databricks Sun-Xi-User_Role_Clustering_Part2

# Continued from the Graph Modeling

The following works are done in this part of notebook:

- Feature Exploration and Preprocessing
  - Features table was merged from topological feature table and user activity table
  - Each feature was carefully explored. The distribution was examined using q-q plot. Corresponding filtering strategies were determined
  - Filtering conditions were applied to clean the data
  - Featurs were scaled into standard normal distribution, for the reason that k-means clustering is sensitive to high variance and skewed features.
- Clustering
  - According to both the guidance paper and other paper comparing multiple clustering algorithms [8], bisecting k-means algroithm was used to cluster the users. According to the papers, bisecting k-means on average has better performance than k-means, and tends to give more stable splits with less skewed cluster sizes.
  - Elbow method was used to determine the optimal number of cluster to use, k was experimented in range [2,20], 8 was determined to be the optimal choice.
  - Users were divided in to 8 clusters, and their centroid were recorded.
- Role Identification and Explaination
  - Cluster centroids were scaled into range [1,10] to facilitate the visualization, for the reason that we only care about the relative magnitude of each feature among clusters.
  - Cluster centroides were visualized, and each role type was identified and explained combined with the unscaled average features of clusters.
- Future Work
  - Possible defects of this analysi are identified, and possible future work to improve the work are proposed.

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

# Importing user feature data

- Define data processing functions
  - Multiple functions are defined to convert input data into corresponding data type.
- Read the .csv file from online storage to the spark RDD
- Define table schema and processing each column accordingly.

```
%scala
/*
Init data type conversion functions to conver string type to destination type.
*/

// Libraries to load data from remote source
import org.apache.commons.io.IOUtils
import java.net.URL
import java.nio.charset.Charset

// Function to cast input data to match the schema
implicit class StringConversion(val s: String) {
def toTypeOrElse[T](convert: String=>T, defaultVal: T) = try {
    convert(s)
  } catch {
    case _: Throwable => defaultVal
  }

  def toIntOrElse(defaultVal: Int = 0) = toTypeOrElse[Int](_.toInt, defaultVal)
  def toDoubleOrElse(defaultVal: Double = 0D) = toTypeOrElse[Double]
(_.toDouble, defaultVal)
  def toFloatOrElse(defaultVal: Float = 0F) = toTypeOrElse[Float](_.toFloat,
defaultVal)
  def toDateOrElse(defaultVal: java.sql.Timestamp =
java.sql.Timestamp.valueOf("1970-01-01 00:00:00")) =
toTypeOrElse[java.sql.Timestamp](java.sql.Timestamp.valueOf(_), defaultVal)
}

//Fix the date format in this dataset
def fixDateFormat(orig: String): String = {
    val splited_date = orig.split(" ")
    val fixed_date_parts = splited_date(0).split("-").map(part => if (part.size
== 1) "0" + part else part)
    val fixed_date = List(fixed_date_parts(0), fixed_date_parts(1),
fixed_date_parts(2)).mkString("-")
    val fixed_time = splited_date(1).split(":").map(part => if (part.size == 1)
"0" + part else part).mkString(":")
    fixed_date + " " + fixed_time + ":00"
}
```

```
import org.apache.commons.io.IOUtils
import java.net.URL
import java.nio.charset.Charset
defined class StringConversion
fixDateFormat: (orig: String)String
```

```
%scala
// Load CSV file from online storage locations
val user_feature_RDD_unfiltered = sc.parallelize( IOUtils.toString( new
URL("https://www.dropbox.com/s/farqszmcrh3ur3x/user_feature.csv?dl=1"),
Charset.forName("utf8")).split("\n"))
```

```
user_feature_RDD_unfiltered: org.apache.spark.rdd.RDD[String] = ParallelCollec
tionRDD[402] at parallelize at command-3238877744563429:2
```

```
%scala
// drop header row
val first_row = user_feature_RDD_unfiltered.first
val user_feature_RDD = user_feature_RDD_unfiltered.filter(row=> row!=first_row)
```

```
first_row: String = "user_id,pull_count,watch_count,commit_count,in_degree,out
_degree,pagerank,betweenness,closeness,transitivity
"
user_feature_RDD: org.apache.spark.rdd.RDD[String] = MapPartitionsRDD[403] at
filter at command-3238877744563430:3
```

```
%scala
user_feature_RDD.take(5)
```

```
res3: Array[String] = Array("1,13,59,55,22,28,1.23E-06,290.0790043,7.91E-07,0.
106312292
", "2,228,706,12846,404,356,3.52E-05,84875.61655,9.27E-07,0.015334079
", "4,148,248,1538,79,149,2.92E-06,5856.666745,8.72E-07,0.058108108
", "5,1224,64,12251,3,285,1.51E-07,306.654159,8.19E-07,0.024017467
", "6,8,49,109913,4,5,3.05E-07,0.95,7.49E-07,0.392857143
")
```

```
%scala
/*
Process the user_feature_RDD in to dataframes
*/
// Define  schema
case class feature(
   USER_ID: Int,     //0
   PULL_COUNT: Int,  //1
   WATCH_COUNT: Int, //2
   COMMIT_COUNT: Int,//3
   IN_DEGREE: Int,   //4
   OUT_DEGREE: Int,  //5
   PAGERANK: Float,  //6
   BETWEENNESS: Float,//7
   CLOSENESS: Float,//8
   TRANSITIVITY: Float //9
  )

// Map input data to schema
def getFeatureCleaned(row:Array[String]):feature = {
return feature(
    row(0).toIntOrElse(),
    row(1).toIntOrElse(),
    row(2).toIntOrElse(),
    row(3).toIntOrElse(),
    row(4).toIntOrElse(),
    row(5).toIntOrElse(),
    row(6).toFloatOrElse(),
    row(7).toFloatOrElse(),
    row(8).toFloatOrElse(),
    row(9).toFloatOrElse()

  )
}
// Create sql table for dataset
val user_feature = user_feature_RDD.map(line => line.split(",").map(elem =>
elem.trim))
val data = user_feature.filter(s => s(1) != 0).map(s =>
getFeatureCleaned(s)).toDF()
data.createOrReplaceTempView("user_feature")
```

```
defined class feature
getFeatureCleaned: (row: Array[String])feature
user_feature: org.apache.spark.rdd.RDD[Array[String]] = MapPartitionsRDD[404]
at map at command-3238877744563432:35
data: org.apache.spark.sql.DataFrame = [USER_ID: int, PULL_COUNT: int ... 8 mo
re fields]
```

```
# show the table schema and row numbers
df = sqlContext.table('user_feature')
print("Records:",df.count())
df.printSchema()
```

```
('Records:', 1048575)
root
 |-- USER_ID: integer (nullable = false)
 |-- PULL_COUNT: integer (nullable = false)
 |-- WATCH_COUNT: integer (nullable = false)
 |-- COMMIT_COUNT: integer (nullable = false)
 |-- IN_DEGREE: integer (nullable = false)
 |-- OUT_DEGREE: integer (nullable = false)
 |-- PAGERANK: float (nullable = false)
 |-- BETWEENNESS: float (nullable = false)
 |-- CLOSENESS: float (nullable = false)
 |-- TRANSITIVITY: float (nullable = false)
```

```
%sql
SELECT * FROM user_feature
WHERE USER_ID != 0
LIMIT 10
```

| USER_ID | PULL_COUNT | WATCH_COUNT | COMMIT_C |
|---|---|---|---|
| 1 | 13 | 59 | 55 |
| 2 | 228 | 706 | 12846 |
| 4 | 148 | 248 | 1538 |
| 5 | 1224 | 64 | 12251 |
| 6 | 8 | 49 | 109913 |
| 7 | 11 | 336 | 243 |
| 8 | 5 | 549 | 30 |
| 9 | 41 | 146 | 3262 |
| 10 | 2556 | 3418 | 8736 |

# Visualize individual feature distribution

Each feature is carefully explored. Based on the type of distribution, the potential abnormal range used to filter the data is determined.
More specifically, following steps were performed.

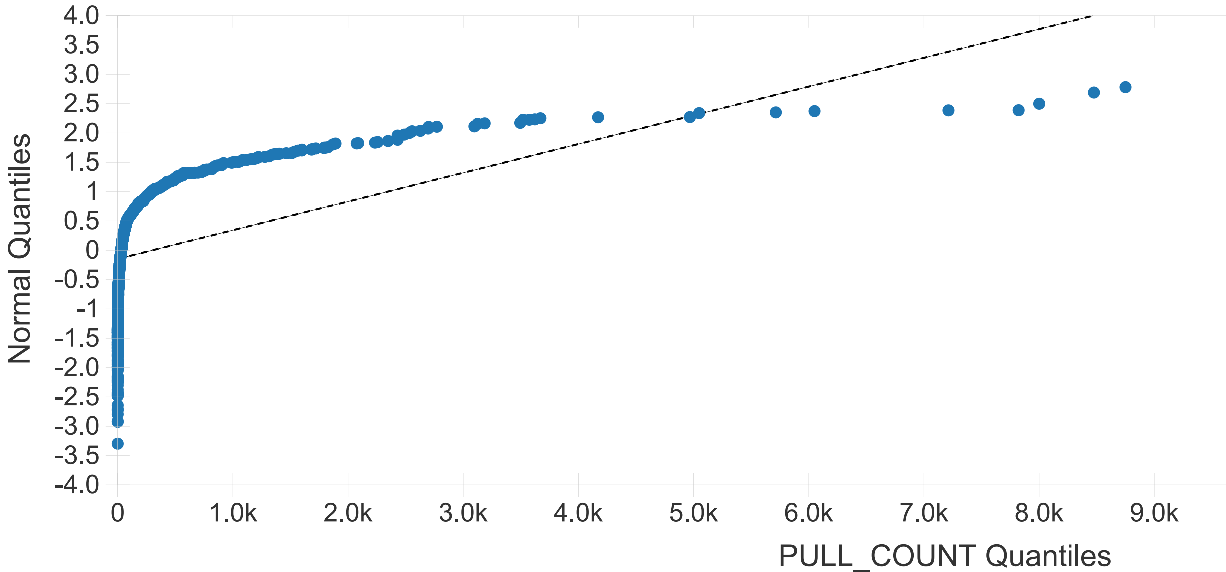
- Analyze the distribution style of each feature
  - **Why understanding feature distribution is important?**
    The algroithm to be used is K-means clustering, which is an unsupervised algorithm based on the distance to the cluster centroids. The algorithm tends to produce spherical shaped clusters. However, if features are not standarized, the effect of each feature to the model will be biased. For example, if some features has significantly large means than others, they will have more effect in the distance calculation, and the features with small mean would more likely to be ignored. Additionally, even each feature has exactly same mean but vastly different variances, the clustering performance would still decrease. Because the large variance means the feature datum would be distorted to an wider range. Therefore, the clustering algorithm would be affected more by the features with large variance.
    The effects of feature mean and variance on the clutering algorithm indicate we should have a throughout understanding of the distribution of each feature, and then transform them to the desired distribution that facilitate the clustering.
  - **Why using Q-Q plot to visualize feature distribution?**
    Based on the above analysis, it's obvious that the desired distribution should be similar to the normal distribution. Therefore, an reasonable approach is to chose standard normal distribution as an benchmark distribution, and then compare each distribution to the benchmark distribution. Q-Q plot, which is also know as quantile-quantile plot is doing exactly what described above.
  - **Understanding Q-Q plot**
    Q-Q plot is an graphical method for comparing two probability distribution by plotting their quantiles against each other. The y axis of an Q-Q plot usually represents the quantile distribution of an standard normal distribution. Each point on the plot is generated by first taking the quantile on the standard normal distribution on y axis, then map it to the value of same quantile in the distribution on the x axis.

- If the distribution to be analyzed follows an roughly standard normal distribution, the Q-Q plot would lie on y=x.
- If the distribution to be analyzed follows an normal distribution with mean value other than 0, the Q-Q plot would still be a line, but not y=x.
- If the distribution to be analyzed is less dispersed than the standard normal distribution, the Q-Q plot would be steeper than y=x.
- If the distribution to be analyzed is more dispersed than the standard normal distribution, the Q-Q plot would be flatter than y=x.
- If the Q-Q plot is in "S" shape, the distribution can be explained combining the above 2 situations.

- Identify potential range used to filter out abnormal data
  - **Why it's important to filter our abnormal data**
    Using unsupervised k-means algorithm, the abnormal data can be still assigned to the most appropriate clusters. However, the extreme values will lead to the distortion of the centroids, which would result in inaccurate cluster interpretation and thus inaccurate role defining.
  - **What is defined as abnormal data in this analysis**
    In this analysis, there are 2 kinds of features: the 1st type is aggregated user activity features, including commit count, pull request count and watch count. The 2nd type is the topological features, including pagerank, clustering coefficient, betweenness and closeness. For both types of feature, we found most of them follows an heavily right skewed distribution with long tail. Meaning their distribution at higher amount are very sparse. After the feature exploration, the abnormal range are determined to be the threshold after which the distribution become super sparse.
  - **How much percentage of datums are discarded**
    Using the above mentioned apporach, about 3% datum were filtered out for each features except for transitivity.

# PULL_COUNT Analysis

```
%sql
-- visualize the pull count distribution
SELECT PULL_COUNT FROM user_feature
```




Showing sample based on the first 1000 rows.

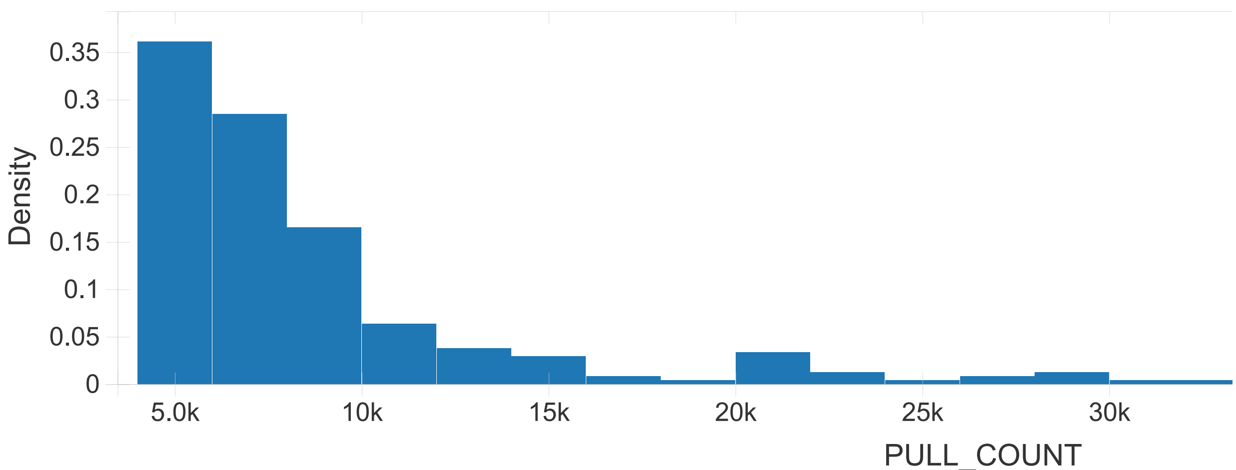
- **Distribution Interpretation**
  - The slope of the Q-Q plot cureve is steeper than y=x, then get dramatically flatter than y=x after passing the median.
  - The initial steep slope indicates the distributiion analyzed is much denser than the normal distribution.
  - The later flat slope indicates the distribution analyzed is much sparser than the normal distribution.
  - Such curve means the distribution is highly right skewed with a very long tail.
  - As indicated by the plot, the distribution get very sparse for the range of >3k, this range will be furthur analyzed below.

```
%sql
-- show average, median, min, max values.
SELECT AVG(PULL_COUNT), percentile_approx(PULL_COUNT, 0.5) median,
MIN(PULL_COUNT), MAX(PULL_COUNT) FROM USER_FEATURE
```

| avg(PULL_COUNT) | median |
|---|---|
| 36.319449729394655 | 3 |

```
%sql
```

```
%sql
-- Take a closer look at pull_count in the range of > 5000
SELECT PULL_COUNT FROM user_feature
WHERE PULL_COUNT>5000
ORDER BY PULL_COUNT DESC
```



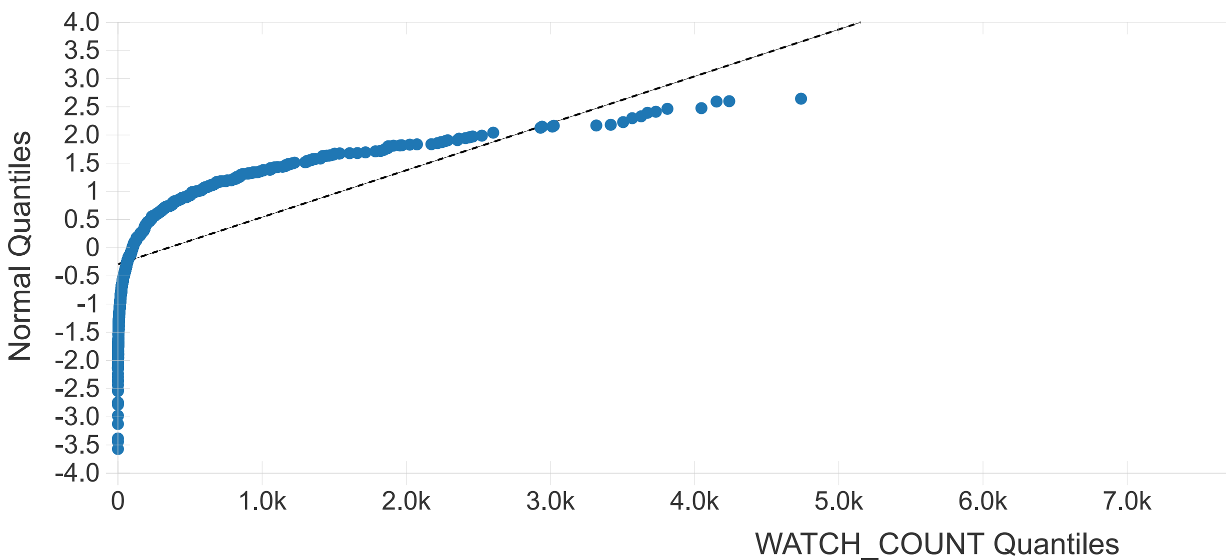

```
%sql
-- count of pull_count > 5000 and >10000
(SELECT COUNT(PULL_COUNT) 5000_10000_count FROM user_feature
WHERE PULL_COUNT>5000)
UNION
(SELECT COUNT(PULL_COUNT)  FROM user_feature
WHERE PULL_COUNT>10000)
```

| 5000_10000_count |
|---|
| 235 |
| 47 |

- **Insights from PULL_COUNT**
  - PULL_COUNT distribution is very skewed, there are 90% users have less than 600 pull_count, while the largest is 52k.
  - Base on observation, distribution in range of >5000 is very sparse, therefore 5000 will be used as filtering condition for pull_count
  - There will be 235 PULL_COUNT filtered out.

# WATCH_COUNT Analysis

```
%sql
SELECT WATCH_COUNT FROM user_feature
```




Showing sample based on the first 1000 rows.

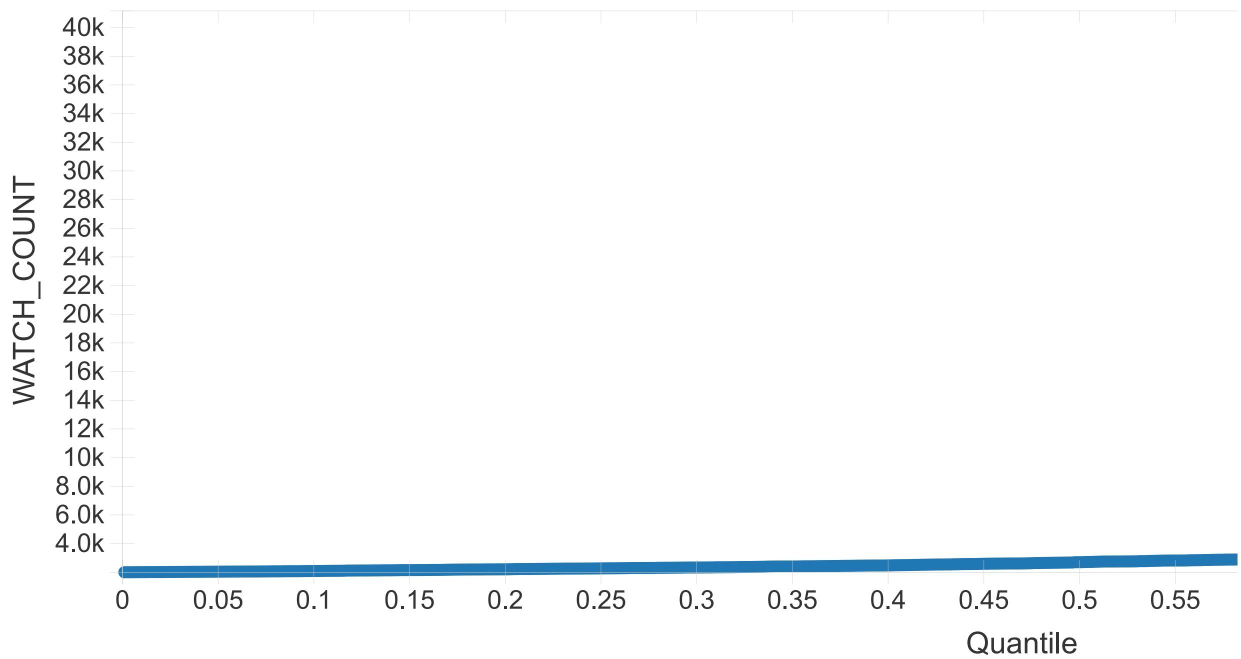
- **Distribution Interpretation**
  - The slope of the Q-Q plot cureve is steeper than y=x, then get dramatically flatter than y=x after passing the median.
  - The initial steep slope indicates the distributiion analyzed is much denser than the normal distribution.

- The later flat slope indicates the distribution analyzed is much sparser than the normal distribution.
- Such curve means the distribution is highly right skewed with a very long tail.
- As indicated by the plot, the distribution get very sparse for the range of >3k, this range will be furthur analyzed below.

```
%sql
-- show average, median, min, max values.
SELECT AVG(WATCH_COUNT), percentile_approx(WATCH_COUNT, 0.5) median,
MIN(WATCH_COUNT), MAX(WATCH_COUNT) FROM USER_FEATURE
```

| avg(WATCH_COUNT) | median |
|---|---|
| 44.07975776649262 | 5 |

```
%sql
-- a closer look at watch_count > 2000, take 80% quantile at 3800
SELECT WATCH_COUNT FROM user_feature
WHERE WATCH_COUNT>2000
```




Showing sample based on the first 1000 rows.

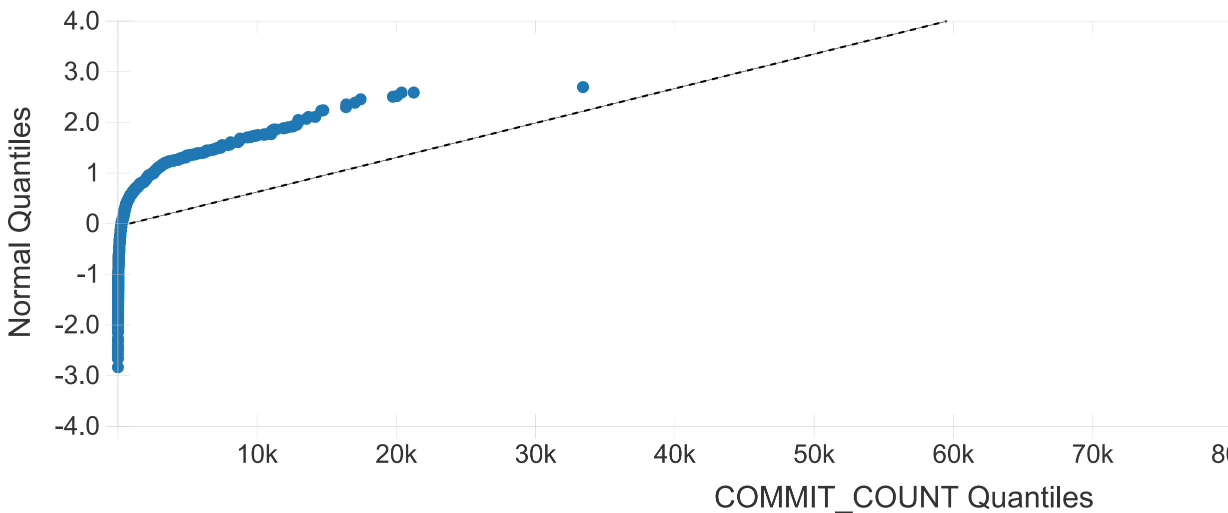
```
%sql
SELECT COUNT(*) FROM user_feature
WHERE WATCH_COUNT>3300
```

| count(1) |
|---|
| 315 |

- **Insights of WATCH_COUNT:**
  - WATCH_COUNT distribution is skewed, with highest 70% less than 300, while the largest number of 14k.
  - The distribution starts to get sparse after 3300, so this will be used as the filter condition for watch_count
  - There will be 315 WATCH_COUNT filtered out.

# COMMIT_COUNT Analysis

```
%sql
-- quantile plot of commit_count
SELECT COMMIT_COUNT FROM user_feature
```



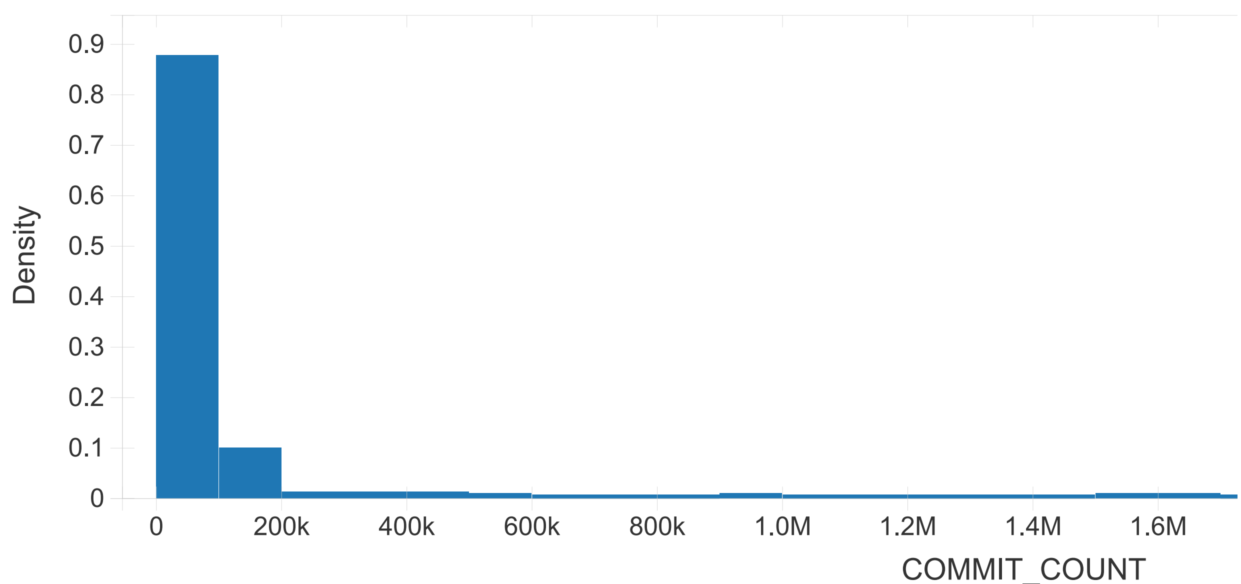

Showing sample based on the first 1000 rows.

- **Distribution Interpretation**
  - The slope of the Q-Q plot cureve is steeper than y=x, then get dramatically flatter than y=x after passing the median.
  - The initial steep slope indicates the distributiion analyzed is much denser than the normal distribution.
  - The later flat slope indicates the distribution analyzed is much sparser than the normal distribution.
  - Such curve means the distribution is highly right skewed with a very long tail.
  - As indicated by the plot, the distribution get very sparse for the range of >20k, this range will be furthur analyzed below.

```
%sql
-- show average, median, min, max values.
SELECT AVG(COMMIT_COUNT), percentile_approx(COMMIT_COUNT, 0.5) median,
MIN(COMMIT_COUNT), MAX(COMMIT_COUNT) FROM USER_FEATURE
```

| avg(COMMIT_COUNT) | median |
|---|---|
| 330.66356912953296 | 88 |

```
%sql
-- Take a closer look at COMMIT_COUNT in the range of > 22000
SELECT COMMIT_COUNT FROM user_feature
WHERE COMMIT_COUNT>22000
ORDER BY COMMIT_COUNT DESC
```




```
%sql
-- count of COMMIT_COUNT > 22000
SELECT COUNT(*) FROM user_feature
WHERE COMMIT_COUNT>22000
```

| count(1) |
|---|

| |
|---|
| 355 |

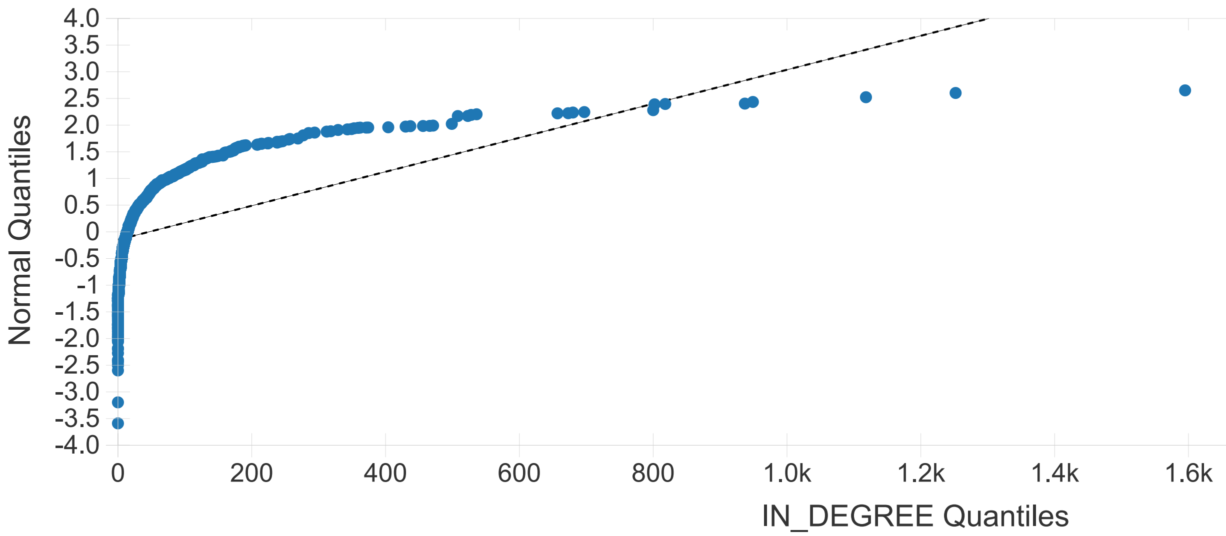
```
%sql
-- closer look at distribution in abnormal range
SELECT COMMIT_COUNT FROM user_feature
WHERE COMMIT_COUNT>22000
ORDER BY COMMIT_COUNT DESC
```

| COMMIT_COUNT |
|---|
| 3071024 |
| 2156287 |
| 2054941 |
| 1660434 |
| 1504395 |
| 995194 |
| 533991 |
| 458195 |
| 435284 |
| 347658 |

- **Insights of COMMIT_COUNT**
  - COMMIT_COUNT distribution is very skewed, there are 80% users have less than 2000 pull_count, while the largest is 3 Million.
  - Base on observation, distribution in range of >22k is very sparse, therefore 22k will be used as filtering condition for COMMIT_COUNT
  - There will be 355 COMMI_COUNT filtered out

# IN_DEGREE Analysis

```
%sql
SELECT IN_DEGREE FROM user_feature
```




Showing sample based on the first 1000 rows.

- **Distribution Interpretation**
  - The slope of the Q-Q plot cureve is steeper than y=x, then get dramatically flatter than y=x after passing the median.
  - The initial steep slope indicates the distributiion analyzed is much denser than the normal distribution.
  - The later flat slope indicates the distribution analyzed is much sparser than the normal distribution.
  - Such curve means the distribution is highly right skewed with a very long tail.
  - As indicated by the plot, the distribution get very sparse for the range of >600, this range will be furthur analyzed below.

```
%sql
-- show average, median, min, max values.
SELECT AVG(IN_DEGREE), percentile_approx(IN_DEGREE, 0.5) median,
MIN(IN_DEGREE), MAX(IN_DEGREE) FROM USER_FEATURE
```

| avg(IN_DEGREE) | median |
|---|---|
| 9.521564981045705 | 2 |

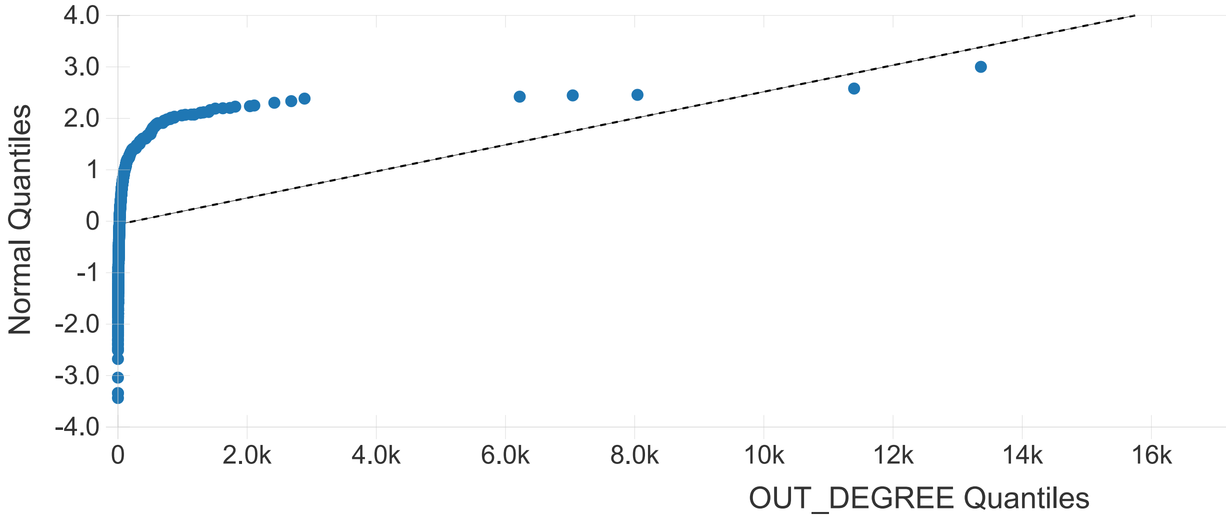
```
%sql
SELECT COUNT(IN_DEGREE) FROM user_feature
WHERE IN_DEGREE > 550
-- ORDER BY IN_DEGREE DESC
```

| count(IN_DEGREE) |
| --- |
| 469 |

- **Insights of IN_DEGREE**
  - IN_DEGREE distribution is very skewed, there are 90% users have less than 115 IN_DEGREE, while the largest is 3k.
  - Base on observation, distribution in range of >550 is very sparse, therefore 550 will be used as filtering condition for IN_DEGREE
  - There will be 469 IN_DEGREE filtered out

# OUT_DEGREE Analysis

```
%sql
SELECT OUT_DEGREE FROM user_feature
```




Showing sample based on the first 1000 rows.

- **Distribution Interpretation**
  - The slope of the Q-Q plot cureve is steeper than y=x, then get dramatically flatter than y=x after passing the median.
  - The initial steep slope indicates the distributiion analyzed is much denser than the normal distribution.
  - The later flat slope indicates the distribution analyzed is much sparser than the normal distribution.
  - Such curve means the distribution is highly right skewed with a very long tail.
  - As indicated by the plot, the distribution get very sparse for the range of >2k, this range will be furthur analyzed below.

```
%sql
-- show average, median, min, max values.
SELECT AVG(OUT_DEGREE), percentile_approx(OUT_DEGREE, 0.5) median,
MIN(OUT_DEGREE), MAX(OUT_DEGREE) FROM USER_FEATURE
```

| avg(OUT_DEGREE) | median |
|---|---|
| 12.136080871659157 | 3 |

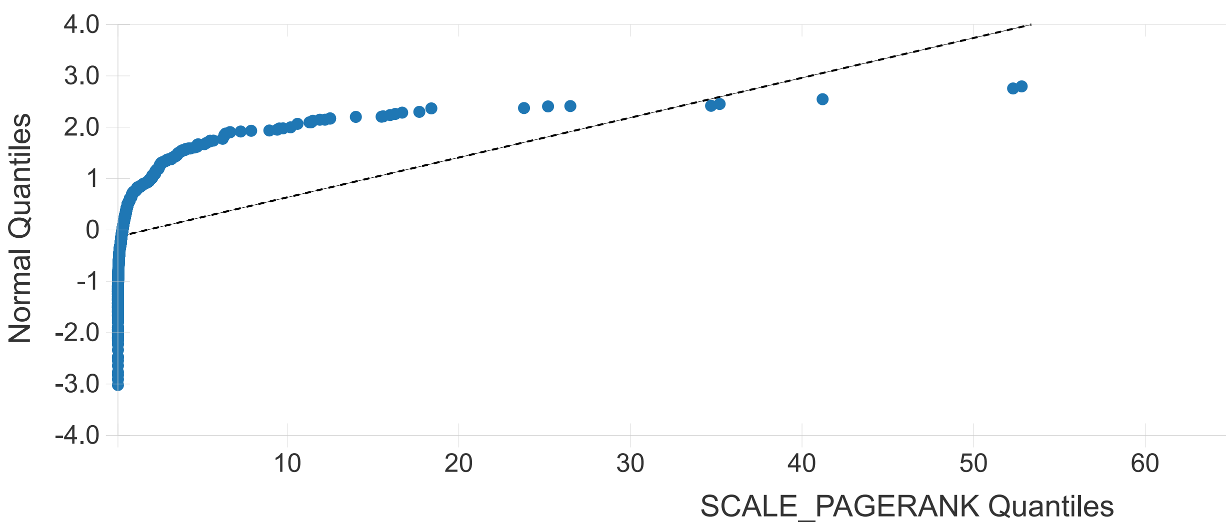
```
%sql
SELECT COUNT(OUT_DEGREE) FROM user_feature
WHERE OUT_DEGREE > 1500
```

| count(OUT_DEGREE) |
|---|
| 451 |

- **Insights of OUT_DEGREE**
  - OUT_DEGREE distribution is very skewed, there are 90% users have less than 200 OUT_DEGREE, while the largest is 30k.
  - Base on observation, distribution in range of >1500 is very sparse, therefore 1500 will be used as filtering condition for OUT_DEGREE
  - There will be 451 OUT_DEGREE filtered out

# PAGERANK Analysis

```
%sql
SELECT PAGERANK*1000000 AS SCALE_PAGERANK FROM user_feature
```




Showing sample based on the first 1000 rows.

```
%sql
-- show average, median, min, max values.
SELECT AVG(PAGERANK)*1000000, percentile_approx(PAGERANK, 0.5)*1000000 median,
MIN(PAGERANK)*1000000,  MAX(PAGERANK)*1000000 FROM USER_FEATURE
```

| (avg(CAST(PAGERANK AS DOUBLE)) * CAST(1000000 AS DOUBLE)) |
|---|
| 0.6703826343907858 |

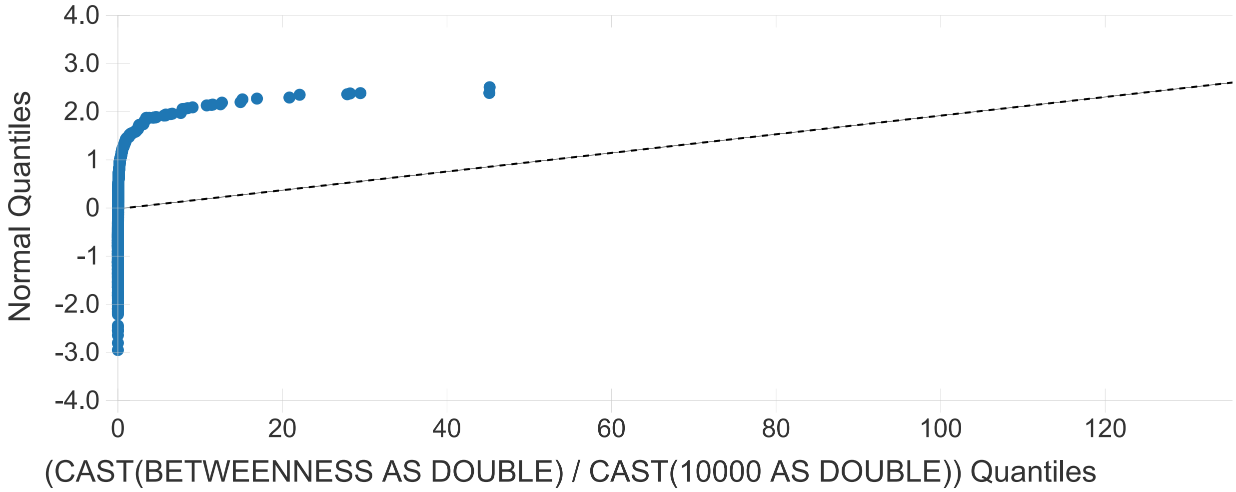
```
%sql
SELECT COUNT(PAGERANK*1000000) FROM user_feature
WHERE PAGERANK*1000000 > 73
```

| count((PAGERANK * CAST(1000000 AS FLOAT))) |
|---|
| 313 |

- **Insights of PAGERANK**
  - PAGERANK distribution is very skewed, there are 90% users have less than 2.*510E-6 PAGERANK, while the largest is 1.3*10E-2.
  - Base on observation, distribution in range of >7.3*10E-5 is very sparse, therefore it will be used as filtering condition for PAGERANK
  - There will be 313 PAGERANK filtered out

# BETWEENNESS Analysis

```
%sql
SELECT BETWEENNESS/10000 FROM user_feature
```




Showing sample based on the first 1000 rows.

- **Distribution Interpretation**
  - The slope of the Q-Q plot cureve is steeper than y=x, then get dramatically flatter than y=x after passing the median.
  - The initial steep slope indicates the distributiion analyzed is much denser than the normal distribution.
  - The later flat slope indicates the distribution analyzed is much sparser than the normal distribution.
  - Such curve means the distribution is highly right skewed with a very long tail.
  - As indicated by the plot, the distribution get very sparse for the range of >2*10E3, this range will be furthur analyzed below.

```
%sql
-- show average, median, min, max values.
SELECT AVG(BETWEENNESS/10000), percentile_approx(BETWEENNESS/10000, 0.5)
MEDIAN, MIN(BETWEENNESS/10000), MAX(BETWEENNESS/10000) FROM USER_FEATURE
```

| avg((CAST(BETWEENNESS AS DOUBLE) / CAST(10000 AS DOUBLE))) | MEDIAN |
|---|---|
| 0.16199118725947784 | 0.000083 |

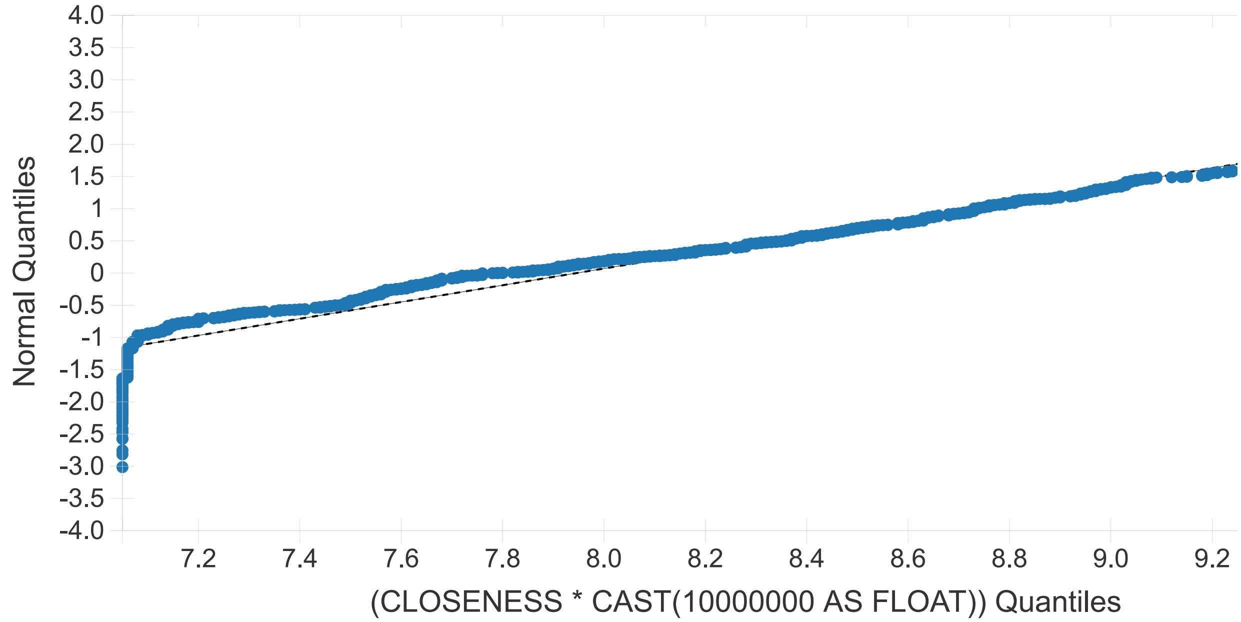
```
%sql
SELECT COUNT(BETWEENNESS/10000) FROM user_feature
WHERE BETWEENNESS/10000 > 10
```

| count((CAST(BETWEENNESS AS DOUBLE) / CAST(10000 AS DOUBLE))) |
|---|
| 347 |

- **Insights of BETWEENNESS**
  - **The BETWEENNESS here is an estimation calculated by considering only edges with length less or equal to 2 for each vertex, therefore it's only meaningful to compare the relative magnitude instead of the accurate value**
  - BETWEENNESS distribution is very skewed, there are 90% users have less than 7*10E3 BETWEENNESS, while the largest is 4.76*10E8.
  - Base on observation, distribution in range of > 10E5 is very sparse, therefore it will be used as filtering condition for BETWEENNESS
  - There will be 347 BETWEENNESS filtered out

# CLOSENESS Analysis

```
%sql
SELECT CLOSENESS*10000000 FROM user_feature
WHERE CLOSENESS > 0
```



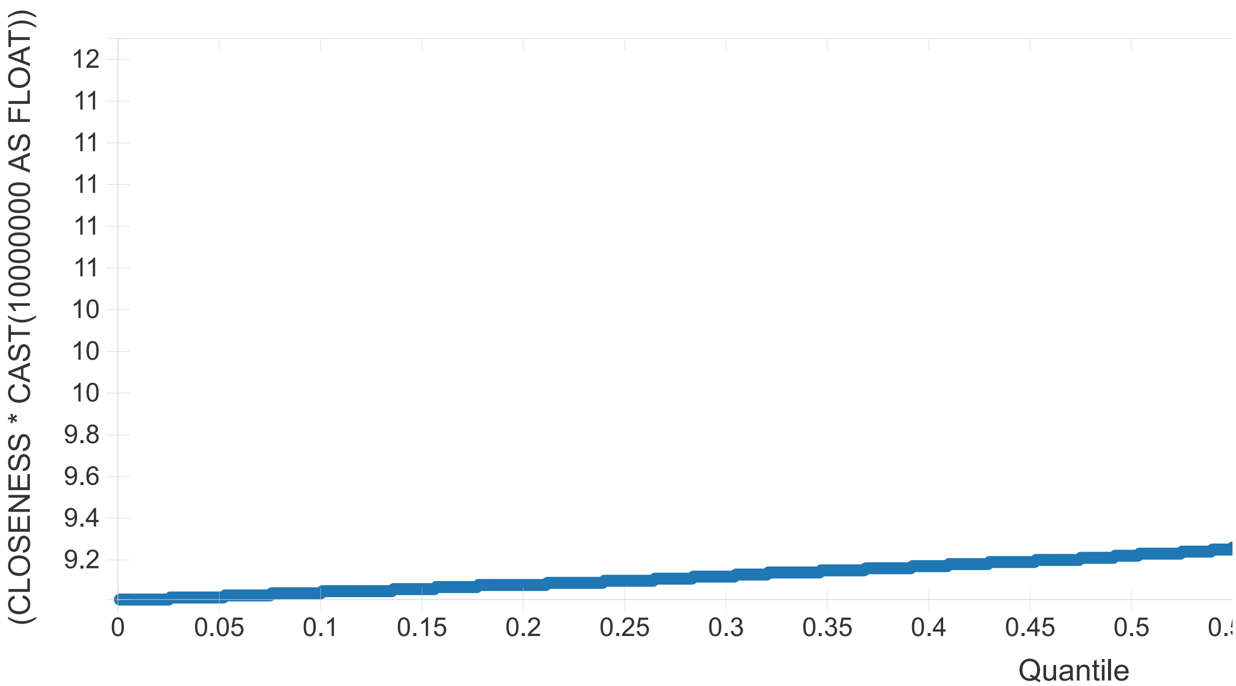

Showing sample based on the first 1000 rows.

- **Distribution Interpretation**
  - The slope of the Q-Q plot cureve is steeper than y=x initally, then the plot lies on a strait line after reaching -sigma in normal distribution.
  - The initial steep slope indicates the distributiion analyzed is much denser than the normal distribution.
  - After the -sigma in the normal distribution, the corresponding distribution roughly lies on the y=x, indicating this part of distribution is very similar to normal distribution, but with a different mean value.
  - Such curve means the distribution is very dense at the left end, however, the distribution behaves very similar to the normal distribution.
  - As indicated by the plot, the distribution get very sparse at the right hand side, this range will be furthur analyzed below.

```
%sql
-- show average, median, min, max values.
SELECT AVG(CLOSENESS*10000000), percentile_approx(CLOSENESS*10000000, 0.5)
median, MIN(CLOSENESS*10000000), MAX(CLOSENESS*10000000) FROM USER_FEATURE
```

| avg((CLOSENESS * CAST(10000000 AS FLOAT))) | medi |
|---|---|
| 7.296325912475131 | 7.06 |

```
%sql
SELECT CLOSENESS*10000000 FROM user_feature
WHERE CLOSENESS*10000000 > 9
```




Showing sample based on the first 1000 rows.

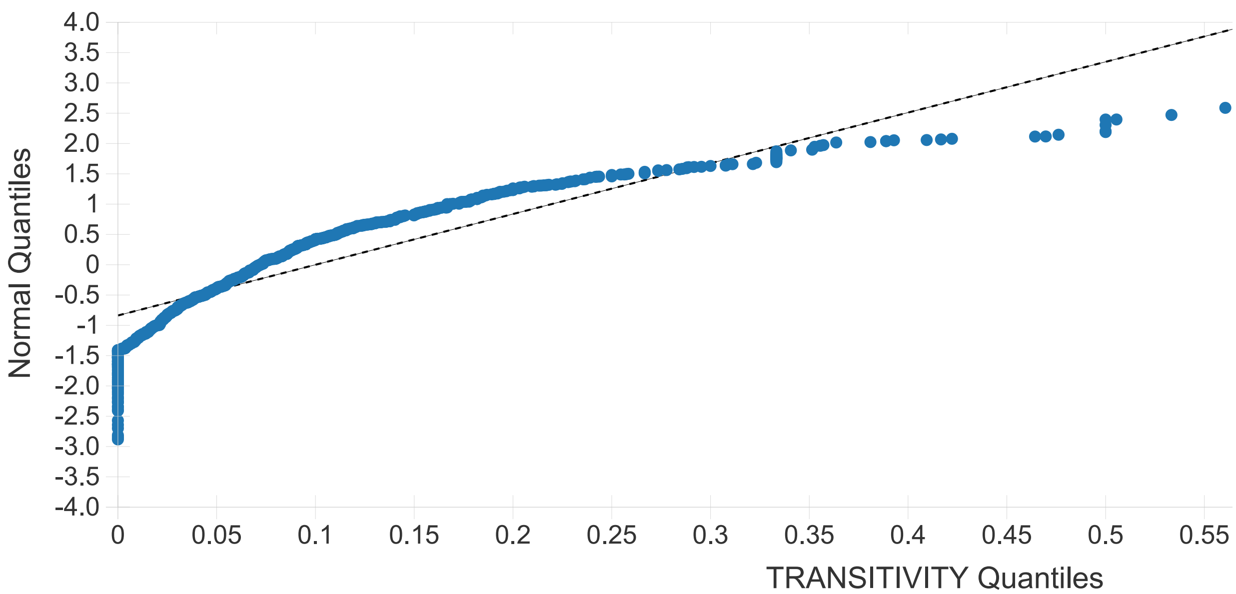
```
%sql
SELECT COUNT(CLOSENESS*10000000) FROM user_feature
WHERE CLOSENESS*10000000 > 9.6
```

| count((CLOSENESS * CAST(10000000 AS FLOAT))) |
|---|
| 360 |

- **Insights of CLOSENESS**
  - CLOSENESS distribution is linearly proportional to normal distribution for the most part.
  - Base on observation, distribution in range of > 9.6E-7 is very sparse, therefore it will be used as filtering condition for CLOSENESS
  - There will be 360 CLOSENESS filtered out

# TRANSITIVITY Analysis

```
%sql
SELECT TRANSITIVITY FROM user_feature
```




Showing sample based on the first 1000 rows.

- **Distribution Interpretation**
  - The slope of the Q-Q plot cureve is steeper than y=x initially, then the plot lies nearly on a straight line in the middle, at the right end, the plot is flatter.
  - The initial steep slope indicates the distributiion analyzed is much denser than the normal distribution.

- In the middle of the plot, the corresponding distribution roughly lies on the y=x, indicating this part of distribution is very similar to normal distribution, but with a different mean value.
- The right end of the plot has an slope flatter than y=x, indicating this part of distribution is sparser than normal distribution.
- Such curve means the distribution is very dense at the left end, then the middle part of the distribution follow a near normal distribution, then the tail of the distribution is sparse.
- Alghouth the distribution is nor exactly normal for transitivity, but they all fall in the range of [0,1]. The definition of transitivity is the percentage of the neighbour vertices connected with each other. It's normal for vertices with high degree have low transitivity, while vertices with low degree have high transitivity. Therefore, the distribution is acceptable.

```
%sql
-- show average, median, min, max values.
SELECT AVG(TRANSITIVITY), percentile_approx(TRANSITIVITY, 0.5) median,
MIN(TRANSITIVITY), MAX(TRANSITIVITY) FROM USER_FEATURE
```

| avg(TRANSITIVITY) | median |
|---|---|
| 0.1246043593383146 | 0.03696919 |

```
%sql
SELECT TRANSITIVITY FROM user_feature
ORDER BY TRANSITIVITY DESC
LIMIT 100
```

| TRANSITIVITY |
|---|
| 1 |
| 1 |
| 1 |
| 1 |
| 1 |
| 1 |

1

- **Insights of transitivity:**
  - The transitivity is accurately calculated for each node
  - The distribution is roughly normal distribution except for the nodes with transitivity of 0.
  - It won't be filtered.

# Data Preparation for K-Means Clustering

In this step, the feature dataframe will be preprocessed, the the k-means clustering algorithm will be applied on the feature data to divide the users into different roles. Following steps will be performed in this step:

- The features will be filtered based on the filtering condition concluded in the above feature exploration
- Z-standaization will be performed to transfer all featutes into standar normal distribution X~N(0,1). The reason is stated below:
  - The k-means is a distance based clustering algorithm, thus the features representing different aspects with different scale should be normalized.
  - k-means tends to produce circular clusters in spatial, thus will be sensitive to higher variance, the features need to be standaized to improve clustering quality.
- k-means clustering will be performed on the preprocessed features. The optimal number of k is determined through "elbow method".
- Resulting cluster centroids will be plotted and analyzed. Various user roles will be identified from the clustering result

## Applying filters

- Apply the filters obtained from feature exploration to the data.

```
# applying filter condition
df =
df.filter("PULL_COUNT<5000").filter("WATCH_COUNT<3300").filter("COMMIT_COUNT<22
000").filter("IN_DEGREE<550").filter("OUT_DEGREE<1500")\

.filter("pagerank<7.3/pow(10,-5)").filter("BETWEENNESS<pow(10,5)").filter("CLOS
ENESS<9.6*pow(10,-7)")


# convert the filtered sparkDF to pandasDF, which will be used in the cluster
centroid interpretation at the end of the analysis
features_filtered_df = df.toPandas()


# intotal 1048575-1046752=1823 rows are filtered out, which accounts for 0.1%
of the total records.
df.count()
```

```
Out[15]: 1046752
```

```
# store column names
headers = df.columns
FEATURES_COL = headers[1:]
FEATURES_COL
```

```
Out[16]:
['PULL_COUNT',
 'WATCH_COUNT',
 'COMMIT_COUNT',
 'IN_DEGREE',
 'OUT_DEGREE',
 'PAGERANK',
 'BETWEENNESS',
 'CLOSENESS',
 'TRANSITIVITY']
```

# Feature Standarizing

- Scale the features to standard normal distribution to improve clustering performance
- **Reason**: The algroithm to be used is K-means clustering, which is an unsupervised algorithm based on the distance to the cluster centroids. The algorithm tends to produce spherical shaped clusters. However, if features are not standarized, the effect of each feature to the model will be biased. For example, if

some features has significantly large means than others, they will have more effect in the distance calculation, and the features with small mean would more likely to be ignored. Additionally, even each feature has exactly same mean but vastly different variances, the clustering performance would still decrease. Because the large variance means the feature datum would be distorted to an wider range. Therefore, the clustering algorithm would be affected more by the features with large variance. Thus, standarizing features to normal distribution X~N(0,1) would increase the clustering quality.

```
from pyspark.ml.feature import StandardScaler,VectorAssembler,MinMaxScaler
from pyspark.ml.linalg import Vectors
from pyspark.ml.clustering import KMeans, BisectingKMeans
from tqdm import tqdm
from sklearn.preprocessing import MinMaxScaler
```

```
# create assembled vectors using the dataframe columns, to be used in spark ML
vecAssembler = VectorAssembler(inputCols=FEATURES_COL,outputCol='featuresVec')
df_kmeans = vecAssembler.transform(df).select("user_id","featuresVec")
df_kmeans.show()
```

```
+-------+--------------------+
|user_id|         featuresVec|
+-------+--------------------+
|      1|[13.0,59.0,55.0,2...|
|      2|[228.0,706.0,1284...|
|      4|[148.0,248.0,1538...|
|      5|[1224.0,64.0,1225...|
|      7|[11.0,336.0,243.0...|
|      8|[5.0,549.0,30.0,1...|
|      9|[41.0,146.0,3262....|
|     12|[223.0,28.0,1888....|
|     13|[72.0,118.0,563.0...|
|     14|[425.0,129.0,1071...|
|     17|[0.0,243.0,24.0,4...|
|     19|[155.0,1162.0,141...|
|     21|[15.0,11.0,146.0,...|
|     24|[717.0,5.0,4392.0...|
|     26|[139.0,683.0,2823...|
|     28|[99.0,61.0,1223.0...|
|     29|[46.0,15.0,174.0,...|
|     33|[402.0,116.0,4733...|
|     35|[65.0,5.0,86.0,0....|
```

```
|     36|[33.0,5.0,147.0,0...|
+-------+--------------------+
only showing top 20 rows
```

```
# z-standarization to re-scale distribution to X~N(0,1)
z_scaler = StandardScaler(inputCol="featuresVec",
outputCol="scaledFeaturesVec",withStd=True,withMean=True)
scalerModel = z_scaler.fit(df_kmeans)
df_kmeans_scaled =
scalerModel.transform(df_kmeans).select("user_id","scaledFeaturesVec")
df_kmeans_scaled.show()
```

```
+-------+--------------------+
|user_id|   scaledFeaturesVec|
+-------+--------------------+
|      1|[-0.1369683589671...|
|      2|[1.29308826960552...|
|      4|[0.76097417525290...|
|      5|[7.91790874429555...|
|      7|[-0.1502712113259...|
|      8|[-0.1901797684023...|
|      9|[0.04927157405629...|
|     12|[1.25983113870848...|
|     13|[0.25546578561792...|
|     14|[2.60341922694883...|
|     17|[-0.2234368992994...|
|     19|[0.80753415850876...|
|     21|[-0.1236655066083...|
|     24|[4.54563567133586...|
|     26|[0.70111133963824...|
|     28|[0.43505429246193...|
|     29|[0.08252870495332...|
|     33|[2.45043642482245...|
|     35|[0.20890580236207...|
|     36|[-0.0039398353789...|
+-------+--------------------+
only showing top 20 rows
```

# Applying k-means clustering

- The referenced paper used bisecting k-means in their experiemnt. Both the referenced paper and other highly cited peer reviewed papers claimed bisecting k-

means has better
performance in user-role identification [1][8].
- Experiment with a range of k values and observe cost function change
- Select the k using elbow method.

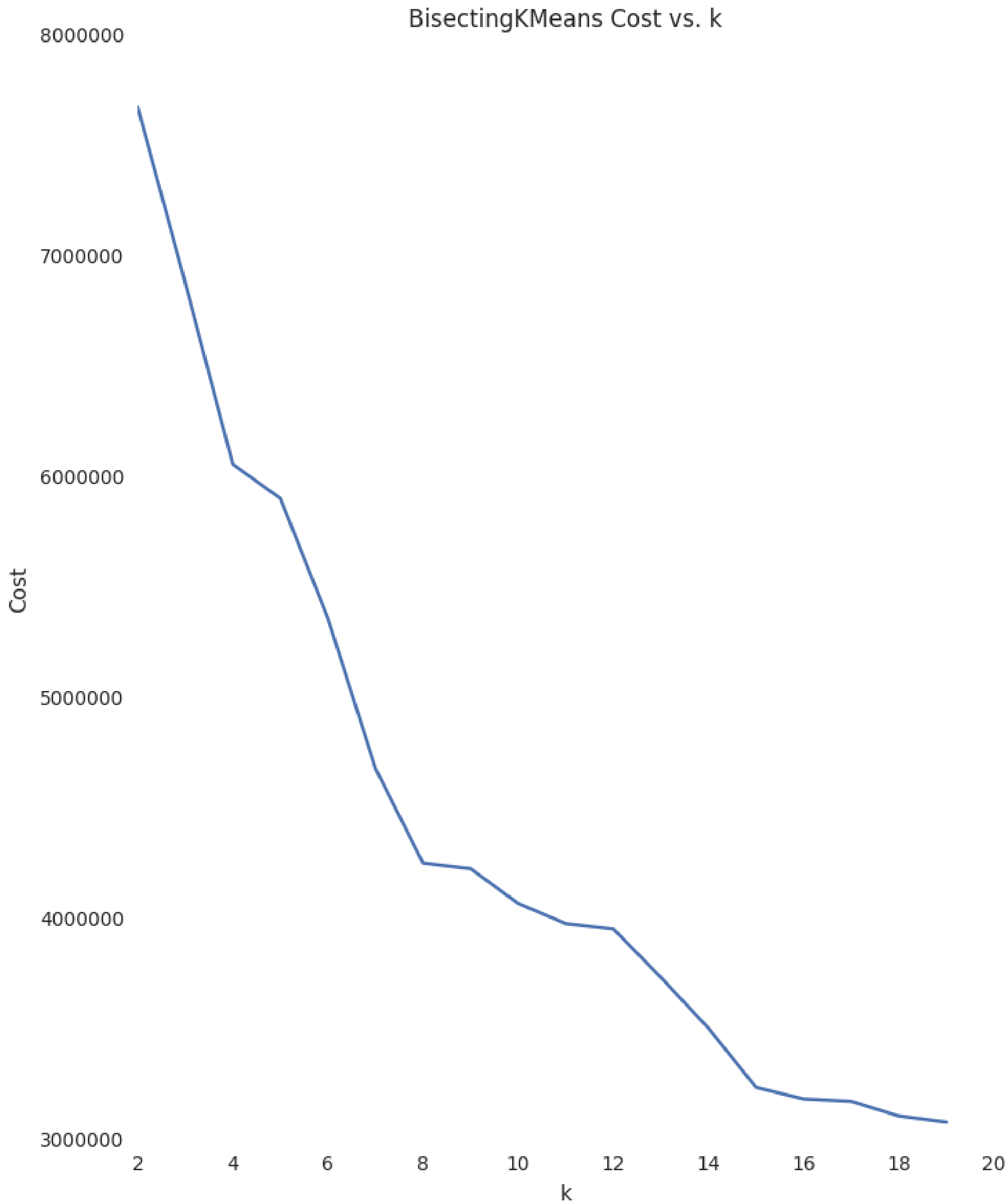
# Experimenting with Bisecting k-means

- Bisecting k-means implementation in Spark ML was used to cluster user features into distinct clusters
- Number of clusters k in range [2,20) were used, and the corresponding cost function were evaluated.
- The relationship of cost function vs. k was plotted, and the optimal k was selecte at the point on the curve, where the slope start to get flatter.

```
# gather cost for models of k in range [2,20)
cost_bisec = []
for k in tqdm(range(2,20)):
  bisec_kmeans=  BisectingKMeans(k=k, seed=1,featuresCol="scaledFeaturesVec")
  model = bisec_kmeans.fit(df_kmeans_scaled)
  cost_bisec.append(model.computeCost(df_kmeans_scaled))
```

```
  0%|          | 0/18 [00:00<?, ?it/s][AException KeyError: KeyError(<weakref at 0x7fa95dba8f70; to 'tqdm' at 0x7fa95dbc0f10>,) in <bound method tqdm.__del__ of   0%|          | 0/18 [08:20<?, ?it/s]> ignored


  6%|5         | 1/18 [01:13<20:56, 73.94s/it][A

 11%|#1        | 2/18 [03:37<28:57, 108.62s/it][A

 17%|#6        | 3/18 [05:59<29:57, 119.86s/it][A

 22%|##2       | 4/18 [09:26<33:03, 141.68s/it][A

 28%|##7       | 5/18 [13:02<33:54, 156.53s/it][A

 33%|###3      | 6/18 [16:40<33:21, 166.82s/it][A

 39%|###8      | 7/18 [20:25<32:05, 175.02s/it][A
```

```
# plot cost function vs. #k
fig, ax = plt.subplots(1,1,figsize=(8,10))
ax.plot(range(2,20),cost_bisec)
ax.set_xlabel('k')
ax.set_ylabel('Cost')
ax.set_title('BisectingKMeans Cost vs. k')
display(fig)
```




**Cost Function Curve Interpretation**

- Cost functions were plotted for k in range of [2,20).
- The curve can be divided into 3 different regions:

- For k in range [2,8], the cost function decreases dramatically.
- For k in range [8,12], the cost function stays steady.
- For k in range [12,20], the cost function starts to decrease again, but with much flatter slope.

- The above analysis indicating that the number of k=8 reaches a good trade-off between the cost function and the model complexity.
- k=8 was chosen as the optimal number, and will be used in the following experiment.

## Model with optimal k=8 with bisecting k-means

```
# model with optimal k = 8
bisec_model =  BisectingKMeans(k=8, seed=42,featuresCol="scaledFeaturesVec")
bisec_trained_model = bisec_model.fit(df_kmeans_scaled)
centroids = bisec_trained_model.clusterCenters()
```

```
# show scaled centroid
centroids
```

```
Out[23]:
[array([-0.17466633, -0.236978  , -0.2413299 , -0.27100268, -0.20222955,
        -0.117041  , -0.09162857, -0.48182674, -0.50583709]),
 array([ 0.13132573, -0.10755457,  0.24780537,  0.01512766,  0.03859984,
         0.03017531, -0.05905921, -0.27347133,  0.13853156]),
 array([-0.12354803, -0.21713497, -0.18870746, -0.31061003, -0.1845108 ,
        -0.16497657, -0.09373468, -0.16472669,  3.26740606]),
 array([-0.03632452,  0.35328306,  0.00254077,  0.2802216 ,  0.12204017,
         0.05842978, -0.03172847,  1.51838086,  0.34717537]),
 array([ 7.19066081,  0.40317156,  6.98228452,  0.43515385,  2.16898288,
         0.08938219,  0.47586917,  0.84127272, -0.22859302]),
 array([ 0.47301266,  3.03694514,  0.80083069,  2.97623889,  1.5559899 ,
         1.30240778,  1.02989927,  2.63269289, -0.18433876]),
 array([  2.70055436,   3.7501474 ,   2.8603574 ,   7.95027217,
          9.85932694,   6.97020128,  10.15198869,   3.30713756,  -0.4239369
2]),
 array([  4.22241314,   4.81945372,   4.37607427,  10.28010826,
         15.95984085,   3.0547539 ,  36.25423307,   3.97865122,  -0.4463974
1])]
```

```
# assginment to centroids
transformed =
bisec_trained_model.transform(df_kmeans_scaled).select("user_id","prediction")
rows = transformed.toPandas() # retrieve all rows to single machine


# find percentage for each cluster
prediction_dist = rows.groupby(["prediction"]).sum().apply(lambda x:
x/x.sum()).reset_index()
prediction_dist
```

```
Out[25]:
   prediction   user_id
0           0  0.610411
1           1  0.184993
2           2  0.069417
3           3  0.114735
4           4  0.002571
5           5  0.017000
6           6  0.000771
7           7  0.000103
```

```
# cluster size by label
rows['prediction'].value_counts()
```

```
Out[26]:
0    561197
1    221209
3    156259
2     63696
5     33656
4      8222
6      2079
7       434
Name: prediction, dtype: int64
```

```
rows['prediction'].value_counts().values
```

```
Out[27]: array([561197, 221209, 156259,  63696,  33656,   8222,   2079,    43
4])
```

### show unscaled feature means for each cluster

- The scaled features were used to cluster the users
- After clustering, the centroids are calculated again with the unscaled features, which can represent the real user behavior for each cluster.

```
# convert filtered sparkdf to pandas df
features_filtered_df = df.toPandas()
# add the cluster prediction to the original featurs
features_filtered_df =
features_filtered_df.merge(rows,left_on='USER_ID',right_on='user_id')
# show average features per cluster, order by cluster size
features_filtered_df =
features_filtered_df.drop(['USER_ID','user_id'],axis=1).groupby("prediction").m
ean().reset_index().merge(prediction_dist,on='prediction')
features_filtered_df =
features_filtered_df.iloc[features_filtered_df['user_id'].argsort()[::-1]]
```

```
temp_header = features_filtered_df.columns.values
temp_header[-1]='percentage'
features_filtered_df.columns = temp_header
features_filtered_df
```

```
Out[29]:
   prediction   PULL_COUNT  WATCH_COUNT  COMMIT_COUNT   IN_DEGREE  OUT_DEGREE  \
0           0     7.332259     9.819726    107.294952    2.075793    2.600443

1           1    53.342581    27.174279    492.968880    7.874096   11.678571

3           3    28.131167    88.981505    299.555027   13.246629   14.823364

2           2    15.017662    12.479449    148.783048    1.272764    3.268055

5           5   104.721862   448.879487    929.253684   67.881329   68.878357

4           4  1114.969715    95.639382   5802.389929   16.380443   91.978716

6           6   439.603656   544.554594   2552.536316  168.688793  381.866282

7           7   668.405530   687.965438   3747.463134  215.905530  611.822581


       PAGERANK   BETWEENNESS     CLOSENESS  TRANSITIVITY  percentage
0  2.887994e-07      3.317114  7.089363e-07      0.016549    0.610411
1  6.097517e-07     48.880920  7.177778e-07      0.154387    0.184993
3  6.713395e-07     87.113373  7.938200e-07      0.199014    0.114735
2  1.842805e-07      0.369239  7.223928e-07      0.823689    0.069417
5  3.383217e-06   1572.330078  8.411071e-07      0.085321    0.017000
```

```
4  7.385885e-07    796.762207  7.650641e-07      0.075826    0.002571
6  1.574002e-05  14333.715820  8.697306e-07      0.034063    0.000771
7  7.203770e-06  50849.683594  8.982281e-07      0.029258    0.000103
```

# Centroids observation & User role identification

- The centroid of clusters calculated in the previous section with unscaled feature data were rescaled into the same range, as we are only inerested in the relative magnitude of each feature across all centroids.
- The relative feature magnitudes were ploted as grouped histograms, and those histograms were observed and explained.
- Each role characteristics were concluded from the observation.

```
# estabilish centroid pandasdf with feature name
centroids_df = pd.DataFrame(centroids)
centroids_df.columns = FEATURES_COL
```

```
# add index column for grouping
centroids_df["Role"] = centroids_df.index
# add percentage for each role type
centroids_df["Percentage"] = prediction_dist['user_id'].apply(lambda x: "%.3f"%
(x*100)+"%")
```

```
# sort by Percentage
centroids_df = centroids_df.iloc[centroids_df.Percentage.apply(lambda x: -
float(x[:-1])).argsort()]
```

```
# scale each feature of the centroids to the range of [1,10], we only care
about the relative magnitude for visualization
mmScaler = MinMaxScaler(feature_range=(1, 10))
centroids_df[FEATURES_COL] = mmScaler.fit_transform(centroids_df[FEATURES_COL])
# create spark dataframe
centroid_spark_df = sqlContext.createDataFrame(centroids_df)
# register sparkSQL view
centroid_spark_df.createOrReplaceTempView("centroids")
```

```
%sql
-- show scaled features in range 1,10
SELECT * FROM centroids
```

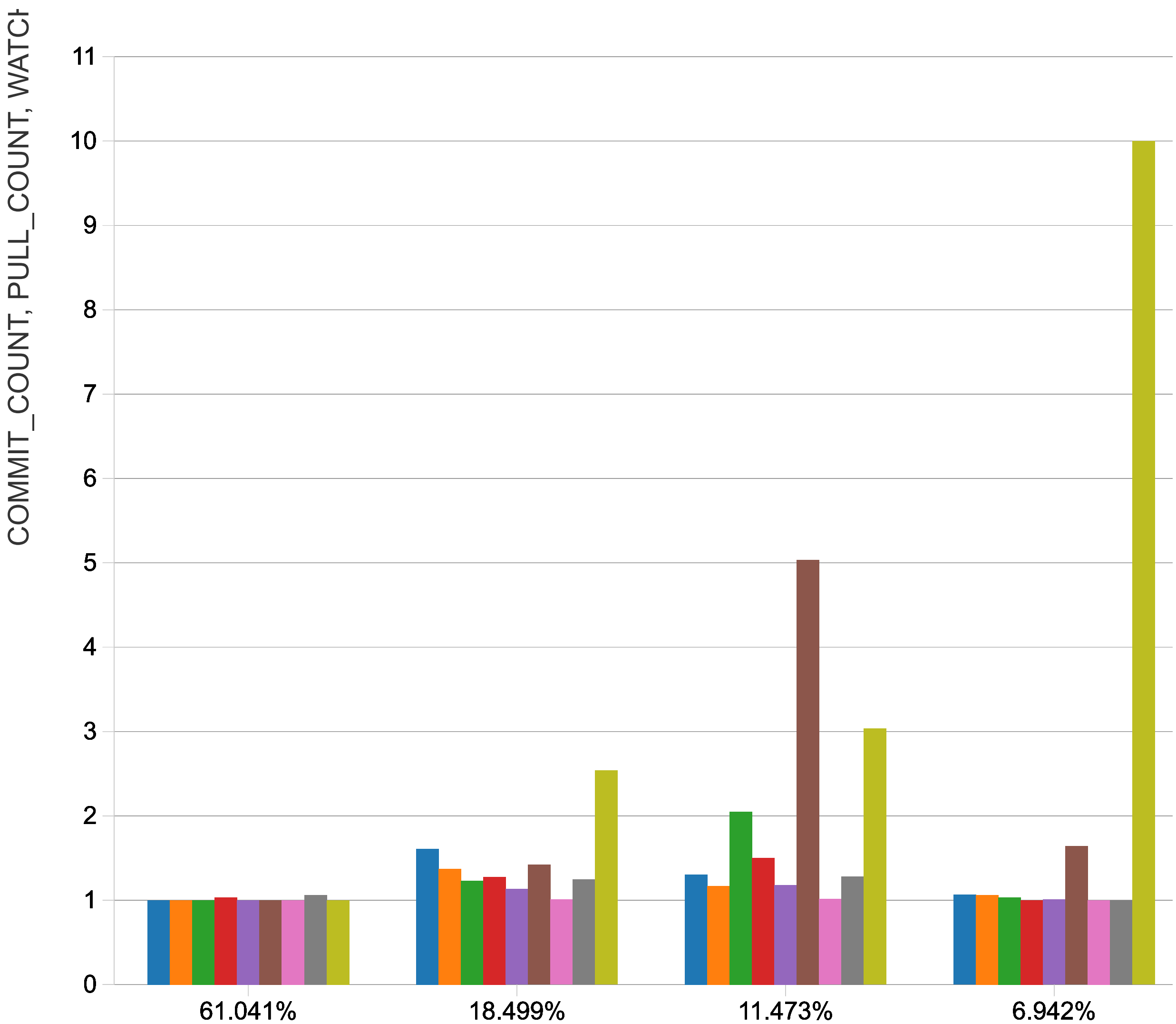
| PULL_COUNT | WATCH_COUNT | COMMIT_COUNT | IN_DEGR |
|---|---|---|---|
| 1 | 1 | 1 | 1.0336583 |
| 1.3739044448028959 | 1.2303622309681603 | 1.6094203210937654 | 1.276812 |
| 1.1690456183050528 | 2.0506123366183573 | 1.303841796358784 | 1.502089 |
| 1.0624635793921604 | 1.035318828016575 | 1.065563017589642 | 1 |
| 1.7914259361744003 | 6.827292823714274 | 2.2984421312385455 | 3.793166 |
| 10.000000000000002 | 2.1394094498038836 | 10 | 1.633750 |
| 4.513351918800711 | 8.096729588785003 | 4.864434628575148 | 8.020103 |
| 6.3729745445146175 | 10 | 6.752887003057271 | 10 |

```
# show aggregated features
features_filtered_df.reset_index(drop=True)
```

```
Out[34]:
   prediction   PULL_COUNT  WATCH_COUNT  COMMIT_COUNT   IN_DEGREE  OUT_DEGREE  \
0           0     7.332259     9.819726    107.294952    2.075793    2.600443

1           1    53.342581    27.174279    492.968880    7.874096   11.678571

2           3    28.131167    88.981505    299.555027   13.246629   14.823364

3           2    15.017662    12.479449    148.783048    1.272764    3.268055

4           5   104.721862   448.879487    929.253684   67.881329   68.878357

5           4  1114.969715    95.639382   5802.389929   16.380443   91.978716

6           6   439.603656   544.554594   2552.536316  168.688793  381.866282

7           7   668.405530   687.965438   3747.463134  215.905530  611.822581


       PAGERANK   BETWEENNESS     CLOSENESS  TRANSITIVITY  percentage
0  2.887994e-07      3.317114  7.089363e-07      0.016549    0.610411
1  6.097517e-07     48.880920  7.177778e-07      0.154387    0.184993
2  6.713395e-07     87.113373  7.938200e-07      0.199014    0.114735
3  1.842805e-07      0.369239  7.223928e-07      0.823689    0.069417
4  3.383217e-06   1572.330078  8.411071e-07      0.085321    0.017000
5  7.385885e-07    796.762207  7.650641e-07      0.075826    0.002571
```

```
6  1.574002e-05  14333.715820  8.697306e-07      0.034063    0.000771
7  7.203770e-06  50849.683594  8.982281e-07      0.029258    0.000103
```

```
%sql
-- group by clusters, sort group by percentage
-- y-axis shows only the relative magnitude of feature after scaling
SELECT * FROM centroids
```




### Experiment Results

After the clustering, the optimal number of cluster was determined to be 8.
Users vertices are assigned to the cluster, whose centroid is closest to the vertex. The clustering model tries to maximize the inter-cluster variance while minimize the intra-

cluster vairance. As an result, each cluster can be treated as an group of vertices/users with similar features. Thus the users in each cluster could be represented by the cluster centroid, and the behvior of users can be explained with their centroid features.

---

### Features Description

The clustering result is shown in the above graph. There are 9 different features been considered in the clustering modeling. According to the well cited paper on social network link prediction by Hasan, Chaoji, and Salem [9], features in social network analysis can be divided to aggregated features and topological features. Aggregated features ar the user-specific feature calculated with some aggregated function, and topological features are features induced from network structure based on graph theory. Among the 9 feature, 5 are aggregated features, and 4 are topological features.

- Aggregated Features:
  - COMMIT_COUNT: number of commit made by the user to date.
  - PULL_COUNT: number of pull request made by the user to date.
  - WATCH_COUNT: number of repository that an user is watching.
  - IN_DEGREE: number of followers owned by an user.
  - OUT_DEGREE: number of other users followed by the user.
- Topological Features:
  - CLOSENESS CENTRALITY:
    - calculated as: $C(u) = \frac{n-1}{\sum_{v=1}^{n-1} d(v,u)}$, where n is number of vertices in the graph, d(u,v) stands for the shortest path distance between nodes u and v.
    - measures how easily can the node be accessed from other nodes, or access to other nodes from this node. The larger the closeness, the higher it's centrality.
  - BETWEENNESS CENTRALITY:
    - calculated as: $c_B(v) = \sum_{s,t \in V} \frac{\sigma(s,t|v)}{\sigma(s,t)}$, it's the number of shortest path passes through specific vertex divided by total number of vertices in the graph.

    - measures the possibility that the information flow in a network passes through the specific vertex.
  - PAGERANK:
    - pageRank computes a ranking of the nodes in the graph G based on the structure of the incoming links. It was originally designed as an algorithm to rank web pages.
    - measures the authority of a user/vetex in a directed graph
  - TRANSITIVITY (CLUSTERING COEFFICIENT)
    - calculated as $C_v = \frac{N_v}{\left(\frac{d_v(d_v - 1)}{2}\right)}$ , where n is the number of triangles contianing an specific vertex, and d is the degree of the vertex.
    - measures the probability of neiberghs of a users/vertex are also connected.

---

**Role Identification**

- Role percentage distribution
  - Out of the total 1.04 million users in the follow network, each cluster accounts for 61%, 18.5%, 11.5%, 7%, 1.7%, 0.26%, 0.077% and 0.01% respectively. The corresponding cluster sized are: 561k, 221k,156k, 63.7k, 33.6k, 8.2k, 2k, and 434.
  - Role Descriptions:
    - Role #1 (Cluster #0): **INACTIVE USERS 1** 61%
      - This group has very low aggregated and topological features. As described in paper: The promises and perils of mining github [6], lots of user only use the Github for personal project, storage or experiement.
      - This group of user is inactive in terms of activity and social networking, and it's account for majority of the users, which is aligned with the arguement in the paper.
    - Role #2 (Cluster #1): **NORMAL PROGRAMMERS** 18.5%

    - This group has reasonable AVG_COMMIT_COUNT of 500, they also use pull request and watch interesting repositories. This group represents the normal programmers on the Github, they use Github frequently, but their work is not distinguishable. ***This group can be used as an benchmark group for role comparing***
    - They focus more on their own repositories, for the reason they have relatively low AVG_IN/OUT_DEGREE of arount 10.
    - None of their PAGERANK, BETWEENNESS, nor CLOSENESS are high indicating they are not really influencial in the follow network
    - They have medium level of CLUSTERING COEFFICIENT (TRANSITIVITY) among all clusters, this can be explained by the normal users tends to follow their familiar people, for example, their colleges or classmates. Thus they form small communities, in which they have high CLUSTERING COEFFICIENT.

- Role #3 (Cluster #3) **ACTIVE LEARNERS** 11.5%
    - This group is less active in contributing to open-source coding projects compared to the NORMAL PROGRAMMERS described above. However, they watch more repositories, and they also follow more other users compared to NORMAL PROGRAMMERS. This means this group is less skilled in programming, but they are actively learning from famous users and repos.
    - This group has less COMMIT_COUNT(300) and PULL_COUNT(30) than the 50 and 500 of NORMAL PROGRAMMERS.
    - This group watch 3 times more repo than NORMAL PROGRAMMERS.
    - This group has really much higher closeess than that of NORMAL PROGRAMMERS, for the reason that this group follows more famous users, so that the shortest path to other nodes are reduces.

- Role #4 (Cluster #2) **MUTUAL FOLLOWING INACTIVE USERS** 7%
    - This is another group of very inactive users that is similar to Role #1, the difference is that this group has very high CLUSTERING COEFFICIENT.
    - This group has average IN/OUT DEGREE of 1 and 3, which is very small.

    - The reason for the high CLUSTERING COEFFICIENT is that those users only had several trails of Github, such as account creation. They only follow very few people they already knew, so that the people in this small community tends to follow each other, which leads to high CLUSTERING COEFFICIENT. This amplification effect is very significant in the large network, for the reason that the denominator in the clustering coefficient equation increase quandraticly as the vertex degree increase. Therefore, it's more likely for vertex with very small vertex degree.

- Role #5 (Cluster #5) **INFLUENTIAL PROGRAMMERS** 1.7%
    - This group has one of the largest IN_DEGREE and PAGERANK, indicating this group of user has some interesting project that attracts many other influential programmers in the follow network.
    - This group has the 3rd highest PAGERANK and IN_DEGREE.

- Role #6 (Cluster #4) **HIGHLY PRODUCTIVE PROGRAMMERS** 0.26%
    - This group contains programmers that are highly productive. They tends to make very high amount of commit and pull requests. However, the high productivity doesn't mean the high quality of their product, for the reason that they are not followed by many other authoritive programmers, and their PAGERANK are relatively low.
    - This group's COMMIT_COUNT and PULL_COUNT are the highest amoung all groups.
    - This group's WATCH_COUNT is relatively low, indicating this group tends to focus on their's own project.

- Role #7 (Cluster #6) **HIGHLY INFLUENTIAL PROGRAMMERS** 0.077%
    - This group contains programmers that are well-known and are very influential in the community. They have the highest PAGERANK, and one of the highest IN_DEGREE.
    - This group has high, but not necessarily the highest PULL_COUNT and COMMIT_COUNT
    - The PAGERANK of this group is the highest while the IN_DEGREE is the 2nd highest. This indicates that this group is followed by many other programmers, and many of those followers are very influential and

authritive too. This leads to the very high authrity of this group, as indicated by the highest PAGERANK.

- Role #8 (Cluster #7) **SOCIABLE PROGRAMMERS** 0.01%
  - This group has 2nd largest amount of PULL_COUNT and COMMIT_COUNT. Moreover, their IN/OUT DEGREE, BETWEENNESS and CLOSENESS are the highest in the network. However, this group doesn't have the highest PAGERANK, indicating the follow connections of this group is not really authoritive.

# Conclusion

---

**Analysis Process**
In this project, a behavior-based clustering technique was performed to identify distinct user roles in the Github follow network. The theoritical foundation for the project is the paper "Role defining using behavior-based clustering in telecommunication network" published in 2011 [1].

Following the similar methodology in the original paper, the analysis went through 4 steps:

- Data Collection
  - User activity and follow relationship data on Github was extracted from GHTorrent data hosted on Goolge Bigquery. The desired information was queried with certain filtering condition.
  - Resulting tables including both the user activity count and follow network relationship.
- Graph Modeling
  - Follow network was generated with python-iGraph. The directed graph contains 1.5 million user nodes and 11 million edges representing follow relationship.
  - Various topological features were calculated, such as pagerank, betweenness, closeness and transitivity.
- Feature Exploration and Preprocessing

- Features table was merged from topological feature table and user activity table
- Each feature was carefully explored. The distribution was examined using q-q plot. Corresponding filtering strategies were determined
- Filtering conditions were applied to clean the data
- Featurs were scaled into standard normal distribution, for the reason that k-means clustering is sensitive to high variance and skewed features.

- Clustering
  - According to both the guidance paper and other paper comparing multiple clustering algorithms [8], bisecting k-means algroithm was used to cluster the users. According to the papers, bisecting k-means on average has better performance than k-means, and tends to give more stable splits with less skewed cluster sizes.
  - Elbow method was used to determine the optimal number of cluster to use, k was experimented in range [2,20], 8 was determined to be the optimal choice.
  - Users were divided in to 8 clusters, and their centroid were recorded.
- Role Identification and Explaination
  - Cluster centroids were scaled into range [1,10] to facilitate the visualization, for the reason that we only care about the relative magnitude of each feature among clusters.
  - Cluster centroides were visualized, and each role type was identified and explained combined with the unscaled average features of clusters.

---

**Analysis Result**

The clustering analysis on the user behavior features indicating that the Github users could be divided into 8 groups. It's interesting to notice that the alghough the number of users been analyzed is as large as 1 million, they can still be well explained by a relatively small number of 8 groups. Those eight groups are: "Inactive Users", "Normal Programmers", "Active Learners", "Mutual Following Inactive Users", "Influential Programmers", "Highly Productive Programmers", "Highly Influential Programmers" and "Socialble Programmers". They each accounts for 61%, 18.5%, 11.5%, 7%, 1.7%, 0.26%, 0.077% and 0.01% of the total population been analyzed. Both the role types and the distribution are intuitively understandable. The role types are analogously comparable with the user types in the real world usage, and the distribution follows an long tail pattern.

The features been used contains both aggregated features and topological features calculated from the follow network. After looking at the group centroids data, as well as the histogram plot of the group centroids, it's found both the aggregated features and topological features played important roles in separating distinct clusters. The aggregated features explain the individual user behaviors, while the topological metrics give insights of user's influence in the whole Github community.

When taking a closer look at the user roles and their proportion. The following points can be found:

- The "Inactive User" role type accounts for majority of the users. They have very less activity counts. This is in consistent with the general trend described in the paper "Promises and Perils of Mining Github Data". Those users might only register Github for an trial after hearing others talk about it.
- In the rest of the users. The normal programmers account for the largest portion. Those users have normal activity count and topological metrics.
- The group "Active Learner" is worth noticing, as this group is identified as users with less programming activity, such as pull request, and commit count. But they are actively watching at trending repositores and popular programmers. They are using Github as an learning platform instead of an colaborative coding platform. This behavior is reasonable and can be found in many real life users too.
- There are 2 types of inactive users been identified. One group accounts for 61% and another accounts for 1.7%. The difference is that the smaller portion inactive users have super high clustering coefficient. This interesting fact can be explained with the definition of the clusterting coefficient. Clustering coefficient implies the probability/percentage for the neighbours of certain user are also connected with each other. Therefore. the larger the node degree is, the less likely it will have high clustering coefficient. However, the inactive users tends to have very less activities and follow/followed count, they tend to form strongly connected componens in a network graph, thus they are likely to have super high cluster coefficient.
- By analyzing the other user roles in the network, including "Influential Programmers", "Highly Influential Programmers", "Socialble Programmers" and "Highly Productive Programmers". The following facts can be concluded:
  - High productivity, large number of code submittion will not necessarily make the programmer influential. However, the programmer could become really influential and followed by lots of other authoritive programmers while only have moderate amount of commits. The potential reason for this is that the

high quality work is easier to be recognized, and will be spread accross the programming community. While the repetitive programming work without any distinguishable contribution or innovation won't make too much influence.
  - The "Influential Programmer" and "Highly Influential Programmer" are clearly distinguished as 2 groups. The reason could be that the former group only have recognizable or interesting projects in certain field, thus the influence is well known, but not to the level that is known to all other groups. While the highly influential users might have project that is inested by vast amount of other groups.
  - There is a small group of "Sociable Programmer" that have highest in/out degree and betweeness/closeness centralities, but lower pagerank. This could be indication that these users are actively deliver information among the network, but not aiming at being the most authoritive users.

---

In conclusion, the behavior-based clustering on Github user data can effectively separate users in to small amount of clusters with distinct characteristics. The cluster results matches the expectation while giving some unexpected insights to the user roles in the Github follow network. The analysis process and result could be used to better understand the user structure in the Github, and to design customized strategy or new products for specific user group.

---

# Possible Improvement and Future Work

- Granularity:
  - The granularity of the data in this analysis is very coarse, the aggregated value for the entire lifecycle of the Github was used. Although it make sense to analyze the overall user behavior, the conclusion been made based on this is too general. The possible improvement can be performing an time sensitive analysis that only consider user roles for certain period. Then the role change pattern in consecutive periods can be furthur analyzed.
- Data filtering:
  - The data filtering in the current version is only based on the commit count, more refined definition can be set to filter the inactive users by consider their recent activities.
- Feature selection:

- Currently there are 3 types of aggregated features been used, including watch count, commit count and pull request count. However, there are much more aggregated features on Github that might also releveant to the analysis. In the future iterations, effort can be put to carry out more refined feature engineering. For example, after collecting the possible relevant aggregated features, the step-wise feature selection can be used to add/delete features one by one. Then by observing the clustering quality, the best features can determined for the role identification.

# Bibliograph:

***[1] T. Zhu, B. Wang, B. Wu, and C. Zhu, "Role defining using behavior- based clustering in telecommunication network," Expert Systems with Applications, vol. 38, no. 4, pp. 3902–3908, 2011.***

[2] M. Maia, J. Almeida, and V. Almeida, "Identifying user behavior in online social networks," in Proceedings of the 1st workshop on Social network systems. ACM, 2008, pp. 1–6.

[3] A. J. Lee, F.-C. Yang, H.-C. Tsai, and Y.-Y. Lai, "Discovering content- based behavioral roles in social networks," Decision Support Systems, vol. 59, pp. 250–261, 2014.

[4] Y. Yu, G. Yin, H. Wang, and T. Wang, "Exploring the patterns of social behavior in github," in Proceedings of the 1st international workshop on crowd-based software development methods and technologies.ACM,2014, pp. 31–36.

[5] Lima, Antonio, Luca Rossi, and Mirco Musolesi. "Coding Together at Scale: GitHub as a Collaborative Social Network." ICWSM. 2014.

[6] E. Kalliamvakou, G. Gousios, K. Blincoe, L. Singer, D. M. German, and D. Damian, "The promises and perils of mining github," in Proceedings of the 11th working conference on mining software repositories. ACM,2014, pp. 92–101.

[7] Arthur, David, and Sergei Vassilvitskii. "k-means++: The advantages of careful seeding." Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms. Society for Industrial and Applied Mathematics, 2007.

[8] Steinbach, Michael, George Karypis, and Vipin Kumar. "A comparison of document clustering techniques." KDD workshop on text mining. Vol. 400. No. 1. 2000.

[9] Al Hasan, Mohammad, et al. "Link prediction using supervised learning." SDM06: workshop on link analysis, counter-terrorism and security. 2006.