DDA08_PySpark_Nabawi309498

July 1, 2022

1 Mansoor Nabawi, 309498

1.1 Exercise 1: Apache Spark Basics (10 points)

ref: https://towardsdatascience.com/pyspark-on-google-colab-101-d31830b238be

[280]:
```
#installing PySpark on Google Colab
!pip install pyspark
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-
wheels/public/simple/
Requirement already satisfied: pyspark in /usr/local/lib/python3.7/dist-packages
(3.3.0)
Requirement already satisfied: py4j==0.10.9.5 in /usr/local/lib/python3.7/dist-
packages (from pyspark) (0.10.9.5)
```

[281]:
```
#importing libraries
from pyspark import SparkConf, SparkContext
from pyspark.sql import *
from pyspark.sql import SQLContext
import pyspark.sql.types as sparktypes
from pyspark.sql.types import DateType
from pyspark.sql.types import IntegerType,FloatType
from pyspark.sql.functions import mean as mean_, stddev as␣
 ↪std_,col,asc,collect_set
from pyspark.sql.functions import to_date, date_format,udf
from pyspark.sql import functions as F
import datetime
from datetime import datetime , date
from functools import reduce
import json
import pandas as pd
from matplotlib import pyplot as plt
import numpy as np
np.set_printoptions(threshold=7)
import os
import io
```

[282]:
```
#configuration and naming the app
conf = SparkConf().setAppName("Spark_Basic")
sc = SparkContext.getOrCreate(conf=conf)
```

[283]:
```
# import SparkSession from pyspark.sql and create a SparkSession
spark = SparkSession.builder\
        .master("local")\
        .appName("Exercise01")\
        .config('spark.ui.port', '4050')\
        .getOrCreate()
```

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

[284]:
```
#lists provided in the exercise sheet
a = ["spark", "rdd", "python", "context", "create", "class"]
b = ["operation", "apache", "scala", "lambda","parallel","partition"]
```

[285]:
```
#key pair for each word, making tuple
a_ = [("spark",1), ("rdd",1), ("python",1), ("context",1), ("create",1),␣
 ↪("class",1)]
b_ = [("operation",1), ("apache",1), ("scala",1),␣
 ↪("lambda",1),("parallel",1),("partition",1)]
```

[286]:
```
#making dataframe in spark, words and key as their columns
df_a = spark.createDataFrame(a_,['words',"key"])
df_b= spark.createDataFrame(b_,['words',"key"])
#The alias, allows you to distinguish where each column is coming from.
dfa = df_a.alias('dfa')
dfb = df_b.alias('dfb')
dfa.show()
dfb.show()
```

```
+-------+---+
|  words|key|
+-------+---+
|  spark|  1|
|    rdd|  1|
| python|  1|
|context|  1|
| create|  1|
|  class|  1|
+-------+---+

+---------+---+
|    words|key|
+---------+---+
```

```
|operation|  1|
|   apache|  1|
|    scala|  1|
|   lambda|  1|
| parallel|  1|
|partition|  1|
+---------+---+
```

[287]:
```
#creating a prallelized collection
# Convert list to RDDs
a = sc.parallelize(a)
b = sc.parallelize(b)

a_ = sc.parallelize(a_)
b_ = sc.parallelize(b_)
```

[288]:
```
print("a_:", a_.collect())
print("b_:", b_.collect())
```

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

1. Perform rightOuterJoin and fullOuterJoin operations between a and b. Briefly explain your solution. (1 point) We convert the list to tuple and a 1 as key is added for each word.

Since a and b or a_ and b_ have no common word, the list after right join get value (None , 1).

rightouterjoin ref: https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.RDD.rightOuterJ

For each element (k, w) in other, the resulting RDD will either contain all pairs (k, (v, w)) for v in this, or the pair (k, (None, w)) if no elements in self have key k.

Hash-partitions the resulting RDD into the given number of partitions.

[289]:
```
rOJoin_a = a_.rightOuterJoin(b_)
print(sorted(rOJoin_a.collect()))
```

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

[290]:
```
rOJoin_a = b_.rightOuterJoin(a_) #right joins B to A
print(sorted(rOJoin_a.collect()))
```

```
[('class', (None, 1)), ('context', (None, 1)), ('create', (None, 1)), ('python',
(None, 1)), ('rdd', (None, 1)), ('spark', (None, 1))]
```

fullouterjoin ref: https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.RDD.fullOuterJoin.

For each element (k, v) in self, the resulting RDD will either contain all pairs (k, (v, w)) for w in other, or the pair (k, (v, None)) if no elements in other have key k.

Similarly, for each element (k, w) in other, the resulting RDD will either contain all pairs (k, (v, w)) for v in self, or the pair (k, (None, w)) if no elements in self have key k.

[291]:
```
full_outer_join_a = b_.fullOuterJoin(a_) #full joins B to A
print(full_outer_join_a.take(20))
```

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

[292]:
```
full_outer_join_b = a_.fullOuterJoin(b_)
print(full_outer_join_a.take(20))
```

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

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

[293]:
```
def reducer(map1,map2):
  for key in map2:
    map1[key] = map1.get(key,0) + map2.get(key,0)
  return map1

def letter_count(list_a, list_b,letter):
  #all words combined into one big string
  all_string = "".join(list_a.collect()+ list_b.collect())
  frequency = reduce(reducer, map(lambda string: dict([[string,1]]),all_string))
  return frequency[letter]
```

[294]:
```
letter = "s"
print(f"Letter '{letter}' frequency is:",letter_count(a,b,letter))
```

```
Letter 's' frequency is: 4
```

3. Using aggregate function to count how many times the character ”s” appears in all a and b. (1 point) https://sparkbyexamples.com/apache-spark-rdd/spark-rdd-aggregate-example/

[295]:
```
def aggregator_func(list_a,list_b,l):
  #all words combined into one big string
  all_list =[(list_a.collect()+ list_b.collect())]
  RDD = sc.parallelize(all_list)
  return RDD.aggregate(0 , lambda x, y: x+y.count(l), lambda x, y: x+y)
```

[296]:
```
letter = "s"
print(f"Letter '{letter}' appears",aggregator_func(a,b,letter), "times in all a␣
 ↪and b with aggregate function")
```

```
Letter 's' appears 0 times in all a and b with aggregate function
```

1.1.2 Part b) Basic Operations on DataFrames (6 points)

[297]:
```
#converting to RDD and then reading the file
students = sc.textFile('students.json')
students_df = spark.read.json(students)
students_df.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|  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)

[298]:
```
#using aggregate to the mean of points
means = students_df.agg({'points': 'mean'}).collect()[0][0]
print(np.round(means))
students_df_1 = students_df.fillna(np.round(means),('points'))
```

```
12.0
```

[299]:
```
students_df_1.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|    12|  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|
+-----------------+-----------------+----------+---------+------+----+
```

Replace the null value(s) in column dob and column last name by ”unknown” and ”--” respectively. (0.5 point)

[300]:
```
#using fillna to solve this problem
students_df_2 = students_df_1.fillna('unknown',('dob'))
students_df_2 = students_df_2.fillna('--',('last_name'))
```

[301]:
```
students_df_2.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|    12|  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|
+------------------+------------------+----------+---------+------+----+
```

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)

[302]:
```
#this function check if there is an input as our date, then convert it into the␣
↪desired format, otherwise "unknown"
def date_convert_func(input_date):
  if(len(input_date.split(' '))>1):
    if(input_date.split(' ')[0].isdigit()):
      #for example if it is like 7 March 1980
      final_date = datetime.strptime(input_date,'%d %B %Y').date().
↪strftime('%d-%m-%Y')
      return final_date
    else:
      #else it could be like this June 2, 1985
      final_date = datetime.strptime(input_date,'%B %d, %Y').date().
↪strftime('%d-%m-%Y')
      return final_date
  else:
    return "unknown"
```

[303]:
```
#creating a user defined function
dt_converter_udf = udf(date_convert_func, sparktypes.StringType())
```

```
students_df_3 = students_df_2.withColumn('dob',␣
 ↪dt_converter_udf(students_df_2['dob']))
students_df_3.show()
```

```
+------------------+----------+----------+---------+------+----+
|            course|       dob|first_name|last_name|points|s_id|
+------------------+----------+----------+---------+------+----+
|Humanities and Art|14-10-1983|      Alan|      Joe|    10|   1|
|  Computer Science|26-09-1980|    Martin|  Genberg|    17|   2|
|    Graphic Design|12-06-1982|     Athur|   Watson|    16|   3|
|    Graphic Design|05-04-1987|  Anabelle|  Sanberg|    12|   4|
|        Psychology|01-11-1978|      Kira| Schommer|    11|   5|
|          Business|17-02-1981| Christian|   Kiriam|    10|   6|
|  Machine Learning|01-01-1984|   Barbara|  Ballard|    14|   7|
|     Deep Learning|13-01-1978|      John|       --|    10|   8|
|  Machine Learning|26-12-1989|    Marcus|   Carson|    15|   9|
|           Physics|30-12-1987|     Marta|   Brooks|    11|  10|
|    Data Analytics|12-06-1975|     Holly| Schwartz|    12|  11|
|  Computer Science|02-07-1985|     April|    Black|    12|  12|
|  Computer Science|22-07-1980|     Irene|  Bradley|    13|  13|
|        Psychology|07-02-1986|      Mark|    Weber|    12|  14|
|       Informatics|18-05-1987|     Rosie|   Norman|     9|  15|
|          Business|10-08-1984|    Martin|   Steele|     7|  16|
|  Machine Learning|16-12-1990|     Colin| Martinez|     9|  17|
|    Data Analytics|   unknown|   Bridget|    Twain|     6|  18|
|          Business|07-03-1980|   Darlene|    Mills|    19|  19|
|    Data Analytics|02-06-1985|   Zachary|       --|    10|  20|
+------------------+----------+----------+---------+------+----+
```

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

[304]:
```
#this function calculates age based on toda's date, if there is no date then␣
 ↪put "-"
def age_calc_func(dob):
  if len(dob.split('-'))>1:
    #taking today's date and calculating based on today's date
    current_time = date.today()
    dob = datetime.strptime(dob, "%d-%m-%Y")
    age = current_time.year - dob.year + ((current_time.month,current_time.
 ↪day)>(dob.month, dob.day))
    return age
  else:
    return "-"
```

[305]:
```
age_calc_udf = udf(age_calc_func, sparktypes.StringType())
```

```
students_age = students_df_3.withColumn('age',␣
 ↪age_calc_udf(students_df_3['dob']))
students_age.show()
```

```
+------------------+----------+----------+---------+------+----+---+
|            course|       dob|first_name|last_name|points|s_id|age|
+------------------+----------+----------+---------+------+----+---+
|Humanities and Art|14-10-1983|      Alan|      Joe|    10|   1| 39|
|  Computer Science|26-09-1980|    Martin|  Genberg|    17|   2| 42|
|    Graphic Design|12-06-1982|     Athur|   Watson|    16|   3| 41|
|    Graphic Design|05-04-1987|  Anabelle|  Sanberg|    12|   4| 36|
|        Psychology|01-11-1978|      Kira| Schommer|    11|   5| 44|
|          Business|17-02-1981| Christian|   Kiriam|    10|   6| 42|
|  Machine Learning|01-01-1984|   Barbara|  Ballard|    14|   7| 39|
|     Deep Learning|13-01-1978|      John|       --|    10|   8| 45|
|  Machine Learning|26-12-1989|    Marcus|   Carson|    15|   9| 33|
|           Physics|30-12-1987|     Marta|   Brooks|    11|  10| 35|
|    Data Analytics|12-06-1975|     Holly| Schwartz|    12|  11| 48|
|  Computer Science|02-07-1985|     April|    Black|    12|  12| 37|
|  Computer Science|22-07-1980|     Irene|  Bradley|    13|  13| 42|
|        Psychology|07-02-1986|      Mark|    Weber|    12|  14| 37|
|       Informatics|18-05-1987|     Rosie|   Norman|     9|  15| 36|
|          Business|10-08-1984|    Martin|   Steele|     7|  16| 38|
|  Machine Learning|16-12-1990|     Colin| Martinez|     9|  17| 32|
|    Data Analytics|   unknown|   Bridget|    Twain|     6|  18|  -|
|          Business|07-03-1980|   Darlene|    Mills|    19|  19| 43|
|    Data Analytics|02-06-1985|   Zachary|       --|    10|  20| 38|
+------------------+----------+----------+---------+------+----+---+
```

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)

[306]:
```
def points_granting(points):
  if points > (ave_points + std_points):
    return int(20)
  else:
    return points
```

[307]:
```
#average points of students
ave_points = students_age.select(mean_(students_age["points"])).collect()[0][0]
print("Mean of points of students:",ave_points)
#std dev points of students
std_points = students_age.select(std_(students_age["points"])).collect()[0][0]
print("Standard deviation of points of students_age:",(std_points))
```

```
points_granter_udf = udf(points_granting, sparktypes.StringType())
students_points = students_age.withColumn("points",␣
 ↪points_granter_udf(students_age["points"]))
students_points.show()
```

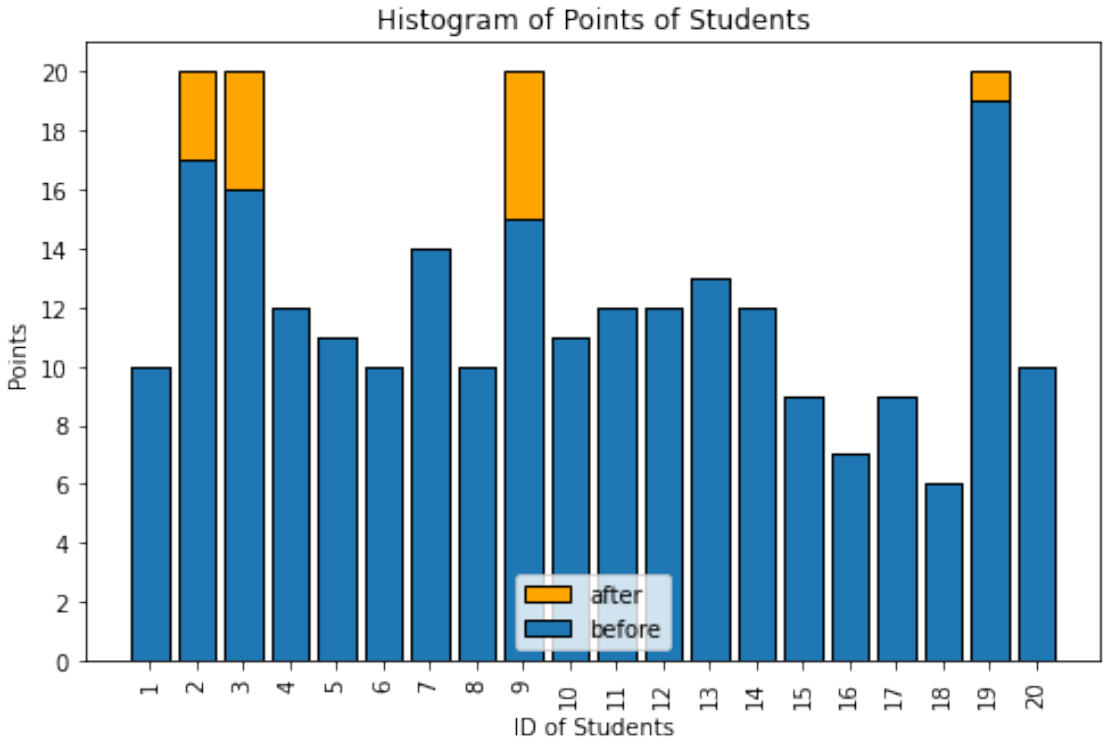
```
Mean of points of students: 11.75
Standard deviation of points of students_age: 3.242400020777332
+------------------+----------+----------+---------+------+----+---+
|            course|       dob|first_name|last_name|points|s_id|age|
+------------------+----------+----------+---------+------+----+---+
|Humanities and Art|14-10-1983|      Alan|      Joe|    10|   1| 39|
|  Computer Science|26-09-1980|    Martin|  Genberg|    20|   2| 42|
|    Graphic Design|12-06-1982|     Athur|   Watson|    20|   3| 41|
|    Graphic Design|05-04-1987|  Anabelle|  Sanberg|    12|   4| 36|
|        Psychology|01-11-1978|      Kira| Schommer|    11|   5| 44|
|          Business|17-02-1981| Christian|   Kiriam|    10|   6| 42|
|  Machine Learning|01-01-1984|   Barbara|  Ballard|    14|   7| 39|
|     Deep Learning|13-01-1978|      John|       --|    10|   8| 45|
|  Machine Learning|26-12-1989|    Marcus|   Carson|    20|   9| 33|
|           Physics|30-12-1987|     Marta|   Brooks|    11|  10| 35|
|    Data Analytics|12-06-1975|     Holly| Schwartz|    12|  11| 48|
|  Computer Science|02-07-1985|     April|    Black|    12|  12| 37|
|  Computer Science|22-07-1980|     Irene|  Bradley|    13|  13| 42|
|        Psychology|07-02-1986|      Mark|    Weber|    12|  14| 37|
|       Informatics|18-05-1987|     Rosie|   Norman|     9|  15| 36|
|          Business|10-08-1984|    Martin|   Steele|     7|  16| 38|
|  Machine Learning|16-12-1990|     Colin| Martinez|     9|  17| 32|
|    Data Analytics|   unknown|   Bridget|    Twain|     6|  18|  -|
|          Business|07-03-1980|   Darlene|    Mills|    20|  19| 43|
|    Data Analytics|02-06-1985|   Zachary|       --|    10|  20| 38|
+------------------+----------+----------+---------+------+----+---+
```

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

[308]:
```
#seing the result after and before adding one point
df = students_points.select("s_id" , "points").toPandas()
df_old = students_age.select("s_id" , "points").toPandas()
df["points"] = pd.to_numeric(df["points"])
df_old["points"] = pd.to_numeric(df_old["points"])
fig, ax = plt.subplots(figsize=(8, 5))

plt.bar(df.s_id,df["points"], label = 'after', color = 'orange',edgecolor =␣
 ↪'black')
plt.bar(df_old.s_id,df_old["points"], label = 'before', edgecolor = 'black')
plt.xticks(range(1,len(df)+1), df.s_id,rotation=90)
plt.yticks(np.arange(0, df.shape[0]+1, step=2))
```

```
plt.title('Histogram of Points of Students')
plt.xlabel('ID of Students')
plt.ylabel('Points')
plt.legend(loc = 'lower center')
plt.show()
```



2 Exercise 2: Manipulating Recommender Dataset with Apache Spark (10 points)

[309]:
```
#downloading the dataset
!wget https://files.grouplens.org/datasets/movielens/ml-10m.zip
```

```
--2022-07-01 22:44:55--
https://files.grouplens.org/datasets/movielens/ml-10m.zip
Resolving files.grouplens.org (files.grouplens.org)… 128.101.65.152
Connecting to files.grouplens.org (files.grouplens.org)|128.101.65.152|:443…
connected.
HTTP request sent, awaiting response… 200 OK
Length: 65566137 (63M) [application/zip]
Saving to: ‘ml-10m.zip.2’
```

```
ml-10m.zip.2        100%[===================>]  62.53M  58.6MB/s    in 1.1s

2022-07-01 22:44:56 (58.6 MB/s) - 'ml-10m.zip.2' saved [65566137/65566137]
```

[310]:
```
#extracting zip file of dataset
!unzip ml-10m.zip
```

```
Archive:  ml-10m.zip
replace ml-10M100K/allbut.pl? [y]es, [n]o, [A]ll, [N]one, [r]ename: nA
replace ml-10M100K/movies.dat? [y]es, [n]o, [A]ll, [N]one, [r]ename: A
  inflating: ml-10M100K/movies.dat
  inflating: ml-10M100K/ratings.dat
  inflating: ml-10M100K/README.html
  inflating: ml-10M100K/split_ratings.sh
  inflating: ml-10M100K/tags.dat
```

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.

[311]:
```
#import libraries
from pyspark import SparkContext
from pyspark.sql import SQLContext
import pandas as pd
from pyspark.sql import Row
import numpy as np
from pyspark.sql.functions import *
from pyspark.sql.types import IntegerType
from pyspark.sql import Window
from pyspark.sql.functions import mean as avg,stddev as stdd
```

[312]:
```
#configuring again with another app name, optional if we need to have different␣
↪appnames
conf = SparkConf().setAppName("Spark_Movie")
sc = SparkContext.getOrCreate(conf=conf)
```

[313]:
```
spark = SparkSession.builder\
        .master("local")\
        .appName("Exercise02")\
        .config('spark.ui.port', '4050')\
        .getOrCreate()
```

[314]:
```
#address of tags.dat
tags_file = "/content/ml-10M100K/tags.dat"
```

```
tags = spark.read.option("delimiter",":").csv(tags_file)
#changing column names
tags = tags.selectExpr("_c0 as UserID","_c2 as MovieID","_c4 as Tag", "_c6 as␣
 ↪Timestamp")
#casting data types
tags = tags.withColumn("UserID", tags["UserID"].cast(IntegerType()))
tags = tags.withColumn("MovieID",tags["MovieID"].cast(IntegerType()))

# Remove duplicate records
tags = tags.distinct()

tags.show(10)
tags.printSchema()
tags.count()
```

```
+------+-------+----------------+----------+
|UserID|MovieID|             Tag| Timestamp|
+------+-------+----------------+----------+
|    78|   4223|     want to own|1176691425|
|   146|    899| black and white|1204202551|
|   146|   1281| black and white|1210473492|
|   146|   1307|            jazz|1196824410|
|   146|   6979|        military|1209983708|
|   146|   7402|      H.G. Wells|1196562263|
|   146|  26554|Post apocalyptic|1204431274|
|   146|  27822|           shark|1195618767|
|   146|  31702|            Iraq|1222145876|
|   146|  43934|          nature|1228356185|
+------+-------+----------------+----------+
only showing top 10 rows

root
 |-- UserID: integer (nullable = true)
 |-- MovieID: integer (nullable = true)
 |-- Tag: string (nullable = true)
 |-- Timestamp: string (nullable = true)
```

[314]: 95376

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.

[315]:
```
#partitioning per use and ordering by timestamp
ts_w = Window.partitionBy("UserID").orderBy(asc("Timestamp"))
```

```
#making a new column lag and using lag window function, it will return the␣
 ↪previous row at any given point in the window partition.
df = tags.withColumn('lag',lag(tags.Timestamp).over(ts_w))
#if difference of the two times is less than 3- minute, output 1, otherwise 0
df = df.withColumn('difference',when((df.Timestamp - df.lag)/60 < 30,1)
                   .otherwise(0))
#session is just the sum of difference
df = df.withColumn('session',sum('difference').over(ts_w))

print("tagging sessions for each user: \n")
df.show()
```

```
tagging sessions for each user:

+------+-------+--------------------+----------+----------+----------+-------+
|UserID|MovieID|                 Tag| Timestamp|       lag|difference|session|
+------+-------+--------------------+----------+----------+----------+-------+
|    15|   4973|          excellent!|1215184630|      null|         0|      0|
|    20|   2947|              action|1188263755|      null|         0|      0|
|    20|   2947|                bond|1188263756|1188263755|         1|      1|
|    20|   7438|           Tarantino|1188263801|1188263756|         1|      4|
|    20|   7438|             kung fu|1188263801|1188263801|         1|      4|
|    20|   7438|              bloody|1188263801|1188263801|         1|      4|
|    20|   2424|               hanks|1188263835|1188263801|         1|      7|
|    20|   2424|                ryan|1188263835|1188263835|         1|      7|
|    20|   2424|     chick flick 212|1188263835|1188263835|         1|      7|
|    20|   1747|            politics|1188263867|1188263835|         1|      9|
|    20|   1747|              satire|1188263867|1188263867|         1|      9|
|    20|   3033|           star wars|1188263880|1188263867|         1|     11|
|    20|   3033|               spoof|1188263880|1188263880|         1|     11|
|    31|   6373|   comedy of manners|1188263644|      null|         0|      0|
|    31|    546|strangely compelling|1188263674|1188263644|         1|      1|
|    31|   2116|                Epic|1188263707|1188263674|         1|      2|
|    31|   1091|         catastrophe|1188263741|1188263707|         1|      3|
|    31|     65|        buddy comedy|1188263759|1188263741|         1|      4|
|    48|  54775|The Director Shou…|1215135517|      null|         0|      0|
|    48|  54290|Why the terrorist…|1215135611|1215135517|         1|      1|
+------+-------+--------------------+----------+----------+----------+-------+
only showing top 20 rows
```

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

[316]:
```
#all the previous action, just at the end we need to groupby UserID and sum the␣
 ↪differences or get the max of the session
df_ts = df.withColumn('lag',lag(df.Timestamp).over(ts_w))
```

```
df_ts = df_ts.withColumn('difference',when((df_ts.Timestamp - df_ts.lag) >␣
 ↪30*60,1).otherwise(0))

#sum of the differences give us total the user sessions per user
max_tag_sess = df_ts.groupBy("UserID").sum("difference")
max_tag_sess = max_tag_sess.orderBy('sum(difference)',ascending=False)
max_tag_sess.show()
```

```
+------+---------------+
|UserID|sum(difference)|
+------+---------------+
| 10555|            884|
| 23172|            476|
|   146|            332|
| 33384|            243|
| 47448|            198|
| 34745|            143|
| 11898|            126|
| 30167|            114|
| 64633|            107|
|  8041|            103|
| 41838|             99|
|  6362|             94|
| 23388|             84|
| 18015|             77|
| 23032|             72|
| 49882|             72|
| 59092|             71|
| 50970|             70|
|  2643|             68|
| 32828|             64|
+------+---------------+
only showing top 20 rows
```

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

[317]:
```
from pyspark.sql.functions import stddev, mean as mean_, count as count_
#groupby UserID and use aggregate function and use mean and stddev to get mean␣
 ↪and standard deviation for each user
df_tagfreq =  df_ts.groupby('UserID').agg(mean_('session').
 ↪alias('mean_session'),\
                                          stddev('session').alias('std_dev')).
 ↪sort('mean_session', ascending=False)

df_tagfreq.show(20)
```

```
+------+-----------------+-----------------+
|UserID|      mean_session|           std_dev|
+------+-----------------+-----------------+
| 23388| 2867.051970400964| 1651.012977443729|
| 10555|2432.3075515635396|1512.0921546437862|
| 23172|1522.9335877862595| 985.9479275152155|
| 52723| 864.7927876359473| 495.5622161484416|
|  9316| 752.3050624589087| 433.1017554338483|
|   146| 695.6559466019418| 391.0253393088701|
| 48717|  668.591642228739| 387.4230319458489|
| 51372| 655.3662182361734| 381.8550933350988|
| 70974| 630.5417956656347| 367.5827151449084|
| 33384| 607.5941457922883| 651.5650408560288|
| 34745| 580.5838607594936|323.90038485893353|
| 66129| 545.2064056939502|310.72778171761195|
| 63604| 522.2611163670766|301.65261968993065|
| 30167| 440.9577006507592| 242.9239711411123|
| 11898|431.30121703853956|245.31714796322677|
| 59092|404.75119047619046| 226.1236358444124|
| 11613| 375.7836107554417|214.17415759687316|
| 59816|354.94266666666664|201.16100445291477|
| 23032|326.20845481049565|185.18407961936205|
| 47448|319.38048780487804|171.41833557894412|
+------+-----------------+-----------------+
only showing top 20 rows
```

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

[318]:
```
#to get mean ans stddev across users, we just need to get the avg/stddev of␣
 ↪session for all users
mean = df_ts.agg(avg("session")).collect()[0]['avg(session)']
std = df_ts.agg(stdd("session")).collect()[0]['stddev_samp(session)']
print("mean and standard deviation of the tagging frequency for across users:␣
 ↪\n")
print("Mean: ",mean)
print("Standard Deviation: ",std)
```

```
mean and standard deviation of the tagging frequency for across users:

Mean:  571.9564880053682
Standard Deviation:  1060.324323980841
```

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

[319]:
```
#using previous data, just a simple filter to limit the data  within the two␣
 ↪standard deviation from the mean frequency of all users.
df_tagfreq.filter((col('mean_session') >= (mean - (2*std)))  &
              (col('std_dev') <= (mean + (2*std)))).distinct().show(20)
```

```
+------+------------------+------------------+
|UserID|      mean_session|           std_dev|
+------+------------------+------------------+
|    20| 6.166666666666667| 3.613946051147701|
|    21|               0.5|0.7071067811865476|
|    25|               0.0|               0.0|
|    31|               2.0|1.5811388300841898|
|    39|               2.0|1.5811388300841898|
|    48|               0.5|0.7071067811865476|
|    49|               7.2| 4.507137197189111|
|   109|              9.64| 4.923413450036468|
|   127|13.576923076923077| 7.360288455885827|
|   133|               2.4|1.3416407864998738|
|   146| 695.6559466019418| 391.0253393088701|
|   147|               0.5|0.7071067811865476|
|   175|               0.0|               0.0|
|   181|               3.0|               0.0|
|   190| 10.73076923076923| 6.988892285950284|
|   233|               1.0|1.2649110640673518|
|   283|0.6666666666666666|0.5773502691896258|
|   284|              14.8|  8.72333375058744|
|   299|               2.6|0.8944271909999159|
|   325|               1.9|1.5238839267549946|
+------+------------------+------------------+
only showing top 20 rows
```

[ ]:
```
!wget -nc https://raw.githubusercontent.com/brpy/colab-pdf/master/colab_pdf.py
from colab_pdf import colab_pdf
colab_pdf('DDA08_PySpark_Nabawi309498.ipynb')
```

```
File ‘colab_pdf.py’ already there; not retrieving.


WARNING: apt does not have a stable CLI interface. Use with caution in scripts.
```