Apache Spark Basics

In [ ]:

```
```

Part a) Basic Operations on Resilient Distributed Dataset (RDD)

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

In [1114]:

```
import findspark
findspark.init()
findspark.find()
import pyspark
findspark.find()
```

executed in 150ms, finished 07:36:01 2020-07-06

Out[1114]:

```
'C:\\Users\\admin\\spark\\spark-2.3.2-bin-hadoop2.7'
```

In [ ]:

```
```

executed in 374ms, finished 07:36:03 2020-07-06

In [ ]:

```
from pyspark import SparkContext, SparkConf
from pyspark.sql import SparkSession
conf = pyspark.SparkConf().setAppName('appName').setMaster('local')
sc = pyspark.SparkContext(conf=conf)
spark = SparkSession(sc)
```

executed in 738ms, finished 12:23:44 2020-07-03

In [ ]:

```
```

executed in 294ms, finished 12:23:49 2020-07-03

In [1116]:

```
#initializing a
a = ["spark", "rdd", "python", "context", "create", "class"]
```

executed in 6ms, finished 07:36:17 2020-07-06

In [ ]:

```
```

In [1117]:

```
#initializing b
b = ["operation", "apache", "scala", "lambda","parallel","partition"]
```

executed in 5ms, finished 07:36:19 2020-07-06

In [ ]:

In [1122]:

```
#creating RDDs from 'a' such that, that RDD is a key value pair
def add_key(value):
    return (value,1)
rdd1 = sc.parallelize(a)
rdd1 = rdd1.map(add_key)
rdd1.collect()
```

executed in 728ms, finished 07:37:01 2020-07-06

Out[1122]:

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

In [1123]:

```
#creating RDDs from 'b' such that, that RDD is a key value pair
rdd2 = sc.parallelize(b)
rdd2 = rdd2.map(add_key)
rdd2.collect()
```

executed in 1.06s, finished 07:37:03 2020-07-06

Out[1123]:

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

In [1124]:

```
#Right outer join
roj = rdd1.rightOuterJoin(rdd2)
```

executed in 36ms, finished 07:37:03 2020-07-06

In [1125]:

```
roj.collect()
```

executed in 5.45s, finished 07:37:10 2020-07-06

Out[1125]:

```
[('scala', (None, 1)),
 ('parallel', (None, 1)),
 ('partition', (None, 1)),
 ('operation', (None, 1)),
 ('apache', (None, 1)),
 ('lambda', (None, 1))]
```

```
In [1126]: #Full outer join
           rdd1.fullOuterJoin(rdd2).collect()
```

executed in 5.65s, finished 07:37:17 2020-07-06

```
Out[1126]: [('python', (1, None)),
            ('class', (1, None)),
            ('scala', (None, 1)),
            ('parallel', (None, 1)),
            ('partition', (None, 1)),
            ('spark', (1, None)),
            ('rdd', (1, None)),
            ('context', (1, None)),
            ('create', (1, None)),
            ('operation', (None, 1)),
            ('apache', (None, 1)),
            ('lambda', (None, 1))]
```

Explanation

- The two inputs given are converted into two RDD's respectively such that each RDD is a key value pair with the key as the words in input and the value as 1.
- Right outer join is then computed on both RDD's
- Full outer join is also computed on both RDD's
- 1 in the value of output represents the presence of the key in the joined table.

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

In [1128]:
```
from operator import add
a = ["spark", "rdd", "python", "context", "create", "class"]
b = ["operation", "apache", "scala", "lambda","parallel","partition"]
#function to count and return the number of occurences of 's' in its parameter
def detect_s(x):
    x = list(x)
    c =0
    for i in x:
        if i == 's':
            c+=1
    return c

rdd3 = sc.parallelize(a)
rdd4 = sc.parallelize(b)
#Combining RDD's for input's a and b
newr = sc.union([rdd3,rdd4])
#Mapping Input words containing 's' to the number of 's's in that word and thereby
rdd_s = newr.map(detect_s).filter(lambda x: x is not None).reduce(add)
print(rdd_s)
```

executed in 2.56s, finished 07:43:15 2020-07-06

```
4
```

Explanation

- Create RDD's from a and b
- Combine the RDD's
- Detect the number of 's's in a word and place it as the value in a key-value pair with the word being the key
- Reduce the values

3. Using aggregate function to count how many times the character "s" appears in all a and b

In [1142]:

```python
from operator import add
a = ["spark", "rdd", "python", "context", "create", "class"]
b = ["operation", "apache", "scala", "lambda","parallel","partition"]
def detect_s(x):
    x = list(x)
    c =0
    for i in x:
        if i == 's':
            c+=1
    return c

rdd3 = sc.parallelize(a)
rdd4 = sc.parallelize(b)
newr = sc.union([rdd3,rdd4])
rdd_s = newr.filter(lambda x: 's' in x).map(detect_s).sum()
print(rdd_s)
```

executed in 1.78s, finished 08:10:53 2020-07-06

```
4
```

Explanation

- Create RDD's from a and b
- Combine the RDD's
- Filter words with s's and pass them to a map function that maps the words with the number of 's's in that word.
- Aggregate the results using sum() aggregate function

Part b) Basic Operations on DataFrames

In [1215]:

```python
#imports
from pyspark.sql import SQLContext
sqlContext = SQLContext(sc)
```

executed in 117ms, finished 09:15:29 2020-07-06

In [1216]:

```python
#Loading the dataset
df = sqlContext.read.json('C:\\Users\\admin\\Downloads\\students.json')
```

executed in 99ms, finished 09:15:30 2020-07-06

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

In [1217]:

```
from pyspark.sql.functions import mean as _mean, stddev as _stddev, col
#calculating mean of points
df_stats = df.select(
    _mean(col('points')).alias("mean"),
    _stddev(col('points')).alias("std deviation")
).collect()

print(df_stats[0]["mean"])
mean = df_stats[0]['mean']
std = df_stats[0]['std deviation']
print(std)
```

executed in 103ms, finished 09:15:32 2020-07-06

```
11.736842105263158
3.3307007147839007
```

In [ ]:

executed in 6ms, finished 20:24:34 2020-07-03

In [1218]:

```
#filling null values with mean of all points
df = df.na.fill(mean,['points'])
df.show()
```

executed in 150ms, finished 09:15:35 2020-07-06

```
+------------------+------------------+----------+---------+------+----+
|            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|
+------------------+------------------+----------+---------+------+----+
```

2.Replace the null value(s) in column dob and column last name by "unknown" and "--" respectively.

In [1219]:

```
df.show()
#filling null values with values specified in the question
df1 = df.na.fill("unknown","dob")
df1.show()
df1 = df1.na.fill("--","last_name")
df1.show()
df = df1
```

executed in 201ms, finished 09:15:37 2020-07-06

```
|        Psychology|   7 February 1980|      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|
+------------------+------------------+----------+---------+------+----+

+------------------+------------------+----------+---------+------+----+
|            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|
```

Explanation for Part b) 1 and 2

- The null values in points are replaced by mean of the column points by using the mean sql function
- The null values in dob and lastname are replaced by 'unknown' and '--' respectively using fill method of pyspark

3.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

4.Insert a new column age and calculate the current age of all students

In [1220]:

```
import pyspark.sql.functions as F
from pyspark.sql.types import *
import re
#function to change the format of the date to DD-MM-YYYY
def changedateformat(value):
    #dictionary with keys as months and the values as the month's number
    months = {'January':1,'February':2,"March":3,"April":4,"May":5,"June":6,"July
    valu = str(value)
    valu1 = str(valu)
    v = valu.split()
    print(type(value))
    #regexes for detecting months, years, days from the dob column
    regmonth = r"([a-zA-Z]+)"
    regday = r"(\d+)"
    regyear = r"([0-9]{4})"

    #matching regexes with the dob column
    matchmonth = re.search(regmonth, valu)
    matchday = re.search(regday, valu)
    matchyear = re.search(regyear, valu)

    if(matchyear != None):
        year = str(matchyear.group(0))
    else:
        return "unknown"
    if(matchday!=None):
        day = str(matchday.group(0))
    else:
        return "unknown"
    #returning the result in DD_MM-YYYY format
    if(matchmonth!=None and matchday!=None and matchyear!=None):
        return day+"/"+str(months[str(matchmonth.group(0))])+"/"+year


def computeage(value):
    #matching year
    regyear = r"([0-9]{4})"
    matchyear = re.search(regyear, str(value))
    #subtracting matched year from 2020
    if(matchyear!=None):
        return 2020-int(str(matchyear.group(0)))
    return None
#convert to a UDF Function by passing in the function and return type of function
udfdate = F.udf(changedateformat, StringType())
udfage = F.udf(computeage, StringType())
df = df.withColumn("dob", udfdate("dob"))
df = df.withColumn("age",udfage("dob"))
df.show()
```

executed in 773ms, finished 09:15:40 2020-07-06

```
+-----------------+---------+----------+---------+------+----+----+
|           course|      dob|first_name|last_name|points|s_id| age|
+-----------------+---------+----------+---------+------+----+----+
|Humanities and Art|14/10/1983|      Alan|      Joe|    10|   1|  37|
|  Computer Science| 26/9/1980|    Martin|  Genberg|    17|   2|  40|
|    Graphic Design| 12/6/1982|     Athur|   Watson|    16|   3|  38|
|    Graphic Design|  5/4/1987|  Anabelle|  Sanberg|    12|   4|  33|
|        Psychology| 1/11/1978|      Kira| Schommer|    11|   5|  42|
|          Business| 17/2/1981| Christian|   Kiriam|    10|   6|  39|
|  Machine Learning|  1/1/1984|   Barbara|  Ballard|    14|   7|  36|
|     Deep Learning| 13/1/1978|      John|       --|    10|   8|  42|
|  Machine Learning|26/12/1989|    Marcus|   Carson|    15|   9|  31|
|           Physics|30/12/1987|     Marta|   Brooks|    11|  10|  33|
|    Data Analytics| 12/6/1975|     Holly| Schwartz|    12|  11|  45|
|  Computer Science|  2/7/1985|     April|    Black|    11|  12|  35|
|  Computer Science| 22/7/1980|     Irene|  Bradley|    13|  13|  40|
|        Psychology|  7/2/1986|      Mark|    Weber|    12|  14|  34|
|       Informatics| 18/5/1987|     Rosie|   Norman|     9|  15|  33|
|          Business| 10/8/1984|    Martin|   Steele|     7|  16|  36|
```

Explanation

- changedateformat() function takes in date of births in any format and changes it to DD/MM/YYYY format.
- UserDefinedFunction library is used to process the dataframe using user written functions
- computeage() function extracts year of birth from the dob column and calculates the age of the person if the dob is given

In [1221]:

```
df.show()
```

executed in 663ms, finished 09:15:51 2020-07-06

```
+------------------+----------+----------+---------+------+----+----+
|            course|       dob|first_name|last_name|points|s_id| age|
+------------------+----------+----------+---------+------+----+----+
|Humanities and Art|14/10/1983|      Alan|      Joe|    10|   1|  37|
|  Computer Science| 26/9/1980|    Martin|  Genberg|    17|   2|  40|
|    Graphic Design| 12/6/1982|     Athur|   Watson|    16|   3|  38|
|    Graphic Design|  5/4/1987|  Anabelle|  Sanberg|    12|   4|  33|
|        Psychology| 1/11/1978|      Kira| Schommer|    11|   5|  42|
|          Business| 17/2/1981| Christian|   Kiriam|    10|   6|  39|
|  Machine Learning|  1/1/1984|   Barbara|  Ballard|    14|   7|  36|
|     Deep Learning| 13/1/1978|      John|       --|    10|   8|  42|
|  Machine Learning|26/12/1989|    Marcus|   Carson|    15|   9|  31|
|           Physics|30/12/1987|     Marta|   Brooks|    11|  10|  33|
|    Data Analytics| 12/6/1975|     Holly| Schwartz|    12|  11|  45|
|  Computer Science|  2/7/1985|     April|    Black|    11|  12|  35|
|  Computer Science| 22/7/1980|     Irene|  Bradley|    13|  13|  40|
|        Psychology|  7/2/1986|      Mark|    Weber|    12|  14|  34|
|       Informatics| 18/5/1987|     Rosie|   Norman|     9|  15|  33|
|          Business| 10/8/1984|    Martin|   Steele|     7|  16|  36|
|  Machine Learning|16/12/1990|     Colin| Martinez|     9|  17|  30|
|    Data Analytics|   unknown|   Bridget|    Twain|     6|  18|null|
|          Business|  7/3/1980|   Darlene|    Mills|    19|  19|  40|
|    Data Analytics|  2/6/1985|   Zachary|       --|    10|  20|  35|
+------------------+----------+----------+---------+------+----+----+
```

In [510]:

executed in 105ms, finished 17:14:46 2020-07-04

5.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

In [1222]:

```
ints > df_stats[0]["mean"]+df_stats[0]["std deviation"] ,20).otherwise(df.points))
```

executed in 673ms, finished 09:15:55 2020-07-06

```
+------------------+----------+----------+---------+------+----+----+
|            course|       dob|first_name|last_name|points|s_id| age|
+------------------+----------+----------+---------+------+----+----+
|Humanities and Art|14/10/1983|      Alan|      Joe|    10|   1|  37|
|  Computer Science| 26/9/1980|    Martin|  Genberg|    20|   2|  40|
|    Graphic Design| 12/6/1982|     Athur|   Watson|    20|   3|  38|
|    Graphic Design|  5/4/1987|  Anabelle|  Sanberg|    12|   4|  33|
|        Psychology| 1/11/1978|      Kira| Schommer|    11|   5|  42|
|          Business| 17/2/1981| Christian|   Kiriam|    10|   6|  39|
|  Machine Learning|  1/1/1984|   Barbara|  Ballard|    14|   7|  36|
|     Deep Learning| 13/1/1978|      John|       --|    10|   8|  42|
|  Machine Learning|26/12/1989|    Marcus|   Carson|    15|   9|  31|
|           Physics|30/12/1987|     Marta|   Brooks|    11|  10|  33|
|    Data Analytics| 12/6/1975|     Holly| Schwartz|    12|  11|  45|
|  Computer Science|  2/7/1985|     April|    Black|    11|  12|  35|
|  Computer Science| 22/7/1980|     Irene|  Bradley|    13|  13|  40|
|        Psychology|  7/2/1986|      Mark|    Weber|    12|  14|  34|
|       Informatics| 18/5/1987|     Rosie|   Norman|     9|  15|  33|
|          Business| 10/8/1984|    Martin|   Steele|     7|  16|  36|
|  Machine Learning|16/12/1990|     Colin| Martinez|     9|  17|  30|
|    Data Analytics|   unknown|   Bridget|    Twain|     6|  18|null|
|          Business|  7/3/1980|   Darlene|    Mills|    20|  19|  40|
|    Data Analytics|  2/6/1985|   Zachary|       --|    10|  20|  35|
+------------------+----------+----------+---------+------+----+----+
```

Explanation

- Calculate mean+standard deviation for each point and replace points with 20 if they have more value than the sum mentioned.

6. Create a histogram on the new points created in the task 5

In [1280]:

```
# Converting column points to list
points = df.select('points').collect()
print(type(points))
points = [int(i.points) for i in points]
print(points)
```

executed in 235ms, finished 12:10:30 2020-07-06

```
<class 'list'>
[10, 20, 20, 12, 11, 10, 14, 10, 15, 11, 12, 11, 13, 12, 9, 7, 9, 6, 20, 10]
```

In [ ]:

```
#plotting a histogram
from matplotlib import pyplot as plt

plt.hist(points)
plt.xlabel('points')
plt.ylabel('frequency')
plt.title('Ex 1. Part b) 6.')
plt.show()
```

executed in 157ms, finished 12:11:02 2020-07-06

In [ ]:

Ex 2: Manipulating Recommender Dataset with Apache Spark

Preprocessing

In [1232]:

```
#reading the dataset and preprocessing
tags = sqlContext.read.format("csv").option("delimiter",":").load('C:\\Users\\adm
tags = tags.drop('_c1').drop('_c3').drop('_c5')
```

executed in 133ms, finished 10:26:41 2020-07-06

In [1233]:

```
from pyspark.sql.window import Window
```

executed in 5ms, finished 10:26:43 2020-07-06

In [1234]:

```
#ordering by users and time
tags = tags.orderBy('_c0','_c6', ascending=True)

Windowspec = Window.orderBy('_c0')
#Windowspec = Windowspec.orderBy('_c6')
```

executed in 25ms, finished 10:26:45 2020-07-06

In [1235]:

```
#creating a new column with value of column '_c6' from t-1
previous_time = tags.withColumn('previous_time', F.lag(tags['_c6']).over(Windowsp
```

executed in 16ms, finished 10:26:48 2020-07-06

In [1236]:

```
previous_time.show()
```

executed in 9.58s, finished 10:26:59 2020-07-06

```
+-----+-----+-----------------+----------+-------------+
|  _c0|  _c2|              _c4|       _c6|previous_time|
+-----+-----+-----------------+----------+-------------+
| 1000|  277|children's story|1188533111|         null|
| 1000| 1994|    sci-fi. dark|1188533136|   1188533111|
| 1000| 5377|         romance|1188533150|   1188533136|
| 1000| 7147|    family bonds|1188533161|   1188533150|
| 1000|  362|animated classic|1188533171|   1188533161|
| 1000|  276|          family|1188533235|   1188533171|
|10003|42013|        Passable|1150432435|   1188533235|
|10003|51662|  FIOS on demand|1207953326|   1150432435|
|10003|54997|  FIOS on demand|1207953335|   1207953326|
|10003|55765|  FIOS on demand|1207953342|   1207953335|
|10003|55363|  FIOS on demand|1207953420|   1207953342|
|10003|56152|  FIOS on demand|1207953526|   1207953420|
|10003|55116|  FIOS on demand|1207953636|   1207953526|
|10003|56174|  FIOS on demand|1207953670|   1207953636|
|10003|55176|  FIOS on demand|1207953755|   1207953670|
|10003|55247|  FIOS on demand|1207953756|   1207953755|
|10003|54881|  FIOS on demand|1207953758|   1207953756|
|10003|55820|  FIOS on demand|1207953873|   1207953758|
|10003|53123|  FIOS on demand|1207953875|   1207953873|
|10003|53550|  FIOS on demand|1207953937|   1207953875|
+-----+-----+-----------------+----------+-------------+
only showing top 20 rows
```

In [1237]:

```
#calculating time difference between values in column _c6 at time t-1 and t
result = previous_time.withColumn('time_difference', (previous_time['_c6']-previo
```

executed in 10ms, finished 10:27:18 2020-07-06

In [1238]:

```
result.show()
```

executed in 7.82s, finished 10:27:27 2020-07-06

```
+-----+-----+----------------+----------+-------------+---------------+
|  _c0|  _c2|             _c4|       _c6|previous_time|time_difference|
+-----+-----+----------------+----------+-------------+---------------+
| 1000|  277|children's story|1188533111|         null|           null|
| 1000| 1994|    sci-fi. dark|1188533136|   1188533111|           25.0|
| 1000| 5377|         romance|1188533150|   1188533136|           14.0|
| 1000| 7147|    family bonds|1188533161|   1188533150|           11.0|
| 1000|  362|animated classic|1188533171|   1188533161|           10.0|
| 1000|  276|          family|1188533235|   1188533171|           64.0|
|10003|42013|        Passable|1150432435|   1188533235|     -3.81008E7|
|10003|51662|  FIOS on demand|1207953326|   1150432435|    5.7520891E7|
|10003|54997|  FIOS on demand|1207953335|   1207953326|            9.0|
|10003|55765|  FIOS on demand|1207953342|   1207953335|            7.0|
|10003|55363|  FIOS on demand|1207953420|   1207953342|           78.0|
|10003|56152|  FIOS on demand|1207953526|   1207953420|          106.0|
|10003|55116|  FIOS on demand|1207953636|   1207953526|          110.0|
|10003|56174|  FIOS on demand|1207953670|   1207953636|           34.0|
|10003|55176|  FIOS on demand|1207953755|   1207953670|           85.0|
|10003|55247|  FIOS on demand|1207953756|   1207953755|            1.0|
|10003|54881|  FIOS on demand|1207953758|   1207953756|            2.0|
|10003|55820|  FIOS on demand|1207953873|   1207953758|          115.0|
|10003|53123|  FIOS on demand|1207953875|   1207953873|            2.0|
|10003|53550|  FIOS on demand|1207953937|   1207953875|           62.0|
+-----+-----+----------------+----------+-------------+---------------+
only showing top 20 rows
```

In [1240]:

```
prev_time= 0
prev_user = 0
session = 0
#assigns session based on time difference
def assign_session(time, user):
    global prev_time
    global prev_user
    global session
    if(user!=prev_user):
        time = float("-inf")
    if(time== None or time > 1800 or user!=prev_user or time< -1800):
        session+=1
    prev_user = user
    return session
udfsession = F.udf(assign_session, StringType())

tags_session = result.withColumn("session",udfsession("time_difference",'_c0'))
```

executed in 35ms, finished 10:36:33 2020-07-06

1. Tagging session for each user

In [1241]:
```
tags_session.show()
```
executed in 9.58s, finished 10:36:45 2020-07-06

```
+-----+-----+---------------+----------+------------+---------------+-------+
|  _c0|  _c2|            _c4|       _c6|previous_time|time_difference|session|
+-----+-----+---------------+----------+------------+---------------+-------+
| 1000|  277|children's story|1188533111|        null|           null|      1|
| 1000| 1994|   sci-fi. dark|1188533136|  1188533111|           25.0|      1|
| 1000| 5377|        romance|1188533150|  1188533136|           14.0|      1|
| 1000| 7147|   family bonds|1188533161|  1188533150|           11.0|      1|
| 1000|  362|animated classic|1188533171|  1188533161|           10.0|      1|
| 1000|  276|         family|1188533235|  1188533171|           64.0|      1|
|10003|42013|       Passable|1150432435|  1188533235|     -3.81008E7|      2|
|10003|51662|  FIOS on demand|1207953326|  1150432435|    5.7520891E7|      3|
|10003|54997|  FIOS on demand|1207953335|  1207953326|            9.0|      3|
|10003|55765|  FIOS on demand|1207953342|  1207953335|            7.0|      3|
|10003|55363|  FIOS on demand|1207953420|  1207953342|           78.0|      3|
|10003|56152|  FIOS on demand|1207953526|  1207953420|          106.0|      3|
|10003|55116|  FIOS on demand|1207953636|  1207953526|          110.0|      3|
|10003|56174|  FIOS on demand|1207953670|  1207953636|           34.0|      3|
|10003|55176|  FIOS on demand|1207953755|  1207953670|           85.0|      3|
|10003|55247|  FIOS on demand|1207953756|  1207953755|            1.0|      3|
|10003|54881|  FIOS on demand|1207953758|  1207953756|            2.0|      3|
|10003|55820|  FIOS on demand|1207953873|  1207953758|          115.0|      3|
|10003|53123|  FIOS on demand|1207953875|  1207953873|            2.0|      3|
|10003|53550|  FIOS on demand|1207953937|  1207953875|           62.0|      3|
+-----+-----+---------------+----------+------------+---------------+-------+
only showing top 20 rows
```

Explanation

- Preprocess the dataset to have an additional column that dscribes the time difference between two tags of a user
- Assign a new session if the time difference is greater than 30 minutes = 1800 seconds.

2.Frequency of tagging for each user session

In [1242]:
```
freq_tag = tags_session.orderBy('session').groupBy('session').count()
```
executed in 21ms, finished 10:37:46 2020-07-06

```
In [1243]: freq_tag.show()
```

executed in 9.65s, finished 10:37:56 2020-07-06

```
+-------+-----+
|session|count|
+-------+-----+
|      1|    6|
|     10|    1|
|    100|    1|
|   1000|    6|
|  10000|    5|
|  10001|    8|
|  10002|    1|
|  10003|    1|
|  10004|    7|
|  10005|    3|
|  10006|    2|
|  10007|    2|
|  10008|    1|
|  10009|    1|
|   1001|    1|
|  10010|    1|
|  10011|    2|
|  10012|    1|
|  10013|    2|
|  10014|    1|
+-------+-----+
only showing top 20 rows
```

Explanation

- Perform groupBy on session attribute and count() such that the frequency of tagging within each user session is obtained.

3.Mean and standard deviation of the tagging frequency for each user

In [1255]:

```
per_user = tags_session.groupBy('_c0','session').count().orderBy('_c0','session')
per_user.show()
```

executed in 7.60s, finished 11:13:26 2020-07-06

```
+-----+-------+-----+
|  _c0|session|count|
+-----+-------+-----+
| 1000|      1|    6|
|10003|      2|    1|
|10003|      3|   18|
|10003|      4|   38|
|10020|      5|    2|
|10025|      6|    1|
|10032|     10|    1|
|10032|     11|    4|
|10032|     12|    1|
|10032|     13|    1|
|10032|     14|    4|
|10032|     15|    1|
|10032|     16|    1|
|10032|     17|    1|
|10032|     18|    1|
|10032|      7|   39|
|10032|      8|    1|
|10032|      9|    1|
|10051|     19|    1|
|10058|     20|   35|
+-----+-------+-----+
only showing top 20 rows
```

- The above table shows the number of tags performed for each session by each user.

In [1269]:

```
#mean of tagging frequency of each user
per_user_mean = per_user.groupBy('_c0').mean().orderBy('_c0')
per_user_mean.show()
```

executed in 2m 2s, finished 11:50:39 2020-07-06

```
+-----+------------------+
|  _c0|        avg(count)|
+-----+------------------+
| 1000|               6.0|
|10003|              19.0|
|10020|               2.0|
|10025|               1.0|
|10032| 4.666666666666667|
|10051|               1.0|
|10058|25.333333333333332|
|10059|               2.5|
|10064|               1.0|
|10084|              3.75|
|10094|               2.0|
| 1010|               4.0|
|10117|               1.5|
|10125|              21.0|
|10132|            1.5625|
|10154|               8.0|
|10167|               1.0|
| 1017|               7.0|
|10181|              11.0|
|10191|2.6666666666666665|
+-----+------------------+
only showing top 20 rows
```

Explanation

- This table calculates the mean of tagging frequency of each user by performing groupBy on user and then calculating the mean

In [1266]:
```
#std of tagging frequency of each user
per_user.groupBy('_c0').agg({'count':'std'}).show()
```
executed in 2m 26s, finished 11:35:05 2020-07-06

```
+-----+------------------+
|  _c0|     stddev(count)|
+-----+------------------+
|11563|               NaN|
| 1436|               NaN|
|17427|0.7071067811865476|
| 2136|               NaN|
|23318|               NaN|
|28473|               0.0|
| 2904|               NaN|
|29549|               NaN|
|32812|1.4142135623730951|
|38271|               NaN|
|39917|               NaN|
|42688|               NaN|
|44446|               NaN|
|48370|               0.0|
| 5325|               NaN|
|57051|               NaN|
|57112|               NaN|
```

Explanation

- This table calculates the std of tagging frequency of each user by performing groupBy on user and then calculating the mean

4. Mean and standard deviation of the tagging frequency across users

In [1270]:
```
freq_stats = freq_tag.select(
    _mean(col('count')).alias("mean"),
    _stddev(col('count')).alias("std deviation")
).collect()

freq_mean = (freq_stats[0]["mean"])
freq_std = (freq_stats[0]["std deviation"])
```
executed in 12.4s, finished 11:50:51 2020-07-06

In [1271]:
```
print(freq_mean, freq_std)
```
executed in 6ms, finished 11:51:12 2020-07-06

```
6.269596589045589 20.519115697996106
```

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

In [1277]:

```
m = per_user_mean.select('_c0').filter(per_user_mean['avg(count)']<= freq_mean+2*
```

executed in 225ms, finished 12:04:13 2020-07-06

In [1278]:

```
m.show()
```

executed in 2m 9s, finished 12:06:23 2020-07-06

```
+-----+
|  _c0|
+-----+
| 1000|
|10003|
|10020|
|10025|
|10032|
|10051|
|10058|
|10059|
|10064|
|10084|
|10094|
| 1010|
|10117|
|10125|
|10132|
|10154|
|10167|
| 1017|
|10181|
|10191|
+-----+
only showing top 20 rows
```

Explanation

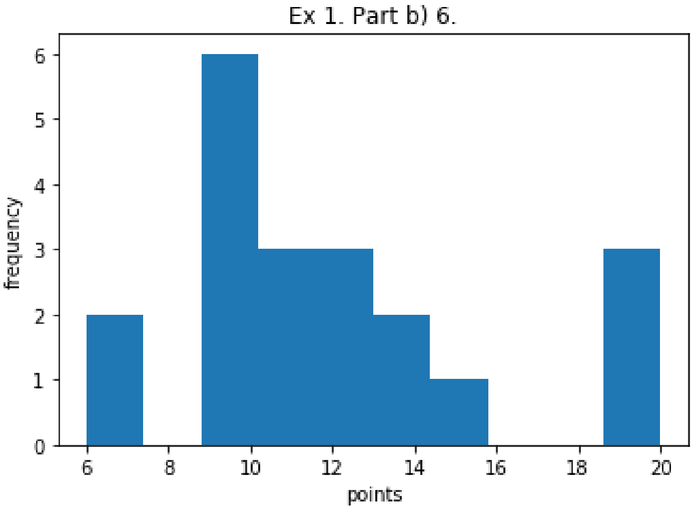
- Filtering users whose per user mean is within two stds from the mean across all users

Solution to Exercise sheet 8

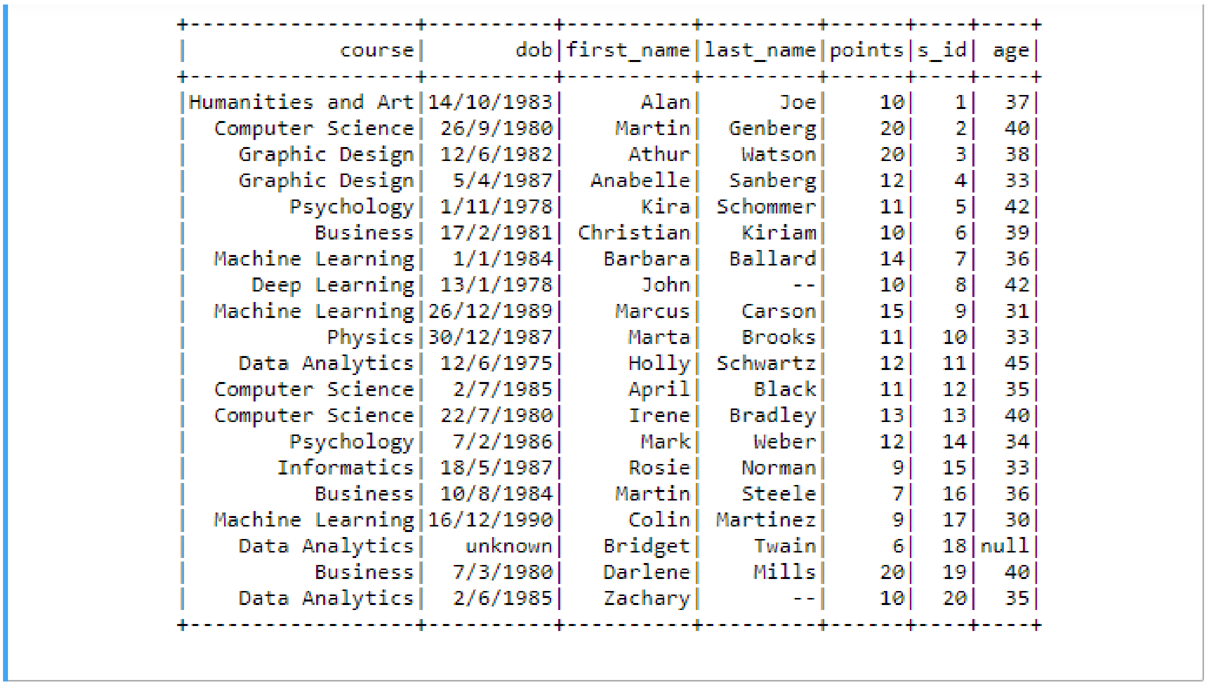
Exercise 1:

Part b)

6. Histogram



Snapshot of final dataset:



```
+------------------+----------+----------+---------+------+----+----+
|            course|       dob|first_name|last_name|points|s_id| age|
+------------------+----------+----------+---------+------+----+----+
|Humanities and Art|14/10/1983|      Alan|      Joe|    10|   1|  37|
|  Computer Science| 26/9/1980|    Martin|  Genberg|    20|   2|  40|
|    Graphic Design| 12/6/1982|     Athur|   Watson|    20|   3|  38|
|    Graphic Design|  5/4/1987|  Anabelle|  Sanberg|    12|   4|  33|
|        Psychology| 1/11/1978|      Kira| Schommer|    11|   5|  42|
|          Business| 17/2/1981| Christian|   Kiriam|    10|   6|  39|
|  Machine Learning|  1/1/1984|   Barbara|  Ballard|    14|   7|  36|
|     Deep Learning| 13/1/1978|      John|       --|    10|   8|  42|
|  Machine Learning|26/12/1989|    Marcus|   Carson|    15|   9|  31|
|           Physics|30/12/1987|     Marta|   Brooks|    11|  10|  33|
|    Data Analytics| 12/6/1975|     Holly| Schwartz|    12|  11|  45|
|  Computer Science|  2/7/1985|     April|    Black|    11|  12|  35|
|  Computer Science| 22/7/1980|     Irene|  Bradley|    13|  13|  40|
|        Psychology|  7/2/1986|      Mark|    Weber|    12|  14|  34|
|       Informatics| 18/5/1987|     Rosie|   Norman|     9|  15|  33|
|          Business| 10/8/1984|    Martin|   Steele|     7|  16|  36|
|  Machine Learning|16/12/1990|     Colin| Martinez|     9|  17|  30|
|    Data Analytics|   unknown|   Bridget|    Twain|     6|  18|null|
|          Business|  7/3/1980|   Darlene|    Mills|    20|  19|  40|
|    Data Analytics|  2/6/1985|   Zachary|       --|    10|  20|  35|
+------------------+----------+----------+---------+------+----+----+
```