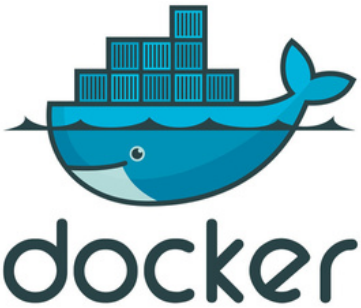
Rapport du Projet Big Data

Hadoop ,Docker et Spark



Élaborée par :

Islem Ben Hassouna

2BD2

ISAMM

Sommaire

I.Procédures d'installation de Docker et installation image Docker:

1.Télécharger docker à travers la commande:

sudo apt install docker.io

```
islem@islem-VirtualBox:~$ sudo apt install docker.io
```

2.Télécharger l'image docker à travers la commande

sudo docker pull liliasfaxi/hadoop-cluster:latest

```
islem@islem-VirtualBox:~$ docker pull liliasfaxi/hadoop-cluster:latest
La commande « docker » n'a pas été trouvée, mais peut être installée avec :
sudo apt install podman-docker  # version 3.4.4+ds1-1ubuntu1.22.04.2, or
sudo apt install docker.io      # version 24.0.5-0ubuntu1~22.04.1
```

3.Créer un réseau qui permettra de relier les trois contenaires:

sudo docker network create --driver=bridge hadoop

```
islem@islem-VirtualBox:~$ sudo docker network create --driver=bridge hadoop
66f7df10862c7eefc32ca1094dd048fc7308af9d35952318034c37afda9ee4ef
```

4.Créer et lancer les trois contenaires :

hadoop-master:

sudo docker run -itd --net=hadoop -p 9870:9870 -p 8088:8088 -p 7077:7077 -p 16010:16010 --name hadoop-master --hostname hadoop-master liliasfaxi/hadoop-cluster:latest

hadoop-worker1:

sudo docker run -itd -p 8040:8042 --net=hadoop --name hadoop-worker1 --hostname hadoop-worker1 liliasfaxi/hadoop-cluster:latest

hadoop-worker2:

sudo docker run -itd -p 8041:8042 --net=hadoop --name hadoop-worker2 --hostname hadoop-worker2 liliasfaxi/hadoop-cluster:latest

5.Vérifier que les trois contenaires tournent bien en lançant la commande

docker ps

```
islem@islem-VirtualBox:~$ sudo docker ps
[sudo] password for islem:
CONTAINER ID   IMAGE                                COMMAND                  CREATED        STATUS        PORTS

                                NAMES
ec634bd70bc4   liliasfaxi/hadoop-cluster:latest   "sh -c 'service ssh …"   45 hours ago   Up 45 hours   0.0.0.0:7077->7077/
tcp, :::7077->7077/tcp, 0.0.0.0:8088->8088/tcp, :::8088->8088/tcp, 0.0.0.0:9870->9870/tcp, :::9870->9870/tcp, 0.0.0.0:16010
->16010/tcp, :::16010->16010/tcp   hadoop-master
0150696b57f7   liliasfaxi/hadoop-cluster:latest   "sh -c 'service ssh …"   45 hours ago   Up 45 hours   0.0.0.0:8040->8042/
tcp, :::8040->8042/tcp
                                hadoop-worker1
94aacf1dca24   liliasfaxi/hadoop-cluster:latest   "sh -c 'service ssh …"   2 days ago     Up 2 days     0.0.0.0:8041->8042/
tcp, :::8041->8042/tcp
                                hadoop-worker2
```

6.charger **purchases.txt** et **Mapper.py** et **Reducer.py** dans le contenaire **hadoop-master** avec les commande suivante:

sudo docker cp /home/islem/Downloads/purchases.txt hadoop-master:/root/purchases.txt

sudo docker cp /home/islem/Downloads/Mapper.py hadoop-master:/root/Mapper.py

sudo docker cp /home/islem/Downloads/Reducer.py hadoop-master:/root/Reducer.py

```
islem@islem-VirtualBox:~$ sudo docker cp /home/islem/Downloads/purchases.txt hadoop-master:/root/purchases.txt
Successfully copied 211MB to hadoop-master:/root/purchases.txt
islem@islem-VirtualBox:~$ sudo docker cp /home/islem/Downloads/Mapper.py  hadoop-master:/root/Mapper.py
Successfully copied 2.05kB to hadoop-master:/root/Mapper.py
islem@islem-VirtualBox:~$ sudo docker cp /home/islem/Downloads/Reducer.py  hadoop-master:/root/Reducer.py
```

II.MapReduce dans le conteneur Master:

1.Entrer dans le contenaire master pour commencer à l'utiliser.

sudo docker exec -it hadoop-master bash

2.La première chose à faire, une fois dans le contenaire, est de lancer hadoop et yarn.

./start-hadoop.sh

```
islem@islem-VirtualBox:~$ sudo docker exec -it hadoop-master bash
[sudo] password for islem:
root@hadoop-master:~# ./start-hadoop.sh


Starting namenodes on [hadoop-master]
hadoop-master: Warning: Permanently added 'hadoop-master' (ED25519) to the list of known hosts.
hadoop-master: WARNING: HADOOP_NAMENODE_OPTS has been replaced by HDFS_NAMENODE_OPTS. Using value of HADOOP_NAMENODE_OPTS.
hadoop-master: namenode is running as process 165.  Stop it first and ensure /tmp/hadoop-root-namenode.pid file is empty be
fore retry.
Starting datanodes
WARNING: HADOOP_SECURE_DN_LOG_DIR has been replaced by HADOOP_SECURE_LOG_DIR. Using value of HADOOP_SECURE_DN_LOG_DIR.
hadoop-worker1: Warning: Permanently added 'hadoop-worker1' (ED25519) to the list of known hosts.
hadoop-worker2: Warning: Permanently added 'hadoop-worker2' (ED25519) to the list of known hosts.
hadoop-worker2: WARNING: HADOOP_SECURE_DN_LOG_DIR has been replaced by HADOOP_SECURE_LOG_DIR. Using value of HADOOP_SECURE_
DN_LOG_DIR.
hadoop-worker2: WARNING: HADOOP_DATANODE_OPTS has been replaced by HDFS_DATANODE_OPTS. Using value of HADOOP_DATANODE_OPTS.
hadoop-worker1: WARNING: HADOOP_SECURE_DN_LOG_DIR has been replaced by HADOOP_SECURE_LOG_DIR. Using value of HADOOP_SECURE_
DN_LOG_DIR.
hadoop-worker1: WARNING: HADOOP_DATANODE_OPTS has been replaced by HDFS_DATANODE_OPTS. Using value of HADOOP_DATANODE_OPTS.
hadoop-worker1: datanode is running as process 70.  Stop it first and ensure /tmp/hadoop-root-datanode.pid file is empty be
fore retry.
hadoop-worker2: datanode is running as process 70.  Stop it first and ensure /tmp/hadoop-root-datanode.pid file is empty be
```

3. lister les processus Java en cours d'exécution dans les conteneurs Docker à travers la commande

```
root@hadoop-master:~# jps
352 SecondaryNameNode
593 ResourceManager
902 Jps
166 NameNode
root@hadoop-master:~#
```

4.À partir du contenaire master, charger le fichier purchases dans le répertoire input (de HDFS) que vous avez créé:

hdfs dfs –put purchases.txt myinput

```
root@hadoop-master:~# hdfs dfs -put purchases.txt myinput
```

5.Vérifier l’existance de purchases.txt dans myinput

hdfs dfs –ls myinput

```
root@hadoop-master:~# hdfs dfs -ls myinput
Found 1 items
-rw-r--r--   2 root supergroup  211312924 2024-05-06 12:41 myinput/purchases.txt
```

6.visualiser les dernières lignes du fichier,

hdfs dfs -tail myinput/purchases.txt

```
root@hadoop-master:~# hdfs dfs -tail myinput/purchases.txt
31      17:59   Norfolk Toys    164.34  MasterCard
2012-12-31      17:59   Chula Vista     Music   380.67  Visa
2012-12-31      17:59   Hialeah Toys    115.21  MasterCard
2012-12-31      17:59   Indianapolis    Men's Clothing  158.28  MasterCard
2012-12-31      17:59   Norfolk Garden  414.09  MasterCard
2012-12-31      17:59   Baltimore       DVDs    467.3   Visa
2012-12-31      17:59   Santa Ana       Video Games     144.73  Visa
2012-12-31      17:59   Gilbert Consumer Electronics    354.66  Discover
2012-12-31      17:59   Memphis Sporting Goods  124.79  Amex
2012-12-31      17:59   Chicago Men's Clothing  386.54  MasterCard
2012-12-31      17:59   Birmingham      CDs     118.04  Cash
2012-12-31      17:59   Las Vegas       Health and Beauty       420.46  Amex
2012-12-31      17:59   Wichita Toys    383.9   Cash
2012-12-31      17:59   Tucson  Pet Supplies    268.39  MasterCard
2012-12-31      17:59   Glendale        Women's Clothing        68.05   Amex
2012-12-31      17:59   Albuquerque     Toys    345.7   MasterCard
2012-12-31      17:59   Rochester       DVDs    399.57  Amex
2012-12-31      17:59   Greensboro      Baby    277.27  Discover
2012-12-31      17:59   Arlington       Women's Clothing        134.95  MasterCard
2012-12-31      17:59   Corpus Christi  DVDs    441.61  Discover
```

7.changer le repertoire et aller à HADOOP_CONF_DIR

cd $HADOOP_CONF_DIR

```
root@hadoop-master:~# cd $HADOOP_CONF_DIR/
root@hadoop-master:/usr/local/hadoop/etc/hadoop# ls
capacity-scheduler.xml      hadoop-user-functions.sh.example  kms-log4j.properties        ssl-client.xml.example
configuration.xsl           hdfs-rbf-site.xml                 kms-site.xml                ssl-server.xml.example
container-executor.cfg      hdfs-site.xml                     log4j.properties            user_ec_policies.xml.template
core-site.xml               httpfs-env.sh                     mapred-env.cmd              workers
hadoop-env.cmd              httpfs-log4j.properties           mapred-env.sh               yarn-env.cmd
hadoop-env.sh               httpfs-site.xml                   mapred-queues.xml.template  yarn-env.sh
hadoop-metrics2.properties  kms-acls.xml                      mapred-site.xml             yarn-site.xml
hadoop-policy.xml           kms-env.sh                        shellprofile.d              yarnservice-log4j.properties
root@hadoop-master:/usr/local/hadoop/etc/hadoop# vi mapred-site.xml
```

8.configurer le fichier mapred-site.xml

vi mapred-site.xml

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>
<!--
  Licensed under the Apache License, Version 2.0 (the "License");
  you may not use this file except in compliance with the License.
  You may obtain a copy of the License at

    http://www.apache.org/licenses/LICENSE-2.0

  Unless required by applicable law or agreed to in writing, software
  distributed under the License is distributed on an "AS IS" BASIS,
  WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
  See the License for the specific language governing permissions and
  limitations under the License. See accompanying LICENSE file.
-->

<!-- Put site-specific property overrides in this file. -->

<configuration> <!-- Configurations for MapReduce Applications: -->
  <property>
       <name>mapreduce.framework.name</name>
       <value>yarn</value>
   </property>
<property>
  <name>yarn.app.mapreduce.am.env</name>
  <value>HADOOP_MAPRED_HOME=/usr/local/hadoop</value>
</property>
<property>
  <name>mapreduce.map.env</name>
  <value>HADOOP_MAPRED_HOME=/usr/local/hadoop</value>
</property>
<property>
  <name>mapreduce.reduce.env</name>
  <value>HADOOP_MAPRED_HOME=/usr/local/hadoop</value>
</property>


</configuration>
```

9.Lancer la commande hadoop jar

hadoop jar $HADOOP_HOME/share/hadoop/tools/lib/hadoop-streaming-3.3.6.jar -mapper Mapper.py -reducer Reducer.py -file Maper.py -file Reducer.py -input myinput -output /joboutput

```
root@hadoop-master:~# hadoop jar $HADOOP_HOME/share/hadoop/tools/lib/hadoop-streaming-3.3.6.jar -mapper Mapper.py -reducer Reducer.p
y -file Mapper.py -file Reducer.py -input myinput -output /joboutput
```

10.voir le contenue de répertoire joboutput

hdfs dfs -ls /joboutput : 2 fichier _SUCCESS ET part-00000

```
root@hadoop-master:~# hdfs dfs -ls /joboutput
Found 2 items
-rw-r--r--   2 root supergroup          0 2024-05-06 13:15 /joboutput/_SUCCESS
-rw-r--r--   2 root supergroup       3100 2024-05-06 13:15 /joboutput/part-00000
```

III.SPARK

1.Présentation de mon dataset:

L'ensemble de données "Forbes Billionaires Evolution" offre un examen complet de la croissance financière et de la situation des milliardaires mondiaux de 1997 à 2023. Il documente les transformations des fortunes de ces titans de la finance sur près de trois décennies.

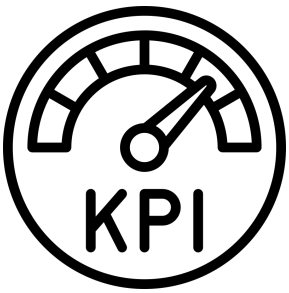
Lien Source: Forbes Billionaires Evolution (1997-2023) (kaggle.com)

Format: CSV

Structure:mon dataset contient 31 732 lignes (points de données) et 18 colonnes (attributs).

Voici une ventilation des colonnes :

- year : Année (1997-2023)
- month : Mois (1-12)
- rank : Classement du milliardaire sur la liste Forbes
- age : age du milliardaire
- net_worth : Fortune nette du milliardaire (peut nécessiter un nettoyage)
- last_name : Nom de famille du milliardaire
- first_name : Prénom du milliardaire
- full_name : Combinaison du nom et du prénom
- birth_date : Date de naissance du milliardaire
- gender : Genre du milliardaire
- country_of_citizenship : Pays de citoyenneté du milliardaire
- country_of_residence : Pays de résidence du milliardaire
- city_of_residence : Ville de résidence du milliardaire
- business_category : Catégorie de l'activité du milliardaire
- business_industries : Secteurs d'activité associés à l'entreprise du milliardaire
- organization_name : Nom de l'organisation du milliardaire
- position_in_organization : Poste du milliardaire au sein de l'organisation
- self_made : Indique si le milliardaire est self-made
- wealth_status : Statut de richesse du milliardaire



2. les indicateurs de performances (KPIs)

KPIs liés à la richesse :

Richesse totale: Somme de toutes les valeurs de fortune nette dans votre jeu de données. Calculable par année ou sur toute la période.
Richesse moyenne: Richesse totale divisée par le nombre de milliardaires. Peut être calculée globalement ou pour des groupes spécifiques (pays, industrie, genre).
Répartition de la richesse: Mesurez la distribution de la richesse entre les milliardaires à l'aide de l'indice de Gini ou de quantiles.
Taux de croissance de la richesse: Calculez le taux de croissance annuel de la richesse totale ou moyenne.
KPIs liés aux entreprises :
Nombre de milliardaires par secteur: Identifiez les secteurs d'activité concentrant le plus de milliardaires. Affinez l'analyse par pays, richesse héritée vs self-made, etc.
Contribution des secteurs à la richesse totale: Évaluez la contribution de chaque secteur à la richesse totale des milliardaires.

KPIs démographiques :

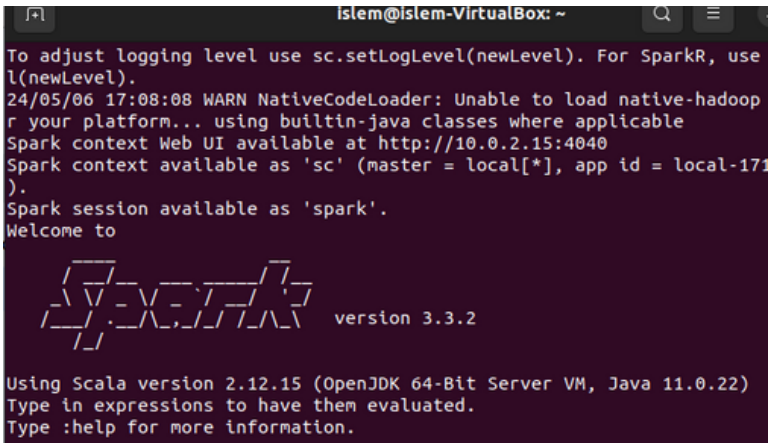
Nombre de milliardaires par pays/région: Localisez la concentration géographique des milliardaires.
Densité des milliardaires: Ratio entre le nombre de milliardaires et la population totale d'un pays ou d'une région.
Répartition par âge: Analysez comment l'accumulation de richesse diffère selon les tranches d'âge.
Répartition par genre: Suivez l'évolution du nombre de femmes milliardaires par rapport aux hommes.

2. les objectifs de mon analyse

1.Comprendre les tendances temporelles : Analyser comment la richesse des milliardaires a évolué au fil du temps, identifier les années où il y a eu des changements significatifs, et comprendre les facteurs économiques ou sociaux qui ont pu influencer ces tendances.
2.Analyser la répartition de la richesse : Examiner la distribution de la richesse parmi les milliardaires au fil des années, identifier les régions géographiques où se concentre la majorité de la richesse, et comprendre les facteurs démographiques ou géopolitiques qui pourraient expliquer ces différences.
3.Disparités de genre parmi les plus riches du monde
4.Étudier les caractéristiques des milliardaires : Analyser les caractéristiques démographiques (âge, genre, pays d'origine, les sources de richesse, les secteurs d'activité) et d'autres facteurs associés aux milliardaires, et identifier les tendances ou les patterns intéressants.
5.Identifier les milliardaires les plus influents : Identifier les milliardaires les plus riches ou les plus influents au fil du temps, examiner leurs parcours professionnels, leurs investissements, et leurs activités philanthropiques pour comprendre leur impact sur l'économie mondiale.

3. Analyse de données interactive avec scala

Spark Scala fournit des outils puissants pour l'analyse de données



a.Démarrer Spark:

Start-master.sh
Spark-shell

b.lire le fichier avec la commande:

```
val
path="file:///home/islem/Téléchargements/archive/all_billi
onaires_1997_2023.csv"
```

c.Accéder et lire le fichier avec la commande:

```
val billionaire = spark.read.option("delimiter",
",").option("header", "true").csv(path)
```

billionaire: est un RDD.

```
scala> val path = "file:///home/islem/Téléchargements/archive/all_billionaires_1
997_2023.csv"
path: String = file:///home/islem/Téléchargements/archive/all_billionaires_1997_
2023.csv

scala> val billionaire = spark.read.option("delimiter", ",").option("header", "t
rue").csv(path)
billionaire: org.apache.spark.sql.DataFrame = [year: string, month: string ... 1
7 more fields]
```

d.Afficher le contenu de RDD avec la commande

billionaire.show()

```
scala> billionaire.show()
+----+-----+----+---------+-----------+--------------------+------------------
-+----------+----+------+--------------------+--------------------+-----------
------+--------------------+--------------------+-----------------+------------
----------+---------+-------------+
|year|month|rank|net_worth|   last_name|          first_name|           full_nam
e|birth_date| age|gender|country_of_citizenship|country_of_residence|city_of_res
idence|   business_category| business_industries|organization_name|position_in_o
rganization|self_made|wealth_status|
+----+-----+----+---------+-----------+--------------------+------------------
-+----------+----+------+--------------------+--------------------+-----------
------+--------------------+--------------------+-----------------+------------
----------+---------+-------------+
|1997|    7|null|    2.0 B|Sophonpanich|              Chatri|Chatri Sophonpani..
.|1934-02-28|  73|  Male|             Thailand|            Thailand|          B
angkok|Finance and Inves...|['Finance and Inv...|             null|
      null|    False|         null|
|1997|    7|null|    1.8 B|   Adulyadej|       King Bhumibol|King Bhumibol Adu..
.|1927-12-05|  69|  Male|             Thailand|                null|
  null|               null|                null|             null|
      null|    False|         null|
|1998|    7|null|    3.3 B|       Safra|              Edmond|        Edmond Safr
```

c.Nettoyage des colonnes

- enléver les valeurs nulles du colonnes net_worth

val filteredBillionnaires = billionaire.filter(col(“net_worth”).isNull)

```
import org.apache.spark.sql.functions._

scala> val filteredBillionaires = billionaire.filter(col("net_worth").isNotNull)

filteredBillionaires: org.apache.spark.sql.Dataset[org.apache.spark.sql.Row] = [
year: string, month: string ... 17 more fields]
```

- enléver B (comme billions) de net_worth poue qu‘il soit un réel:

val cleanedBillionaires = filteredBillionaires.withColumn(“net_worth”,regex_extract(col(“net_worth”,”^([0-9.]”,1).cast(“double”)

```
scala> val cleanedBillionaires = filteredBillionaires.withColumn(
     |   "net_worth",
     |   regexp_extract(col("net_worth"), "^([0-9.]+)", 1).cast("double")
     | )
cleanedBillionaires: org.apache.spark.sql.DataFrame = [year: string, month: stri
ng ... 17 more fields]
```

c.Calcul de la richesse totale des milliardaires chaque année :

- val totalWealthPerYear = cleanedBillionaires.groupBy("year")
  .agg(sum("net_worth").as("total_wealth"))

```
scala> val totalWealthPerYear = cleanedBillionaires.groupBy("year").agg(sum("net
_worth").as("total_wealth"))
totalWealthPerYear: org.apache.spark.sql.DataFrame = [year: string, total_wealth
: double]
```

- totalWealthPerYear.show()

Interprétation:
la richesse totale augmente au cours du temps(on remarque dans le résultat 2022>2020>2018>2016....

```
scala> totalWealthPerYear.show()
+----+-----------------+
|year|      total_wealth|
+----+-----------------+
|2016| 6484.700000000026|
|2012| 4574.500000000015|
|2020| 8037.500000000031|
|2019| 8669.200000000055|
|2017|  7666.10000000005|
|2014| 6446.490000000028|
|2013| 5431.809999999994|
|2005|1817.5999999999963|
|2000|              16.2|
|2002|1113.4999999999977|
|2009|2290.5999999999954|
|2018| 9059.600000000026|
|2006|2228.4999999999995|
|2004|1472.6999999999985|
|2011| 4495.300000000002|
|2022|12705.950000000015|
|2008|3926.4000000000005|
|1999|              18.7|
```

d.Richesse maximale par année

- val maxWealthPerYear = cleanedBillionaires.groupBy("year")
  .agg(max("net_worth").as("max_wealth"))

mais cette fois ci je vais les trier d'ordre décroissant de max_wealth pour une intérpretation plus facile

```
scala> val maxWealthPerYear = cleanedBillionaires.groupBy("year").agg(max("net_w
orth").as("max_wealth"))
maxWealthPerYear: org.apache.spark.sql.DataFrame = [year: string, max_wealth: do
uble]

scala> val sortedMaxWealthPerYear = maxWealthPerYear.orderBy(col("max_wealth").d
esc)
sortedMaxWealthPerYear: org.apache.spark.sql.Dataset[org.apache.spark.sql.Row] =
 [year: string, max_wealth: double]
```

- sortedMaxWealthPerYear = maxWealthPerYear.orderBy(col("max_wealth").desc)

- sortedMaxWealthPerYear.show()

Interprétation:
la richesse maximale augmente au cours du temps(on remarque dans le résultat 2022>2023>2021>2019....

```
scala> sortedMaxWealthPerYear.show()
+----+----------+
|year|max_wealth|
+----+----------+
|2022|     219.0|
|2023|     211.0|
|2021|     177.0|
|2019|     131.0|
|2020|     113.0|
|2018|     112.0|
|2017|      86.0|
|2015|      79.2|
|2014|      76.0|
|2016|      75.0|
|2011|      74.0|
|2013|      73.0|
|2012|      69.0|
|2008|      62.0|
|2001|      58.7|
|2007|      56.0|
|2010|      53.5|
```

d.Richesse maximale par année

- import org.apache.spark.sql.functions._ (importer les librairies)

- val avgWealthPerIndustry = cleanedBillionaires
.groupBy("business_category")
.agg(sum("net_worth").as("total_wealth"),
count("net_worth").as("num_billionaires"))
.withColumn("avg_wealth", col("total_wealth") / col("num_billionaires"))
.select("business_category", "avg_wealth")

```
scala> val avgWealthPerIndustry = cleanedBillionaires.groupBy("business_category
").agg(sum("net_worth").as("total_wealth"), count("net_worth").as("num_billionai
res")) .withColumn("avg_wealth", col("total_wealth") / col("num_billionaires")).
select("business_category", "avg_wealth")
avgWealthPerIndustry: org.apache.spark.sql.DataFrame = [business_category: strin
g, avg_wealth: double]
```

avgWealthPerIndustry.show()

Interprétation:
la richesse maximale augmente au cours du temps(on remarque dans le résultat 2022>2023>2021>2019....

```
scala> avgWealthPerIndustry.orderBy(col("avg_wealth").desc).show()
+--------------------+------------------+
|   business_category|        avg_wealth|
+--------------------+------------------+
|            Politics|             20.22|
|              Gaming| 7.500000000000002|
|  Fashion and Retail| 6.109999999999998|
|            Business| 5.988636363636363|
|          Technology| 5.926490421455958|
|             Telecom|  5.77295321637427|
|         Billionaire| 5.386363636363637|
|    Fashion & Retail| 5.276185609157804|
|          Automotive| 5.120588235294114|
|     Metals & Mining| 4.959527027027023|
|Media & Entertain...| 4.510064102564103|
|  Gambling & Casinos| 4.494117647058824|
|     Food & Beverage| 4.408622828784116|
|              Energy|4.1382079459002545|
|         Diversified| 4.075399999999993|
|Finance & Investm...| 4.048097826086967|
|         Investments| 3.972247360482656|
|   Food and Beverage|3.7572362685265843|
|                null| 3.735461235666599|
```

e. les milliardaires les plus riches chaque année

- import org.apache.spark.sql.expressions.Window (importer les librairies)

- val topPerYear = cleanedBillionaires .withColumn("rank",
dense_rank().over(Window.partitionBy("year").orderBy(col("net_worth").desc
))).groupBy("year", "full_name")
.agg(max("net_worth").alias("max_net_worth"),
first("rank").alias("rank")).orderBy("year", "rank")

- topPerYear.show()

Interprétation:

on identifiant les milliardaires les plus riches chaque année , on constate qu'il ya un changement remarquable dans les derniéres années , apparition des nouveaux top riches comme **Elon musk** et ca etait la valeur maximal de net_worth a traves les anneés . aussi, **Bernard**

```
scala> val topPerYear = cleanedBillionaires  .withColumn("rank", dense_rank().ov
er(Window.partitionBy("year").orderBy(col("net_worth").desc))).groupBy("year", "
full_name") .agg(max("net_worth").alias("max_net_worth"), first("rank").alias("r
ank")).orderBy("year", "rank")
topPerYear: org.apache.spark.sql.Dataset[org.apache.spark.sql.Row] = [year: stri
ng, full_name: string ... 2 more fields]

scala> topPerYear.show()
+----+--------------------+-------------+----+
|year|           full_name|max_net_worth|rank|
+----+--------------------+-------------+----+
|1997|Chatri Sophonpani...|          2.0|   1|
|1997|King Bhumibol Adu...|          1.8|   2|
|1998|        Edmond Safra|          3.3|   1|
|1999|      Dieter Schwarz|          7.1|   1|
|1999|        Martin Ebner|          2.9|   2|
|1999|        Edmond Safra|          2.5|   3|
|1999|         Serge Kampf|          1.8|   4|
|1999| Carlos Ardila Lülle|          1.3|   5|
|1999|          Lee family|          1.1|   6|
|1999|Srichand & Gopich...|          1.0|   7|
|1999|   Jose Luis Cutrale|          1.0|   7|
|2000|    Christoph Henkel|          4.5|   1|
```

f.Richesse par groupe d'age:

- val avgWealthPerAgeGroup = billionairesWithAgeGroups.groupBy("age_group").agg(avg("net_worth").alias("average_wealth")).select("age_group", "average_wealth")

- avgWealthPerAgeGroup.show()

```
scala> val billionairesWithAgeGroups = cleanedBillionaires.withColumn("age_group
", when(col("age") < 30, "Under 30").when(col("age") >= 30 && col("age") < 50, "
30-49").when(col("age") >= 50 && col("age") < 65, "50-64").otherwise("65+"))
billionairesWithAgeGroups: org.apache.spark.sql.DataFrame = [year: string, month
: string ... 18 more fields]

scala> val avgWealthPerAgeGroup = billionairesWithAgeGroups.groupBy("age_group")
.agg(avg("net_worth").alias("average_wealth")).select("age_group", "average_weal
th")
avgWealthPerAgeGroup: org.apache.spark.sql.DataFrame = [age_group: string, avera
ge_wealth: double]

scala> avgWealthPerAgeGroup.show()
+---------+------------------+
|age_group|    average_wealth|
+---------+------------------+
|    30-49|3.6738658563056346|
|      65+| 4.425841690785437|
| Under 30| 3.439057750759882|
|    50-64|3.6706214925872813|
+---------+------------------+
```

Interprétation:

les plus riches sont dans lintervalle [65,80] ans(+65 ans)

f.Richesse par genre:

- val genderDistribution = cleanedBillionaires.groupBy("gender")
  .count()

- genderDistribution.show()

```
scala> val genderDistribution = cleanedBillionaires.groupBy("gender") .count()
genderDistribution: org.apache.spark.sql.DataFrame = [gender: string, count: big
int]

scala> genderDistribution.show()
+------+-----+
|gender|count|
+------+-----+
|  null| 3829|
|Female| 3035|
|  Male|24868|
+------+-----+


scala>
```

Interprétation:

les hommes sont nettement plus nombreux que les femmes parmi les milliardaires dans votre ensemble de données cela peut etre du à : Historiquement, les hommes ont eu un accès plus large aux opportunités économiques, à l'éducation et aux réseaux d'affaires Dans certains secteurs et cultures d'entreprise, les femmes peuvent être sous-représentées aux postes de direction et de prise de décision, ce qui limite leur capacité à accumuler des richesses à grande échelle...

10.9% des billionaires sont des femelle
89.1% des billionaires sont des males

h.nombres de milliardaires par pays:

- val billionaireCountsPerCountry = cleanedBillionaires.groupBy("country_of_citizenship")
.count()
.filter(col("count") > 1)

```
scala> val billionaireCountsPerCountry = cleanedBillionaires.groupBy("country_of
_citizenship").count().filter(col("count") > 1)
billionaireCountsPerCountry: org.apache.spark.sql.Dataset[org.apache.spark.sql.R
ow] = [country_of_citizenship: string, count: bigint]
```

- val sortedBillionaireCounts = billionaireCountsPerCountry.orderBy(col("count").desc)

- sortedBillionaireCounts.show()

```
scala> val sortedBillionaireCounts = billionaireCountsPerCountry.orderBy(col("co
unt").desc)
sortedBillionaireCounts: org.apache.spark.sql.Dataset[org.apache.spark.sql.Row]
= [country_of_citizenship: string, count: bigint]

scala> sortedBillionaireCounts.show()
+---------------------+-----+
|country_of_citizenship|count|
+---------------------+-----+
|        United States|10142|
|                China| 4162|
|              Germany| 1688|
|               Russia| 1584|
|                India| 1485|
|            Hong Kong|  902|
|       United Kingdom|  805|
|               Brazil|  730|
|               Canada|  706|
|                Italy|  597|
|               France|  574|
|                Japan|  559|
|               Turkey|  513|
```

Interprétation:

La pays ou il ya plus de milliardaires est US puis la chine puis Deutshland... c'est du au activités industrielles dans ces pays