature set.

length	level	location	method	...	userAgent	userId	date	previous_page	last_event_ts	last_page	register_page	first_ts	ts_elapsed	session_duration
NaN	free	Hartford-West Hartford-East Hartford, CT	GET	...	"Mozilla/5.0 (Windows NT 6.3; WOW64) AppleWebK...	1528635	2018-10-01	None	1543412707000	Roll Advert	None	1538400348000	0	1365000
196.28363	free	Hartford-West Hartford-East Hartford, CT	PUT	...	"Mozilla/5.0 (Windows NT 6.3; WOW64) AppleWebK...	1528635	2018-10-01	Home	1543412707000	Roll Advert	None	1538400348000	32000	1365000
251.55873	free	Hartford-West Hartford-East Hartford, CT	PUT	...	"Mozilla/5.0 (Windows NT 6.3; WOW64) AppleWebK...	1528635	2018-10-01	NextSong	1543412707000	Roll Advert	None	1538400348000	228000	1365000
		Hartford-			"Mozilla/5.0									

Data Visualization

All may happened event pages

- About
- Add Friend
- Add to Playlist
- Cancel
- **Cancellation Confirmation**
- Downgrade
- Error
- Help
- Home
- Login
- Logout
- NextSong
- Register
- Roll Advert
- Save Settings
- Settings
- Submit Downgrade
- Submit Registration
- Submit Upgrade
- Thumbs Down
- Thumbs Up
- Upgrade

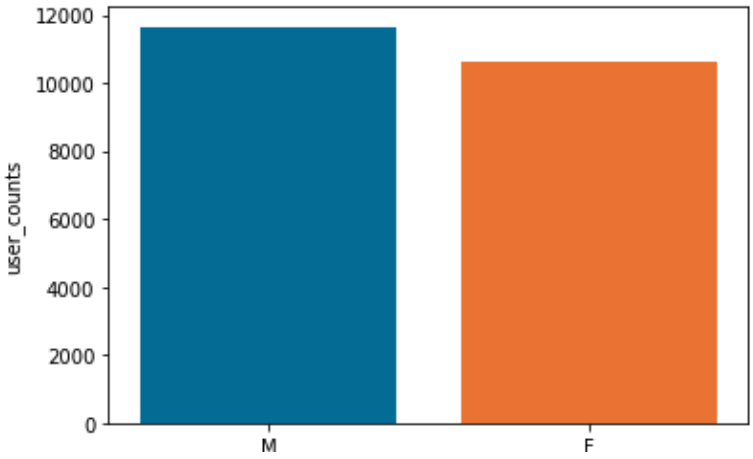
We may pay specific attention in Cancellation Confirmation, it's an event that we never like to see a user doing.We use this event to flag and create a churn column.

The data divide will check the user churn ratio, to define how many users have actually churned and how many users are still using the music app.

Churned: 22.46%

Gender Count

The dataset contains both male and female users and thus presents a ratio divide in our dataset.



Thus, we can see there are more male users than female users in our dataset.

Unique user sessions (Count)

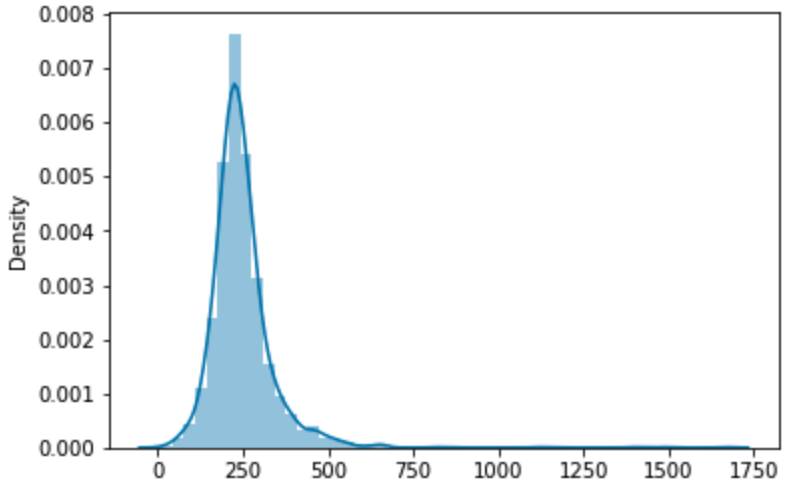
The dataset contains values in reference to a huge number of sessions, but also includes the values of each page visit for each user. Thus, using the PySpark Query, we count the number of distinct sessions for each user.

```
+-----------+
|item_counts|
+-----------+
|       1429|
+-----------+
```

Unique user sessions (Length)

The dataset contains values in reference to a huge number of sessions, but also includes the values of each page visit for each user. Thus, using the PySpark Query, we count the number of distinct sessions length for each user. In other words, sorting the highest amount of time a user spent on the app in a single session.

```
+----------+
|    length|
+----------+
| 524.32934|
| 238.39302|
| 140.35546|
| 277.15873|
|1121.25342|
| 229.25016|
|  313.5473|
|  250.8273|
| 203.88526|
| 183.87546|
|  265.9522|
| 199.81016|
| 131.91791|
| 248.78975|
| 126.40608|
| 355.57832|
|  204.7473|
| 293.85098|
| 267.12771|
| 256.80934|
+----------+
only showing top 20 rows
```



Length looks like a gaussian distribution with a long write tail. Most length values are concentrated between 0 and 500.

User Level (Free Vs Paid)

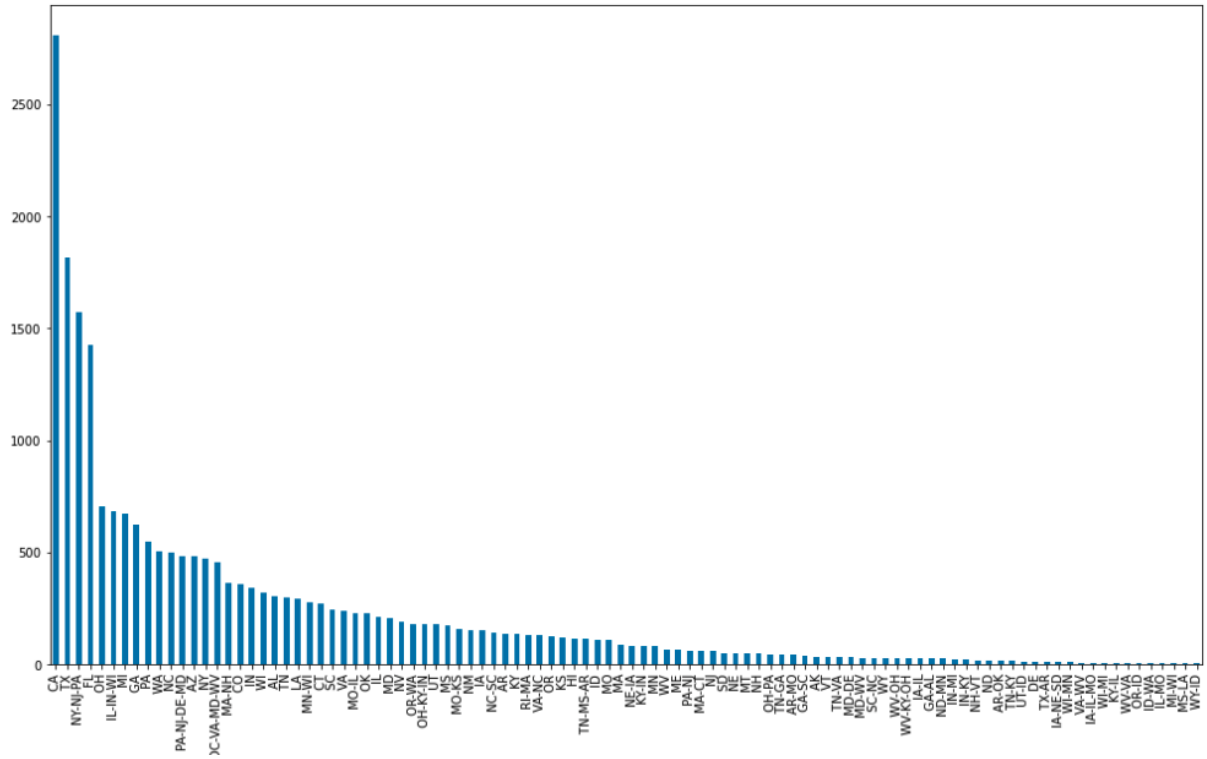
The user on the app is either a paid subscriber or a free user and thus would be helpful to determine if it is the reason for the user to churn. We would explore the distribution of the free and paid subscribers in our event log distribution.

```
+-----+-----------+
|level|user_counts|
+-----+-----------+
| free|      18793|
| paid|      16185|
+-----+-----------+
```

Also, the user interaction was found to be the value of 12700 which signifies the amount of time these free users on the app integrated with the paid subscription users.

Local Analysis (State)

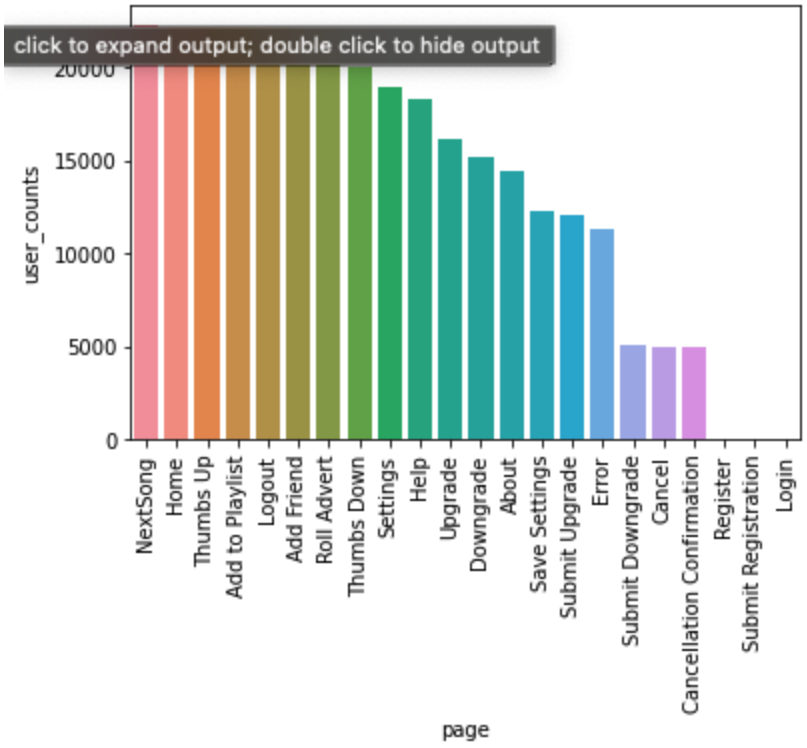
It is very important to determine where the users on our app are present and situated. This would help the marketing team to better understand how the user base in those areas can be grown either by applying discounts or giving free trials.



California has the widest user base, followed by texas and new york state.

Page Count Analysis

The number of user counts that visited different pages or how they interacted with the app, most likely the page events they visited.



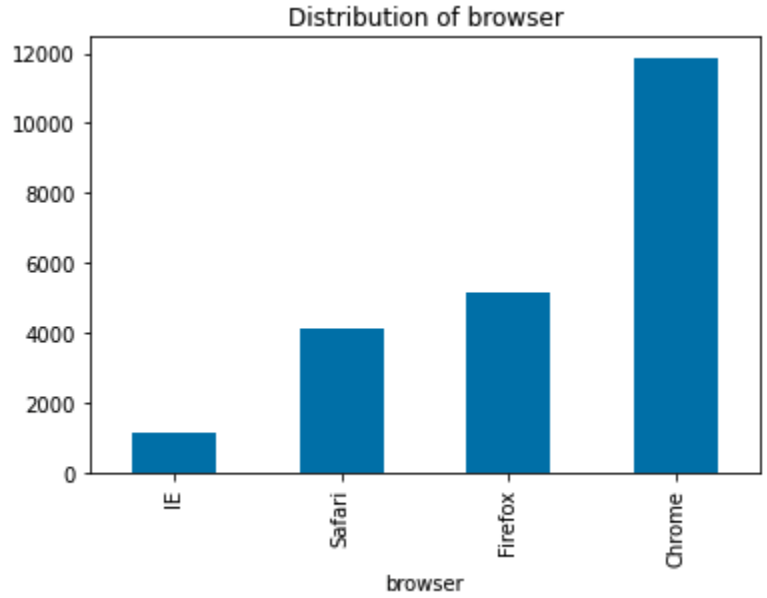
Most of the users are in NextSong, Home, Thumbs Up or Add to Playlist When we have few that cancel, or in Cancellation Confirmation or Submit Downgrade.

User Device Analysis

The user device analysis is an important part of our churn prediction as it would help us to answer that are the users churning because of the device they are using. Each device has its different UI, so is that affecting the user behavior on the platform.

userAgent	user_counts
"Mozilla/5.0 (Win...	2024
Mozilla/5.0 (Wind...	1661
"Mozilla/5.0 (Win...	1395
"Mozilla/5.0 (Mac...	1340
"Mozilla/5.0 (Mac...	1229
"Mozilla/5.0 (Mac...	1084
Mozilla/5.0 (Maci...	1000
"Mozilla/5.0 (Mac...	998
"Mozilla/5.0 (Mac...	913
"Mozilla/5.0 (Win...	795
Mozilla/5.0 (Wind...	529
"Mozilla/5.0 (iPh...	517
"Mozilla/5.0 (Win...	510
Mozilla/5.0 (Wind...	487
"Mozilla/5.0 (Win...	410
Mozilla/5.0 (X11;...	361
Mozilla/5.0 (Wind...	317
"Mozilla/5.0 (Win...	281
"Mozilla/5.0 (iPa...	259
Mozilla/5.0 (Wind...	239

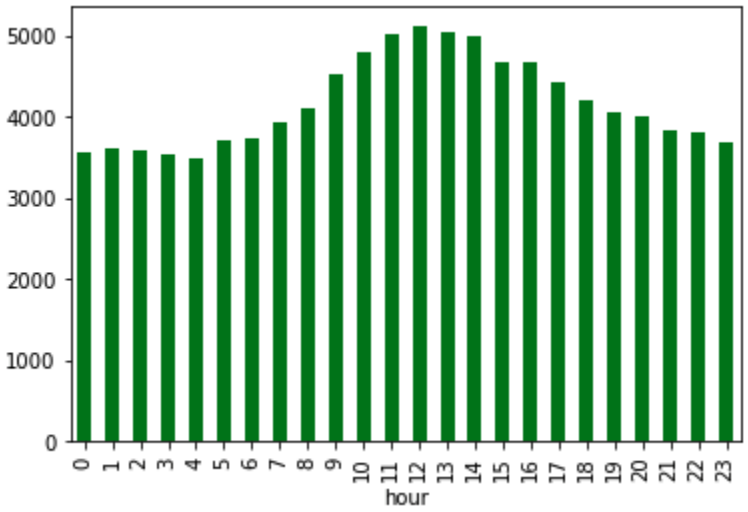
Classifying devices used by each user while being present on the platform.

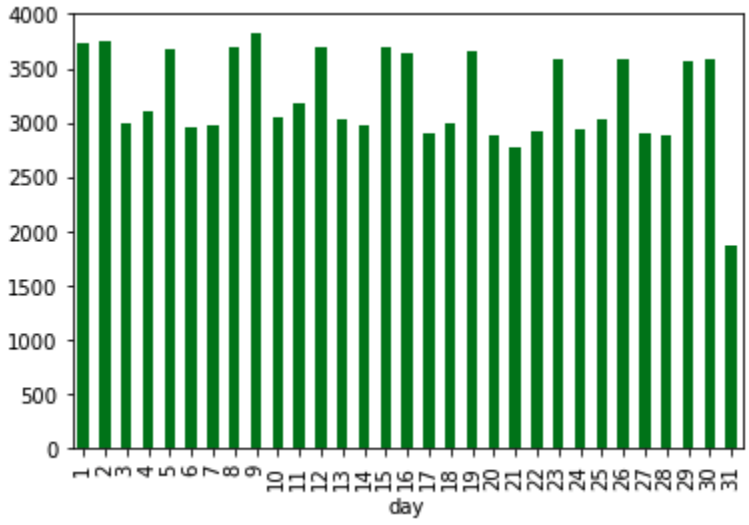


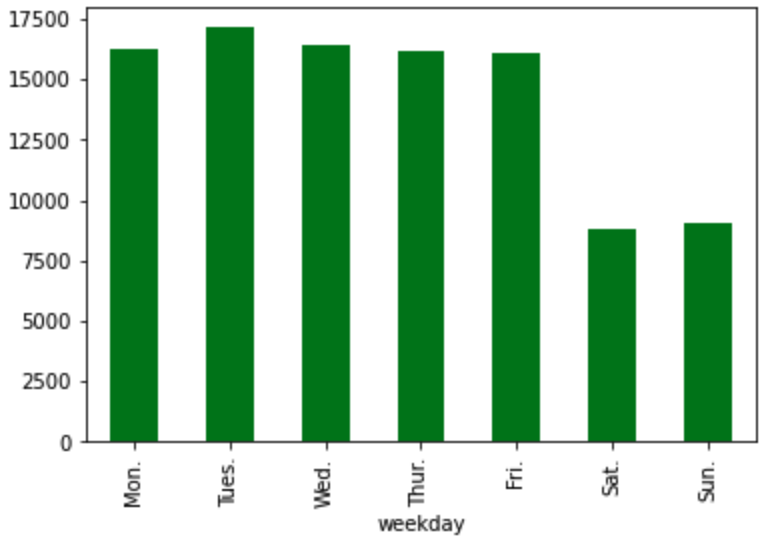
We see that Chrome browser is far most used, followed by firefox and safari. IE is very little used.

User Analysis (Day & Time)

Below depicted charts help users to understand the user timestamp behavior on the platform. It helps us to track the highest period of activity on the user app. At what day or time, does the user use the music app the most.







Users' behaviors are periodic, they use Sparkify more often on weekdays than weekends. In one single day, they use Sparkify more often after 14 o'clock.

Effect of each feature on churn (Analysis)

Let’s analyze how these features had the effect on the churn. The below references present how each feature in our event dataset was used to affect the user churn like the questions were answered using Pyspark queries whether the male users churned more in reference to the other female users or does the user session length had any effect on the churn prediction.

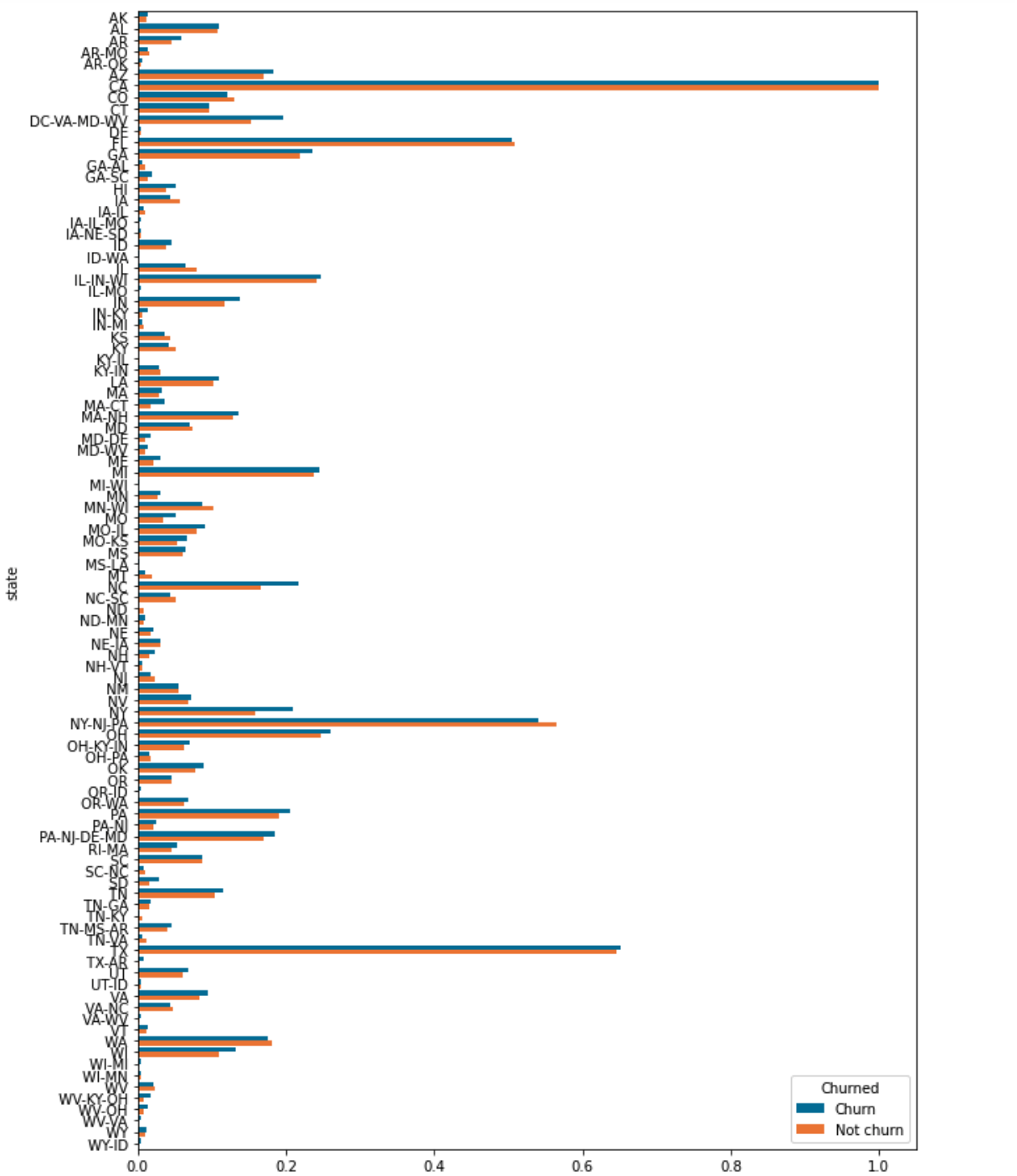
Churned	level	count
Churn	paid	1239
Churn	free	3764
Not churn	free	13484
Not churn	paid	3791

Churned	mean_length	stdev_length	max_length	min_length
Churn	248.680031572098	97.31322630186042	3024.66567	0.522
Not churn	248.73752863664356	97.28014262879445	3024.66567	0.522

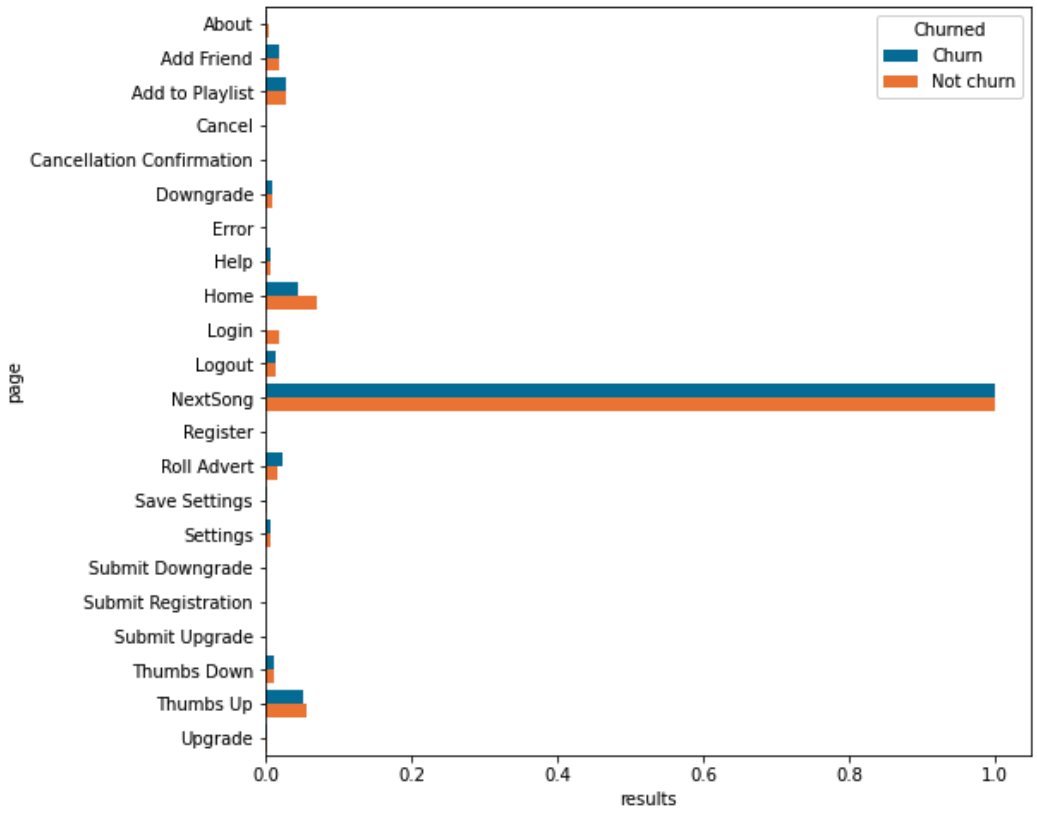
Churned	gender	count
Churn	F	2347
Churn	M	2656
Not churn	null	1
Not churn	M	8995
Not churn	F	8279

Churned	auth	count
Churn	Logged In	5003
Not churn	Logged In	17274
Not churn	Logged Out	1

Location (Churn Effect)



Page (Churn Effect)



NextSong,Thumbs Up/Down, Home, Add to Playlist page's seems to have an effect on Churn or not.

Feature Engineering

Feature engineering is the process of using domain knowledge of the data to create features that make machine learning algorithms work. If feature engineering is done correctly, it increases the predictive power of machine learning algorithms by creating features from raw data that help facilitate the machine learning process. Feature Engineering is an art. Some of the ideas we have used are mentioned below about how the original set of features were changed to derive a new set of features that were more relevant in predicting the user churn.

In this part, we create other features from the existing ones. For example, the number of times that the user has an event on the page **Submit Downgrade**

becomes the number of downgrades. Some features are ignored like the firstName and lastName because they appear to be irrelevant by the aspect of churn. Others, like the location and the userAgent, are complex to extract something important.

User Session (Count of each page visit as feature set for new data)

Using the pyspark order by feature to create new column features based on the actions users have taken in the event logs.

```
df_sessions = df.orderBy(df.sessionId).groupBy('sessionId', 'userId').agg(
        smax(df.ts).alias('max_event_ts'),
        smin(df.ts).alias('min_event_ts'),
        ssum(df.length).alias('session_n_total_playback'), # Based on songs length
        count(when(df.page == 'Thumbs Up', True)).alias("session_n_likes"),
        count(when(df.page == 'Thumbs Down', True)).alias("session_n_dislikes"),
        count(when(df.page == 'NextSong', True)).alias("session_n_songs"),
        count(when(df.page == 'Add Friend', True)).alias("session_n_friends"),
        count(when(df.page == 'Add to Playlist', True)).alias("session_n_add_playlist")
        count(when(df.page == 'Home', True)).alias("session_n_home"),
        count(when(df.page == 'Roll Advert', True)).alias("session_n_ads"),
        count(when(df.page == 'Help', True)).alias("session_n_help"),
        count(when(df.page == 'Error', True)).alias("session_n_error"),
        count(when(df.page == 'Settings', True)).alias("session_n_sets"),
        count(col('page')).alias('session_n_actions'),
        first(col('session_duration')).alias('session_duration')
    )
```

Interval until next session (Adding the interval between sessions as an feature)

w_user_sessions_interval = Window.partitionBy('userId').orderBy('min_event_ts')
df_sessions = df_sessions.withColumn('interval_to_session', col('min_event_ts') - lag(col('max_event_ts')).over(w_user_sessions_interval))

Adding the average playback time as a feature to check how users have played songs per session

```
df_session_time = df_sessions.groupBy('userId').agg(
(avg(df_sessions.session_n_total_playback) /
minutes_to_hours).alias('avg_playback_time'))

df_sessions = df_sessions.join(df_session_time, on = 'userId')
```

User Profile

Creating user dimension features for subscription, streaming, community and page visit counts based on user interaction

```
CHURN_CANCELLATION_PAGE = 'Cancellation Confirmation'
REGISTRATION_PAGE = 'Submit Registration'
```

Subscription

a.) No of downgrades

b.) No of upgrades

c.) Paid User

d.) User Canceled

Streaming

a.) Next Song

b.) Likes on each song

c.) Dislikes on each song

Community

a.) Add Friend

b.) Add to playlist

```
df_user_profile = df.groupby('userId')\
        .agg(first(when(col('gender') == 'M', TRUE).otherwise(FALSE)).alias('male'),

            smin(col('first_ts')).alias('ts_start'),
            smax(col('last_event_ts')).alias('ts_end'),

            ((smax(col('last_event_ts')) - smin(col('first_ts'))) / milliseconds_to_hours).alias('time_window'),

            # Subscription
            count(when(col('page') == 'Submit Downgrade', True)).alias('n_downgrades'),
            count(when(col('page') == 'Submit Upgrade', True)).alias('n_upgrades'),
            last(when(col('level') == 'paid', TRUE).otherwise(FALSE)).alias('paid'),
            first(when(col('last_page') == CHURN_CANCELLATION_PAGE, TRUE).otherwise(FALSE)).alias('canceled'),

            # Streaming
            count(when(col('page') == 'NextSong', True)).alias('n_songs'),
            count(when(col('page') == 'Thumbs Up', True)).alias('n_likes'),
            count(when(col('page') == 'Thumbs Down', True)).alias('n_dislikes'),

            (count(when(col('page') == 'NextSong', True))/count(when(col('page') == 'Roll Advert', True))).alias('n_ads
            (count(when(col('page') == 'NextSong', True))/count(when(col('page') == 'Thumbs Up', True))).alias('n_likes
            (count(when(col('page') == 'NextSong', True))/count(when(col('page') == 'Thumbs Down', True))).alias('n_dis
            (count(when(col('page') == 'Thumbs Up', True))/count(when(col('page') == 'Thumbs Down', True))).alias('n_li

            countDistinct(col('sessionId')).alias('n_sess'),
            (avg(col('session_duration')) / milliseconds_to_hours).alias('avg_session_duration'),

            # Community
            count(when(col('page') == 'Add Friend', True)).alias('n_friends'),
            count(when(col('page') == ' ', True)).alias('n_added_to_playlist'),

            # Other
            count(when(col('page') == 'Home', True)).alias('n_home'),
            count(when(col('page') == 'Roll Advert', True)).alias('n_ads'),
            count(when(col('page') == 'Help', True)).alias('n_help'),
            count(when(col('page') == 'Error', True)).alias('n_errors'),
            count(when(col('page') == 'Settings', True)).alias('n_settings'),
            count(col('page')).alias('n_actions')
        )
```

User Dimension (Daily Data) - Adding daily

```
df_unique_days = df.groupby('userId').agg(countDistinct('date').alias('n_days'))
df_daily_actions = df.groupby('userId', 'date').agg(count('page').alias('total'))
df_daily_actions = df_daily_actions.groupby('userId').agg(avg('total').alias('avg_daily_actions'))
df_days = df_unique_days.join(df_daily_actions, df_unique_days.userId == df_daily_actions.userId)
df_days = df_days.drop(df_daily_actions.userId)
```

```
df_users = df_user_profile.orderBy(df_user_profile.userId).join(df_days, on = 'userId')
```

The final features

- **male:** 1 — male or 0 — female
- **paid:** 1- paid subscription or 0 — free subscription
- **avg_daily_actions:** Average actions by day
- **avg_session_duration:** Average duration in hours of each session
- **avg_playback_time:** Average time listening to music for each session

- **n_actions:** The number of actions
- **n_added_to_playlist:** Number of songs added to the playlist
- **n_ads:** Number of ads viewed
- **n_days:** Number of days of the observed window,
- **n_dislikes:** Number of dislikes
- **n_downgrades:** Number of downgrades
- **n_errors:** Number of experienced errors
- **n_friends:** Number of friends added
- **n_help:** Number of times has accessed the help page
- **n_home:** Number of times has accessed the home page
- **n_likes:** Number of songs liked
- **n_sess:** Number of sessions
- **n_settings:** Number of times has accessed the help page
- **n_songs:** Number of songs played
- **n_upgrades:** Number of upgrades
- **n_ads_over_songs:** Number of ads viewed divided by Number of songs played
- **n_likes_over_songs:** Number of songs liked divided by the number of Songs
- **n_dislikes_over_songs:** Number of songs disliked divided by the number of Songs
- **n_likes_over_dislikes:** Number of songs liked divided by the number of songs disliked
- **time_window:** Time time in hours of observed data

Data Modeling

Gradient Boost and Random Forest

The idea of boosting came out of the idea of whether a weak learner can be modified to become better. A weak hypothesis or weak learner is defined as one whose performance is at least slightly better than random chance. Gradient Boost Trees is the modified and improved version of AdaBoost.

AdaBoost, which is the abbreviation of the Adaptive Boosting, is the first practical boosting algorithm proposed by Freund and Schapire in 1996.

AdaBoost is an algorithm for constructing a “strong” classifier as a linear combination of “simple” “weak” classifiers. In other words, the main working principle is to convert a set of weak classifiers into a strong one. Weak classifier is described as less than 50% error over any distribution and strong classifier is thresholded linear combination of the weak classifier outputs. AdaBoost works by weighting the observations, putting more weight on difficult to classify instances and less on those already handled well. New weak learners are added sequentially that focus their training on the more difficult patterns. Predictions are made by majority vote of the weak learners’ predictions, weighted by their individual accuracy.

Gradient boosting involves three elements:

- A loss function to be optimized. The loss function used depends on the type of problem being solved. It must be differentiable, but many standard loss functions are supported and you can define your own. For example, regression may use a squared error and classification may use logarithmic loss
- A weak learner to make predictions. Decision trees are used as the weak learner in gradient boosting
- An additive model to add weak learners to minimize the loss function. Trees are added one at a time, and existing trees in the model are not changed. A gradient descent procedure is used to minimize the loss when adding trees.

GBT builds trees one at a time, where each new tree helps to correct errors made by previously trained trees. A great application of GBM is anomaly detection in supervised learning settings where data is often highly unbalanced such as DNA sequences, credit card transactions or cybersecurity.

The differences between the GBT and RF can be explained as Gradient boosting uses regression trees for prediction purposes where a random forest uses decision trees. The random forest is easy to parallelize but boosted trees are

hard to do. Random forests overfit a sample of the training data and then reduce the overfit by simply averaging the predictors. If you carefully tune parameters, gradient boosting can result in better performance than random forests. However, gradient boosting may not be a good choice if you have a lot of noise, as it can result in overfitting. They also tend to be harder to tune than random forests.

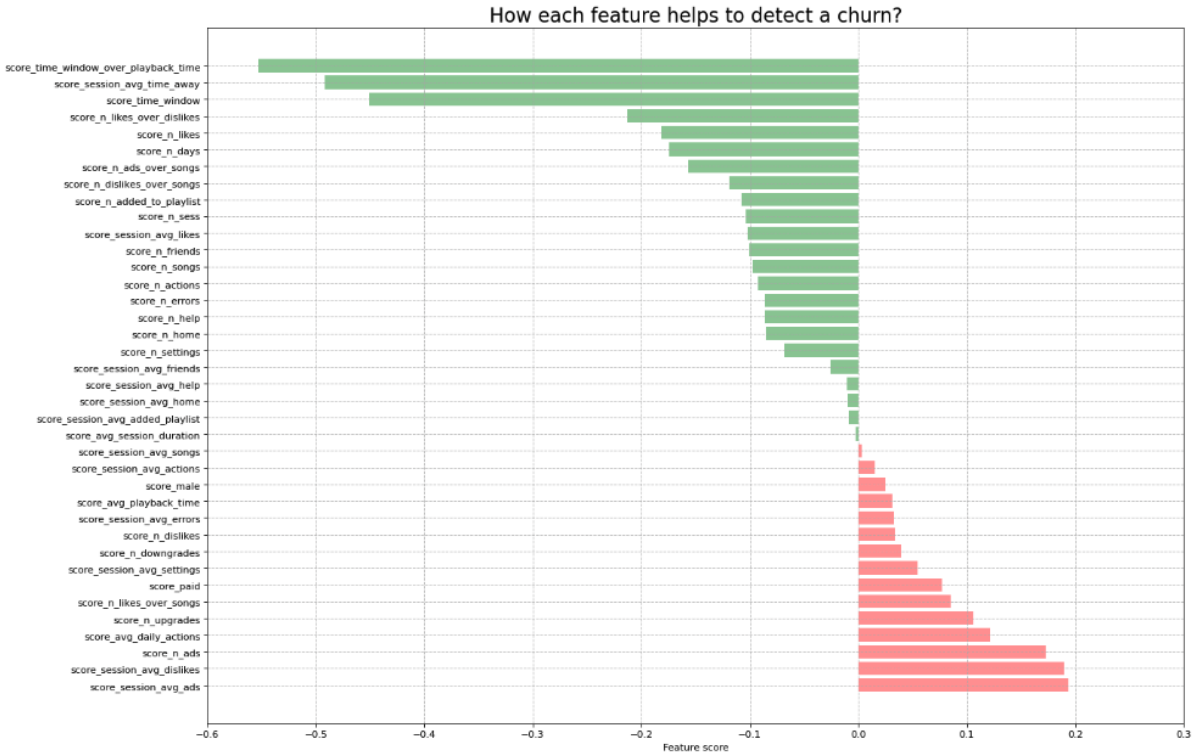
Decision Tree Classifier

In decision analysis, one of the visual and explicit representations of decision and decision making procedure can be performed by using decision trees. The decision tree is using a tree-like model of decisions such that a flowchart like tree structure in which each node internal node denotes a test and each branch represents an outcome of the test and each leaf node holds a corresponding class label. Decision trees are one of the most powerful and popular tools for classification and prediction. Decision trees are commonly used in medical diagnosis, failure prediction, credit scoring and crime risk investigation. Several business analysis examples can be found but one of the widely used ones is the detection of fraudulent financial statements.

Logistic Regression

When working with our data that accumulates to a binary separation, we want to classify our observations as the customer “will churn” or “won’t churn” from the platform. A logistic regression model will try to guess the probability of belonging to one group or another. The logistic regression is essentially an extension of a linear regression, only the predicted outcome value is between [0, 1]. The model will identify relationships between our target feature, Churn, and our remaining features to apply probabilistic calculations for determining which

class the customer should belong to. We will be using the ‘ScikitLearn’ package in Python.



Results

Machine learning modeling has a success in predicting the customer's churn activity. That can help the application owners to improve the user lifetime. Despite the relatively good results of other selected machine learning models such as SVM and Decision Trees, the GBT algorithm is selected and tuned with hyperparameters. For future work, the Random Forest algorithm can be adjusted with different settings to overcome the over-fitting.

Possible Improvements for Current Model

Even if we have almost 89% accuracy in current model, there is still a chance to improve it by

- Working with more observations by increasing the dataset size.
- Investigating the different parameter settings to have more accuracy.
- Analyzing location information has an effect on the churned user since location can provide us information about lifestyle and give company owners a chance to provide location based promotions to keep the users.
- Used environments can also provide useful information. If android users are more likely to be churned, this means the application needs an improvement for that environment.

Decision Tree Classifier

```
In [87]: pipeline1 = create_decision_tree_pipeline()
         model_pipeline1 = pipeline1.fit(train_df)
         predictions1 = model_pipeline1.transform(test_df)
         show_results_and_save(model_pipeline1, predictions1)
```

```
[[ 841  497]
 [ 139 4411]]

accuracy..... 0.8920
precision..... 0.8582
recall..... 0.6286
auc..... 0.7990
beta..... 0.6641
F1 macro..... 0.8292
F1 micro..... 0.8920
F1 weighted..... 0.8857
F1 binary..... 0.7256
```

Gradient Boost

```
pipeline3 = create_gradient_boost_pipeline()
model_pipeline3 = pipeline3.fit(train_df)
predictions3 = model_pipeline.transform(test_df)
show_results_and_save(model_pipeline3, predictions3)
```

```
Number of models to train: 150
[[ 876  401]
 [ 193 4447]]

accuracy..... 0.8996
precision..... 0.8195
recall..... 0.6860
auc..... 0.8222
beta..... 0.7091
F1 macro..... 0.8421
F1 micro..... 0.8996
F1 weighted..... 0.8963
F1 binary..... 0.7468
```

Random Forest

```
In [89]: pipeline2 = create_random_forest_pipeline()
         model_pipeline2 = pipeline2.fit(train_df)
         predictions2 = model_pipeline2.transform(test_df)
         show_results_and_save(model_pipeline2, predictions2)
```

```
[[ 804  534]
 [ 100 4450]]

accuracy..... 0.8923
precision..... 0.8894
recall..... 0.6009
auc..... 0.7895
beta..... 0.6426
F1 macro..... 0.8254
F1 micro..... 0.8923
F1 weighted..... 0.8844
F1 binary..... 0.7172
```

Logistic Regression

```
In [86]: pipeline = create_logistic_regression_pipeline()
         model_pipeline = pipeline.fit(train_df)
         predictions = model_pipeline.transform(test_df)
         show_results_and_save(model_pipeline, predictions)
```

```
[[1085  253]
 [ 609 3941]]

accuracy..... 0.8536
precision..... 0.6405
recall..... 0.8109
auc..... 0.8385
beta..... 0.7699
F1 macro..... 0.8086
F1 micro..... 0.8536
F1 weighted..... 0.8592
F1 binary..... 0.7157
```

Conclusion

In this project, we were able to study the service dataset and create functions for the modeling process. To begin with, we studied different levels of the dataset, which was the logs of each user session. The dataset allowed us to study churn and create suitable predictive features. In addition, feature selection was not a trivial task. We trained four models: Logistic Regression, Random Forest, Gradient Boosted Trees, and Decision Tree Classifier.

Feasibility and cost become very important in real applications. A model like this can be run weekly or monthly depending on the data latency, business requirements. Operational cost should be monitored and model results should be validated with testing (A/B testing). Experiment results(evaluation metrics, KPIs) should be tracked so that our model and following actions could bring value to business.

Acknowledgements

We thank Professor Juan Rodriguez/TAs for teaching the material on big data analysis, for answering our questions and for helping us develop an immense interest in big data. They equipped us with the necessary skills and knowledge to finish a project of this magnitude and complexity. We also are extremely thankful to the teaching assistants for their contribution towards strengthening our conceptual and technical understanding of the various topics in big data.

References

1) https://kth.diva-portal.org/smash/get/diva2:1149077/FULLTEXT01.pdf
2) https://www.sciencedirect.com/science/article/abs/pii/S0305054805003503
3) https://www.profitwell.com/customer-churn/analysis
4) https://www.width.ai/post/churn-prediction-model