Sreyashi Saha

Matriculation Number : 1747271

Exercise 1 Part A

In [1]:

```
import pyspark
from pyspark.sql import SparkSession
from pyspark import SparkContext
from pyspark.sql import SQLContext
```

In [2]:

```
list_a = ["spark", "rdd", "python", "context", "create", "class"]
list_b = ["operation", "apache", "scala", "lambda","parallel","partition"
```

Create two RDD objects of a, b and do the following tasks. Words should be remained in the results of join operations.

In [4]:

```
sc =SparkContext()
a_rdd = sc.parallelize(list_a)
```

```
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use set
LogLevel(newLevel).
22/07/03 10:33:28 WARN NativeCodeLoader: Unable to load native-hadoop lib
rary for your platform... using builtin-java classes where applicable
22/07/03 10:33:30 WARN Utils: Service 'SparkUI' could not bind on port 40
40. Attempting port 4041.
22/07/03 10:33:30 WARN Utils: Service 'SparkUI' could not bind on port 40
41. Attempting port 4042.
22/07/03 10:33:30 WARN Utils: Service 'SparkUI' could not bind on port 40
42. Attempting port 4043.
22/07/03 10:33:30 WARN Utils: Service 'SparkUI' could not bind on port 40
43. Attempting port 4044.
22/07/03 10:33:30 WARN Utils: Service 'SparkUI' could not bind on port 40
44. Attempting port 4045.
22/07/03 10:33:30 WARN Utils: Service 'SparkUI' could not bind on port 40
45. Attempting port 4046.
22/07/03 10:33:30 WARN Utils: Service 'SparkUI' could not bind on port 40
46. Attempting port 4047.
22/07/03 10:33:30 WARN Utils: Service 'SparkUI' could not bind on port 40
47. Attempting port 4048.
```

In [5]:

```
b_rdd = sc.parallelize(list_b)
```

In [7]:

```
a_rdd.collect()
```

```
Out[7]: ['spark', 'rdd', 'python', 'context', 'create', 'class']
```

```
In [8]: b_rdd.collect()
```

```
Out[8]: ['operation', 'apache', 'scala', 'lambda', 'parallel', 'partition']
```

In order to perform the joins we need to have key value pairs. Therefore, we map each word from both the list and assign their values as 1 using the map function.

```
In [9]: # map_a = a_rdd.map(lambda x: (x, 1)).collect()
        map_a = a_rdd.map(lambda x: (x, 1))
```

```
In [10]: # map_b = b_rdd.map(lambda x: (x, 1)).collect()
         map_b = b_rdd.map(lambda x: (x, 1))
```

```
In [11]: map_a.collect()
```

```
Out[11]: [('spark', 1),
          ('rdd', 1),
          ('python', 1),
          ('context', 1),
          ('create', 1),
          ('class', 1)]
```

```
In [12]: map_b.collect()
```

```
Out[12]: [('operation', 1),
          ('apache', 1),
          ('scala', 1),
          ('lambda', 1),
          ('parallel', 1),
          ('partition', 1)]
```

We can see that all the words in both rdds now have values assigned as 1

1. Perform rightOuterJoin and fullOuterJoin operations between a and b. Briefly explain your solution. (1 point)

```
In [13]: join1 = map_a.rightOuterJoin(map_b)
         Right_Outer_Join = join1.map(lambda x: x[0])
```

```
In [14]: Right_Outer_Join.collect()
```

```
Out[14]: ['parallel', 'lambda', 'scala', 'operation', 'apache', 'partition']
```

A right outer join is a method of combining tables. The result includes unmatched rows from only the table that is specified after the RIGHT OUTER JOIN phrase. Here in our case we have map_b after the phrase, it returns all records from the right rdd (map_b), and the matching records from the left rdd (map_a). The result is 0 records from the left side, if there is no match.

In [18]:

```
join2 = map_a.fullOuterJoin(map_b)
Full_Outer_join = join2.map(lambda x: x[0]).collect()
```

In [16]:

```
Full_Outer_join.collect()
```

Out[16]:

```
['python',
 'spark',
 'context',
 'create',
 'parallel',
 'lambda',
 'class',
 'rdd',
 'scala',
 'operation',
 'apache',
 'partition']
```

The FULL OUTER JOIN keyword returns all records when there is a match in left rdd (map_a) or right rdd(map_b) table records.

In [19]:

```
new_list = sc.parallelize(Full_Outer_join)
all_words = new_list.map(lambda word: (word, 1)).collect()
all_words
```

Out[19]:

```
[('python', 1),
 ('spark', 1),
 ('context', 1),
 ('create', 1),
 ('parallel', 1),
 ('lambda', 1),
 ('class', 1),
 ('rdd', 1),
 ('scala', 1),
 ('operation', 1),
 ('apache', 1),
 ('partition', 1)]
```

Here I map all the words after performing Full_Outer_join and take them as key value pairs for further implementations.

1. Using map and reduce functions to count how many times the character "s" appears in all a and b. (1 point)

In [22]:
```
words = new_list.map(lambda x: (x, 1))
```

In the function below I check if the word received as input has the character 's' in it or not. If it does then I count the number of times I get 's' for that word and then I return the count.

In [23]:
```
def check_for_s(temp):
    c=0
    for idx in temp:
        if idx == 's':
                c+=1
    return c
```

Here I map the words and the number of 's' that the word contains

In [24]:
```
count_s_words = words.map(lambda x:(x[0], check_for_s(x[0])))
```

In [25]:
```
count_s_words.collect()
```

Out[25]:
```
[('python', 0),
 ('spark', 1),
 ('context', 0),
 ('create', 0),
 ('parallel', 0),
 ('lambda', 0),
 ('class', 2),
 ('rdd', 0),
 ('scala', 1),
 ('operation', 0),
 ('apache', 0),
 ('partition', 0)]
```

In [27]:
```
count = count_s_words.map(lambda a:a[1]).reduce(lambda a,b:a+b)
```

In [29]:
```
print(" The number of times s occurs in the list of words using reduce fu
```

```
 The number of times s occurs in the list of words using reduce funtion i
s   4
```

1. Using aggregate function to count how many times the character "s" appears in all a and b. (1 point)

In [30]:
```
count1 = words.aggregate(0,lambda a,x:a+check_for_s(x[0]), lambda a,b: a+
```

In [31]:
```
print(" The number of times s occurs in the list of words using aggregate
 The number of times s occurs in the list of words using aggregate funtio
n is  4
```

In [ ]:

Part b) Basic Operations on DataFrames (6 points)

Use dataset students.json (download from learnweb) for this exercise. First creating DataFrames from the dataset and do several tasks as follows:

In [2]:

```
import pyspark
from pyspark.sql import SparkSession
from pyspark import SparkContext
from pyspark.sql import SQLContext
```

In [3]:

```
# Read JSON file into dataframe
spark = SparkSession.builder.appName(
    'Read Json File into DataFrame').getOrCreate()
df = spark.read.json("students.json")
df.printSchema()
df.show()
```

```
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use set
LogLevel(newLevel).
22/07/03 10:58:48 WARN NativeCodeLoader: Unable to load native-hadoop lib
rary for your platform... using builtin-java classes where applicable
22/07/03 10:58:50 WARN Utils: Service 'SparkUI' could not bind on port 40
40. Attempting port 4041.
22/07/03 10:58:50 WARN Utils: Service 'SparkUI' could not bind on port 40
41. Attempting port 4042.
22/07/03 10:58:50 WARN Utils: Service 'SparkUI' could not bind on port 40
42. Attempting port 4043.
22/07/03 10:58:50 WARN Utils: Service 'SparkUI' could not bind on port 40
43. Attempting port 4044.
22/07/03 10:58:50 WARN Utils: Service 'SparkUI' could not bind on port 40
44. Attempting port 4045.
22/07/03 10:58:50 WARN Utils: Service 'SparkUI' could not bind on port 40
45. Attempting port 4046.
22/07/03 10:58:50 WARN Utils: Service 'SparkUI' could not bind on port 40
46. Attempting port 4047.
22/07/03 10:58:50 WARN Utils: Service 'SparkUI' could not bind on port 40
47. Attempting port 4048.
22/07/03 10:58:50 WARN Utils: Service 'SparkUI' could not bind on port 40
48. Attempting port 4049.
```

```
root
 |-- course: string (nullable = true)
 |-- dob: string (nullable = true)
 |-- first_name: string (nullable = true)
 |-- last_name: string (nullable = true)
 |-- points: long (nullable = true)
 |-- s_id: long (nullable = true)

+------------------+------------------+----------+---------+------+----+
|            course|               dob|first_name|last_name|points|s_id|
+------------------+------------------+----------+---------+------+----+
|Humanities and Art|  October 14, 1983|      Alan|      Joe|    10|   1|
|  Computer Science|September 26, 1980|    Martin|  Genberg|    17|   2|
|    Graphic Design|     June 12, 1982|     Athur|   Watson|    16|   3|
|    Graphic Design|     April 5, 1987|  Anabelle|  Sanberg|    12|   4|
|        Psychology|  November 1, 1978|      Kira| Schommer|    11|   5|
|          Business|  17 February 1981| Christian|   Kiriam|    10|   6|
|  Machine Learning|    1 January 1984|   Barbara|  Ballard|    14|   7|
|     Deep Learning|  January 13, 1978|      John|     null|    10|   8|
|  Machine Learning|  26 December 1989|    Marcus|   Carson|    15|   9|
|           Physics|  30 December 1987|     Marta|   Brooks|    11|  10|
|    Data Analytics|     June 12, 1975|     Holly| Schwartz|    12|  11|
|  Computer Science|      July 2, 1985|     April|    Black|  null|  12|
|  Computer Science|     July 22, 1980|     Irene|  Bradley|    13|  13|
|        Psychology|   7 February 1986|      Mark|    Weber|    12|  14|
|       Informatics|      May 18, 1987|     Rosie|   Norman|     9|  15|
|          Business|   August 10, 1984|    Martin|   Steele|     7|  16|
|  Machine Learning|  16 December 1990|     Colin| Martinez|     9|  17|
|    Data Analytics|              null|   Bridget|    Twain|     6|  18|
|          Business|      7 March 1980|   Darlene|    Mills|    19|  19|
|    Data Analytics|      June 2, 1985|   Zachary|     null|    10|  20|
+------------------+------------------+----------+---------+------+----+
```

1. Replace the null value(s) in column points by the mean of all points. (0.5 point)

In [5]:

```
#Replace mean for null on only points column
mean = df.agg({'points': 'mean'}).collect()
mean = mean[0][0]
print("The mean of the points column is ", mean)
new_df = df.na.fill(value=mean,subset=["points"])
```

```
The mean of the points column is  11.736842105263158
```

In [6]:

```
new_df.show() # we can see that the null values has been replaced by the
```

```
+------------------+------------------+----------+---------+------+----+
|            course|               dob|first_name|last_name|points|s_id|
+------------------+------------------+----------+---------+------+----+
|Humanities and Art|  October 14, 1983|      Alan|      Joe|    10|   1|
|  Computer Science|September 26, 1980|    Martin|  Genberg|    17|   2|
|    Graphic Design|     June 12, 1982|     Athur|   Watson|    16|   3|
|    Graphic Design|     April 5, 1987|  Anabelle|  Sanberg|    12|   4|
|        Psychology|  November 1, 1978|      Kira| Schommer|    11|   5|
|          Business|  17 February 1981| Christian|   Kiriam|    10|   6|
|  Machine Learning|    1 January 1984|   Barbara|  Ballard|    14|   7|
|     Deep Learning|  January 13, 1978|      John|     null|    10|   8|
|  Machine Learning|  26 December 1989|    Marcus|   Carson|    15|   9|
|           Physics|  30 December 1987|     Marta|   Brooks|    11|  10|
|    Data Analytics|     June 12, 1975|     Holly| Schwartz|    12|  11|
|  Computer Science|      July 2, 1985|     April|    Black|    11|  12|
|  Computer Science|     July 22, 1980|     Irene|  Bradley|    13|  13|
|        Psychology|   7 February 1986|      Mark|    Weber|    12|  14|
|       Informatics|      May 18, 1987|     Rosie|   Norman|     9|  15|
|          Business|   August 10, 1984|    Martin|   Steele|     7|  16|
|  Machine Learning|  16 December 1990|     Colin| Martinez|     9|  17|
|    Data Analytics|              null|   Bridget|    Twain|     6|  18|
|          Business|      7 March 1980|   Darlene|    Mills|    19|  19|
|    Data Analytics|      June 2, 1985|   Zachary|     null|    10|  20|
+------------------+------------------+----------+---------+------+----+
```

1. Replace the null value(s) in column dob and column last name by "unknown" and "--" respec- tively. (0.5 point)

In [8]:

```
df1 = new_df.na.fill(value="unknown",subset=["dob"])
df1.show()
```

```
+------------------+------------------+----------+---------+------+----+
|            course|               dob|first_name|last_name|points|s_id|
+------------------+------------------+----------+---------+------+----+
|Humanities and Art|  October 14, 1983|      Alan|      Joe|    10|   1|
|  Computer Science|September 26, 1980|    Martin|  Genberg|    17|   2|
|    Graphic Design|     June 12, 1982|     Athur|   Watson|    16|   3|
|    Graphic Design|     April 5, 1987|  Anabelle|  Sanberg|    12|   4|
|        Psychology|  November 1, 1978|      Kira| Schommer|    11|   5|
|          Business|  17 February 1981| Christian|   Kiriam|    10|   6|
|  Machine Learning|    1 January 1984|   Barbara|  Ballard|    14|   7|
|     Deep Learning|  January 13, 1978|      John|     null|    10|   8|
|  Machine Learning|  26 December 1989|    Marcus|   Carson|    15|   9|
|           Physics|  30 December 1987|     Marta|   Brooks|    11|  10|
|    Data Analytics|     June 12, 1975|     Holly| Schwartz|    12|  11|
|  Computer Science|      July 2, 1985|     April|    Black|    11|  12|
|  Computer Science|     July 22, 1980|     Irene|  Bradley|    13|  13|
|        Psychology|   7 February 1986|      Mark|    Weber|    12|  14|
|       Informatics|      May 18, 1987|     Rosie|   Norman|     9|  15|
|          Business|   August 10, 1984|    Martin|   Steele|     7|  16|
|  Machine Learning|  16 December 1990|     Colin| Martinez|     9|  17|
|    Data Analytics|           unknown|   Bridget|    Twain|     6|  18|
|          Business|      7 March 1980|   Darlene|    Mills|    19|  19|
|    Data Analytics|      June 2, 1985|   Zachary|     null|    10|  20|
+------------------+------------------+----------+---------+------+----+
```

In [9]:

```
df2 = df1.na.fill(value="--",subset=["last_name"])
df2.show()
```

```
+------------------+------------------+----------+---------+------+----+
|            course|               dob|first_name|last_name|points|s_id|
+------------------+------------------+----------+---------+------+----+
|Humanities and Art|  October 14, 1983|      Alan|      Joe|    10|   1|
|  Computer Science|September 26, 1980|    Martin|  Genberg|    17|   2|
|    Graphic Design|     June 12, 1982|     Athur|   Watson|    16|   3|
|    Graphic Design|     April 5, 1987|  Anabelle|  Sanberg|    12|   4|
|        Psychology|  November 1, 1978|      Kira| Schommer|    11|   5|
|          Business|  17 February 1981| Christian|   Kiriam|    10|   6|
|  Machine Learning|    1 January 1984|   Barbara|  Ballard|    14|   7|
|     Deep Learning|  January 13, 1978|      John|       --|    10|   8|
|  Machine Learning|  26 December 1989|    Marcus|   Carson|    15|   9|
|           Physics|  30 December 1987|     Marta|   Brooks|    11|  10|
|    Data Analytics|     June 12, 1975|     Holly| Schwartz|    12|  11|
|  Computer Science|      July 2, 1985|     April|    Black|    11|  12|
|  Computer Science|     July 22, 1980|     Irene|  Bradley|    13|  13|
|        Psychology|   7 February 1986|      Mark|    Weber|    12|  14|
|       Informatics|      May 18, 1987|     Rosie|   Norman|     9|  15|
|          Business|   August 10, 1984|    Martin|   Steele|     7|  16|
|  Machine Learning|  16 December 1990|     Colin| Martinez|     9|  17|
|    Data Analytics|           unknown|   Bridget|    Twain|     6|  18|
|          Business|      7 March 1980|   Darlene|    Mills|    19|  19|
|    Data Analytics|      June 2, 1985|   Zachary|       --|    10|  20|
+------------------+------------------+----------+---------+------+----+
```

1. In the dob column, there exist several formats of dates, e.g. October 14, 1983 and 26 December 1989. Let’s convert all the dates into DD-MM-YYYY format where DD, MM and YYYY are two digits for day, two digits for months and four digits for year respectively. (2 points)

In [17]:
```
from dateutil import parser
from datetime import datetime
import datetime
```

In this function below I convert the provided date format as "%m-%d-%Y" format and update the values in a new column name new_date. For those rows which have values as 'unknown' I returned "1" as their date of birth.

In [21]:
```
def date(input):
    if input == "unknown":
        return "1"
    else:
        obj=str(parser.parse(input))
        d = obj.split(" ")
        d = datetime.datetime.strptime(d[0], '%Y-%m-%d').strftime('%m-%d-
        return d
```

In [22]:
```
from pyspark.sql import functions as F
from pyspark.sql.functions import to_date
from pyspark.sql.types import StringType
```

In [23]:
```
data = df2.withColumn("new_date",  F.udf(date, StringType())(F.col("dob"))
data.show()
```

```
+------------------+------------------+----------+---------+------+----+-
---------+
|            course|               dob|first_name|last_name|points|s_id|
new_date|
+------------------+------------------+----------+---------+------+----+-
---------+
|Humanities and Art|  October 14, 1983|      Alan|      Joe|    10|   1|1
0-14-1983|
|  Computer Science|September 26, 1980|    Martin|  Genberg|    17|   2|0
9-26-1980|
|    Graphic Design|     June 12, 1982|     Athur|   Watson|    16|   3|0
6-12-1982|
|    Graphic Design|     April 5, 1987|  Anabelle|  Sanberg|    12|   4|0
4-05-1987|
|        Psychology|  November 1, 1978|      Kira| Schommer|    11|   5|1
1-01-1978|
|          Business|  17 February 1981| Christian|   Kiriam|    10|   6|0
2-17-1981|
|  Machine Learning|    1 January 1984|   Barbara|  Ballard|    14|   7|0
1-01-1984|
|     Deep Learning|  January 13, 1978|      John|       --|    10|   8|0
1-13-1978|
|  Machine Learning|  26 December 1989|    Marcus|   Carson|    15|   9|1
2-26-1989|
|           Physics|  30 December 1987|     Marta|   Brooks|    11|  10|1
2-30-1987|
|    Data Analytics|     June 12, 1975|     Holly| Schwartz|    12|  11|0
6-12-1975|
|  Computer Science|      July 2, 1985|     April|    Black|    11|  12|0
7-02-1985|
|  Computer Science|     July 22, 1980|     Irene|  Bradley|    13|  13|0
7-22-1980|
|        Psychology|   7 February 1986|      Mark|    Weber|    12|  14|0
2-07-1986|
|       Informatics|      May 18, 1987|     Rosie|   Norman|     9|  15|0
5-18-1987|
|          Business|   August 10, 1984|    Martin|   Steele|     7|  16|0
8-10-1984|
|  Machine Learning|  16 December 1990|     Colin| Martinez|     9|  17|1
2-16-1990|
|    Data Analytics|           unknown|   Bridget|    Twain|     6|  18|
1|
|          Business|      7 March 1980|   Darlene|    Mills|    19|  19|0
3-07-1980|
|    Data Analytics|      June 2, 1985|   Zachary|       --|    10|  20|0
6-02-1985|
+------------------+------------------+----------+---------+------+----+-
---------+
```

1. Insert a new column age and calculate the current age of all students. (1 point)

```
In [27]: from dateutil import relativedelta
```

In this function I calculate the current age of each student and store them in a new column Current_age. For those rows which have birthdate as "1" I didn't calculate the current age adn just returned a null value.

In [32]:

```
 def calculateAge(birthDate):

    if birthDate == "1":
#         print('this is 1')
        return ""
    else:
        start_date = datetime.datetime.strptime(birthDate, '%m-%d-%Y')
        today = datetime.datetime.today()
        delta = relativedelta.relativedelta(today,start_date)
        return delta.years
```

In [33]:

```
data1 = data.withColumn("Current_age",  F.udf(calculateAge, StringType())
data1.show()
```

```
+------------------+------------------+----------+---------+------+----+-
---------+-----------+
|            course|               dob|first_name|last_name|points|s_id|
new_date|Current_age|
+------------------+------------------+----------+---------+------+----+-
---------+-----------+
|Humanities and Art|  October 14, 1983|      Alan|      Joe|    10|   1|1
0-14-1983|         38|
|  Computer Science|September 26, 1980|    Martin|  Genberg|    17|   2|0
9-26-1980|         41|
|    Graphic Design|     June 12, 1982|     Athur|   Watson|    16|   3|0
6-12-1982|         40|
|    Graphic Design|     April 5, 1987|  Anabelle|  Sanberg|    12|   4|0
4-05-1987|         35|
|        Psychology|  November 1, 1978|      Kira| Schommer|    11|   5|1
1-01-1978|         43|
|          Business|  17 February 1981| Christian|   Kiriam|    10|   6|0
2-17-1981|         41|
|  Machine Learning|    1 January 1984|   Barbara|  Ballard|    14|   7|0
1-01-1984|         38|
|     Deep Learning|  January 13, 1978|      John|       --|    10|   8|0
1-13-1978|         44|
|  Machine Learning|  26 December 1989|    Marcus|   Carson|    15|   9|1
2-26-1989|         32|
|           Physics|  30 December 1987|     Marta|   Brooks|    11|  10|1
2-30-1987|         34|
|    Data Analytics|     June 12, 1975|     Holly| Schwartz|    12|  11|0
6-12-1975|         47|
|  Computer Science|      July 2, 1985|     April|    Black|    11|  12|0
7-02-1985|         37|
|  Computer Science|     July 22, 1980|     Irene|  Bradley|    13|  13|0
7-22-1980|         41|
|        Psychology|   7 February 1986|      Mark|    Weber|    12|  14|0
2-07-1986|         36|
|       Informatics|      May 18, 1987|     Rosie|   Norman|     9|  15|0
5-18-1987|         35|
|          Business|   August 10, 1984|    Martin|   Steele|     7|  16|0
8-10-1984|         37|
|  Machine Learning|  16 December 1990|     Colin| Martinez|     9|  17|1
2-16-1990|         31|
|    Data Analytics|           unknown|   Bridget|    Twain|     6|  18|
1|            |
|          Business|      7 March 1980|   Darlene|    Mills|    19|  19|0
3-07-1980|         42|
|    Data Analytics|      June 2, 1985|   Zachary|       --|    10|  20|0
6-02-1985|         37|
+------------------+------------------+----------+---------+------+----+-
---------+-----------+
```

In [35]:

```
sd = data1.agg({'points': 'stddev'}).collect()
print("The standard deviation of the points column is ",sd[0][0])
```

```
The standard deviation of the points column is  3.246050231475656
```

In [37]:

```
mean = data1.agg({'points': 'mean'}).collect()
print("The mean of the points column is ",mean[0][0])
```

```
The mean of the points column is  11.7
```

1. Let’s consider granting some points for good performed students in the class. For each student, if his point is larger than 1 standard deviation of all points, then we update his current point to 20, which is the maximum. See Annex 1 for a tutorial on how to calculate standard deviation. (2 points)

In [39]:

```
std_1 = mean[0][0]+sd[0][0]
std_2 = mean[0][0]-sd[0][0]
```

Here for each student, I check if his point is larger than 1 standard deviation of all points, then I update his current point to 20.

In [41]:

```
from pyspark.sql.functions import when
df3 = data1.withColumn("points", when(data1.points>std_1,20).otherwise(da
df3.show()
```

```
+------------------+------------------+----------+---------+------+----+-
---------+-----------+
|            course|               dob|first_name|last_name|points|s_id|
new_date|Current_age|
+------------------+------------------+----------+---------+------+----+-
---------+-----------+
|Humanities and Art|  October 14, 1983|      Alan|      Joe|    10|   1|1
0-14-1983|         38|
|  Computer Science|September 26, 1980|    Martin|  Genberg|    20|   2|0
9-26-1980|         41|
|    Graphic Design|     June 12, 1982|     Athur|   Watson|    20|   3|0
6-12-1982|         40|
|    Graphic Design|     April 5, 1987|  Anabelle|  Sanberg|    12|   4|0
4-05-1987|         35|
|        Psychology|  November 1, 1978|      Kira| Schommer|    11|   5|1
1-01-1978|         43|
|          Business|  17 February 1981| Christian|   Kiriam|    10|   6|0
2-17-1981|         41|
|  Machine Learning|    1 January 1984|   Barbara|  Ballard|    14|   7|0
1-01-1984|         38|
|     Deep Learning|  January 13, 1978|      John|       --|    10|   8|0
1-13-1978|         44|
|  Machine Learning|  26 December 1989|    Marcus|   Carson|    20|   9|1
2-26-1989|         32|
|           Physics|  30 December 1987|     Marta|   Brooks|    11|  10|1
2-30-1987|         34|
|    Data Analytics|     June 12, 1975|     Holly| Schwartz|    12|  11|0
6-12-1975|         47|
|  Computer Science|      July 2, 1985|     April|    Black|    11|  12|0
7-02-1985|         37|
|  Computer Science|     July 22, 1980|     Irene|  Bradley|    13|  13|0
7-22-1980|         41|
|        Psychology|   7 February 1986|      Mark|    Weber|    12|  14|0
2-07-1986|         36|
|       Informatics|      May 18, 1987|     Rosie|   Norman|     9|  15|0
5-18-1987|         35|
|          Business|   August 10, 1984|    Martin|   Steele|     7|  16|0
8-10-1984|         37|
|  Machine Learning|  16 December 1990|     Colin| Martinez|     9|  17|1
2-16-1990|         31|
|    Data Analytics|           unknown|   Bridget|    Twain|     6|  18|
1|           |
|          Business|      7 March 1980|   Darlene|    Mills|    20|  19|0
3-07-1980|         42|
|    Data Analytics|      June 2, 1985|   Zachary|       --|    10|  20|0
6-02-1985|         37|
+------------------+------------------+----------+---------+------+----+-
---------+-----------+
```

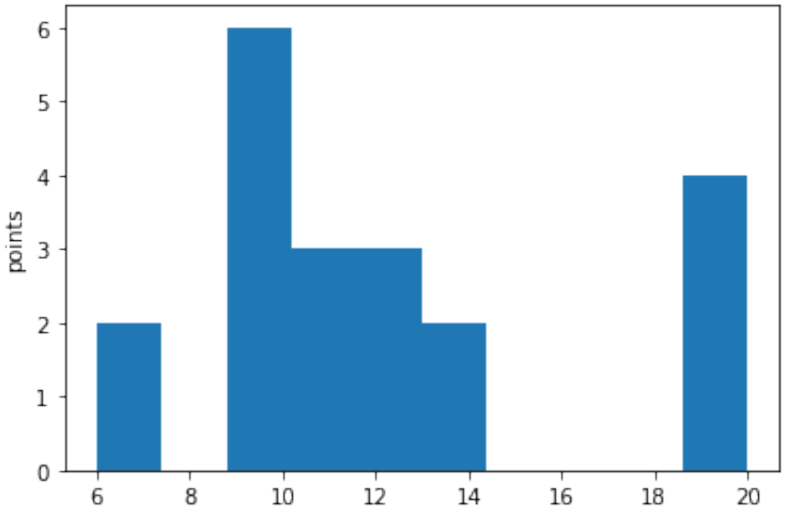
1. Create a histogram on the new points created in the task 5. (1 point)

In [42]:

```
import numpy as np
import matplotlib.pyplot as plt
points=[]
points_array = np.array(df3.select('points').collect())
# points_array
for value in points_array:
    points.append(value[0])
print(points)
student_id=[]
id_array = np.array(df3.select('s_id').collect())
# points_array
for value in id_array:
    student_id.append(value[0])
print(student_id)

plt.hist(points)
# plt.xlabel("sd_id")
plt.ylabel("points")
plt.show()
```

```
[10, 20, 20, 12, 11, 10, 14, 10, 20, 11, 12, 11, 13, 12, 9, 7, 9, 6, 20,
10]
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20]
```



In [ ]:

```
```

Exercise 2

First importing all the libraries required for Spark session in python. A SparkContext represents the connection to a Spark cluster, and is used to create RDDs, accumulators and broadcast variables on that cluster. The sql function on a SQLContext enables applications to run SQL queries programmatically and returns the result as a DataFrame .

In [1]:

```
import pyspark
from pyspark import SparkContext
sc =SparkContext()
from pyspark.sql import Row
from pyspark.sql import SQLContext

sqlContext = SQLContext(sc)
from pyspark.sql import SparkSession
```

```
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use set
LogLevel(newLevel).
22/07/03 08:32:12 WARN NativeCodeLoader: Unable to load native-hadoop lib
rary for your platform... using builtin-java classes where applicable
22/07/03 08:32:13 WARN Utils: Service 'SparkUI' could not bind on port 40
40. Attempting port 4041.
22/07/03 08:32:13 WARN Utils: Service 'SparkUI' could not bind on port 40
41. Attempting port 4042.
22/07/03 08:32:13 WARN Utils: Service 'SparkUI' could not bind on port 40
42. Attempting port 4043.
22/07/03 08:32:13 WARN Utils: Service 'SparkUI' could not bind on port 40
43. Attempting port 4044.
22/07/03 08:32:13 WARN Utils: Service 'SparkUI' could not bind on port 40
44. Attempting port 4045.
22/07/03 08:32:13 WARN Utils: Service 'SparkUI' could not bind on port 40
45. Attempting port 4046.
22/07/03 08:32:13 WARN Utils: Service 'SparkUI' could not bind on port 40
46. Attempting port 4047.
/opt/homebrew/lib/python3.9/site-packages/pyspark/sql/context.py:112: Fut
ureWarning: Deprecated in 3.0.0. Use SparkSession.builder.getOrCreate() i
nstead.
  warnings.warn(
```

A SparkSession can be used create DataFrame, register DataFrame as tables, execute SQL over tables, cache tables, and read data files. Class builder is Builder for SparkSession. appName() sets a name for the application, which will be shown in the Spark web UI. If no application name is set, a randomly generated name will be used. getOrCreate() gets an existing SparkSession or, if there is no existing one, creates a new one based on the options set in this builder.

Using spark.read.csv("path") we can read a CSV file with fields delimited by pipe, comma, tab (and many more) into a Spark DataFrame, These methods take a file path to read from as an argument.

In [2]:

```
spark = SparkSession.builder.appName(
    'Read CSV File into DataFrame').getOrCreate()

data = spark.read.csv('ml-10M100K-2/tags.dat', sep='||',inferSchema=True,

# Show the first 5 values of the dataframe
data.show(5)
```

```
22/07/03 08:32:21 WARN SparkSession: Using an existing Spark session; onl
y runtime SQL configurations will take effect.
```

```
+------+-------+---------------+----------+
|UserID|MovieID|            Tag| Timestamp|
+------+-------+---------------+----------+
|    15|   4973|     excellent!|1215184630|
|    20|   1747|      politics|1188263867|
|    20|   1747|         satire|1188263867|
|    20|   2424|chick flick 212|1188263835|
|    20|   2424|          hanks|1188263835|
+------+-------+---------------+----------+
only showing top 5 rows
```

In [3]:

```
from pyspark import SparkFiles
from pyspark.sql.functions import col
```

In [4]:

```
# Ordering or Sorting the dataframe base din Timestamp column
data.orderBy(col("Timestamp").asc()).show(10)
```

```
+------+-------+--------------------+----------+
|UserID|MovieID|                 Tag| Timestamp|
+------+-------+--------------------+----------+
| 19106|   1247|       mrs. robinson|1135313387|
| 19106|    145|           car chase|1135313403|
| 19106|   3481|          jack black|1135313411|
| 19106|   3481|               funny|1135313411|
| 19106|  34405|the man they call...|1135313453|
| 19106|   1396|    too many secrets|1135313492|
| 71331|   1396|     setec astronomy|1135361580|
| 22198|   2788|        monty python|1135429210|
| 22198|   1732|       coen brothers|1135429236|
| 22198|   1206|     stanley kubrick|1135429248|
+------+-------+--------------------+----------+
only showing top 10 rows
```

In [8]:

```
# In order to convert timestamp to readable date we import this in built
from datetime import datetime
```

In this function I take my timestamp value from the dataframe which is in string and convert it into integer. After that I use datetime.utcfromtimestamp().strftime() function to convert the time stamp into a format as day-month-Year Hours:Minutes:Seconds. Then I return this readable date.

In [9]:

```
def date_time(timestamp):
    timesatmp = int (timestamp)
    date = datetime.utcfromtimestamp(timesatmp).strftime('%d-%m-%Y %H:%M:
    return date
```

In [10]:

```
date_time("1135429236")
```

Out[10]:

```
'24-12-2005 13:00:36'
```

In [11]:

```
from pyspark.sql import functions as F
from pyspark.sql.functions import to_date
from pyspark.sql.types import StringType
```

In the code below I have taken the Timestamp column and performed the timestamp to date conversion by calling my user defined program date_time and saved the returned value from the function in a separate column named as Date_Time. We can see that we have obtained a new coolumn which contains the new readable date and time.

In [12]:

```
data1 = data.withColumn("Date_Time",  F.udf(date_time, StringType())(F.co
data1.show()
```

```
[Stage 3:>                                                          (0 +
1) / 1]
```

```
+------+-------+--------------------+----------+-------------------+
|UserID|MovieID|                 Tag| Timestamp|          Date_Time|
+------+-------+--------------------+----------+-------------------+
|    15|   4973|          excellent!|1215184630|04-07-2008 15:17:10|
|    20|   1747|            politics|1188263867|28-08-2007 01:17:47|
|    20|   1747|              satire|1188263867|28-08-2007 01:17:47|
|    20|   2424|      chick flick 212|1188263835|28-08-2007 01:17:15|
|    20|   2424|               hanks|1188263835|28-08-2007 01:17:15|
|    20|   2424|                ryan|1188263835|28-08-2007 01:17:15|
|    20|   2947|              action|1188263755|28-08-2007 01:15:55|
|    20|   2947|                bond|1188263756|28-08-2007 01:15:56|
|    20|   3033|               spoof|1188263880|28-08-2007 01:18:00|
|    20|   3033|           star wars|1188263880|28-08-2007 01:18:00|
|    20|   7438|              bloody|1188263801|28-08-2007 01:16:41|
|    20|   7438|             kung fu|1188263801|28-08-2007 01:16:41|
|    20|   7438|           Tarantino|1188263801|28-08-2007 01:16:41|
|    21|  55247|                   R|1205081506|09-03-2008 16:51:46|
|    21|  55253|               NC-17|1205081488|09-03-2008 16:51:28|
|    25|     50|        Kevin Spacey|1166101426|14-12-2006 13:03:46|
|    25|   6709|         Johnny Depp|1162147221|29-10-2006 18:40:21|
|    31|     65|        buddy comedy|1188263759|28-08-2007 01:15:59|
|    31|    546|strangely compelling|1188263674|28-08-2007 01:14:34|
|    31|   1091|         catastrophe|1188263741|28-08-2007 01:15:41|
+------+-------+--------------------+----------+-------------------+
only showing top 20 rows
```

1. A tagging session for a user can be defined as the duration in which he/she generated tagging activities. Typically, an inactive duration of 30 mins is considered as a termination of the tagging session. Your task is to separate out tagging sessions for each user.

In [14]:
```
from pyspark.sql import Window
```

PYSPARK LAG is a function in PySpark that works as the offset row returning the value of the before row of a column with respect to the current row in PySpark. This is an operation in PySpark that returns the row just before the current row. Here I have used lag in the timestamp column and partitioned by the userIds. Here I get the previous timestamp for each userid.

In [28]:
```
data1 = data1.withColumn(
    "session_id",
    F.lag("Timestamp",1).over(Window.partitionBy("UserID").orderBy("Times
)
```

In [29]:
```
data1.show(10)
```

```
[Stage 37:>                                                     (0 + 
1) / 1]
```

```
+------+-------+---------------+----------+-------------------+----------
+
|UserID|MovieID|            Tag| Timestamp|          Date_Time|session_id
|
+------+-------+---------------+----------+-------------------+----------
+
|    20|   2947|         action|1188263755|28-08-2007 01:15:55|      null
|
|    20|   2947|           bond|1188263756|28-08-2007 01:15:56|1188263755
|
|    20|   7438|         bloody|1188263801|28-08-2007 01:16:41|1188263756
|
|    20|   7438|        kung fu|1188263801|28-08-2007 01:16:41|1188263801
|
|    20|   7438|      Tarantino|1188263801|28-08-2007 01:16:41|1188263801
|
|    20|   2424|chick flick 212|1188263835|28-08-2007 01:17:15|1188263801
|
|    20|   2424|          hanks|1188263835|28-08-2007 01:17:15|1188263835
|
|    20|   2424|           ryan|1188263835|28-08-2007 01:17:15|1188263835
|
|    20|   1747|       politics|1188263867|28-08-2007 01:17:47|1188263835
|
|    20|   1747|         satire|1188263867|28-08-2007 01:17:47|1188263867
|
+------+-------+---------------+----------+-------------------+----------
+
only showing top 10 rows
```

Define if the session is the 1st one (more than 1800s after the previous one). Here I subtract the timestamps from the timestamp column and session id column created and check it it is less than 1800 seconds or 30 minutes. If it is less then I assign it as 0 or else I assign it as 1.

In [30]:
```
df = data1.withColumn("session_id",
    F.when(F.col("Timestamp") - F.col("session_id") <= 1800, 0).otherwise
)
```

In [31]:
```
df.show(10)
```

```
[Stage 40:>                                                         (0 +
1) / 1]
```

```
+------+-------+---------------+----------+-------------------+----------
+
|UserID|MovieID|            Tag| Timestamp|          Date_Time|session_id
|
+------+-------+---------------+----------+-------------------+----------
+
|    20|   2947|         action|1188263755|28-08-2007 01:15:55|         1
|
|    20|   2947|           bond|1188263756|28-08-2007 01:15:56|         0
|
|    20|   7438|         bloody|1188263801|28-08-2007 01:16:41|         0
|
|    20|   7438|        kung fu|1188263801|28-08-2007 01:16:41|         0
|
|    20|   7438|      Tarantino|1188263801|28-08-2007 01:16:41|         0
|
|    20|   2424|chick flick 212|1188263835|28-08-2007 01:17:15|         0
|
|    20|   2424|          hanks|1188263835|28-08-2007 01:17:15|         0
|
|    20|   2424|           ryan|1188263835|28-08-2007 01:17:15|         0
|
|    20|   1747|       politics|1188263867|28-08-2007 01:17:47|         0
|
|    20|   1747|         satire|1188263867|28-08-2007 01:17:47|         0
|
+------+-------+---------------+----------+-------------------+----------
+
only showing top 10 rows
```

Here I find out the unique session id based on the above result. So basically I sum over the session Id column and partiotion by User Id and again order by the timestamp column. Note that we can get same ids for different users.

In [32]:

```
# create a unique id per session
df = df.withColumn(
    "session_id",
    F.sum("session_id").over(Window.partitionBy("userid").orderBy("timest
)
```

In [33]:

```
df.show(10)
```

```
[Stage 43:>                                                        (0 +
1) / 1]
```

```
+------+-------+---------------+----------+-------------------+----------
+
|UserID|MovieID|            Tag| Timestamp|          Date_Time|session_id
|
+------+-------+---------------+----------+-------------------+----------
+
|    20|   2947|         action|1188263755|28-08-2007 01:15:55|         1
|
|    20|   2947|           bond|1188263756|28-08-2007 01:15:56|         1
|
|    20|   7438|         bloody|1188263801|28-08-2007 01:16:41|         1
|
|    20|   7438|        kung fu|1188263801|28-08-2007 01:16:41|         1
|
|    20|   7438|      Tarantino|1188263801|28-08-2007 01:16:41|         1
|
|    20|   2424|chick flick 212|1188263835|28-08-2007 01:17:15|         1
|
|    20|   2424|          hanks|1188263835|28-08-2007 01:17:15|         1
|
|    20|   2424|           ryan|1188263835|28-08-2007 01:17:15|         1
|
|    20|   1747|       politics|1188263867|28-08-2007 01:17:47|         1
|
|    20|   1747|         satire|1188263867|28-08-2007 01:17:47|         1
|
+------+-------+---------------+----------+-------------------+----------
+
only showing top 10 rows
```

In the code below I finally create a unique id per session per user where F.dense_rank() returns the rank of rows within a window partition without any gaps.

In [34]:

```
df = df.withColumn(
    "session_id", F.dense_rank().over(Window.orderBy("userid", "session_i
)
```

In [36]:

```
df.show(10)
```

```
22/07/03 10:19:04 WARN WindowExec: No Partition Defined for Window operat
ion! Moving all data to a single partition, this can cause serious perfor
mance degradation.
22/07/03 10:19:04 WARN WindowExec: No Partition Defined for Window operat
ion! Moving all data to a single partition, this can cause serious perfor
mance degradation.
22/07/03 10:19:04 WARN WindowExec: No Partition Defined for Window operat
ion! Moving all data to a single partition, this can cause serious perfor
mance degradation.
```

```
[Stage 47:>                                                         (0 + 
1) / 1]
```

```
22/07/03 10:19:06 WARN WindowExec: No Partition Defined for Window operat
ion! Moving all data to a single partition, this can cause serious perfor
mance degradation.
22/07/03 10:19:06 WARN WindowExec: No Partition Defined for Window operat
ion! Moving all data to a single partition, this can cause serious perfor
mance degradation.
22/07/03 10:19:06 WARN WindowExec: No Partition Defined for Window operat
ion! Moving all data to a single partition, this can cause serious perfor
mance degradation.
[Stage 47:>                                                      (0 + 
1) / 1]
22/07/03 10:19:13 WARN WindowExec: No Partition Defined for Window operat
ion! Moving all data to a single partition, this can cause serious perfor
mance degradation.
22/07/03 10:19:13 WARN WindowExec: No Partition Defined for Window operat
ion! Moving all data to a single partition, this can cause serious perfor
mance degradation.
22/07/03 10:19:13 WARN WindowExec: No Partition Defined for Window operat
ion! Moving all data to a single partition, this can cause serious perfor
mance degradation.
+------+-------+---------------+----------+-------------------+----------
+
|UserID|MovieID|            Tag| Timestamp|          Date_Time|session_id
|
+------+-------+---------------+----------+-------------------+----------
+
|    15|   4973|     excellent!|1215184630|04-07-2008 15:17:10|         1
|
|    20|   2947|         action|1188263755|28-08-2007 01:15:55|         2
|
|    20|   2947|           bond|1188263756|28-08-2007 01:15:56|         2
|
|    20|   7438|         bloody|1188263801|28-08-2007 01:16:41|         2
|
|    20|   7438|        kung fu|1188263801|28-08-2007 01:16:41|         2
|
|    20|   7438|      Tarantino|1188263801|28-08-2007 01:16:41|         2
|
|    20|   2424|chick flick 212|1188263835|28-08-2007 01:17:15|         2
|
|    20|   2424|          hanks|1188263835|28-08-2007 01:17:15|         2
|
|    20|   2424|           ryan|1188263835|28-08-2007 01:17:15|         2
|
|    20|   1747|       politics|1188263867|28-08-2007 01:17:47|         2
|
+------+-------+---------------+----------+-------------------+----------
+
only showing top 10 rows
```

1. Once you have all the tagging sessions for each user, calculate the frequency of tagging for each user session.

In [37]:

```
# Here I have obtained the frequency of tagging for each user session
df1 = df.groupBy(['UserID','session_id']).count()
df1.show()
```

```
22/07/03 10:19:19 WARN WindowExec: No Partition Defined for Window operat
ion! Moving all data to a single partition, this can cause serious perfor
mance degradation.
22/07/03 10:19:19 WARN WindowExec: No Partition Defined for Window operat
ion! Moving all data to a single partition, this can cause serious perfor
mance degradation.
22/07/03 10:19:19 WARN WindowExec: No Partition Defined for Window operat
ion! Moving all data to a single partition, this can cause serious perfor
mance degradation.
22/07/03 10:19:19 WARN WindowExec: No Partition Defined for Window operat
ion! Moving all data to a single partition, this can cause serious perfor
mance degradation.
22/07/03 10:19:20 WARN WindowExec: No Partition Defined for Window operat
ion! Moving all data to a single partition, this can cause serious perfor
mance degradation.
22/07/03 10:19:20 WARN WindowExec: No Partition Defined for Window operat
ion! Moving all data to a single partition, this can cause serious perfor
mance degradation.
+------+----------+-----+
|UserID|session_id|count|
+------+----------+-----+
|    15|         1|    1|
|    20|         2|   12|
|    21|         3|    2|
|    25|         4|    1|
|    25|         5|    1|
|    31|         6|    5|
|    32|         7|    1|
|    39|         8|    5|
|    48|         9|    2|
|    49|        10|   15|
|    75|        11|    1|
|    78|        12|    1|
|   109|        13|   11|
|   109|        14|    2|
|   109|        15|    3|
|   109|        16|    1|
|   109|        17|    1|
|   109|        18|    1|
|   109|        19|    4|
|   109|        20|    1|
+------+----------+-----+
only showing top 20 rows
```

1. Find a mean and standard deviation of the tagging frequency of each user.

In [40]:

```
df2 = df1.groupBy('UserID','session_id').agg({'count': 'mean'})
df2.show()
```

```
22/07/03 10:20:43 WARN WindowExec: No Partition Defined for Window operat
ion! Moving all data to a single partition, this can cause serious perfor
mance degradation.
22/07/03 10:20:43 WARN WindowExec: No Partition Defined for Window operat
ion! Moving all data to a single partition, this can cause serious perfor
mance degradation.
22/07/03 10:20:43 WARN WindowExec: No Partition Defined for Window operat
ion! Moving all data to a single partition, this can cause serious perfor
mance degradation.
22/07/03 10:20:43 WARN WindowExec: No Partition Defined for Window operat
ion! Moving all data to a single partition, this can cause serious perfor
mance degradation.
22/07/03 10:20:44 WARN WindowExec: No Partition Defined for Window operat
ion! Moving all data to a single partition, this can cause serious perfor
mance degradation.
22/07/03 10:20:44 WARN WindowExec: No Partition Defined for Window operat
ion! Moving all data to a single partition, this can cause serious perfor
mance degradation.
+------+----------+----------+
|UserID|session_id|avg(count)|
+------+----------+----------+
|    15|         1|       1.0|
|    20|         2|      12.0|
|    21|         3|       2.0|
|    25|         4|       1.0|
|    25|         5|       1.0|
|    31|         6|       5.0|
|    32|         7|       1.0|
|    39|         8|       5.0|
|    48|         9|       2.0|
|    49|        10|      15.0|
|    75|        11|       1.0|
|    78|        12|       1.0|
|   109|        13|      11.0|
|   109|        14|       2.0|
|   109|        15|       3.0|
|   109|        16|       1.0|
|   109|        17|       1.0|
|   109|        18|       1.0|
|   109|        19|       4.0|
|   109|        20|       1.0|
+------+----------+----------+
only showing top 20 rows
```

In [41]:

```
df3 = df1.groupBy('UserID').agg({'session_id': 'stddev'}).show()
```

```
22/07/03 10:21:03 WARN WindowExec: No Partition Defined for Window operat
ion! Moving all data to a single partition, this can cause serious perfor
mance degradation.
22/07/03 10:21:03 WARN WindowExec: No Partition Defined for Window operat
ion! Moving all data to a single partition, this can cause serious perfor
mance degradation.
22/07/03 10:21:04 WARN WindowExec: No Partition Defined for Window operat
ion! Moving all data to a single partition, this can cause serious perfor
mance degradation.
22/07/03 10:21:04 WARN WindowExec: No Partition Defined for Window operat
ion! Moving all data to a single partition, this can cause serious perfor
mance degradation.
22/07/03 10:21:04 WARN WindowExec: No Partition Defined for Window operat
ion! Moving all data to a single partition, this can cause serious perfor
mance degradation.
22/07/03 10:21:04 WARN WindowExec: No Partition Defined for Window operat
ion! Moving all data to a single partition, this can cause serious perfor
mance degradation.
+------+------------------+
|UserID|stddev(session_id)|
+------+------------------+
|    15|              null|
|    20|              null|
|    21|              null|
|    25|0.7071067811865476|
|    31|              null|
|    32|              null|
|    39|              null|
|    48|              null|
|    49|              null|
|    75|              null|
|    78|              null|
|   109|2.7386127875258306|
|   127|              null|
|   133|              null|
|   146| 96.27304918823336|
|   147|              null|
|   170|              null|
|   175|0.7071067811865476|
|   181|              null|
|   190|1.2909944487358056|
+------+------------------+
only showing top 20 rows
```

1. Find a mean and standard deviation of the tagging frequency for across users.

In [42]:

```
mean = df1.agg({'count': 'mean'}).collect()
print("The mean across all users = ",mean[0][0])
```

```
22/07/03 10:21:29 WARN WindowExec: No Partition Defined for Window operat
ion! Moving all data to a single partition, this can cause serious perfor
mance degradation.
22/07/03 10:21:29 WARN WindowExec: No Partition Defined for Window operat
ion! Moving all data to a single partition, this can cause serious perfor
mance degradation.
22/07/03 10:21:29 WARN WindowExec: No Partition Defined for Window operat
ion! Moving all data to a single partition, this can cause serious perfor
mance degradation.
22/07/03 10:21:29 WARN WindowExec: No Partition Defined for Window operat
ion! Moving all data to a single partition, this can cause serious perfor
mance degradation.
22/07/03 10:21:30 WARN WindowExec: No Partition Defined for Window operat
ion! Moving all data to a single partition, this can cause serious perfor
mance degradation.
22/07/03 10:21:30 WARN WindowExec: No Partition Defined for Window operat
ion! Moving all data to a single partition, this can cause serious perfor
mance degradation.
The mean across all users =  7.300084014358817
```

In [43]:

```
stddev = df1.agg({'count': 'stddev'}).collect()
print("The standard deviation across all users = ",stddev[0][0])
```

```
22/07/03 10:21:32 WARN WindowExec: No Partition Defined for Window operat
ion! Moving all data to a single partition, this can cause serious perfor
mance degradation.
22/07/03 10:21:32 WARN WindowExec: No Partition Defined for Window operat
ion! Moving all data to a single partition, this can cause serious perfor
mance degradation.
22/07/03 10:21:32 WARN WindowExec: No Partition Defined for Window operat
ion! Moving all data to a single partition, this can cause serious perfor
mance degradation.
22/07/03 10:21:32 WARN WindowExec: No Partition Defined for Window operat
ion! Moving all data to a single partition, this can cause serious perfor
mance degradation.
22/07/03 10:21:33 WARN WindowExec: No Partition Defined for Window operat
ion! Moving all data to a single partition, this can cause serious perfor
mance degradation.
22/07/03 10:21:33 WARN WindowExec: No Partition Defined for Window operat
ion! Moving all data to a single partition, this can cause serious perfor
mance degradation.
The standard deviation across all users =  22.264293050264985
```

1. Provide the list of users with a mean tagging frequency within the two standard deviation from the mean frequency of all users.

In [45]:

```
# Range of two standard deviation from mean frequency of all users
std_1 = mean[0][0]+2*stddev[0][0]
std_2 = mean[0][0]-2*stddev[0][0]
```

Here I check if the tagging frequency lies between the above range of values. If it does then I assign it as 1 or else I assign it as 0

In [46]:
```
df1 = df1.withColumnRenamed("count","Frequency")
from pyspark.sql.functions import when
df4 = df1.withColumn("count", when(df1.Frequency<std_1, when(df1.Frequenc
df4.show(100)
```

```
22/07/03 10:26:01 WARN WindowExec: No Partition Defined for Window operat
ion! Moving all data to a single partition, this can cause serious perfor
mance degradation.
22/07/03 10:26:01 WARN WindowExec: No Partition Defined for Window operat
ion! Moving all data to a single partition, this can cause serious perfor
mance degradation.
22/07/03 10:26:01 WARN WindowExec: No Partition Defined for Window operat
ion! Moving all data to a single partition, this can cause serious perfor
mance degradation.
22/07/03 10:26:01 WARN WindowExec: No Partition Defined for Window operat
ion! Moving all data to a single partition, this can cause serious perfor
mance degradation.
22/07/03 10:26:02 WARN WindowExec: No Partition Defined for Window operat
ion! Moving all data to a single partition, this can cause serious perfor
mance degradation.
22/07/03 10:26:02 WARN WindowExec: No Partition Defined for Window operat
ion! Moving all data to a single partition, this can cause serious perfor
mance degradation.
+------+----------+---------+-----+
|UserID|session_id|Frequency|count|
+------+----------+---------+-----+
|    15|         1|        1|    1|
|    20|         2|       12|    1|
|    21|         3|        2|    1|
|    25|         4|        1|    1|
|    25|         5|        1|    1|
|    31|         6|        5|    1|
|    32|         7|        1|    1|
|    39|         8|        5|    1|
|    48|         9|        2|    1|
|    49|        10|       15|    1|
|    75|        11|        1|    1|
|    78|        12|        1|    1|
|   109|        13|       11|    1|
|   109|        14|        2|    1|
|   109|        15|        3|    1|
|   109|        16|        1|    1|
|   109|        17|        1|    1|
|   109|        18|        1|    1|
|   109|        19|        4|    1|
|   109|        20|        1|    1|
|   109|        21|        1|    1|
|   127|        22|       26|    1|
|   133|        23|        5|    1|
|   146|        24|        1|    1|
|   146|        25|        1|    1|
|   146|        26|        2|    1|
|   146|        27|       12|    1|
|   146|        28|        2|    1|
|   146|        29|        4|    1|
|   146|        30|       18|    1|
|    146|         31|        53|     0|
|    146|         32|         3|     1|
|    146|         33|         3|     1|
|    146|         34|         2|     1|
|    146|         35|         2|     1|
|    146|         36|         2|     1|
|    146|         37|         1|     1|
|    146|         38|         1|     1|
|    146|         39|         1|     1|
|    146|         40|         4|     1|
|    146|         41|         1|     1|
|    146|         42|         1|     1|
|    146|         43|         2|     1|
|    146|         44|         4|     1|
|    146|         45|         2|     1|
|    146|         46|         2|     1|
|    146|         47|         2|     1|
|    146|         48|         8|     1|
|    146|         49|         4|     1|
|    146|         50|         1|     1|
|    146|         51|         1|     1|
|    146|         52|         1|     1|
|    146|         53|         1|     1|
|    146|         54|        10|     1|
|    146|         55|         1|     1|
|    146|         56|         1|     1|
|    146|         57|         4|     1|
|    146|         58|         1|     1|
|    146|         59|        18|     1|
|    146|         60|         1|     1|
|    146|         61|         6|     1|
|    146|         62|         4|     1|
|    146|         63|         1|     1|
|    146|         64|         1|     1|
|    146|         65|         3|     1|
|    146|         66|        14|     1|
|    146|         67|        69|     0|
|    146|         68|         3|     1|
|    146|         69|        39|     1|
|    146|         70|        27|     1|
|    146|         71|         7|     1|
|    146|         72|         4|     1|
|    146|         73|         2|     1|
|    146|         74|         4|     1|
|    146|         75|         8|     1|
|    146|         76|         1|     1|
|    146|         77|        40|     1|
|    146|         78|         2|     1|
|    146|         79|         3|     1|
|    146|         80|         1|     1|
|    146|         81|         3|     1|
|    146|         82|         1|     1|
|    146|         83|        63|     0|
|    146|         84|        56|     0|
|    146|         85|         5|     1|
|    146|         86|         4|     1|
|    146|         87|         2|     1|
|   146|        88|        2|    1|
|   146|        89|        1|    1|
|   146|        90|        2|    1|
|   146|        91|        2|    1|
|   146|        92|        3|    1|
|   146|        93|        1|    1|
|   146|        94|        4|    1|
|   146|        95|        2|    1|
|   146|        96|        2|    1|
|   146|        97|        1|    1|
|   146|        98|        3|    1|
|   146|        99|        6|    1|
|   146|       100|       16|    1|
+------+----------+---------+-----+
only showing top 100 rows
```

In [47]:

```
df4 = df4.withColumnRenamed("count","Users_in_the_range")
df5 = df4.filter(df4.Users_in_the_range == 1)
```

this dataframe df5 contains all the users whose mean frequency lies within 2 standard devaitions away from mean frequency of all users.

In [48]:

```
df5.show(10)
```

```
22/07/03 10:27:19 WARN WindowExec: No Partition Defined for Window operat
ion! Moving all data to a single partition, this can cause serious perfor
mance degradation.
22/07/03 10:27:19 WARN WindowExec: No Partition Defined for Window operat
ion! Moving all data to a single partition, this can cause serious perfor
mance degradation.
22/07/03 10:27:19 WARN WindowExec: No Partition Defined for Window operat
ion! Moving all data to a single partition, this can cause serious perfor
mance degradation.
22/07/03 10:27:19 WARN WindowExec: No Partition Defined for Window operat
ion! Moving all data to a single partition, this can cause serious perfor
mance degradation.
22/07/03 10:27:20 WARN WindowExec: No Partition Defined for Window operat
ion! Moving all data to a single partition, this can cause serious perfor
mance degradation.
22/07/03 10:27:20 WARN WindowExec: No Partition Defined for Window operat
ion! Moving all data to a single partition, this can cause serious perfor
mance degradation.
+------+----------+---------+------------------+
|UserID|session_id|Frequency|Users_in_the_range|
+------+----------+---------+------------------+
|    15|         1|        1|                 1|
|    20|         2|       12|                 1|
|    21|         3|        2|                 1|
|    25|         4|        1|                 1|
|    25|         5|        1|                 1|
|    31|         6|        5|                 1|
|    32|         7|        1|                 1|
|    39|         8|        5|                 1|
|    48|         9|        2|                 1|
|    49|        10|       15|                 1|
+------+----------+---------+------------------+
only showing top 10 rows
```

In [ ]: