Spark Project - 1

First Steps with Spark and PySpark

In this project, we will work with a dataset containing the names of all the guests who have appeared on The Daily Show hosted by Jon Stewart.

The details of all the headers is as follows:

Header	Definition
`YEAR`	The year the episode aired
`GoogleKnowlege_Occupation`	Their occupation or office, according to Google's Knowledge Graph or, if they're not in there, how Stewart introduced them on the program.
`Show`	Air date of episode. Not unique, as some shows had more than one guest
`Group`	A larger group designation for the occupation. For instance, us senators, us presidents, and former presidents are all under "politicians"
`Raw_Guest_List`	The person or list of people who appeared on the show, according to Wikipedia. The GoogleKnowlege_Occupation only refers to one of them in a given row.

We will try to get some valuable insights from the data and more importantly, learn how to work with PySpark.

1. To start off, we'll load the data set into an RDD. We're using the TSV version of the original data. TSV files use a tab character ("\t") as the delimiter, instead of the comma (",") that CSV files use.

```
raw_data = sc.textFile("daily_show.tsv")
raw_data.take(5)
```

The output that we will get is something like this:

```
Out[1]:
['YEAR\tGoogleKnowlege_Occupation\tShow\tGroup\tRaw_Guest_List',
 '1999\tactor\t1/11/99\tActing\tMichael J. Fox',
 '1999\tComedian\t1/12/99\tComedy\tSandra Bernhard',
 '1999\ttelevision actress\t1/13/99\tActing\tTracey Ullman',
 '1999\tfilm actress\t1/14/99\tActing\tGillian Anderson']
```

We automatically have access to the SparkContext object sc. The RDD object raw_data closely resembles a list of string objects, with one object for each line in the data set. We use the take() method to print the first five elements of the RDD.

2. We will then use the map() function to iterate on each line of the entire RDD, use a lambda function to split each line using the tab delimiter (\t) and assigne the result to the resulting RDD ‘daily show’. We call map a transformation function.

```
daily_show = raw_data.map(lambda line: line.split('\t'))
daily_show.take(5)
```

The output will be something like this:

```
Out[1]:
[['YEAR', 'GoogleKnowlege_Occupation', 'Show', 'Group', 'Raw_Guest_List'],
 ['1999', 'actor', '1/11/99', 'Acting', 'Michael J. Fox'],
 ['1999', 'Comedian', '1/12/99', 'Comedy', 'Sandra Bernhard'],
 ['1999', 'television actress', '1/13/99', 'Acting', 'Tracey Ullman'],
 ['1999', 'film actress', '1/14/99', 'Acting', 'Gillian Anderson']]
```

3. We'd like to tally up the number of guests who have appeared on The Daily Show during each year. To achieve this, we will use the Map step then a ReduceByKey step.

```
tally = daily_show.map(lambda x: (x[0], 1)).reduceByKey(lambda x,y: x+y)
print(tally)
```

And the output will be:

```
PythonRDD[21] at RDD at PythonRDD.scala:43
```

Printing tally didn't return the histogram we were hoping for. Because of lazy evaluation, PySpark delayed executing the map and reduceByKey steps until we actually need them.

4. To see the results, we'll use the take command, which forces lazy code to run immediately. Because tally is an RDD, we can't use Python's len function to find out how many elements are in the collection. Instead, we'll need to use the RDD count() function.

```
tally.take(tally.count())
```

With the output being:

```
Out[1]:
[('YEAR', 1),
 ('2013', 166),
 ('2001', 157),
 ('2004', 164),
 ('2000', 169),
 ('2015', 100),
 ('2010', 165),
 ('2006', 161),
 ('2014', 163),
 ('2003', 166),
 ('2002', 159),
 ('2011', 163),
 ('2012', 164),
 ('2008', 164),
 ('2007', 141),
 ('2005', 162),
 ('1999', 166),
 ('2009', 163)]
```

5. Spark knows nothing about column headers, and didn't set them aside. We need a way to remove the element ('YEAR', 1) from our collection. We'll need a workaround, though, because RDD objects are immutable once we create them. The only way to remove that tuple is to create a new RDD object that doesn't have it.

Spark comes with a filter(f) function that creates a new RDD by filtering an existing one for specific criteria. If we specify a function f that returns a binary value, True or False, the resulting RDD will consist of elements where the function evaluated to True.

```
def filter_year(line):
    # Write your logic here
    if line[0] != "YEAR":
        return True
    else:
        return False
    return True

filtered_daily_show = daily_show.filter(lambda line: filter_year(line))
```

6. We'll filter out actors for whom the profession is blank, lowercase each profession, generate a histogram of professions, and output the first five tuples in the histogram. For this we will chain together a series of data transformations into a pipeline, and observe Spark managing everything in the background.

```
filtered_daily_show.filter(lambda line: line[1] != '') \
                   .map(lambda line: (line[1].lower(), 1)) \
                   .reduceByKey(lambda x,y: x+y) \
                   .take(5)
```

```
Out[1]:
[('radio personality', 3),
 ('television writer', 1),
 ('american political figure', 2),
 ('former united states secretary of state', 6),
 ('mathematician', 1)]
```

In this project, we learnt the basics of operating Spark and PySpark with an introduction to RDDs along with transformation and action functions.