8 Travaux pratiques

Durant cette semaine, nous continuerons notre étude de les applications Mapreduce en réalisant 3 projets de différents sujets. Dans la plupart du temps, vous devez compléter un morceau de code et ensuite l'appliquer sur un jeu de données. (Copiez le code dans JupyterNote book et complétez). Vous pouvez vérifier si votre code est correctement implémenté à l'aide des expected outputs.

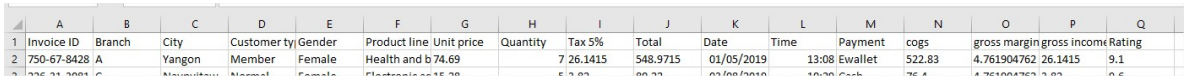
8.1 TP 01: Les ventes d'un supermarché

La croissance des supermarchés dans la plupart des villes peuplées est en augmentation et la concurrence sur le marché est également élevée. L'ensemble de données est l'une des ventes historiques de l'entreprise de supermarché qui a enregistré dans 3 succursales différentes pendant 3 mois des données.

8.1.1 Le fichier d'import: supermarket_sales_Module7.csv

Le fichier est sous la forme suivant:

Invoice ID	Branch	City	Customer type	Gender	Product line	Unit price	Quantity	Tax 5%	Total	Date	Time	Payment	cogs	gross margin percentage	gross income	Rating

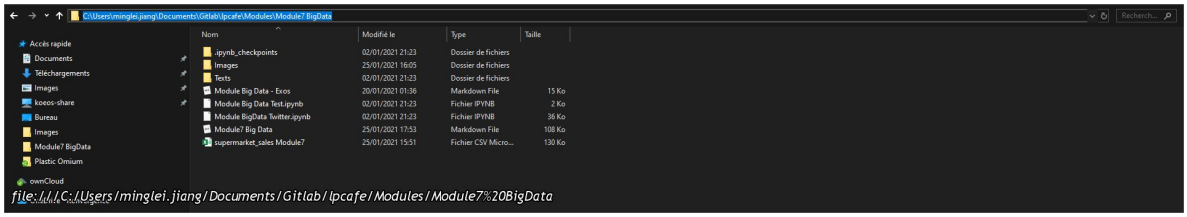


	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
1	Invoice ID	Branch	City	Customer ty	Gender	Product line	Unit price	Quantity	Tax 5%	Total	Date	Time	Payment	cogs	gross margin	gross income	Rating
2	750-67-8428	A	Yangon	Member	Female	Health and b	74.69	7	26.1415	548.9715	01/05/2019	13:08	Ewallet	522.83	4.761904762	26.1415	9.1

> L'objective de ce TP est donc de calculer le total des ventes par villes.

8.1.2 Copier des fichiers sur un conteneur Docker

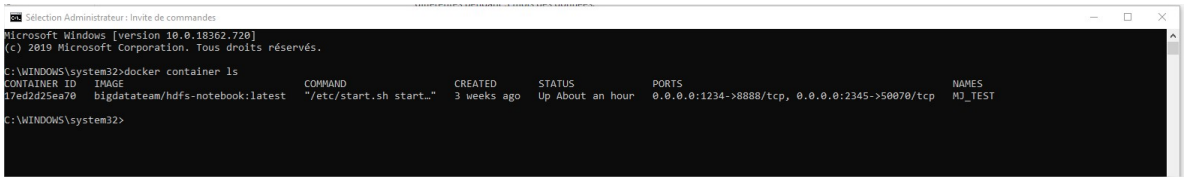
Mon fichier d'import csv est localisé sur mon disque C, C:\Users\minglei.jiang\Documents\Gitlab\lpcafe\Modules\Module7 BigData, maintenant la première étape est donc de copier ce fichier d'import dans notre container docker.



8.1.2.1 Les containers en run

Ouvrir votre Invite de commandes et taper

```
docker container ls
```



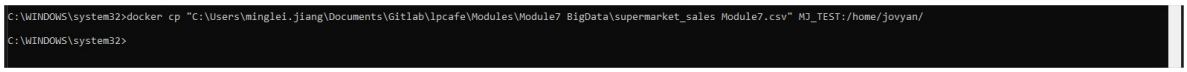
Vous devez normalement voir tous les containers en run sur votre ordinateur. Dans mon exemple, nous avons utiliser le container **MJ_TEST** pour ce module. Nous allons donc copier le fichier d'import **supermarket_sales_Module7.csv** dans ce container pour la suite du TP.

8.1.2.2 Copier le fichier

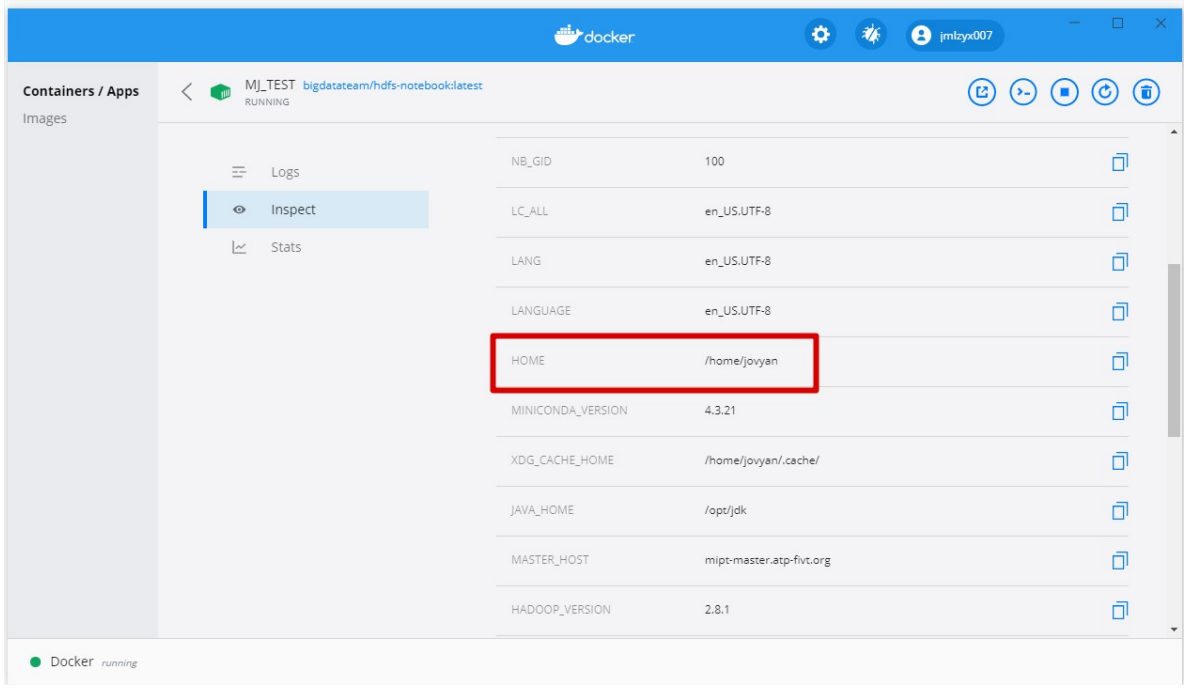
Nous allons donc maintenant copier le fichier d'import csv dans notre container de test: **MJ_TEST**

Vous pouvez donc exécuter le code suivant pour le faire:

```
docker cp "C:\Users\minglei.jiang\Documents\Gitlab\lpcafe\Modules\Module7 BigData\supermarket_sales_Module7.csv" MJ_TEST:/home/jovyan/
```



Où "C:\Users\minglei.jiang\Documents\Gitlab\lpcafe\Modules\Module7 BigData\supermarket_sales_Module7.csv" est le chemin complet de notre fichier csv, MJ_TEST est le nom du container, cp signifie que "copy" et /home/jovyan/ est le path du page d'accueil de container que vous pouvez trouvez dans les configurations du container:



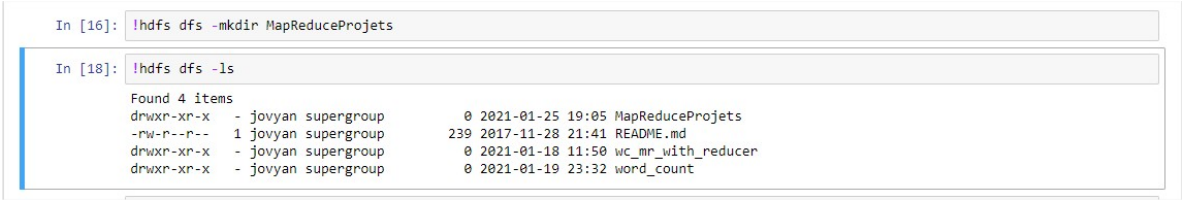
Expected Output 8.1.2:

Vous devez normalement trouver que le fichier est présent, après l'exécution de la commande, dans le page d'accueil de votre container via Jupyternotebook lancé sur localhost:

8.1.3 Copier le fichier d'import dans HDFS

Nous allons tout d'abord créer un nouveau dossier **MapReduceProjets** pour centraliser nos fichiers d'import:

```
!hdfs dfs -mkdir MapReduceProjets
!hdfs dfs -ls
```

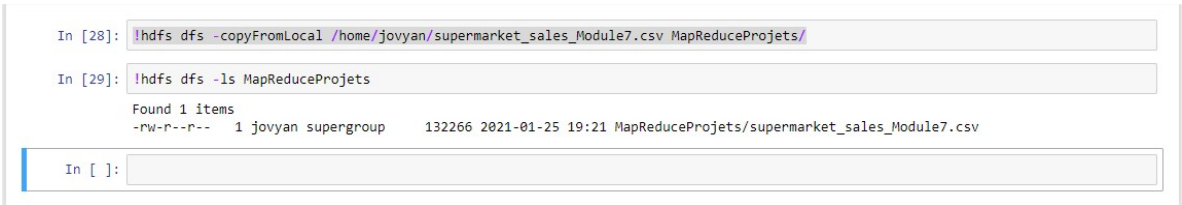


Nous utilisons la commende -copyFromLocal pour copier un fichier local dans le HDFS.

```
!hdfs dfs -copyFromLocal /home/jovyan/supermarket_sales_Module7.csv
MapReduceProjets/
```

Expected Output 8.1.3:

Vous devez normalement trouver que le fichier est présent, après l'exécution de la commande, dans le dossier MapReduceProjets.



8.1.4 mapperTP01.py

L'objective de mapperTP01.py est de définir les paires clé/ valeur . Dans notre cas, il s'agit de se concentrer sur les deux colonnes: **City** et **Total**.

```
#!/usr/bin/env python
from __future__ import print_function
import sys

# input comes from STDIN (standard input)
for 01 in 02:
    # remove leading and trailing whitespace
    line = 03
    # split the line into words
    words = 04
    # Afficher cle valeur city and Total
    print(words[04],words[05], sep = "\t")
```

Une fois que vous aurez terminé l'implémentation(en complétant les 5 champs vides, 01 ... 05), testez mapperTP01.py en complétant le code suivant(Champ vide 06), sachant que nous voulons négliger la phase reduce pour le moment:

```
!hdfs dfs -rm -r outTp01
HADOOP_STREAMING_JAR="../../opt/cloudera/parcels/CDH/lib/hadoop-mapreduce/hadoop-streaming.jar"
!yarn jar $HADOOP_STREAMING_JAR \
    -files mapperTP01.py \
    -mapper 'python mapperTP01.py' \
    06
    -input MapReduceProjets/supermarket_sales_Module7.csv \
    -output outTp01
```

Si tout se passe bien, vous devez avoir les détails d'exécution suivant:

```
    -output outTp01
Deleted outTp01
packageJobJar: [/tmp/hadoop-unjar8640114373007532045/] [] /tmp/streamjob2281832589545419504.jar tmpDir=null
21/01/25 22:19:03 INFO client.RMProxy: Connecting to ResourceManager at /0.0.0.0:8032
21/01/25 22:19:03 INFO client.RMProxy: Connecting to ResourceManager at /0.0.0.0:8032
21/01/25 22:19:04 INFO mapred.FileInputFormat: Total input files to process : 1
21/01/25 22:19:04 INFO mapreduce.JobSubmitter: number of splits:2
21/01/25 22:19:04 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_1611590084730_0009
21/01/25 22:19:04 INFO impl.YarnClientImpl: Submitted application application_1611590084730_0009
21/01/25 22:19:04 INFO mapreduce.Job: The url to track the job: http://17ed2d25ea70:8088/proxy/application_1611590084730_0009/
21/01/25 22:19:04 INFO mapreduce.Job: Running job: job_1611590084730_0009
21/01/25 22:19:09 INFO mapreduce.Job: Job job_1611590084730_0009 running in uber mode : false
21/01/25 22:19:09 INFO mapreduce.Job:  map 0% reduce 0%
21/01/25 22:19:16 INFO mapreduce.Job:  map 100% reduce 0%
21/01/25 22:19:17 INFO mapreduce.Job: Job job_1611590084730_0009 completed successfully
21/01/25 22:19:17 INFO mapreduce.Job: Counters: 31
        File System Counters
                FILE: Number of bytes read=0
                FILE: Number of bytes written=277810
                FILE: Number of read operations=0
                FILE: Number of large read operations=0
                FILE: Number of write operations=0
                HDFS: Number of bytes read=136626
                HDFS: Number of bytes written=24520
                HDFS: Number of read operations=10
                HDFS: Number of large read operations=0
                HDFS: Number of write operations=4
        Job Counters
                Killed map tasks=1
                Launched map tasks=2
                Data-local map tasks=2
                Total time spent by all maps in occupied slots (ms)=7485
                Total time spent by all reduces in occupied slots (ms)=0
                Total time spent by all map tasks (ms)=7485
                Total vcore-milliseconds taken by all map tasks=7485
                Total megabyte-milliseconds taken by all map tasks=7664640
        Map-Reduce Framework
                Map input records=1001
                Map output records=1001
                Input split bytes=264
                Spilled Records=0
                Failed Shuffles=0
                Merged Map outputs=0
                GC time elapsed (ms)=175
                CPU time spent (ms)=680
                Physical memory (bytes) snapshot=330526720
                Virtual memory (bytes) snapshot=3900993536
                Total committed heap usage (bytes)=165150720
        File Input Format Counters
                Bytes Read=136362
        File Output Format Counters
                Bytes Written=24520
21/01/25 22:19:17 INFO streaming.StreamJob: Output directory: outTp01
```

Vérifier le résultat de lancement votre application streaming en exécutant le code suivant:

```
!hdfs dfs -ls outTp01
print ("--------------------------------")
!hdfs dfs -text outTp01/part-00000 | head -5
print ("--------------------------------")
!hdfs dfs -text outTp01/part-00001 | tail -5
```

Expected Output 8.1.4:

```
Found 3 items
-rw-r--r--   1 jovyan supergroup          0 2021-01-25 23:06 outTp01/_SUCCESS
-rw-r--r--   1 jovyan supergroup       8278 2021-01-25 23:06 outTp01/part-00000
-rw-r--r--   1 jovyan supergroup       8234 2021-01-25 23:06 outTp01/part-00001
------------------------------
City    Total
Yangon  548.9715
Naypyitaw   80.22
Yangon  340.5255
Yangon  489.048
------------------------------
Naypyitaw   42.3675
Mandalay    1022.49
Yangon  33.432
```

```
Yangon  69.111
Yangon  649.299
```

```
In [58]: !hdfs dfs -ls outTp01
         print ("-------------------------------")
         !hdfs dfs -text outTp01/part-00000 | head -5
         print ("-------------------------------")
         !hdfs dfs -text outTp01/part-00001 | tail -5

         Found 3 items
         -rw-r--r--   1 jovyan supergroup          0 2021-01-25 23:06 outTp01/_SUCCESS
         -rw-r--r--   1 jovyan supergroup       8278 2021-01-25 23:06 outTp01/part-00000
         -rw-r--r--   1 jovyan supergroup       8234 2021-01-25 23:06 outTp01/part-00001
         -------------------------------
         City    Total
         Yangon  548.9715
         Naypyitaw       80.22
         Yangon  340.5255
         Yangon  489.048
         -------------------------------
         Naypyitaw       42.3675
         Mandalay        1022.49
         Yangon  33.432
         Yangon  69.111
         Yangon  649.299
```

8.1.5 reducerTP01.py

Passons à la phase reduce. L'objective de reducerTP01 est bien évidemment d'agréger les nombres total des ventes ville par ville. Regardons le code suivant:

```
#!/usr/bin/env python
from __future__ import print_function
from operator import itemgetter
import sys

current_city = None
current_prix = 0
city = None

# input comes from STDIN
01
    # remove leading and trailing whitespace
    02

    # parse the input we got from mapper.py
    city, prix = line.split("\t", 1)

    try:
        prix = float(prix)
    except ValueError:
            # prix was not a number, so silently
            #ignore/discard this line
            continue

    if 03:
        04  #calculer la somme des prix par ville
    else:
        if 05:
             # write result to STDOUT
            print(current_city, current_prix)  #print cle valeur
        06   #passer a la cle suivant (ville)
```

```
        07  #initialisation le prix pour le ville suivant

# do not forget to output the last word if needed!
if 08:
    print(current_city, current_prix) #print cle valeur
```

Une fois que vous aurez terminé l'implémentation(en complétant les 8 champs vides, 01 ... 08), testez mapperTP01.py et reducerTP01.py en complétant le code suivant(Champs vide 09 et 10):

```
!hdfs dfs -rm -r outTp01
HADOOP_STREAMING_JAR="../../opt/cloudera/parcels/CDH/lib/hadoop-mapreduce/hadoop-streaming.jar"
!yarn jar $HADOOP_STREAMING_JAR \
    09
    -mapper 'python mapperTP01.py' \
    10
    -input MapReduceProjets/supermarket_sales_Module7.csv \
    -output outTp01
```

Vérifier le résultat de lancement de votre application streaming en exécutant le code suivant:

```
!hdfs dfs -ls outTp01
print ("--------------------------------")
!hdfs dfs -text outTp01/part-00000
```

Expected Output 8.1.5:

```
Found 2 items
-rw-r--r--   1 jovyan supergroup          0 2021-01-25 23:17 outTp01/_SUCCESS
-rw-r--r--   1 jovyan supergroup         64 2021-01-25 23:17 outTp01/part-00000
--------------------------------
Mandalay 106197.672
Naypyitaw 110568.7065
Yangon 106200.3705
```

8.1.6 Conclusion

Avec les deux programmes réalisés dans ce TP, nous arrivons à localiser les ventes totales par ville qui nous permettra d'identifier plus clairement la répartition des ventes entre les villes et quelle(s) villes nous rapporte la plus de vente.

8.2 TP 02: Calculer maximum et minimum du salaire par profession

Une façon de comprendre comment fonctionne une administration municipale consiste à examiner qui elle emploie et comment ses employés sont rémunérés. Ces données contiennent les noms, le titre du poste et la rémunération des employés de la ville de San Francisco sur une base annuelle de 2011 à 2014.

Nous avons pris les fichiers bruts ici[https://transparentcalifornia.com/salaries/san-francisco/] et les avons combinés / normalisés dans un seul fichier CSV.

8.2.1 Le fichier d'import: salaries_Module7.csv

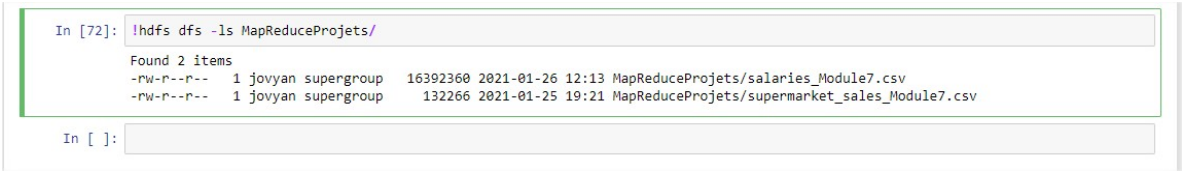
Le fichier est sous la forme suivant:

Id	EmployeeName	JobTitle	BasePay	OvertimePay	OtherPay	Benefits	TotalPay	Year	Notes	Agency	Status

De la même manière que `8.1.2` et `8.1.3`, nous copions le fichier **salaries_Module7.csv** dans notre dossier HDFS: **MapReduceProjets**

Expected result 8.2.1:

```
!hdfs dfs -ls MapReduceProjets/
```



L'objective de ce TP02 est déterminer les salaires maximum et minimum pour chaque profession ainsi de compter combien de personnes ont touché ce max/min.

8.2.2 mapperTP02.py

L'objective de mapperTP02.py est également, comme TP01, de définir les paires clé/ valeur . Dans notre cas, il s'agit de se concentrer sur les deux colonnes: **JobTitle** et **TotalPay**. En plus, il faut aussi afficher un compteur de 1. Donc ça fait en total une clé deux valeurs pour le flux de sortie.

```
#!/usr/bin/env python
from __future__ import print_function
import sys

##################
#Avec l'aide de mapperTP01, compléter vous même mapperTP02
##################
# input comes from STDIN (standard input)
    # remove leading and trailing whitespace
    # split the line into words
    # Afficher cle valeur city and Total
```

Une fois que vous aurez codé mapperTP02, testez mapperTP02.py, sachant que nous voulons négliger la phase reduce pour le moment:

```
!hdfs dfs -rm -r outTp02
HADOOP_STREAMING_JAR="../../opt/cloudera/parcels/CDH/lib/hadoop-mapreduce/hadoop-streaming.jar"
!yarn jar $HADOOP_STREAMING_JAR \
    #Complétez vous les commandes CLI.
    -output outTp02
```

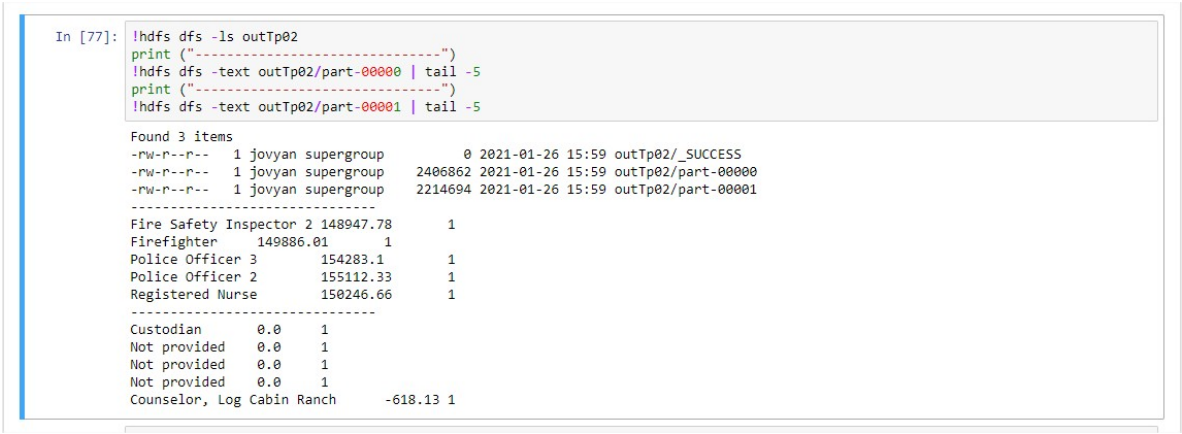
Vérifier le résultat de lancement votre application streaming en exécutant le code suivant:

```
!hdfs dfs -ls outTp02
print ("------------------------------")
!hdfs dfs -text outTp02/part-00000 | tail -5
print ("------------------------------")
!hdfs dfs -text outTp02/part-00001 | tail -5
```

Expected Output 8.2.2:

```
Found 3 items
-rw-r--r--   1 jovyan supergroup          0 2021-01-26 15:59 outTp02/_SUCCESS
-rw-r--r--   1 jovyan supergroup    2406862 2021-01-26 15:59 outTp02/part-00000
-rw-r--r--   1 jovyan supergroup    2214694 2021-01-26 15:59 outTp02/part-00001
------------------------------
Fire Safety Inspector 2 148947.78   1
Firefighter 149886.01   1
Police Officer 3    154283.1    1
Police Officer 2    155112.33   1
Registered Nurse    150246.66   1
------------------------------
Custodian   0.0 1
Not provided    0.0 1
Not provided    0.0 1
```

```
Not provided    0.0 1
Counselor, Log Cabin Ranch  -618.13 1
```



```
In [77]: !hdfs dfs -ls outTp02
         print ("--------------------------------")
         !hdfs dfs -text outTp02/part-00000 | tail -5
         print ("--------------------------------")
         !hdfs dfs -text outTp02/part-00001 | tail -5

         Found 3 items
         -rw-r--r--   1 jovyan supergroup          0 2021-01-26 15:59 outTp02/_SUCCESS
         -rw-r--r--   1 jovyan supergroup    2406862 2021-01-26 15:59 outTp02/part-00000
         -rw-r--r--   1 jovyan supergroup    2214694 2021-01-26 15:59 outTp02/part-00001
         --------------------------------
         Fire Safety Inspector 2 148947.78       1
         Firefighter     149886.01       1
         Police Officer 3        154283.1        1
         Police Officer 2        155112.33       1
         Registered Nurse        150246.66       1
         --------------------------------
         Custodian       0.0     1
         Not provided    0.0     1
         Not provided    0.0     1
         Not provided    0.0     1
         Counselor, Log Cabin Ranch      -618.13 1
```

8.2.3 reducerTP02.py

Passons à la phase reduce. L'objective de reducerTP02 est donc de garder le salaire maximum et minimum pour chaque profession et de compter combien de personnes ont touché ce max/min.

Regardons le code suivant:

```
#!/usr/bin/env python

from operator import itemgetter
import sys

current_pro = None
max_salaire = 0
min_salaire = 0
current_counter = 0
pro = None

# input comes from STDIN
01:
    # remove leading and trailing whitespace
    02

    # parse the input we got from mapper.py
    pro, salaire, counter = line.split("\t", 3)

    try:
        salaire = float(salaire)
    except ValueError:
            # salaire was not a number, so silently
            #ignore/discard this line
            continue
    try:
        counter = float(counter)
    except ValueError:
            # counter was not a number, so silently
```

```
            #ignore/discard this line
            continue
    if min_salaire == 0:
        03  #initialiser  une valeur au minimum pour la premier cle,valeur
    if current_pro == pro:
        if max_salaire < salaire:  # le maximum
            04
        if min_salaire > salaire: #le minimum
            05
        06 #calcule le nombre du employer par profession
    else:
        if current_pro:
             # write result to STDOUT
            07
        08  #passer a la cle suivant
        09  #initialisation du conteur de la profession suivante
        10 #initialisation du salaire de la profession suivante
        11 #intialiser premier salaire de la cle suivante comme un minimum
        12  #intialiser premier salaire de la cle suivante comme un maximum

    # do not forget to output the last word if needed!
if current_pro == pro:
    print(current_pro, max_salaire , min_salaire , current_counter)
```

Une fois que vous aurez terminé l'implémentation(en complétant les 8 champs vides, 01 ... 12), testez mapperTP02.py et reducerTP02.py en complétant le code suivant:

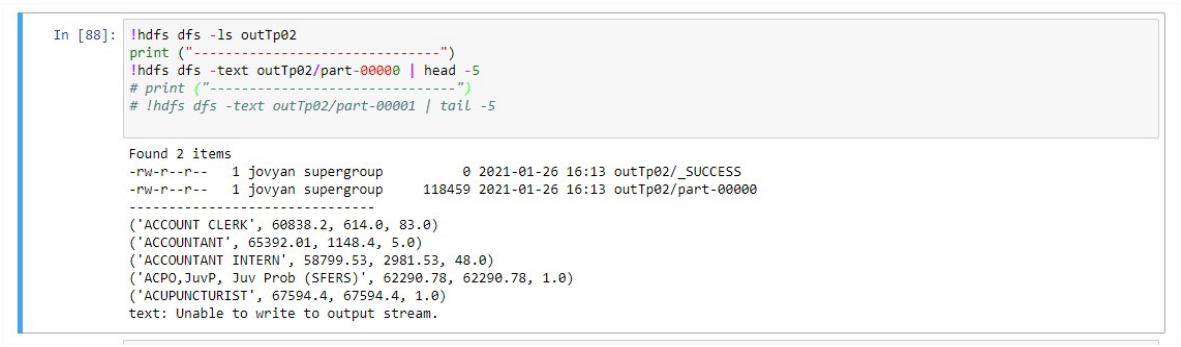
```
!hdfs dfs -rm -r outTp02
HADOOP_STREAMING_JAR="../../opt/cloudera/parcels/CDH/lib/hadoop-mapreduce/hadoop-streaming.jar"
!yarn jar $HADOOP_STREAMING_JAR \
    #Complétez vous les commandes CLI.
    -output outTp02
```

Vérifier le résultat de lancement de votre application streaming en exécutant le code suivant:

```
!hdfs dfs -ls outTp02
print ("-------------------------------")
!hdfs dfs -text outTp02/part-00000 | head -5
```

Expected Output 8.2.3:

```
Found 2 items
-rw-r--r--   1 jovyan supergroup          0 2021-01-26 16:13 outTp02/_SUCCESS
-rw-r--r--   1 jovyan supergroup     118459 2021-01-26 16:13 outTp02/part-00000
------------------------------
('ACCOUNT CLERK', 60838.2, 614.0, 83.0)
('ACCOUNTANT', 65392.01, 1148.4, 5.0)
('ACCOUNTANT INTERN', 58799.53, 2981.53, 48.0)
('ACPO,JuvP, Juv Prob (SFERS)', 62290.78, 62290.78, 1.0)
('ACUPUNCTURIST', 67594.4, 67594.4, 1.0)
text: Unable to write to output stream.
```



8.3 TP03: Déterminer l'âge moyen des personnes par sexe. dans la catastrophe du Titanic

Nous connaissons tous le désastre qui s'est produit le 14 avril 1912. Le gros navire géant de 46 000 tonnes a sombré à une profondeur de 13 000 pieds dans l'océan Atlantique Nord. Notre objectif est d'analyser les données obtenues après cette catastrophe. Hadoop MapReduce peut être utilisé pour traiter efficacement ces grands ensembles de données afin de trouver une solution à un problème particulier.

L'objective de ce TP est de faire une analyse du jeu de données Titanic Disaster, pour trouver l'âge moyen des hommes et des femmes décédés dans cette catastrophe avec MapReduce Hadoop.

8.3.1 Le fichier d'import: titanic_data.txt

Le fichier est sous la forme suivant:

PassengerID	Survived(0/1) /*1 If Person is dead */	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked

De la même manière que `8.1.2` et `8.1.3`, nous copions le fichier **salaries_Module7.csv** dans notre dossier HDFS: **MapReduceProjets**

Expected result 8.3.1:

```
!hdfs dfs -ls MapReduceProjets/
```

8.3.2 mapperTP03.py

L'objective de mapperTP02.py est de définir les paires clé/ valeur. Mais cette fois ci avec une condition en plus qui est donc de filtrer sur toutes les personnes décédées.

```
#!/usr/bin/env python
from __future__ import print_function
import sys

##################
#Avec l'aide de mapperTP01 et mapperTP02, compléter le code
##################
```

Une fois que vous aurez codé mapperTP02, testez mapperTP02.py, sachant que nous voulons négliger la phase reduce pour le moment.

```
!hdfs dfs -rm -r outTp03
HADOOP_STREAMING_JAR="../../opt/cloudera/parcels/CDH/lib/hadoop-mapreduce/hadoop-streaming.jar"
!yarn jar $HADOOP_STREAMING_JAR \
    #Complétez vous les commandes CLI.
    -output outTp03
```

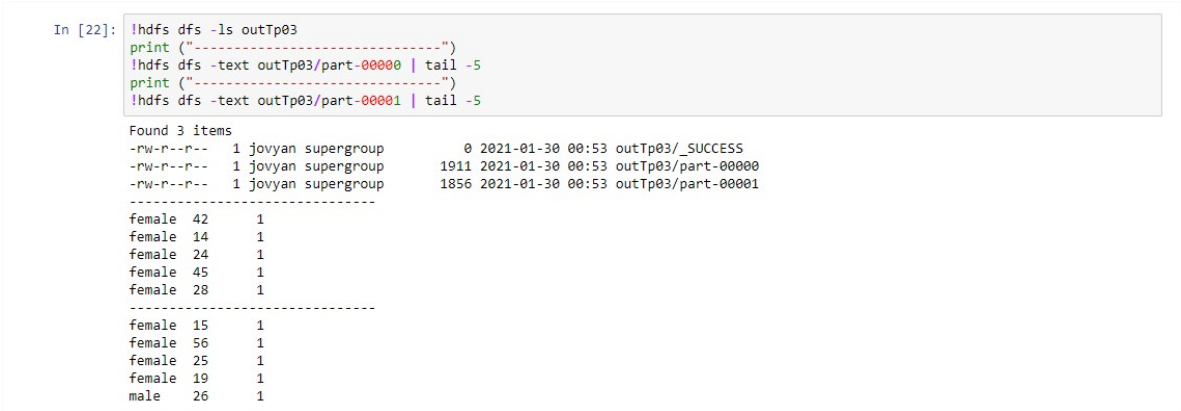
Vérifier le résultat de lancement votre application streaming en exécutant le code suivant:

```
!hdfs dfs -ls outTp03
print ("-------------------------------")
!hdfs dfs -text outTp03/part-00000 | tail -5
print ("-------------------------------")
!hdfs dfs -text outTp03/part-00001 | tail -5
```

Expected result 8.3.2:

```
Found 3 items
-rw-r--r--   1 jovyan supergroup          0 2021-01-30 00:53 outTp03/_SUCCESS
-rw-r--r--   1 jovyan supergroup       1911 2021-01-30 00:53 outTp03/part-00000
-rw-r--r--   1 jovyan supergroup       1856 2021-01-30 00:53 outTp03/part-00001
-------------------------------
female  42  1
female  14  1
```

```
female  24  1
female  45  1
female  28  1
--------------------------------
female  15  1
female  56  1
female  25  1
female  19  1
male    26  1
```



8.3.3 reducerTP03.py

Passons à la phase reduce. L'objective de reducerTP03 est donc de calculer l'âge moyen par sexe des personnes décédées pendant la catastrophe. En déduit de la même logique que TP01 et TP02, implémentez le reducerTP03.py

```
#!/usr/bin/env python
from __future__ import print_function
import sys

##################
#Avec l'aide de reducerTP01 et reducerTP02, compléter le code
##################
```

Une fois que vous aurez terminé l'implémentation, testez mapperTP03.py et reducerTP03.py.

```
!hdfs dfs -rm -r outTp03
HADOOP_STREAMING_JAR="../../opt/cloudera/parcels/CDH/lib/hadoop-mapreduce/hadoop-streaming.jar"
!yarn jar $HADOOP_STREAMING_JAR \
    #Complétez vous les commandes CLI.
    -output outTp03
```

Vérifier le résultat de lancement de votre application streaming en exécutant le code suivant:

```
!hdfs dfs -ls outTp03
print ("-------------------------------")
!hdfs dfs -text outTp03/part-00000 | head -5
```

Expected Output 8.3.3:

```
Found 2 items
-rw-r--r--   1 jovyan supergroup          0 2021-01-30 01:04 outTp03/_SUCCESS
-rw-r--r--   1 jovyan supergroup         61 2021-01-30 01:04 outTp03/part-00000
-------------------------------
('female', 28.84771573604061)
('male', 27.276021505376345)
```