ABSTRACT

The concept of big data has been around for years; most organizations now understand that if they capture all the data that streams into their businesses, they can apply analytics and get significant value from it. But even in the 1950s, decades before anyone uttered the term “big data,” businesses were using basic analytics (essentially numbers in a spreadsheet that were manually examined) to uncover insights and trends.

The new benefits that big data analytics brings to the table, however, are speed and efficiency. Whereas a few years ago a business would have gathered information, run analytics and unearthed information that could be used for future decisions, today that business can identify insights for immediate decisions. The ability to work faster – and stay agile – gives organizations a competitive edge they didn’t have before.

Through our project we intend to carry out analysis on a preferably large dataset. So we have chosen the dataset obtained from several Facebook users. By carrying out certain operations, we intend to harness their data and use it to identify new opportunities.

TABLE OF CONTENTS

Chapter 1: INTRODUCTION

A Social network is defined as a network of relationships or interactions, where the nodes consist of people or actor, and the edges or arcs consist of the relationships or interactions between these actors.

Social networks and the techniques to analyse them existed since decades. There can be several type of social networks like email network, telephone network, collaboration network.

But recently online social networks like Facebook, Twitter, LinkedIn, MySpace etc have been developed which gained popularity within very short amount of time and gathered large number of users.

Facebook is said to have more than 500 million users in 2010. The field of social networks and their analysis has evolved from graph theory, statistics and sociology and it is used in several other fields like information science, business application, communication, economy etc.

Analysing a social network is similar to the analysis of a graph because social networks form the topology of a graph. Graph analysis tools have been there for decades. But they are not designed for analysing a social network graph which has complex properties. An online social network graph may be very large. It may contain millions of nodes and edges. Social networks are dynamic i.e. there is continuous evolution and expansion. A node in social network usually has several attributes. There are small and large communities within the social graph. Old graph analysis tools are not designed to manage such large and complex social network graph.

Facebook is a preferred social network by marketers, not only because of the sheer number of users represented but also because of its incredibly insightful analytics suite. It's important to be able to analyze customers and their behavior on a micro level due to Facebook's ever-changing algorithm, and the implications for our content and business. If we refuse to adapt our approach based on these insights, we're doomed to obscurity on the news feed.

A deep Facebook data analysis shouldn't be a one and done situation. Ideally, we'll be auditing our efforts every few months or so at most. This will help us predict the likings, and the general summary of many users as a whole.

Chapter 2: RESEARCH METHODOLOGY

Apache Spark:

Apache Spark is a lightning fast cluster computing system. It provides the set of high-level API namely Java, Scala, Python, and R for application development. Apache Spark is a tool for speedily executing Spark Applications. Spark utilizes Hadoop in two different ways – one is for Storage and second is for Process handling. Just because Spark has its own Cluster Management, so it utilizes Hadoop for Storage objective.

Spark is intended to cover an extensive variety of remaining loads, for example, cluster applications, iterative calculations, intuitive questions, and streaming. Aside from supporting all these remaining tasks at hand in a particular framework, it decreases the administration weight of keeping up isolated apparatuses.

Spark is one of Hadoop's sub venture created in 2009 in UC Berkeley's AMPLab by Matei Zaharia. It was Open Sourced in 2010 under a BSD license. It was given to Apache programming establishment in 2013, and now Apache Spark has turned into the best level Apache venture from Feb-2014. And now the results are pretty booming.

Market rules and big agencies already tend to use Spark for their solutions. Flabbergast to know that the list includes - Netflix, Uber, Pinterest, Conviva, Yahoo, Alibaba, eBay, MyFitnessPal, OpenTable, TripAdvisor and much more. It can be said that Apache Spark Use Case stretch is spread right from Finance, Healthcare, Travel, e-commerce to Media & Entertainment industry.

Spark is not fit for a multi-user environment. Spark as of now is not capable of handling more users concurrency, maybe in future updates this issue will be overcome. Yet an alternate engine like Hive for handling large batch projects.

Spark is accessible, intense, powerful and proficient Big Data tool for handling different enormous information challenges. Apache Spark takes after an ace/slave engineering with two primary Daemons and a Cluster Manager –

- Master Daemon – (Master/Driver Process)
- Worker Daemon – (Slave Process)

Most advanced and popular product of Apache Community, Spark decreases the time complexity of the system. Fast Computations, increase Performance, structured and unstructured Data Streaming, Graph Analytics, richer Resource Scheduling capabilities ensures smooth, engaging and customer experience compatible with the system.

MapReduce:

MapReduce is a software framework and programming model used for processing huge amounts of data. MapReduce program work in two phases, namely, Map and Reduce. Map tasks deal with splitting and mapping of data while Reduce tasks shuffle and reduce the data.

Hadoop is capable of running MapReduce programs written in various languages: Java, Ruby, Python, and C++. MapReduce programs are parallel in nature, thus are very useful for performing large-scale data analysis using multiple machines in the cluster.

The input to each phase is key-value pairs. In addition, every programmer needs to specify two functions: Map function and Reduce function.

The main advantage of the MapReduce framework is its fault tolerance, where periodic reports from each node in the cluster are expected when work is completed.

A task is transferred from one node to another. If the master node notices that a node has been silent for a longer interval than expected, the main node performs the reassignment process to the frozen/delayed task.

The MapReduce framework is inspired by the "Map" and "Reduce" functions used in functional programming. Computational processing occurs on data stored in a file system or within a database, which takes a set of input key values and produces a set of output key values.

Each day, numerous MapReduce programs and MapReduce jobs are executed on Google's clusters. Programs are automatically parallelized and executed on a large cluster of commodity machines. The runtime system deals with partitioning the input data, scheduling the program's execution across a set of machines, machine failure handling and managing required inter-machine communication. Programmers without any experience with parallel and distributed systems can easily use the resources of a large distributed system.

MapReduce is used in distributed grep, distributed sort, Web link-graph reversal, Web access log stats, document clustering, machine learning and statistical machine translation.

Apache Drill:

Apache Drill is an open source distributed SQL query engine integrated into the MapR Data Platform that delivers fast and secure self-service BI SQL analytics at scale. Drill's distributed shared-nothing architecture enables incremental scale out with low-cost hardware to meet the increasing demands of query response and user concurrency.

Drill was designed from the ground up to support high-performance analysis on the semi structured and rapidly evolving data coming from modern big data applications, while still providing the familiarity and ecosystem of industry standard ANSI SQL.

With Drill, users can query the raw data in situ. There's no need to load the data, create and maintain schemas, or transform the data before it can be processed. Instead, simply include the path to a Hadoop directory, MongoDB collection, or S3 bucket in the SQL query.
Drill supports a variety of SQL and NoSQL databases and file systems, including MapR Database, MapR XD Distributed File and Object Store, HDFS, HBase, MongoDB, Amazon S3, Azure Blob Storage, Google Cloud Storage, Swift, NAS, and local files. A single query can join data from multiple datastores.

Drill features a JSON data model that enables queries on complex/nested data as well as rapidly evolving structures commonly seen in modern applications and non-relational datastores.

Drill is the only columnar query engine that supports complex data. It features an in-memory shredded columnar representation for complex data, which allows Drill to achieve columnar speed with the flexibility of an internal JSON document model.

When Drill plans and executes a query, all operations are fully distributed only to available Drill nodes in the cluster. All Drill nodes are intelligent enough to accept, plan, and perform distributed query execution. When queries are submitted, an available node in the cluster is chosen as the foreman. The foreman then factors in the number of available nodes before it creates a distributed query plan. When nodes go down, they are automatically excluded from query planning and execution. Likewise, when they come back up, no manual intervention is required to reinstate them back into the cluster. This avoids manual administrative cluster corrections for specific lost nodes.

Apache Zeppelin:

A completely open web-based notebook that enables interactive data analytics Apache Zeppelin is a new and incubating multi-purposed web-based notebook which brings data ingestion, data exploration, visualization, sharing and collaboration features to Hadoop and Spark.

Interactive browser-based notebooks enable data engineers, data analysts and data scientists to be more productive by developing, organizing, executing, and sharing data code and visualizing results without referring to the command line or needing the cluster details. Notebooks allow these users not only allow to execute but to interactively work with long workflows. There are a number of notebooks available with Spark. iPython remains a mature choice and great example of a data science notebook. The Hortonworks Gallery provides an Ambari stack definition to help our customers quickly set up iPython on their Hadoop clusters.

Apache Zeppelin is a new and upcoming web-based notebook which brings data exploration, visualization, sharing and collaboration features to Spark. It support Python, but also a growing list of programming languages such as Scala, Hive, SparkSQL, shell and markdown. The various languages are supported via Zeppelin language interpreters.

Data discovery, exploration, reporting and visualization are key components of the data science workflow. Zeppelin provides a "Modern Data Science Studio" that supports Spark and Hive out of the box. Actually, Zeppelin supports multiple language backends which has support for a growing ecosystem of data sources. Zeppelin's notebooks provides interactive snippet-at-time experience to data scientist.

Some basic charts are already included in Apache Zeppelin. Visualizations are not limited to SparkSQL query, any output from any language backend can be recognized and visualized.

Apache Zeppelin aggregates values and displays them in pivot chart with simple drag and drop. You can easily create chart with multiple aggregated values including sum, count, average, min, max.

Apache Zeppelin can dynamically create some input forms in your notebook.

Even with notebooks the data wrangling process remains challenging. Often data scientists struggle with feature engineering, algorithm selection, tuning, sharing their work with others and deploying their work into production.

Chapter 3: IMPLEMENTATION AND OBSERVATIONS

Data Source: https://www.kaggle.com/sheenabatra/facebook-data

1. Sanity Check(Using Spark 2):

Code:

```
from pyspark.sql import SparkSession
from pyspark.sql import Row
from pyspark.sql import functions

def parseInput(line):
    fields = line.split(',')
    return Row(value = str(fields[i]))

if __name__ == "__main__":
    # Create a SparkSession (the config bit is only for Windows!)
    spark = SparkSession.builder.appName("SanityCheck").getOrCreate()

    # Get the raw data
    lines = spark.sparkContext.textFile("hdfs:///tmp/facebook_data/pseudo_facebook.csv")


a=["userid","age","dob_day","dob_year","dob_month","gender","tenure","friend_count","frien
dships_initiated","likes","likes_received","mobile_likes","mobile_likes_received","www_likes",$
    for i in range(15):
        # Convert it to a RDD of Row objects with (value)
        x = lines.map(parseInput)
        # Convert that to a DataFrame
        xDF = spark.createDataFrame(x)

        # Compute count of Null Values
        counts = xDF.filter(xDF["value"]=="NA").count()

        # Print them out
        print ("%s : %d"%(a[i],counts))

    # Stop the session
    spark.stop()
```

Command:

export SPARK_MAJOR_VERSION=2

spark-submit SanityCheck.py

Output:

Observation: Gender has null values, we should not delete these as users might have kept it blank.

2: Facebook popularity based on ages(Using MaRreduce (Python language))

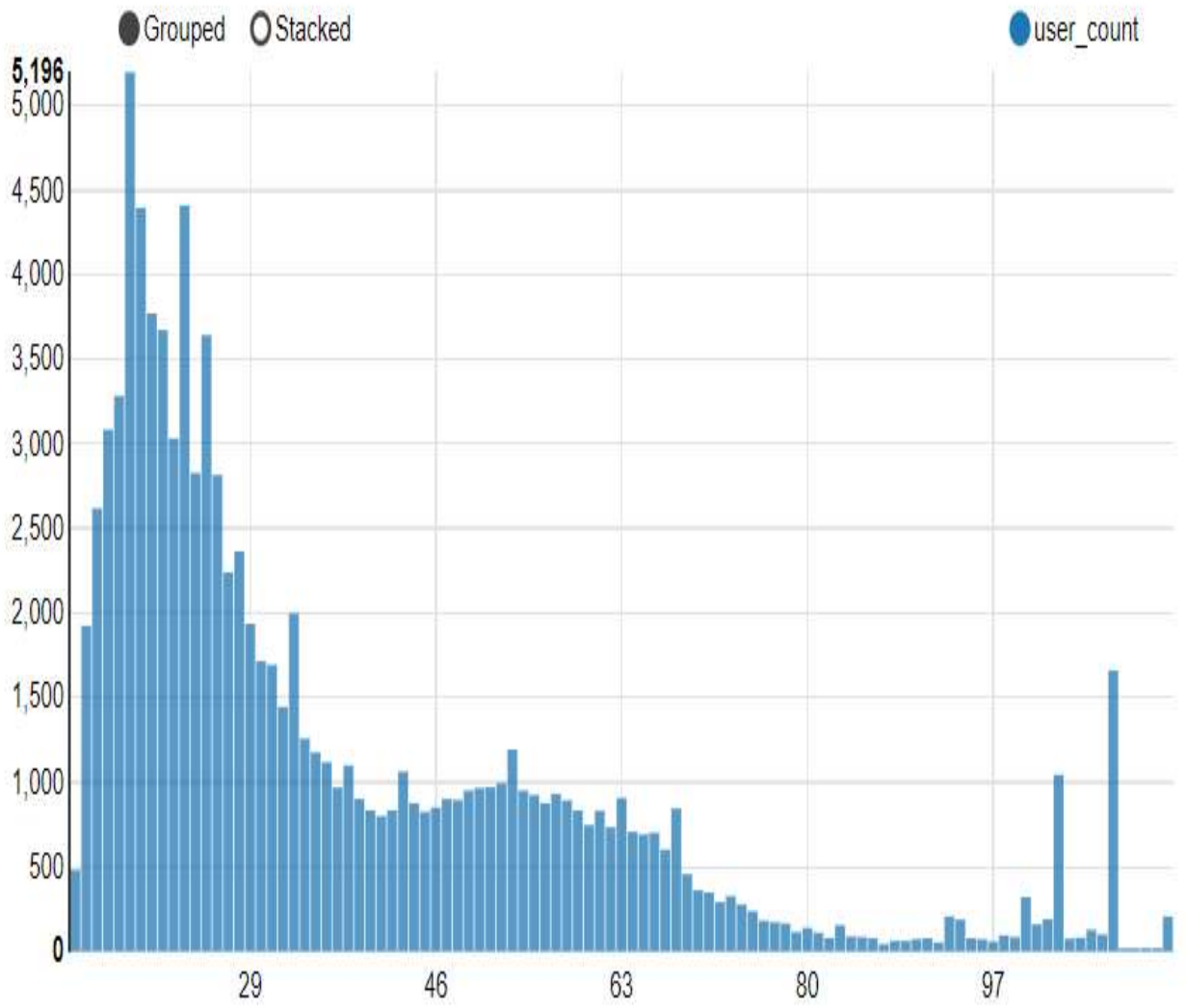
Code:

```
from mrjob.job import MRJob
from mrjob.step import MRStep

class WhatAgeUsesFacebook(MRJob):
   def steps(self):
      return [
         MRStep(mapper=self.mapper_get_ages,
               reducer=self.reducer_count_ages),
         MRStep(reducer=self.reducer_sorted_output)
      ]

   def mapper_get_ages(self, _, line):
      (userid, age, dob_day, dob_year, dob_month, gender, tenure, friend_count,
friendships_initiated, likes, likes_received, mobile_likes, mobile_likes_received, www_likes,
www_likes_receved) = line.split(',')
      yield age, 1

   def reducer_count_ages(self, age, ones):
      yield str(sum(ones)).zfill(5), age
```

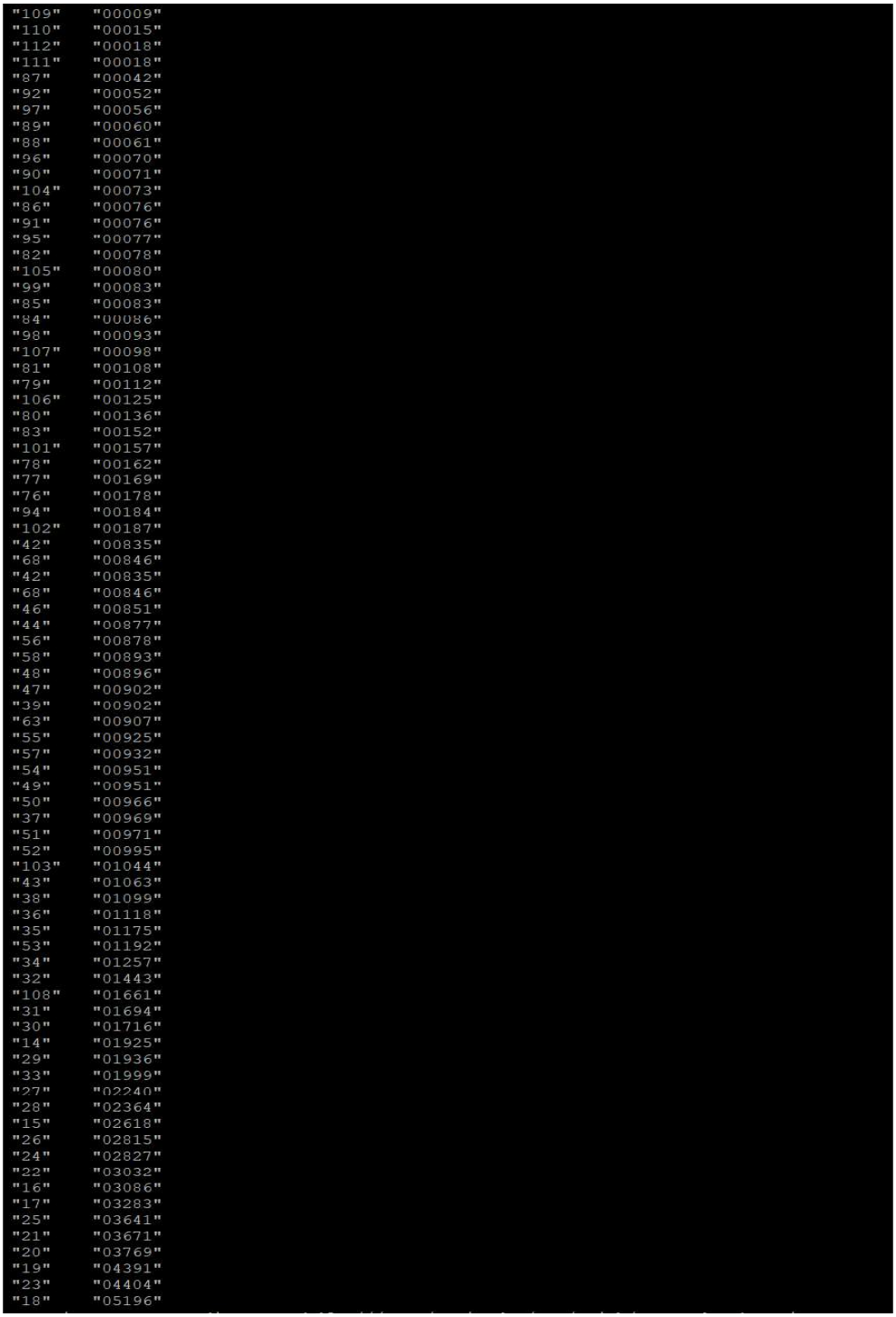
```
  def reducer_sorted_output(self, count, ages):
    for age in ages:
      yield age, count

if __name__ == '__main__':
  WhatAgeUsesFacebook.run()
```

Command: python map_reduce1.py -r hadoop --hadoop-streaming-jar /usr/hdp/current/hadoop-mapreduce-client/hadoop-streaming.jar hdfs:///tmp/facebook_data/pseudo_facebook.csv

Age wise distribution of users:



Output: (Age,Count)



```
"109"   "00009"
"110"   "00015"
"112"   "00018"
"111"   "00018"
"87"    "00042"
"92"    "00052"
"97"    "00056"
"89"    "00060"
"88"    "00061"
"96"    "00070"
"90"    "00071"
"104"   "00073"
"86"    "00076"
"91"    "00076"
"95"    "00077"
"82"    "00078"
"105"   "00080"
"99"    "00083"
"85"    "00083"
"84"    "00086"
"98"    "00093"
"107"   "00098"
"81"    "00108"
"79"    "00112"
"106"   "00125"
"80"    "00136"
"83"    "00152"
"101"   "00157"
"78"    "00162"
"77"    "00169"
"76"    "00178"
"94"    "00184"
"102"   "00187"
"42"    "00835"
"68"    "00846"
"42"    "00835"
"68"    "00846"
"46"    "00851"
"44"    "00877"
"56"    "00878"
"58"    "00893"
"48"    "00896"
"47"    "00902"
"39"    "00902"
"63"    "00907"
"55"    "00925"
"57"    "00932"
"54"    "00951"
"49"    "00951"
"50"    "00966"
"37"    "00969"
"51"    "00971"
"52"    "00995"
"103"   "01044"
"43"    "01063"
"38"    "01099"
"36"    "01118"
"35"    "01175"
"53"    "01192"
"34"    "01257"
"32"    "01443"
"108"   "01661"
"31"    "01694"
"30"    "01716"
"14"    "01925"
"29"    "01936"
"33"    "01999"
"27"    "02240"
"28"    "02364"
"15"    "02618"
"26"    "02815"
"24"    "02827"
"22"    "03032"
"16"    "03086"
"17"    "03283"
"25"    "03641"
"21"    "03671"
"20"    "03769"
"19"    "04391"
"23"    "04404"
"18"    "05196"
```

Observation : Facebook is most popular between age groups 15 and 28.

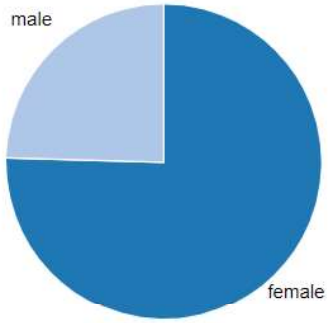
3. Likes Given (Using Drill)

CMD: apache-drill-1.12.0/bin/drillbit.sh start -Ddrill.exec.http.port=8765

Query 1: SELECT gender,avg(likes) AS AVG_Likes_Given
FROM hive.facebook_db.facebook
GROUP BY gender
ORDER BY AVG_Likes_Given DESC

Output: gender vs likes given :

gender	AVG_Likes_Given
female	260.0513240920157
NA	138.50857142857143
male	84.6778946290163



Query 2: SELECT userid, gender, likes AS Total_Likes_Given
FROM hive.facebook_db.facebook
ORDER BY Total_likes_Given DESC LIMIT 10

Output : Top 10 users with most likes received

userid	gender	Total_Likes_Given
1684195	male	25111
1656477	male	21652
1489463	female	16732
1429178	female	16583
1267229	female	14799
1783264	male	14355
1002588	female	14050
1412849	female	14039
1878566	female	13692
2104503	female	13622

Analysis Result: Females give more likes then men

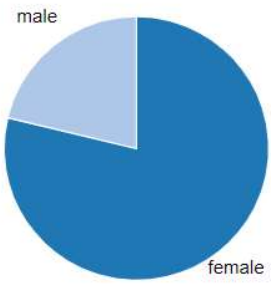
4. Likes Received (Using Drill)

CMD: apache-drill-1.12.0/bin/drillbit.sh start -Ddrill.exec.http.port=8765

Query 1: SELECT gender,avg(likes_received) AS AVG_Likes_Received
FROM hive.facebook_db.facebook
GROUP BY gender
ORDER BY AVG_Likes_Received DESC

Output: gender vs total likes received :

gender	AVG_Likes_Received
female	251.4354349878273
NA	157.38285714285715
male	67.91154778570697



Query 2: SELECT userid, gender, likes_received AS Total_Likes_Received
FROM hive.facebook_db.facebook
ORDER BY likes_received DESC
LIMIT 10

Output : Top 10 users with most likes received

userid	gender	Total_Likes_Received
1674584	female	261197
1441676	female	178166
1715925	female	152014
2063006	female	106025
1053087	male	82623
1432020	male	53534
2042824	male	52964
1559908	female	45633
1781243	female	42449
1015907	male	39536

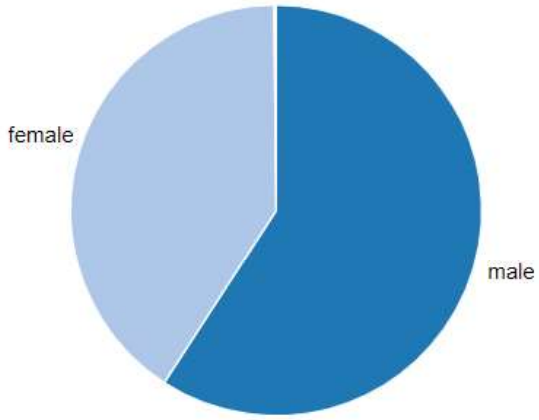
Analysis Result: Females receive more likes then men

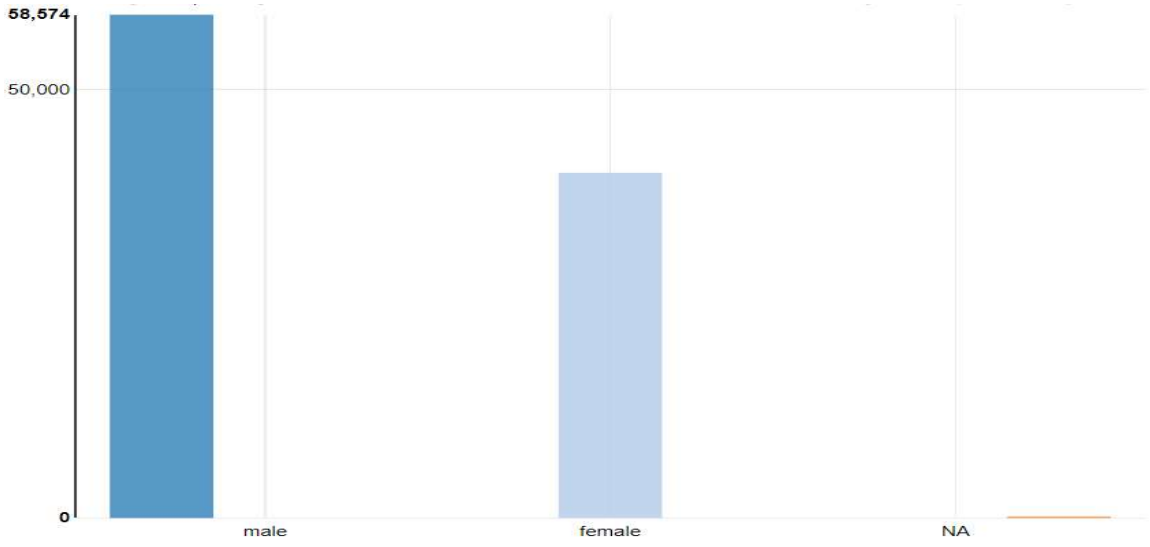
5. Gender Count (Using Zeppelin(Spark code)):

```
val x = fbDF.groupBy("gender").count().orderBy(desc("count")).cache()

x.show()
```

Output:

```
+---------+--------+
|gender|count|
+---------+--------+
| male  |58574|
|female|40254|
| NA     | 175   |
+---------+--------+
```





Analysis : There are more male users than female .

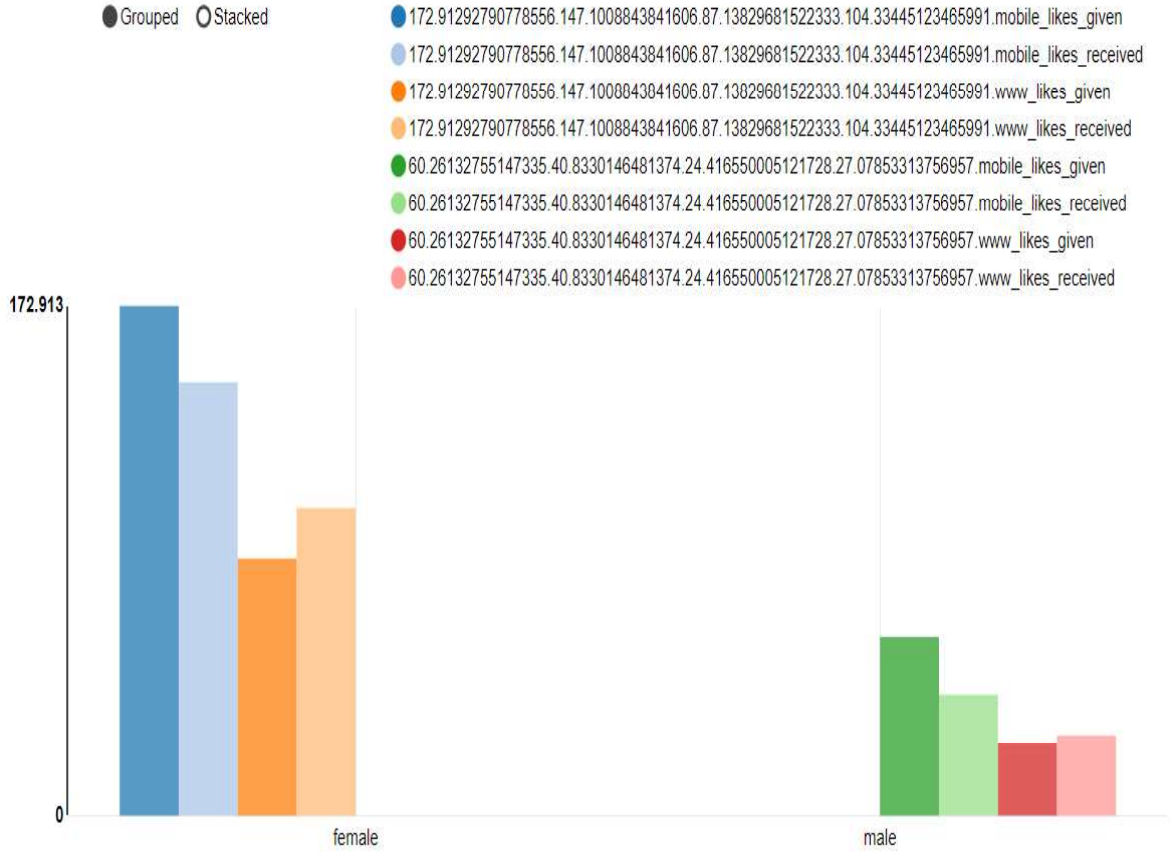
6. Likes Split Up (Using Zeppelin-SQL code)

Query 1:

```
SELECT gender,avg(mobile_likes) AS mobile_likes_given,
avg(mobile_likes_received) AS mobile_likes_received, avg(www_likes) AS
www_likes_given, avg(www_likes_received) AS www_likes_received
FROM fb
WHERE gender <> "NA"
GROUP BY gender
```

Output:

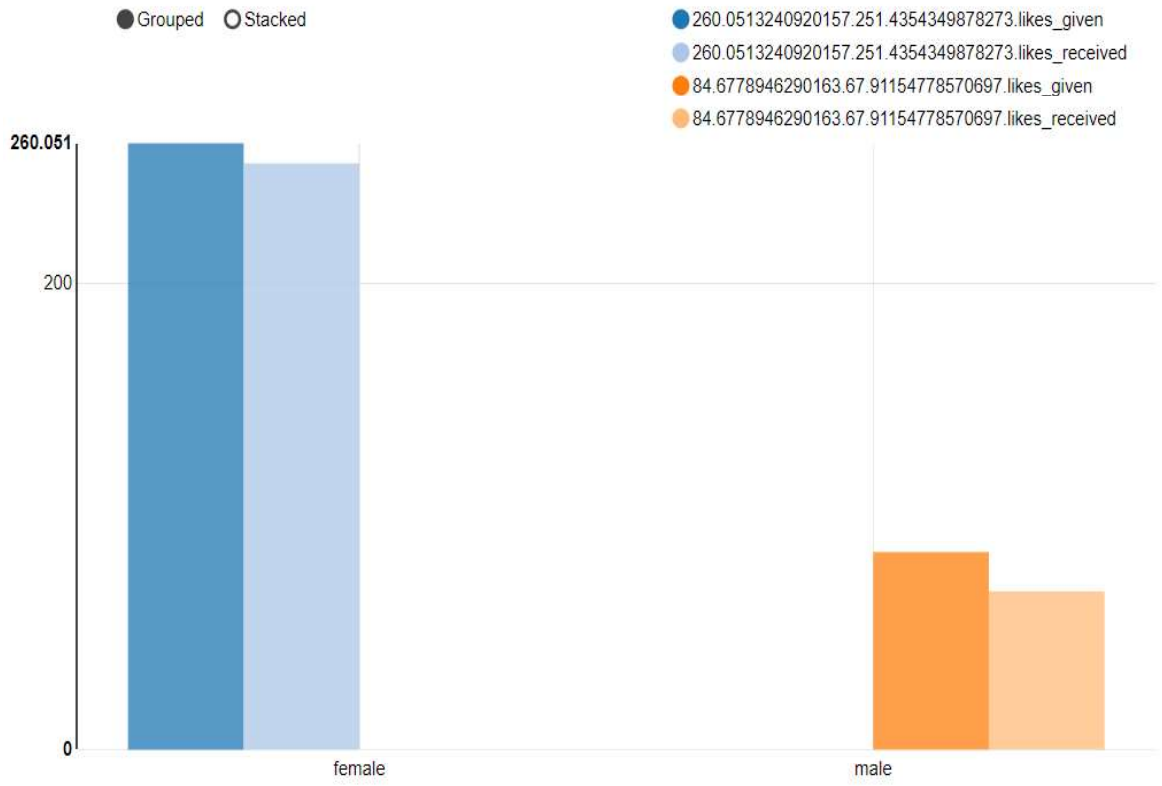
gender	mobile_likes_given	mobile_likes_received	www_likes_given	www_likes_received
female	172.91293	147.10088	87.1383	104.33445
male	60.26133	40.83301	24.41655	27.07853



Query2:

```
%sql
SELECT gender,avg(likes) AS likes_given ,avg(likes_received) AS likes_received
FROM fb
WHERE gender <> "NA"
GROUP BY gender
```

Output:(Likes vs Likes Recived by gender)



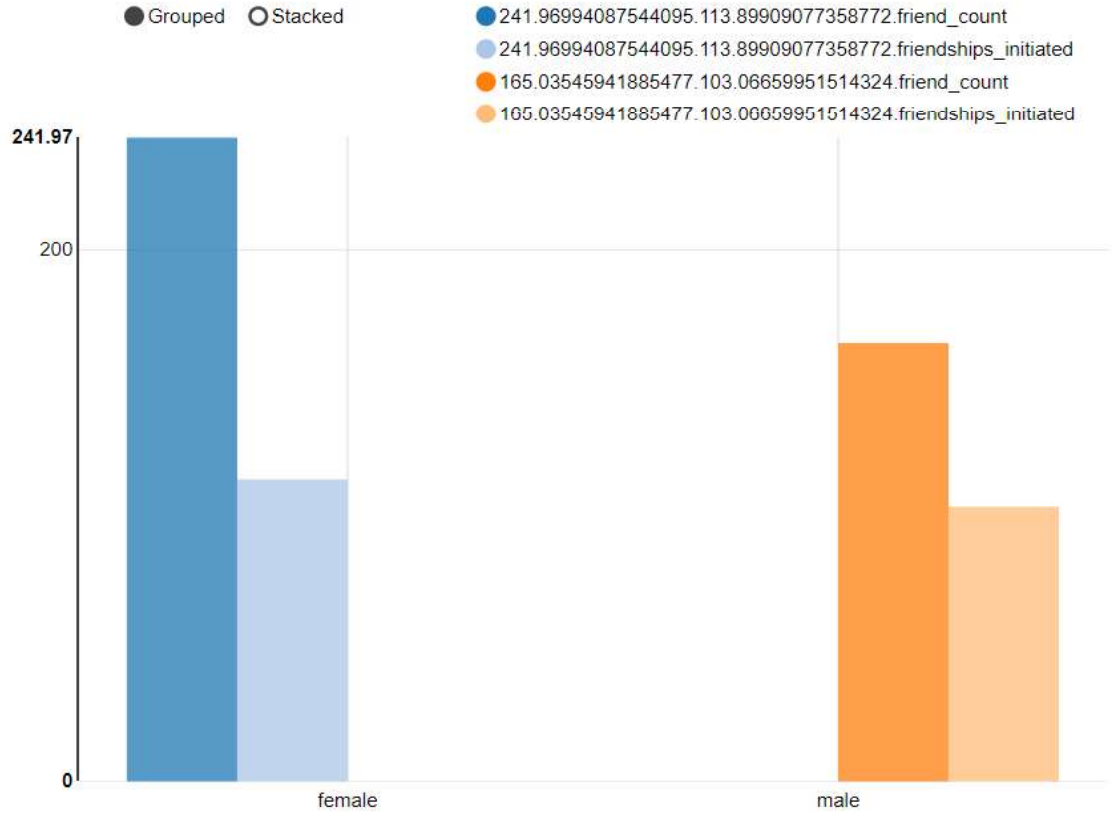
Analysis: Interesting obsservation for gender specific interaction with facebook: women like as well as are liked a lot more than men (nearly 2.5 as much).

7. Friends Counts & Friendships initiated(Using Zepplin -SQL code)

Query :

```
SELECT gender,avg(friend_count) AS friend_count ,avg(friendships_initiated) AS
friendships_initiated
FROM fb
WHERE gender <> "NA"
GROUP BY gender
```

Output: (Friends Count vs Friendships Initiated)

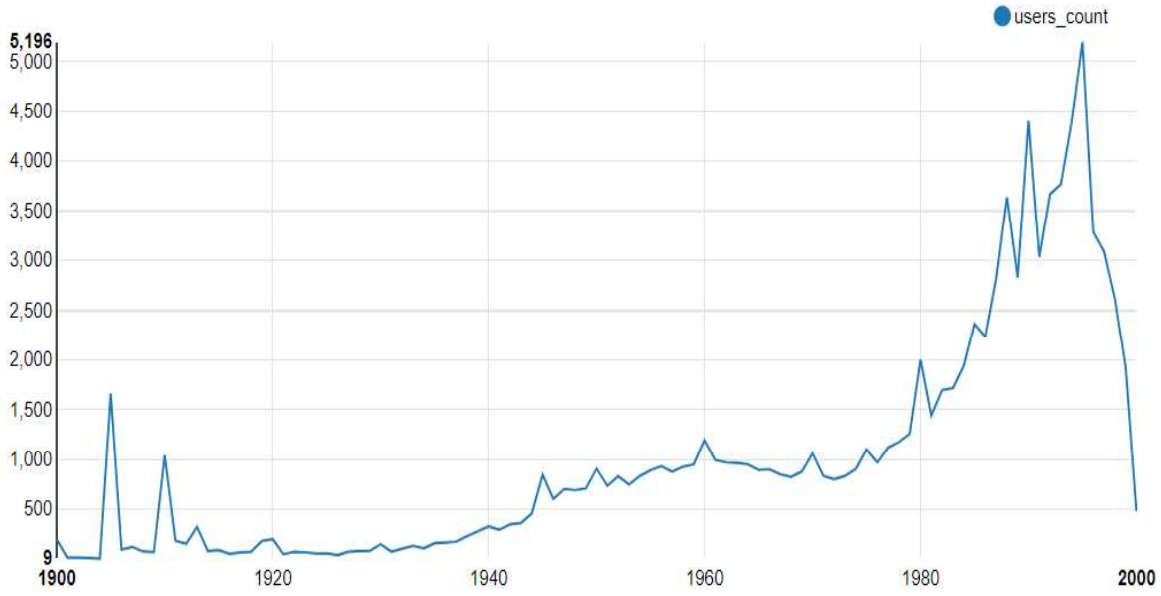


Analysis: Women have more friends than men on facebook, the friendships initiated in proportion to friend count are more in case of men than women.

8. Users w.r.t birth year(Using Zepplin -SQL code)

Query: SELECT dob_year,count(userid) AS users_count
FROM fb
GROUP BY dob_year

Output:



Analysis:
We see bumps between 1940 to 1980. After 1980 the no. users rocket. Since the data is till 2000 (we see miniscule value in 2000)

Chapter 4: RESULT AND CONCLUSION

Key Findings from the data :

- Gender Interaction :
    - Women interact more with fb then men
    - Women receive & give more likes than men on average
    - Women initiate less friendships than men compared proportionally to friend count
- Likes Split Up :
    - Shows inclination towards mobile apps
    - More prominence of likes from mobile compared to site though few users still interact with sites
    - There can be a gradual shift from mobile to site in years to come seeing the trend
- User Counts :
    - Bump between 1940 to 1980
    - Age Distribution shows peak between 15-28 years, then small peak between 45-55 years. It seems like ages of parents and kids of a generation.