Traitement des données massives avec Hadoop Map-Reduce

ISSA BABAN CHAWAÏ, ABDOULAYE

Table des matières

1 Présentation du projet

Dans cette étude, nous avons fait le choix de travailler sur des données massives en utilisant l'approche `MapReduce` de `Hadoop`. Nous appliquerons cette approche sur le jeu de données `Titanic` (que l'on connaît bien), disponible sur Kaggle (https://www.kaggle.com/competitions/titanic/data) afin d'en découvrir des informations cachées qui ne sont pas nécessairement perceptibles à la seule lecture de notre jeu de données.

Notre exploration de ce jeu de données a pour objectif de prédire la survie ou le décès des passagers du Titanic grâce à plusieurs variables (âge, sexe, classe sociale, ...). Nous avons donc pour objectif principal de traiter et d'analyser ce jeu de données à grande échelle afin d'en extraire des informations utiles.

Nous avons voulu pousser un peu plus loin l'analyse de ces données en faisant ressortir certaines statistiques qui permettront de mieux cerner ce qui a pu impacter la survie des passagers. Dans ce cadre, différentes thématiques ont été explorées :

— Le taux de survie par rapport au sexe, à la classe, par rapport à l'âge,

— Le nombre de survivants selon la taille de la famille,

— Les tranches d'âge,

— Le prix moyen déboursé par le voyageur selon les classes,

— Le prix moyen du billet par classe.

Ainsi, grâce à `Hadoop` et à la méthode `MapReduce`, nous avons mis en place un traitement efficace de ces données tout en mettant en œuvre quelques outils de traitement adaptés à un contexte de Big Data.

2 Dataset Information

L'ensemble de données comprend des informations sur les passagers, telles que leur nom, leur âge, leur sexe, leur classe socio-économique, etc. Le fichier `Train` que nous utilisons contiendra les informations d'un sous-ensemble des passagers à bord (891 pour être exact) et, surtout, révélera s'ils ont survécu ou non.

2.1 Dictionnaire de données

La signification de nos différentes colonnes est la suivante :

Variable	Définition	Clé
survival	Survie	0 = Non, 1 = Oui
pclass	Classe de billet	1 = 1ère, 2 = 2ème, 3 = 3ème
sex	Sexe	
age	Âge (années)	
sibsp	Nombre de frères et sœurs / conjoints à bord du Titanic	
parch	Nombre de parents/enfants à bord du Titanic	
ticket	Numéro de billet	
fare	Tarif passager	
cabin	Numéro de cabine	
embarked	Port d'embarquement	C = Cherbourg, Q = Queenstown, S = Southampton

2.2 Notes sur les variables

- **pclass** : est un indicateur du statut socio-économique (SSE). 1st renvoie à la première classe, 2nd à celle de la classe moyenne, et 3rd renvoie à la classe inférieure.
- **Âge** : L'âge est fractionnaire s'il est inférieur à 1. Si l'âge est estimé, il est au format xx.5.
- **sibsp** : L'ensemble de données définit les liens familiaux de la manière suivante :
  - **Fratrie** = frère, sœur, demi-frère, demi-sœur.
  - **Conjoint** = mari, femme (les maîtresses et les fiancés ont été ignorés).
- **parch** : L'ensemble de données définit les liens familiaux de la manière suivante :
  - **Parent** = mère, père.
  - **Enfant** = fille, fils, belle-fille, beau-fils.
- Certains enfants voyageaient uniquement avec une nounou, la valeur de leur **parch** = 0.

Table 1 – Base de donnees Titanic

ID	Surv.	Cl.	Nom	Sexe	Âge	SibSp	Parch	Ticket	Tarif	Cabine	Emb.
1	0	3	Braund, Mr. Owen Harris	M	22	1	0	A/5 21171	7.25	-	S
2	1	1	Cumings, Mrs. John Bradley	F	38	1	0	PC 17599	71.28	C85	C
3	1	3	Heikkinen, Miss. Laina	F	26	0	0	STON/O2. 3101282	7.93	-	S
4	1	1	Futrelle, Mrs. Jacques Heath	F	35	1	0	113803	53.10	C123	S
5	0	3	Allen, Mr. William Henry	M	35	0	0	373450	8.05	-	S
6	0	3	Moran, Mr. James	M	-	0	0	330877	8.46	-	Q
7	0	1	McCarthy, Mr. Timothy J	M	54	0	0	17463	51.86	E46	S
8	0	3	Palsson, Master. Gosta Leonard	M	2	3	1	349909	21.08	-	S
9	1	3	Johnson, Mrs. Oscar W	F	27	0	2	347742	11.13	-	S

2.3 Data Format

Notre jeu de données que nous allons utiliser dans le cadre de ce projet est sous la forme d'un fichier `csv` et se présente comme suit :

Légende

— **ID** : Identifiant du passager

— **Survie** : 0 = Décédé, 1 = Survivant

— **Classe** : 1 = Première classe, 2 = Deuxième classe, 3 = Troisième classe

— **SibSp** : Nombre de frères/sœurs/époux/épouses à bord

— **Parch** : Nombre de parents/enfants à bord

— **Embarq.** : Port d'embarquement (S = Southampton, C = Cherbourg, Q = Queenstown)

— **-** : Données manquantes

3 Bibliothèques utilisées :

Pour réaliser ce projet, nous utiliserons le langage de programmation Java, accompagné des bibliothèques ci-dessous :

3.1 Classes de types de données Hadoop (Writable) :

• **org.apache.hadoop.io.Text** : Cette classe stocke le texte en utilisant le codage UTF8 standard. Il fournit des méthodes pour sérialiser, désérialiser et comparer des textes au niveau des octets. Cette classe Hadoop est l'équivalent de String en Java. Elle est utilisée pour stocker les types de données texte sérialisables dans Hadoop.

• **org.apache.hadoop.io.IntWritable** : Cette classe d'Entrée/Sortie générique est utilisée lors de la lecture et de l'écriture de données sur le réseau, dans des bases de données et dans des fichiers. C'est l'équivalent Hadoop du type de donnée Integer utilisé en Java qui est un type de données sérialisables pour stocker des entiers et utilisé couramment pour les compteurs ou valeurs numériques.

• **org.apache.hadoop.io.LongWritable** : Nous avons utilisé cette classe d'Entrée/Sortie générique utilisée pour la lecture et l'écriture de données sur le réseau, dans les bases de données, etc. Elle est l'équivalent Hadoop de Long en Java utilisée pour le stockage de données entières de type Long. Elle est souvent utilisée pour les offsets de fichiers ou des identifiants.

• **org.apache.hadoop.io.NullWritable** : Cette classe est utilisée lors de la lecture et de l'écriture de données sur le réseau, dans des bases de données et dans des fichiers. Elle représente une valeur nulle sérialisable, elle est utilisée quand on n'a besoin que d'une clé ou d'une valeur (pas les deux) et elle est utile pour les opérations où seule la clé compte.

3.2 Les classes de traitement MapReduce :

Pour réaliser ce projet, nous utiliserons le langage de programmation Java, accompagné des bibliothèques ci-dessous :

• **org.apache.hadoop.mapreduce.Mapper** : Elle mappe les paires clé/valeur d'entrée à un ensemble de paires clé/valeur intermédiaires. Les classes mapper sont des classes de tâches individuelles qui transforment les enregistrements d'entrée en enregistrements intermédiaires. C'est une classe de base abstraite pour créer des 'mappers', elle permet de définir la logique de transformation des données d'entrée la méthode principale : map() permet de traiter chaque enregistrement. •

org.apache.hadoop.mapreduce.Reducer : Cette classe récupère les paires clé/valeurs générées par le Mapper afin de les regrouper par clé, et d'appliquer une logique de réduction tel que la somme, la moyenne, le maximum, etc.) à toutes les valeurs associées à chaque clé a sa méthode principale qui est la fonction reduce()

3.3 La gestion des exceptions :

• **java.io.IOException** : C'est une classe d'exception standard Java pour les erreurs d'entrée/sortie. Elle est présente dans les méthodes MapReduce car elles manipulent des fichiers. Elle permet de gérer les erreurs d'entrée/sortie.

4 Stockage et analyse des données :

Le modèle utilisé est celui du MapReduce. MapReduce fournit un modèle de programmation qui abstrait le problème des lectures et écritures sur le disque, le transformant en un calcul sur des ensembles de clés et de valeurs. MapReduce est un processeur de requêtes par lots, et la possibilité d'exécuter une requête ad hoc sur l'ensemble du jeu de données et d'obtenir des résultats dans un délai raisonnable est révolutionnaire.

5 Pourquoi pas un système de gestion de bases de données relationnelles (SGBDR) ?

MapReduce est idéal pour les problèmes nécessitant l'analyse par lots de l'ensemble des données, notamment pour les analyses ad hoc. Un SGBDR est idéal pour les requêtes ou les mises à jour ponctuelles, où l'ensemble de données a été indexé pour offrir une récupération et des temps de mise à jour à faible latence pour une quantité relativement faible de données. MapReduce convient aux applications où les données sont écrites une fois et lues plusieurs fois, tandis qu'une base de données relationnelle est adaptée aux ensembles de données continuellement mis à jour.[7] Les Framework de traitement distribué comme MapReduce évitent au programmeur de se soucier des pannes, car l'implémentation détecte les tâches en échec et reprogramme les remplacements sur les machines qui fonctionnent correctement.

6 Analyse des données avec Hadoop Map et Reduce

MapReduce décompose le traitement en deux phases : la phase de mappage (map) et la phase de réduction (reducer). Chaque phase utilise des paires clé-valeur en entrée et en sortie. Dans chacun de nos codes MapReduce, nous avons quatre classes :

• **La classe 'Parser' :** La classe 'Parser' qui nous permet de récupérer les différentes colonnes qui nous intéressent et que nous aurons besoin dans notre analyse, de faire un pré-traitement de données pour rendre ces données aptes à l'utilisation dans les prochaines étapes de notre projet .

• **La classe 'mapper' :** La classe 'mapper' est en quelque sorte une classe de préparation dans le but de formater les données pour que la classe 'reducer' puisse faire son travail. Les valeurs de sortie de la fonction map sont utilisés par le Framework MapReduce avant de les envoyer à la fonction 'reduce'. Cette utilisation, permet de regrouper les pairs clé-valeur par clé. .

• **La classe 'reducer' :** a pour but de récupérer les combinaisons clés- valeurs de sortie de la classe mapper et de produire des valeurs-clés de sortie qui nous permettra de mieux comprendre nos données comme : le taux de survie par sexe, par classe, par âge... Il est à noter que les types d'entrée de notre fonction 'reduce' doivent correspondre aux types sorties de notre fonction map .

• **La classe 'driver' :** est en quelque sorte comme un chef d'orchestre qui permet d'exécuter les différentes classe afin d'en produire les résultats escomptés.

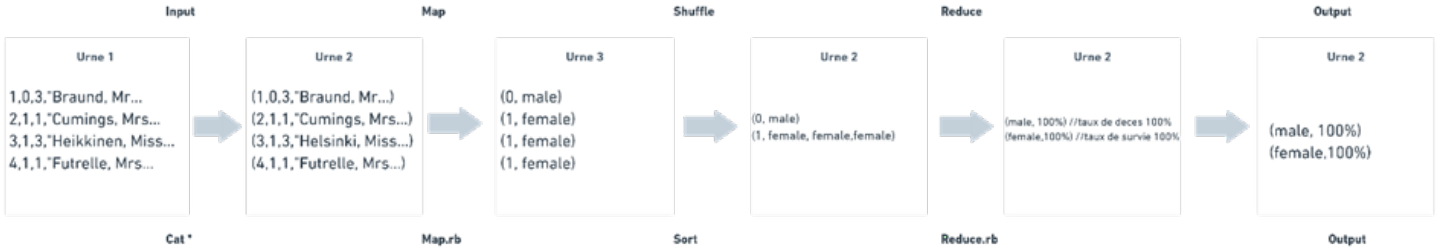

FIGURE 1 – Flux de données logiques MapReduce

La classe 'driver' est en quelque sorte comme un chef d'orchestre qui permet d'exécuter les différentes classe afin d'en produire les résultats escomptés. Les processus entre les différentes classes peuvent être résumé comme suit :

7 Prétraitement :

Cette phase est primordiale avant de réaliser des opérations ou requêtes sur nos données, car elle vise à nettoyer et à formater le jeu de données pour s'adapter à l'analyse. Pour assurer la qualité et la cohérence de données traitées, nous avons traité les colonnes dont les valeurs étaient nulles ou vides en utilisant diverses fonctions parmi lesquelles :

7.1 Parser CSV intelligent :

Cette fonction gère les virgules à l'intérieur des guillemets qui apparaissent dans les colonnes (comme dans la colonne des noms) .

Puisque nos données sont formatées en CSV, il nous faut un parseur qui puisse délivrer chaque ligne sous la forme d'éléments, en considérant la virgule comme séparateur. Cependant, certaines lignes peuvent contenir des virgules placées au sein de champs entre guillemets, ce qui rendrait impossible un découpage correct. Dans le but d'éviter une interprétation erronée, nous avons besoin d'un parseur qui ne tienne pas compte des virgules présentes à l'intérieur des guillemets. La ligne suivante nous permet de faire cela :

```
String[] champs = ligne.split(",(?=(?:[^\"]*\"[^\"]*\")*[^\"]*$)");
```

Un exemple concret est la colonne des noms qui peut être caractérisé par : **"Braund, Mr. Owen Harris"**.

Ici nous avons le nom et le prénom du voyageur séparé par une virgule. A cet effet, on s'assure que notre parseur ne considère pas cette colonne en deux champs : l'un constitue de **"Braund"** et l'autre de **"Mr. Owen Harris"** .

7.2 Gestion d'erreurs :

Elle permet d'ignorer les lignes malformées et l'en-tête. Lorsque nous récupérons une colonne, on s'assure que cette colonne ne contient pas des valeurs non valides ou nulles. A cet effet nous utilisons des ''try and catch '' pour ignorer des lignes qui pourront nous causer des problèmes plus tard lors de la manipulation de nos données.

```
            try {
                totalAge += Double.parseDouble(data[3]);
                nombreAge++;
            } catch (NumberFormatException e) {
                // Ignorer
            }
```

A chaque fois qu'on parse notre jeu de données, on ignore la première ligne qui contient l'entête de nos données grâce à cette boucle if.

```
                if (ligne.startsWith("PassengerId") || ligne.isEmpty()) {
                    return;
                }
```

7.3 Extraction des Titres de civilité :

Nous nous devons de distinguer les hommes des femmes et des femmes mariées des femmes non mariées dans le but de faire ressortir les informations selon le sexe des voyageurs :

```
        String[] parts = nom.split(",");
        if (parts.length > 1) {
            String secondPart = parts[1].trim();
            if (secondPart.contains(".")) {
                return secondPart.substring(0, secondPart.indexOf(".")).trim();
            }
        }
```

Supposons que notre colonne **"nom"** se présente sous cette forme : **"Braund, Mr. Owen Harris"**

Ici on se donne pour objectif de récupérer la deuxième partie celle contenant le titre de civilité du voyageur **"Mr. Owen Harris"** . On retourne la partie de notre chaine de caractère qui contient le point (.) qui est **"Mr."**

7.4 Suppressions des espaces blancs :

Les espaces blancs dans nos données sont caractérisés par des espaces, des tabulations, des retours à la ligne, etc. au début et à la fin d'une chaîne de caractères. Grâce à la fonction ''trim()", nous éliminons les espaces blancs dans nos données.

```
    String ligne = value.toString().trim();
```

7.5 Ignorer l'entête dans les données :

Nous contrôlons le nombre de colonne que nous récupérons de nos données grâce à cette condition.

```
    if (fields.length >= 7) {...}
```

En général, cette condition dépend de la quantité des colonnes que nous voulons récupérer de notre ensemble de données. Si par exemple on veut récupérer juste 3 colonnes alors notre condition devient :

```
    if (fields.length >= 3) {...}
```

7.6 Valeurs manquantes :

Certaines colonnes peuvent avoir des valeurs manquantes ou alors renvoyer des valeurs nulles, nous nous devons de traiter ces valeurs. Dans notre cas de figure, on contrôle les différentes valeurs de notre colonne ''Nom" par :

```
    if (nom == null || nom.isEmpty()) {
            return "";  // Retourne une chaîne vide si le nom est null ou vide
    }
```

De la même manière on s'assure que les valeurs des colonnes nom, sexe, age ne sont pas nulles. Si nulles alors retourner une chaine vide :

```
private  String extraireNom(String nom){
    if (nom ==null || nom.isEmpty()) {
        return nom =’”’ }
    If (sexe ==null) nom =’”’;
    If (age ==null) nom =’”’;
```

8 L'analyse des données proprement dite :

8.1 Analyse Familialle :

Nous avons utilisé le modèle MapReduce dans cette section d'analyse pour étudier la structure familiale des passagers du Titanic, en utilisant deux colonnes de notre jeu de données :

- **SibSp :** le nombre de frères, sœurs ou conjoints à bord avec le passager.
- **Parch :** la présence éventuelle (ou pas) de parents (père/mère) ou d'enfants à bord avec le passager.

En combinant ces deux informations, nous avons ensuite traité le potentiel rôle de la structure familiale sur la survie des passagers du Titanic. La finalité du processus MapReduce que nous avons conçu se propose d'étudier la structure familiale des passagers du Titanic à partir des relations de parenté qui les unissent. En d'autres termes, il s'agit de regrouper les passagers par classement de leur famille et de leur statut dans celle-ci (enfant, parent, conjoint, etc.). A cette fin, nous nous appuyons sur les colonnes suivantes du jeu de données :

```
        String sexe = fields[4].replace("\"", "").trim(); // Colonne Sex
        int survie = Integer.parseInt(fields[1].trim());
        int classe = Integer.parseInt(fields[2].trim());
        String age =  fields[5].trim();
        int sibSp = Integer.parseInt(fields[6].trim());
        int parch = Integer.parseInt(fields[7].trim());
        String nom = fields[3].trim();
```

Ces colonnes représentent successivement les colonnes : sexe, survie, classe, age, sibSp (frères/sœurs/conjoints), parch(parents/enfan et nom.

8.1.1 Catégories de familles :

Nous avons mis en place 4 catégories de voyageurs : les voyageurs se déplaçant seul, à petit nombre, à moyen nombre et en groupe plus large.

- **Seul :** Voyageurs seuls (Taille de famille = 1).
- **Petit :** Petites familles (Taille de famille = 2 personnes).
- **Moyenne :** Familles moyennes (Taille de famille = 3-4 personnes).
- **Grande :** Grandes familles (Taille de famille = 5+ personnes).

```
    private String getCategorieFamille(int grandeurFamille) {
        if (grandeurFamille == 1) {
            return "Seul";
        } else if (grandeurFamille == 2) {
            return "Petit";
        } else if (grandeurFamille <= 4) {
            return "Moyen";
        } else {
            return "Grande";
        }

```

8.1.2 Rôles familiaux :

Nous avons également déterminé les rôles familiaux de chaque voyageur et nous les avons partagés en sous-groupes suivants :

- **Les enfants :** qui sont caractérisé par un âge < 18 ou par le titre "Master"/"Miss" pour dire qu'ils sont célibataires.
- **Les parents :** qui sont des adultes avec enfants, dont l'âge est supérieur ou égale à 25 **(âge $>=$ 25)** et dont la valeur de la colonne ''Parch" est superieur à 0 **(Parch $>$ 0)**.
- **Les époux(ses) :** qui sont des voyageurs ayant des conjoints et dont la valeur de la colonne 'SibSp est supérieure à 0 **(SibSp $>$ 0)**.
- **Les adultes seuls.** .

```
private String getRoleFamille(String nom, String sexe, String age, int sibSp, int parch) {
        if (age != null && !age.isEmpty()) {
            try {
                double age = Double.parseDouble(ageIntervalle);
                // Enfant
                if (age < 18) {
                    return "Enfant";
                }
                // Adulte avec enfants
                if (parch > 0 && age >= 25) {
                    return "Parent";
                }
                // Marié sans enfants ou avec conjoint
                if (sibSp > 0) {
                    return "Epoux";
                }
                // sinon renvoyer Adulte seul
                return "Adulte";

            } catch (NumberFormatException e) {
                // Si l'âge n'est pas parsable, utiliser le titre
            }
        }

```

Nos statistiques sur le taux de survie, la moyenne d'âge et la taille moyenne des familles sont donnés par les formules suivantes :

$$\left(tauxSurvie = \frac{survivant}{nombreTotalVoyageur}\right) * 100$$

$$(moyenneAge = \frac{SommeDesPrix}{nombreDesPrix}) * 100$$

$$\left(moyenneTailleFamille = \frac{totalGrandeurFamille}{nombreTotalVoyageur}\right) * 100$$

8.1.3 Résultats :

TABLE 2 – Statistiques de survie par catégorie de passagers

Catégorie	Effectif	Survivants	Taux (%)	H/F	Cl. 1/2/3	Âge moy.	Taille fam. moy.
Grande_Adulte	6	0	0.0	0/6	0/0/6	0.0	8.0
Grande_Enfant	34	3	8.8	22/12	0/0/34	7.5	6.5
Grande_Epoux	10	5	50.0	4/6	5/1/4	21.4	7.2
Grande_Parent	12	2	16.7	3/9	1/1/10	43.3	6.2
Moyen_Adulte	10	9	90.0	0/10	3/2/5	21.3	3.2
Moyen_Enfant	43	35	81.4	24/19	6/16/21	5.7	3.3
Moyen_Epoux	28	11	39.3	18/10	3/10/15	28.3	3.3
Moyen_Parent	50	25	50.0	21/29	19/16/15	37.7	3.2
Petit_Adulte	11	7	63.6	3/8	4/1/6	20.6	2.0
Petit_Enfant	18	15	83.3	4/14	5/3/10	11.2	2.0
Petit_Epoux	112	56	50.0	58/54	50/25/37	33.5	2.0
Petit_Parent	20	11	55.0	9/11	11/5/4	43.4	2.0
Seul_Adulte	488	133	27.3	399/89	106/98/284	33.2	1.0
Seul_Enfant	45	27	60.0	12/33	2/6/37	15.3	1.0
Seul_Epoux	4	3	75.0	0/4	1/0/3	0.0	1.0

8.1.4 Représentation graphique :

FIGURE 2 – Analyse Familiale

Nos graphes peuvent être interprété somme il suit :

- Les familles moyennes (constituées de 3-4 personnes) comportant essentiellement des adultes ont une meilleure chance de survie soit 90%.
- Les grandes familles constituées d'adultes ont le plus mauvais taux de survie soit 0.0%.
- Les grandes familles constituées d'enfants ont un taux de survie de 8.8%. .
- Le taux de survie des grandes familles est de 9.9% comme l'illustre le graphe 'taux de survie par taille famille'.
- Le taux de survie des enfants est de 57.1% qui représente le plus grand taux de survie par groupe d'âge.

8.2 Analyse économique :

Les résultats découlant de l'exécution de cette classe MapReduce nous permettront d'analyser les différents aspects économiques des passagers à bord du Titanic au moyen d'un traitement sur différentes variables telles que : la classe, le tarif, la cabine et le port d'embarquement. Pour ce faire, nous allons récupérer les données des colonnes suivantes :

```
String sexe = fields[4].replace("\"", "").trim(); // Colonne Sex
int survie = Integer.parseInt(fields[1].trim());   // Colone Survived
int classe = Integer.parseInt(fields[2].trim());
int age = Integer.parseInt(fields[5].trim());
int sibSp = Integer.parseInt(fields[6].trim());
int parch = Integer.parseInt(fields[7].trim());
String nom = fields[3].trim();
String ticket = fields[8].trim();
String fareStr = fields[9].trim(); // le prix
String cabin = fields[10].trim();
String embarked = fields[11].trim();
```

8.2.1 Catégorisation des classes :

Nous avons opté de déterminer la catégorie économique en se basant sur la classe et tarif payé par le voyageur. Comme exemple, si le voyageur appartient à la classe 1(Pclass = 1) et le prix qu'il a payé est supérieure à 100 alors le voyageur appartient à la classe luxe.

- **Luxe :** 1ère classe avec un tarif supérieur (>) à £100.
- **Première classe :** 1ère classe avec un tarif entre £50-100.
- **Classe Moyenne supérieure :** 1ère classe avec un tarif inférieur (<) £50.
- **Classe moyenne :** 2e classe avec un tarif supérieur (>) £25 .
- **Classe Moyenne inférieure :** 2e classe avec un tarif inférieur (<) £25.
- **Classe ouvrière :** 3e classe avec un tarif supérieur (>) £15.
- **Classe à Faible revenu :** 3e classe avec un tarif entre £7-15.
- **Classe à Très faible revenu :** 3e classe avec un tarif inférieur (<) £7.

```
private String getCategorieEconomie(int classe, double fare) {
    if (classe == 1) {
        if (fare > 100) return "Luxe";
        else if (fare > 50) return "PremiereClasse";
        else return "UpperMiddle";
    } else if (classe == 2) {
        if (fare > 25) return "MiddleClass";
        else return "LowerMiddle";
    } else {
        if (fare > 15) return "WorkingClass";
        else if (fare > 7) return "FaibleRevenu";
        else return "VeryLowIncome";
    }
}
```

8.2.2 Catégorisation des sieges :

Nous avons aussi voulu catégoriser les sièges en les répartissant en 5 différentes catégories :

- **Pont supérieur (A, B) :** Ponts supérieurs.
- **Pont intermédiaire (C, D) :** Ponts moyens.
- **Pont inférieur (E, F) :** Ponts inférieurs.
- **Pont inférieur (G) :** Pont le plus bas
- **Sans cabine :** Pas de cabine assignée .

```
private String getTypeCabine(String cabine) {
        if (cabine == null || cabine.trim().isEmpty()) {
            return "NoCabin";
        }

        char deck = cabine.charAt(0);
        switch (deck) {
            case 'A': return "PontLePlusHaut";
            case 'B': return "PontSuperieur";
            case 'C': return "PontSuperieurBas";
            case 'D': return "PontIntermediaire";
            case 'E': return "PontIntermédiaireBas";
            case 'F': return "PontInferieur";
            case 'G': return "PontLePlusBas";
            default: return "PasDeCabine";
        }
    }

```

8.2.3 Catégorisation des ports :

Selon le port ou le voyageur est embarqué, ceci aura aussi une influence sur le prix dont il payera.

```
    private String getCategoriePort(String embarked) {
        switch (embarked.trim().toUpperCase()) {
            case "C": return "Cherbourg"; // Port français, souvent plus cher
            case "Q": return "Queenstown"; // Port irlandais, souvent moins cher
            case "S": return "Southampton"; // Port principal anglais
            default: return "Unknown";
        }
    }
```

Nos statistiques sur le taux de survie, la moyenne des prix et la taille moyenne des familles sont donnés par les formules suivantes :

$$\left(tauxSurvie = \frac{survivant}{nombreTotalVoyageur}\right) * 100$$

$$\left(MoyennePrix = \frac{SommeDesPrix}{nombreDesPrix}\right) * 100$$

$$\left(moyenneTailleFamille = \frac{TotalTailleFamille}{nombreDesFamilles}\right) * 100$$

8.2.4 Résultats :

Table 3 – Statistiques des passagers du Titanic par classe et port d'embarquement

Classe_Pont_Port	Nbre	Survécu (%)	H/F	Tarif Moy.	Gamme Tarifs	Taille Fam.
PremiereClasse_PontInferieur_Cherbourg	3	3 (100,0%)	1/2	£65,18	£55,44-£83,16	2,0
PremiereClasse_PontInferieur_Southampton	10	6 (60,0%)	5/5	£62,50	£51,86-£79,65	2,2
PremiereClasse_PontIntermediaire_Cherbourg	12	12 (100,0%)	4/8	£77,36	£63,36-£89,10	1,8
PremiereClasse_PontIntermediaire_Queenstown	2	1 (50,0%)	1/1	£90,00	£90,00-£90,00	2,5
PremiereClasse_PontIntermediaire_Southampton	21	13 (61,9%)	12/9	£67,61	£51,48-£90,00	2,0
PremiereClasse_PasDeCabine_Cherbourg	3	1 (33,3%)	2/1	£67,65	£59,40-£82,17	2,0
PremiereClasse_PasDeCabine_Southampton	3	2 (66,7%)	1/2	£60,95	£52,00-£78,85	1,7
PremiereClasse_PontSuperieur_Cherbourg	11	8 (72,7%)	6/5	£72,11	£56,93-£91,08	1,7
PremiereClasse_PontSuperieur_Southampton	12	9 (75,0%)	5/7	£73,91	£50,50-£93,50	1,9
ClasseFaibleRevenu_PontInferieur_Southampton	6	3 (50,0%)	5/1	£9,33	£7,65-£12,48	1,3

Suite sur la page suivante

Table 3 – suite de la page précédente

Classe_Pont_Port	Nbre	Survécu (%)	H/F	Tarif Moy.	Gamme Tarifs	Taille Fam.
ClasseFaibleRevenu_PontLePlusBas_Southampton	2	0 (0,0%)	0/2	£10,46	£10,46-£10,46	2,5
ClasseFaibleRevenu_PasDeCabine_Cherbourg	24	9 (37,5%)	16/8	£9,77	£7,23-£14,46	1,5
ClasseFaibleRevenu_PasDeCabine_Queenstown	12	6 (50,0%)	6/6	£7,78	£7,73-£8,03	1,0
ClasseFaibleRevenu_PasDeCabine_Southampton	189	38 (20,1%)	150/39	£8,39	£7,05-£14,50	1,2
ClasseMoyenInférieure_PontInferieur_Southampton	5	4 (80,0%)	0/5	£11,00	£10,50-£13,00	1,0
ClasseMoyenInférieure_PontIntermediaire_Cherbourg	2	1 (50,0%)	1/1	£13,34	£12,88-£13,79	1,0
ClasseMoyenInférieure_PontIntermediaire_Southampton	2	2 (100,0%)	1/1	£13,00	£13,00-£13,00	1,0
ClasseMoyenInférieure_PasDeCabine_Cherbourg	5	3 (60,0%)	2/3	£17,78	£12,00-£24,00	1,6
ClasseMoyenInférieure_PasDeCabine_Queenstown	2	1 (50,0%)	1/1	£12,35	£12,35-£12,35	1,0
ClasseMoyenInférieure_PasDeCabine_Southampton	87	32 (36,8%)	57/30	£13,77	£10,50-£23,00	1,4
ClasseLuxe_PontInferieur_Cherbourg	2	2 (100,0%)	0/2	£134,50	£134,50-£134,50	2,0
ClasseLuxe_PontIntermediaire_Cherbourg	13	8 (61,5%)	6/7	£138,38	£106,43-£227,53	2,2
ClasseLuxe_PontIntermediaire_Southampton	11	6 (54,5%)	3/8	£192,36	£135,63-£263,00	3,7
ClasseLuxe_PasDeCabine_Cherbourg	4	3 (75,0%)	1/3	£245,48	£106,43-£512,33	1,0
ClasseLuxe_PasDeCabine_Southampton	3	3 (100,0%)	1/2	£150,02	£133,65-£164,87	2,3
ClasseLuxe_PontSuperieur_Cherbourg	7	6 (85,7%)	3/4	£313,00	£146,52-£512,33	2,6
ClasseLuxe_PontSuperieur_Southampton	7	7 (100,0%)	2/5	£159,15	£120,00-£211,34	3,0
ClasseMoyenne_PontInferieur_Southampton	4	4 (100,0%)	3/1	£32,50	£26,00-£39,00	3,5
ClasseMoyenne_PasDeCabine_Cherbourg	7	4 (57,1%)	4/3	£36,65	£27,72-£41,58	3,1
ClasseMoyenne_PasDeCabine_Southampton	53	28 (52,8%)	25/28	£33,53	£26,00-£73,50	2,4
ClasseMoyenneSuperieure_PontInferieur_Southampton	9	6 (66,7%)	9/0	£27,67	£25,59-£38,50	1,0
ClasseMoyenneSuperieure_PontIntermediaire_Cherbourg	5	2 (40,0%)	3/2	£33,13	£27,75-£49,50	1,2
ClasseMoyenneSuperieure_PontIntermediaire_Southampton	12	7 (58,3%)	7/5	£29,54	£25,93-£39,40	1,3
ClasseMoyenneSuperieure_PasDeCabine_Cherbourg	4	2 (50,0%)	3/1	£33,69	£26,55-£49,50	1,0
ClasseMoyenneSuperieure_PasDeCabine_Southampton	9	3 (33,3%)	9/0	£29,54	£26,00-£47,10	1,0
ClasseMoyenneSuperieure_PontSuperieur_Cherbourg	10	6 (60,0%)	7/3	£34,90	£27,72-£49,50	1,3
ClasseMoyenneSuperieure_PontSuperieur_Southampton	8	3 (37,5%)	7/1	£26,76	£5,00-£35,50	1,0
ClasseMoyenneSuperieure_PontNonConnu_Southampton	1	0 (0,0%)	1/0	£35,50	£35,50-£35,50	1,0
TresFaibleRevenu_PasDeCabine_Cherbourg	1	0 (0,0%)	1/0	£4,01	£4,01-£4,01	1,0
TresFaibleRevenu_PasDeCabine_Queenstown	2	0 (0,0%)	1/1	£6,75	£6,75-£6,75	1,0
TresFaibleRevenu_PasDeCabine_Southampton	10	2 (20,0%)	10/0	£6,61	£6,24-£6,98	1,1
ClasseOuvriere_PontLePlusBas_Southampton	2	2 (100,0%)	0/2	£16,70	£16,70-£16,70	3,0
ClasseOuvriere_PasDeCabine_Cherbourg	8	6 (75,0%)	3/5	£17,26	£15,25-£19,26	2,8
ClasseOuvriere_PasDeCabine_Queenstown	7	0 (0,0%)	5/2	£25,24	£15,50-£29,13	5,1
ClasseOuvriere_PasDeCabine_Southampton	75	16 (21,3%)	43/32	£28,35	£15,55-£56,50	4,4

8.2.5 Réprésentation Graphique

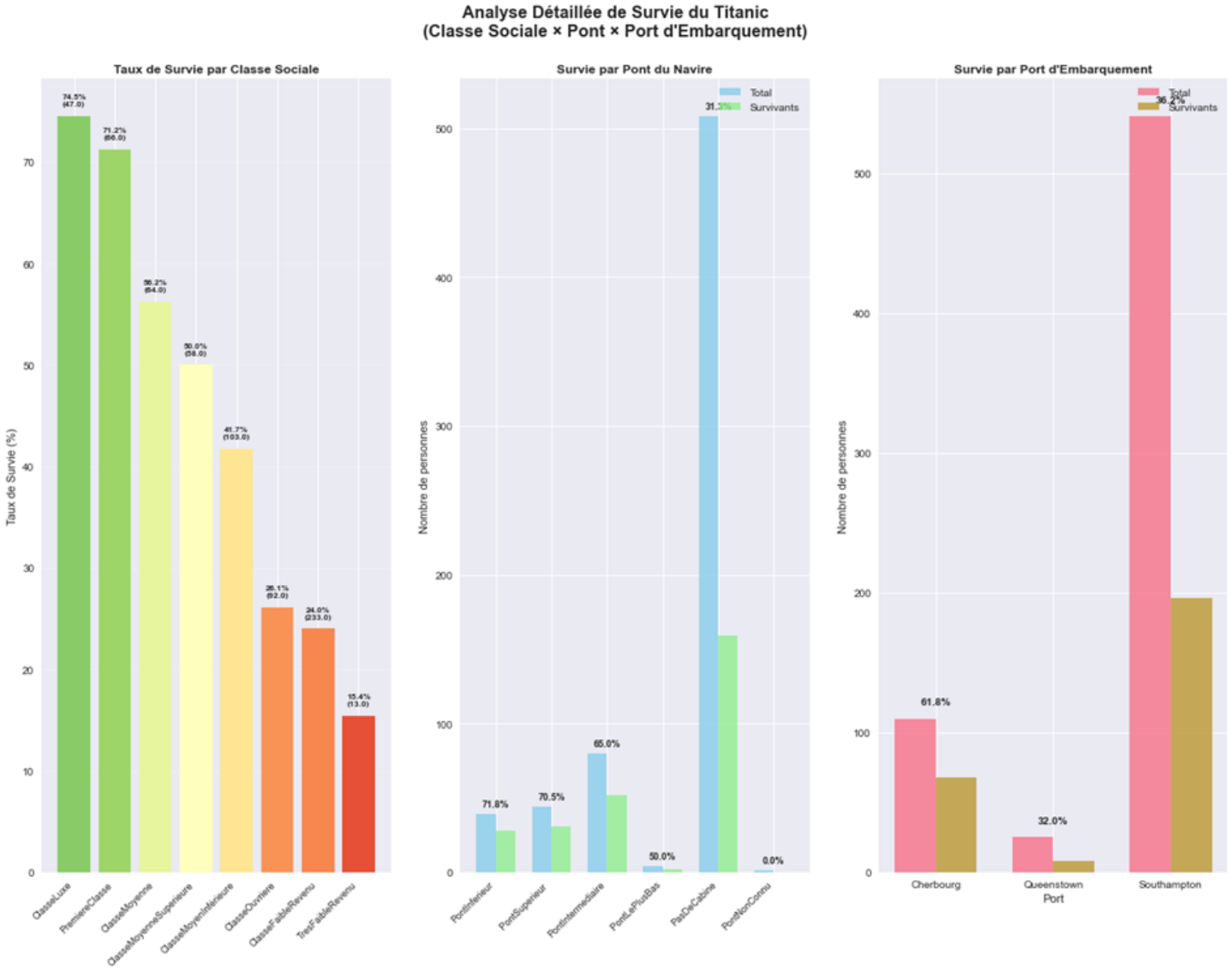

FIGURE 3 – Analyse Economique

Nos graphes nos donnent encore plus de précisions de taux de survie par classe , par groupe social et selon le port d'embarquement :

- Les personnes se trouvant dans la classe de luxe ont un meilleur taux de survie (74.5%) tant dis que les personnes appartenant à la classe des très faibles revenu ont le taux de survie le plus bas soit 15.4%
- Le port d'embarquement de Cherbourg, est le port avec le plus haut taux de survie soit 61.8% et le port de Queenstown a un taux de survie de 32.0%.
- Le pont inférieur a le meilleur taux de survie 71.8% et le pont le plus bas a taux de survie le plus bas qui est de 50.0% si on exclut le pont inconnu car il y'a qu'une seule personne dont on ne connait pas le pont .
- Les familles moyennes (constituées de 3-4 personnes) comportant essentiellement des adultes ont une meilleure chance de survie soit 90%.
- Les grandes familles constituées d'adultes ont le plus mauvais taux de survie soit 0.0%.
- Les grandes familles constituées d'enfants ont un taux de survie de 8.8%.
- Le taux de survie des grandes familles est de 9.9% comme l'illustre le graphe 'taux de survie par taille famille'.
- Le taux de survie des enfants est de 57.1% qui représente le plus grand taux de survie par groupe d'âge.

8.3 Survie selon le sexe :

Cette classe MapReduce nous permettra de comparer le taux de survie par sexe des voyageurs. Nous allons récupérer les données des colonnes qui nous permettra de conclure si la personne a survécu ou elle est morte et la colonne sexe de chaque voyageur :

```
1           String sexe = fields[4].replace("\"", "").trim(); // Colonne Sexe (male ou female)
2           int survie = Integer.parseInt(fields[1].trim());   // Colonne survie (0 ou 1)
```

Nos statistiques sur le taux de survie et le taux de décès selon le sexe des voyageurs sont donnés par les formules suivantes :

$$\left(tauxSurvie = \frac{survivant}{nombreTotalVoyageur}\right) * 100$$

$$\left(tauxDeces = \frac{nombreDeMort}{nombreTotalVoyageur}\right) * 100$$

Les résultats découlant de cette classe MapReduce sont les suivantes :

8.3.1 Résultats :

TABLE 4 – Taux de survie par sexe - Titanic

Sexe	Total	Survivants	Pourcentage de Survie
Femme	314	233	74,20%
Homme	577	109	18,89%

8.3.2 Réprésentation Graphique :

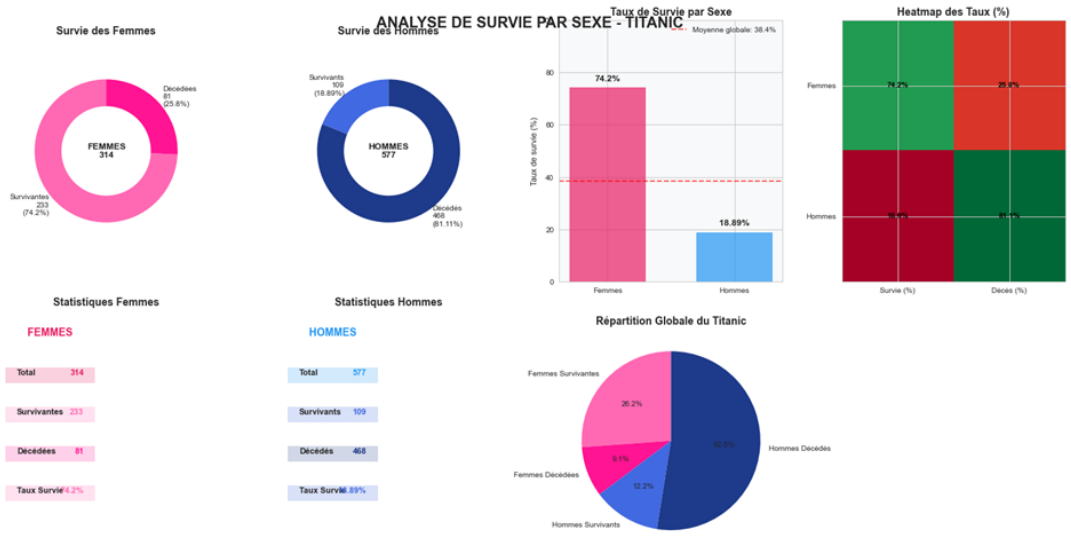

FIGURE 4 – Survie selon le sexe

Grace à nos résultats qui sont illustrés sur le graphe , on constate que :

- Les femmes ont un meilleur taux de survie (74.2%) comparé à celui des hommes (18.89%).
- Sur 314 femmes, 233 ont survécus et sur 577 hommes, 109 ont survécus.

8.4 Segmentation des classes selon l'âge

Le but de cette classe MapReduce est de trouver des paramètres qui nous permettront de faire des interprétations sur les relations entre la classe, l'âge et le taux de survie. Nous voulons savoir si le fait d'appartenir à une certaine classe et d'avoir un certain âge a un avantage sur la survie du voyageur. Nous aurons à cet effet besoin de récupérer les colonnes : survie, classe et âge.

```
                    int survie = Integer.parseInt(fields[1].trim());   // Colonne Survived
                    int classe = Integer.parseInt(fields[2].trim());
                    //int age = Integer.parseInt(fields[5].trim());
                    String age =  fields[5].trim();
```

Notre taux de survie et de décès selon les classes et les catégories d'âge sont calculés par les formules suivantes :

$$\left(tauxSurvie = \frac{mort}{nombreTotalVoyageur}\right) * 100$$

$$\left(tauxDeces = \frac{SommeDesPrix}{nombreDesPrix}\right) * 100$$

$$\left(moyenneTailleFamille = \frac{TotalTailleFamille}{nombreDesFamilles}\right) * 100$$

8.4.1 Résultats

Les résultats de sortie de notre 'reducer' se présente comme suit :

TABLE 5 – Statistiques de survie du Titanic par classe et groupe d'âge

Classe_Groupe d'âge	Passagers	Survivants	Taux de Survie
Classe 1			
Class1_Non spécifié	30	14	46,7%
Class1_Adolescent	17	14	82,4%
Class1_Enfant	4	3	75,0%
Class1_Jeune	52	38	73,1%
Class1_Moyen âge	86	57	66,3%
Class1_Vieillard	27	10	37,0%
Classe 2			
Class2_Non spécifié	11	4	36,4%
Class2_Adolescent	18	9	50,0%
Class2_Enfant	17	17	100,0%
Class2_Jeune	85	37	43,5%
Class2_Moyen âge	45	18	40,0%
Class2_Vieillard	8	2	25,0%
Classe 3			
Class3_Non spécifié	136	34	25,0%
Class3_Adolescent	60	16	26,7%
Class3_Enfant	48	20	41,7%
Class3_Jeune	178	42	23,6%
Class3_Moyen âge	62	6	9,7%
Class3_Vieillard	7	1	14,3%

Avec taux de survie (par défaut) : Montre la distribution et le taux de survie par segment
Cette segmentation révélera des patterns intéressants comme l'avantage de la première classe et l'impact de l'âge sur la survie.

8.4.2 Réprésentation Graphique :

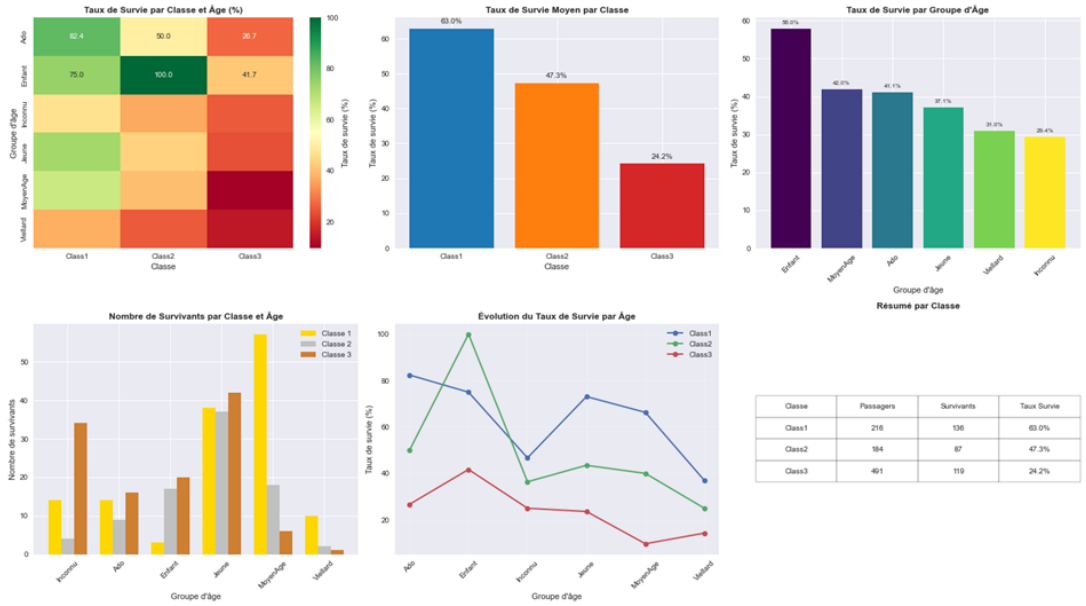

FIGURE 5 – Segmentation des classes selon l'âge

Ici, d'après nos résultats :

- les voyageurs voyageant en classe 1 ont un meilleur taux de survie (63.0%).
- Tous les enfants appartenant à la classe 2 ont survécu (100.0%).
- Plus notre âge avance moins on a des chances de survivre d'après le graphe 'évolution du taux de survie par âge'.
- Les enfants ont le meilleur taux de survie.

Références