



République du Sénégal
Un Peuple – Un But – Une Foi
Ministère l'Enseignement Supérieur de la
Recherche et de l'Innovation
École Polytechnique de Thiès
B.P.A 10 Thiès

Tél: (221) 76 223 61 74 – Fax: (221) 33 951 14 67

RAPPORT PROJET MODELISATION STOCHASTIQUE

Présenté par :

Mohamed Massamba
SENE

Professeur
Dr. Mamadou Thiongane

Matière
Modélisation Stochastique

Table des matières

Introduction.....	3
I. Prétraitement des données.....	4
II. Création d'un jeu de données contenant les métriques.....	8
III. Création du modèle ANN	22

Introduction

Dans ce travail, notre objectif consiste à créer des prédicteurs de délai capable de prédire avec une bonne précision les temps d'attente des clients au niveau d'un centre d'appel.

Du point de vue du service client, les prédictions de temps d'attente permettent aux clients de mieux planifier leur emploi du temps, réduisant ainsi leur frustration et leur insatisfaction. En étant informés du temps approximatif qu'ils devront patienter, les clients se sentent plus pris en considération, ce qui améliore leur expérience globale avec l'entreprise.

Développer des prédicteurs de délai précis pour prédire le temps d'attente des clients dans les entreprises et les centres d'appels est essentiel pour améliorer l'expérience client, optimiser l'efficacité opérationnelle et renforcer la performance globale de l'entreprise. Cela favorise une meilleure gestion des ressources, une satisfaction client accrue et une compétitivité renforcée sur le marché.

I. Prétraitement des données

Pour commencer nous effectuons un prétraitement sur les données consistant à apporter les transformations suivantes :

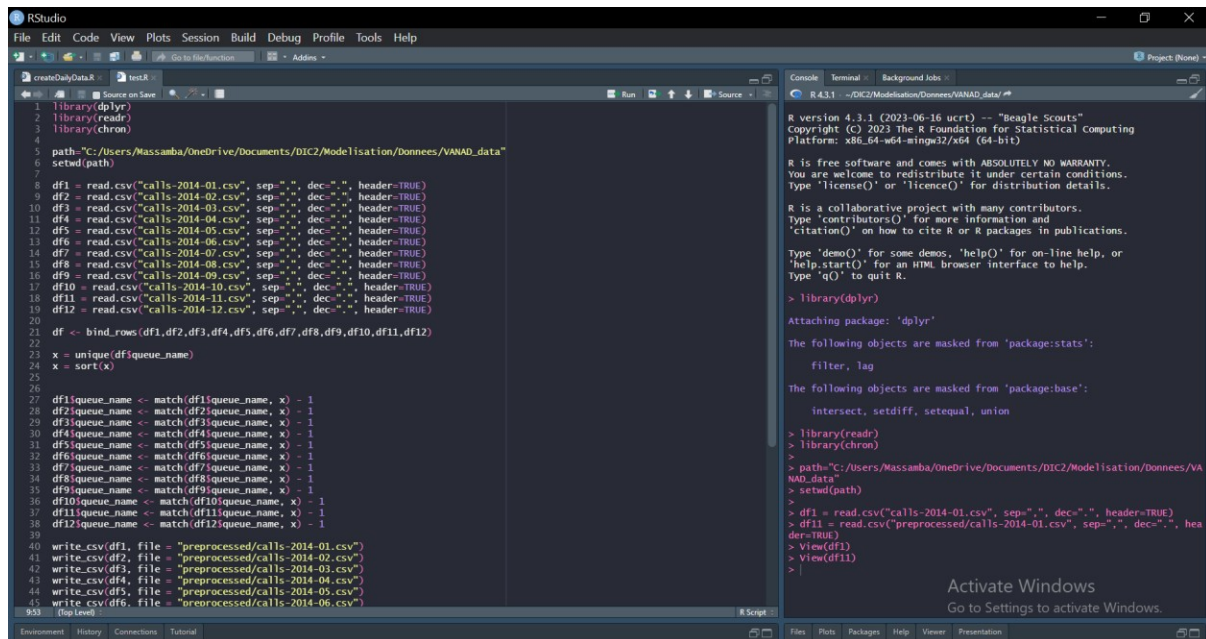
- **Transformer les types d'appels de sorte à obtenir des valeurs[] entre 0 et 26**

Pour effectuer cette opération, nous avons commencé par charger tous les fichiers de calls allant de Janvier à Décembre dans des dataframes distincts. On les regroupe ensuite en un seul et on crée une liste contenant tous les types présents que l'on trie.

Pour finir sur les dataframes initiaux on établit un mapping entre le numéro de type d'appel et la position dans la liste.

Ceci nous permet d'obtenir au final nos nouveaux dataframes avec les types numérotés de 1 à 27.

Le mapping peut être trouvé dans le fichier envoyé en pièce jointe.



```
1 library(dplyr)
2 library(readr)
3 library(chron)
4
5 path="C:/Users/Massamba/OneDrive/Documents/DIC2/Modelisation/Donnees/VANAD_data"
6 setwd(path)
7
8 df1 = read.csv("calls-2014-01.csv", sep=";", dec=".", header=TRUE)
9 df2 = read.csv("calls-2014-02.csv", sep=";", dec=".", header=TRUE)
10 df3 = read.csv("calls-2014-03.csv", sep=";", dec=".", header=TRUE)
11 df4 = read.csv("calls-2014-04.csv", sep=";", dec=".", header=TRUE)
12 df5 = read.csv("calls-2014-05.csv", sep=";", dec=".", header=TRUE)
13 df6 = read.csv("calls-2014-06.csv", sep=";", dec=".", header=TRUE)
14 df7 = read.csv("calls-2014-07.csv", sep=";", dec=".", header=TRUE)
15 df8 = read.csv("calls-2014-08.csv", sep=";", dec=".", header=TRUE)
16 df9 = read.csv("calls-2014-09.csv", sep=";", dec=".", header=TRUE)
17 df10 = read.csv("calls-2014-10.csv", sep=";", dec=".", header=TRUE)
18 df11 = read.csv("calls-2014-11.csv", sep=";", dec=".", header=TRUE)
19 df12 = read.csv("calls-2014-12.csv", sep=";", dec=".", header=TRUE)
20
21 df <- bind_rows(df1,df2,df3,df4,df5,df6,df7,df8,df9,df10,df11,df12)
22
23 x = unique(df$queue_name)
24 x = sort(x)
25
26
27 df1$queue_name <- match(df1$queue_name, x) - 1
28 df2$queue_name <- match(df2$queue_name, x) - 1
29 df3$queue_name <- match(df3$queue_name, x) - 1
30 df4$queue_name <- match(df4$queue_name, x) - 1
31 df5$queue_name <- match(df5$queue_name, x) - 1
32 df6$queue_name <- match(df6$queue_name, x) - 1
33 df7$queue_name <- match(df7$queue_name, x) - 1
34 df8$queue_name <- match(df8$queue_name, x) - 1
35 df9$queue_name <- match(df9$queue_name, x) - 1
36 df10$queue_name <- match(df10$queue_name, x) - 1
37 df11$queue_name <- match(df11$queue_name, x) - 1
38 df12$queue_name <- match(df12$queue_name, x) - 1
39
40 write_csv(df1, file = "preprocessed/calls-2014-01.csv")
41 write_csv(df2, file = "preprocessed/calls-2014-02.csv")
42 write_csv(df3, file = "preprocessed/calls-2014-03.csv")
43 write_csv(df4, file = "preprocessed/calls-2014-04.csv")
44 write_csv(df5, file = "preprocessed/calls-2014-05.csv")
45 write_csv(df6, file = "preprocessed/calls-2014-06.csv")
```

```
R version 4.3.1 (2023-06-16 ucrt) -- "Beagle Scouts"
Copyright (C) 2023 The R Foundation for Statistical Computing
Platform: x86_64-w64-mingw32/x64 (64-bit)

R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.

R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

> library(dplyr)

Attaching package: 'dplyr'

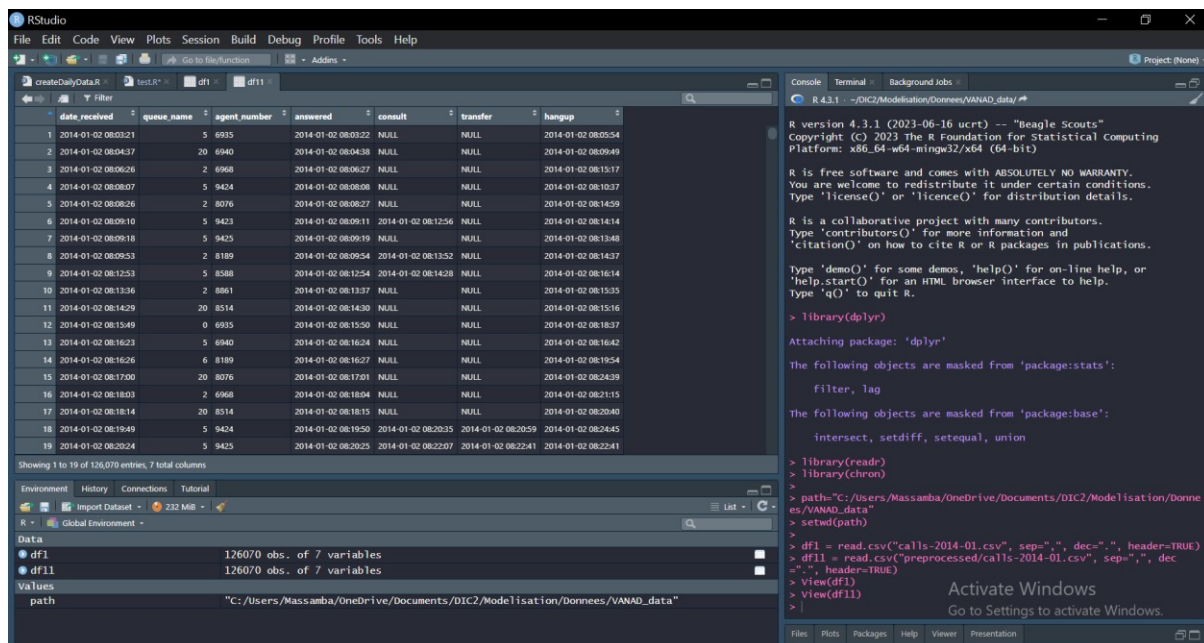
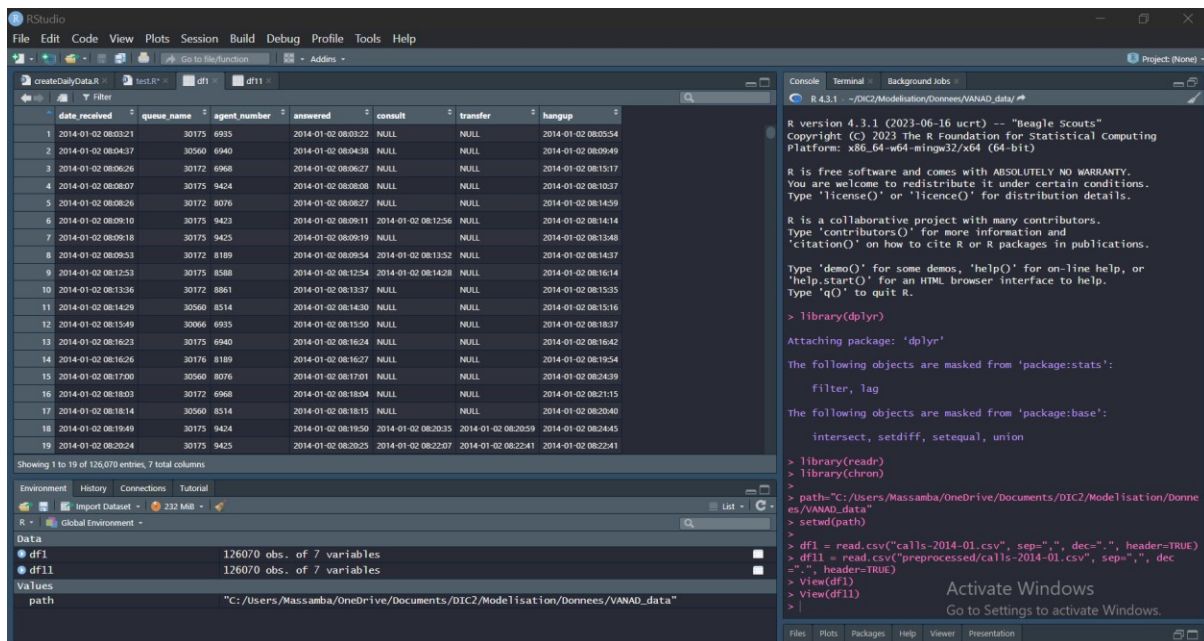
The following objects are masked from 'package:stats':

  filter, lag

The following objects are masked from 'package:base':

  intersect, setdiff, setequal, union

> library(readr)
> library(chron)
>
> path="C:/Users/Massamba/OneDrive/Documents/DIC2/Modelisation/Donnees/VANAD_data"
> setwd(path)
>
> df1 = read.csv("calls-2014-01.csv", sep=";", dec=".", header=TRUE)
> df11 = read.csv("preprocessed/calls-2014-01.csv", sep=";", dec=".", header=TRUE)
> View(df1)
> View(df11)
> |
```



➤ Grouper les données de chaque mois par jour

A la fin de la première étape nous obtenons donc un fichier pour chaque mois contenant le nouveau dataframe avec les types modifiés.

Dans cette partie pour chacun de ces fichiers, nous allons tenter de les séparer en plusieurs fichiers contenant les données journalières pour le mois en question.

C'est-à-dire pour le mois de Janvier nous aurons un fichier pour chaque journée de travail.

Pour réaliser ceci, nous bouclons sur la colonne date_received en formattant pour récupérer spécifiquement le jour afin de récupérer tous les jours distincts.

Ensuite pour chacun de ces jours, nous extrayons un sous df ne contenant que les enregistrements du jour que nous écrivons ensuite dans un fichier.

```

1 library(readr)
2
3 path="C:/Users/Massamba/OneDrive/Documents/DIC2/Modelisation/Donnees/VANAD_data/preprocessed"
4 setwd(path)
5
6 df1 = read.csv("calls-2014-12.csv", sep=";", dec=".", header=TRUE)
7 df1 = na.omit(df1)
8
9 path="12"
10 setwd(path)
11 i = 1
12 for (date in unique(as.Date(df1$date_received))) {
13   subset_df = df1[as.Date(df1$date_received) == date,]
14   subset_df$date_received = as.POSIXct(subset_df$date_received, format="%Y-%m-%d %H:%M:%S")
15   subset_df = subset_df[order(subset_df$date_received),]
16   subset_df$date_received <- format(subset_df$date_received, "%Y-%m-%d %H:%M:%S")
17   file_name <- paste0(i, ".csv")
18   write_csv(subset_df, file = file_name)
19   i <- i + 1;
20 }
21
22
23
24

```

Environment: 113012 obs. of 7 variables (df1), 2580 obs. of 7 variables (subset_df). Variables: date, file_name, i, path.

date_received	queue_name	agent_number	answered	consult	transfer	hangup
2014-12-31 08:03:12	2	6922	2014-12-31 08:03:13	NULL	NULL	2014-12-31 08:05:30
2014-12-31 08:04:09	5	1610	2014-12-31 08:04:10	NULL	NULL	2014-12-31 08:09:23
2014-12-31 08:04:17	5	1618	2014-12-31 08:04:18	NULL	NULL	2014-12-31 08:09:55
2014-12-31 08:04:28	25	1569	2014-12-31 08:04:29	2014-12-31 08:05:22	2014-12-31 08:05:33	2014-12-31 08:06:41
2014-12-31 08:05:42	20	1641	2014-12-31 08:05:42	NULL	NULL	2014-12-31 08:10:07
2014-12-31 08:07:19	5	6925	2014-12-31 08:07:20	NULL	NULL	2014-12-31 08:08:27
2014-12-31 08:08:07	5	1045	2014-12-31 08:08:08	NULL	NULL	2014-12-31 08:15:15
2014-12-31 08:08:14	5	6922	2014-12-31 08:08:15	NULL	NULL	2014-12-31 08:10:18
2014-12-31 08:09:06	5	1569	2014-12-31 08:09:07	2014-12-31 08:15:33	NULL	2014-12-31 08:16:34
2014-12-31 08:09:27	2	6925	2014-12-31 08:09:28	NULL	NULL	2014-12-31 08:11:12
2014-12-31 08:09:47	5	1610	2014-12-31 08:09:48	NULL	NULL	2014-12-31 08:10:15
2014-12-31 08:09:56	5	1618	2014-12-31 08:09:57	2014-12-31 08:10:46	2014-12-31 08:11:13	2014-12-31 08:12:07
2014-12-31 08:10:38	20	1647	2014-12-31 08:10:39	NULL	NULL	2014-12-31 08:11:31
2014-12-31 08:10:48	20	1640	2014-12-31 08:10:48	NULL	NULL	2014-12-31 08:12:07
2014-12-31 08:10:50	20	1639	2014-12-31 08:10:51	NULL	NULL	2014-12-31 08:12:07
2014-12-31 08:11:00	5	1610	2014-12-31 08:11:01	2014-12-31 08:15:16	2014-12-31 08:16:38	2014-12-31 08:22:38
2014-12-31 08:11:25	26	6922	2014-12-31 08:11:26	NULL	NULL	2014-12-31 08:13:12
2014-12-31 08:11:27	5	6925	2014-12-31 08:11:28	NULL	NULL	2014-12-31 08:14:57
2014-12-31 08:11:47	2	6922	2014-12-31 08:11:47	NULL	NULL	2014-12-31 08:15:06
2014-12-31 08:11:50	5	1618	2014-12-31 08:11:51	NULL	NULL	2014-12-31 08:15:54
2014-12-31 08:12:14	30	1646	2014-12-31 08:12:15	NULL	NULL	2014-12-31 08:26:18

Ce qui au final nous donne donc 12 dossiers pour chaque mois avec dans chaque dossier un fichier csv pour chaque journée. Comme on peut le voir ci-dessous.

File Explorer window showing the directory structure of the 'preprocessed' folder. The address bar shows the path: C:\Users\Massamba\OneDrive\Documents\DIC2\Modélisation\Donnees\VANAD_data\preprocessed.

The left sidebar shows the navigation pane with 'Documents' selected. The main pane displays a list of files and folders:

Name	Date modified	Type	Size
1	6/24/2023 4:32 PM	File folder	
2	6/24/2023 4:33 PM	File folder	
3	6/24/2023 4:34 PM	File folder	
4	6/24/2023 4:35 PM	File folder	
5	6/24/2023 4:36 PM	File folder	
6	6/24/2023 4:38 PM	File folder	
7	6/24/2023 4:40 PM	File folder	
8	6/24/2023 4:41 PM	File folder	
9	6/24/2023 4:44 PM	File folder	
10	6/24/2023 4:45 PM	File folder	
11	6/24/2023 4:47 PM	File folder	
12	6/24/2023 4:49 PM	File folder	
RDData	6/23/2023 6:20 PM	R Workspace	1 KB
Rhistory	6/28/2023 9:56 PM	R History Source File	1 KB
calls-2014-01.csv	6/24/2023 4:30 PM	Comma Separated V...	10,575 KB
calls-2014-02.csv	6/24/2023 4:30 PM	Comma Separated V...	9,566 KB
calls-2014-03.csv	6/24/2023 4:30 PM	Comma Separated V...	11,256 KB
calls-2014-04.csv	6/24/2023 4:30 PM	Comma Separated V...	10,645 KB
calls-2014-05.csv	6/24/2023 4:30 PM	Comma Separated V...	11,589 KB
calls-2014-06.csv	6/24/2023 4:30 PM	Comma Separated V...	12,012 KB
calls-2014-07.csv	6/24/2023 4:30 PM	Comma Separated V...	12,124 KB

26 items | 12 items selected

File Explorer window showing the contents of the '1' subfolder. The address bar shows the path: C:\Users\Massamba\OneDrive\Documents\DIC2\Modélisation\Donnees\VANAD_data\preprocessed\1.

The left sidebar shows the navigation pane with 'Documents' selected. The main pane displays a list of CSV files:

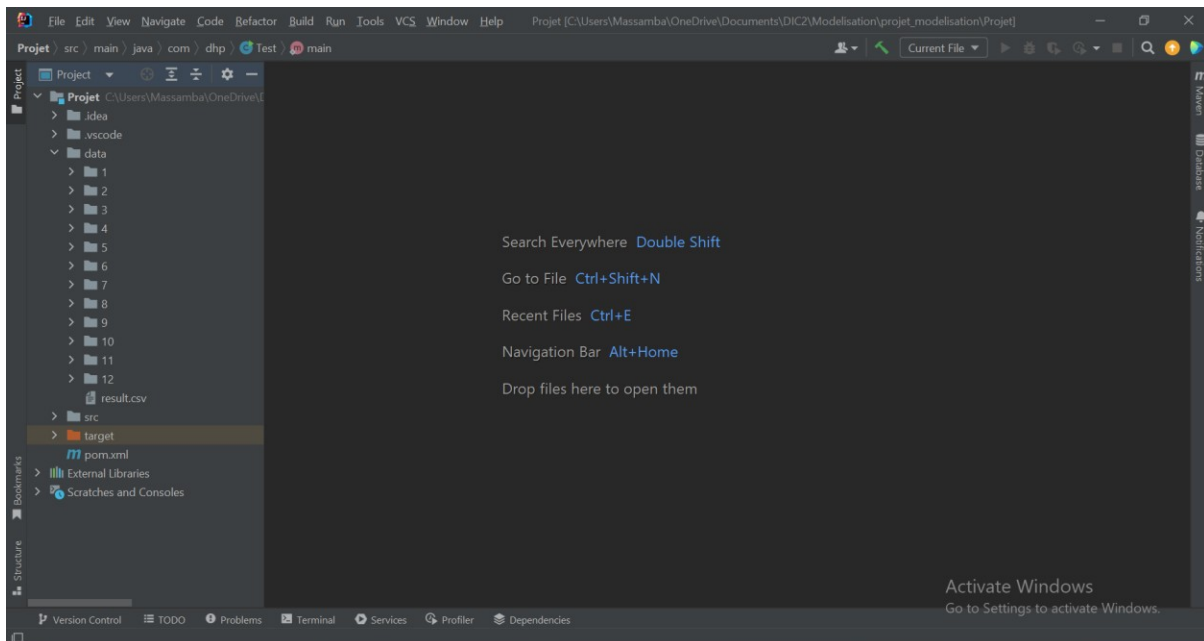
Name	Date modified	Type	Size
1.csv	6/24/2023 4:31 PM	Comma Separated V...	391 KB
2.csv	6/24/2023 4:31 PM	Comma Separated V...	343 KB
3.csv	6/24/2023 4:31 PM	Comma Separated V...	24 KB
4.csv	6/24/2023 4:31 PM	Comma Separated V...	580 KB
5.csv	6/24/2023 4:32 PM	Comma Separated V...	463 KB
6.csv	6/24/2023 4:32 PM	Comma Separated V...	443 KB
7.csv	6/24/2023 4:32 PM	Comma Separated V...	416 KB
8.csv	6/24/2023 4:32 PM	Comma Separated V...	375 KB
9.csv	6/24/2023 4:32 PM	Comma Separated V...	20 KB
10.csv	6/24/2023 4:32 PM	Comma Separated V...	506 KB
11.csv	6/24/2023 4:32 PM	Comma Separated V...	477 KB
12.csv	6/24/2023 4:32 PM	Comma Separated V...	446 KB
13.csv	6/24/2023 4:32 PM	Comma Separated V...	418 KB
14.csv	6/24/2023 4:32 PM	Comma Separated V...	384 KB
15.csv	6/24/2023 4:32 PM	Comma Separated V...	19 KB
16.csv	6/24/2023 4:32 PM	Comma Separated V...	544 KB
17.csv	6/24/2023 4:32 PM	Comma Separated V...	515 KB
18.csv	6/24/2023 4:32 PM	Comma Separated V...	530 KB
19.csv	6/24/2023 4:32 PM	Comma Separated V...	501 KB
20.csv	6/24/2023 4:32 PM	Comma Separated V...	474 KB
21.csv	6/24/2023 4:32 PM	Comma Separated V...	21 KB

26 items | 26 items selected 10.3 MB

II. Création d'un jeu de données contenant les métriques

Nous créons donc un nouveau projet dans lequel l'objectif est de rejouer l'activité du centre d'appel à l'aide de nos données prétraitées.

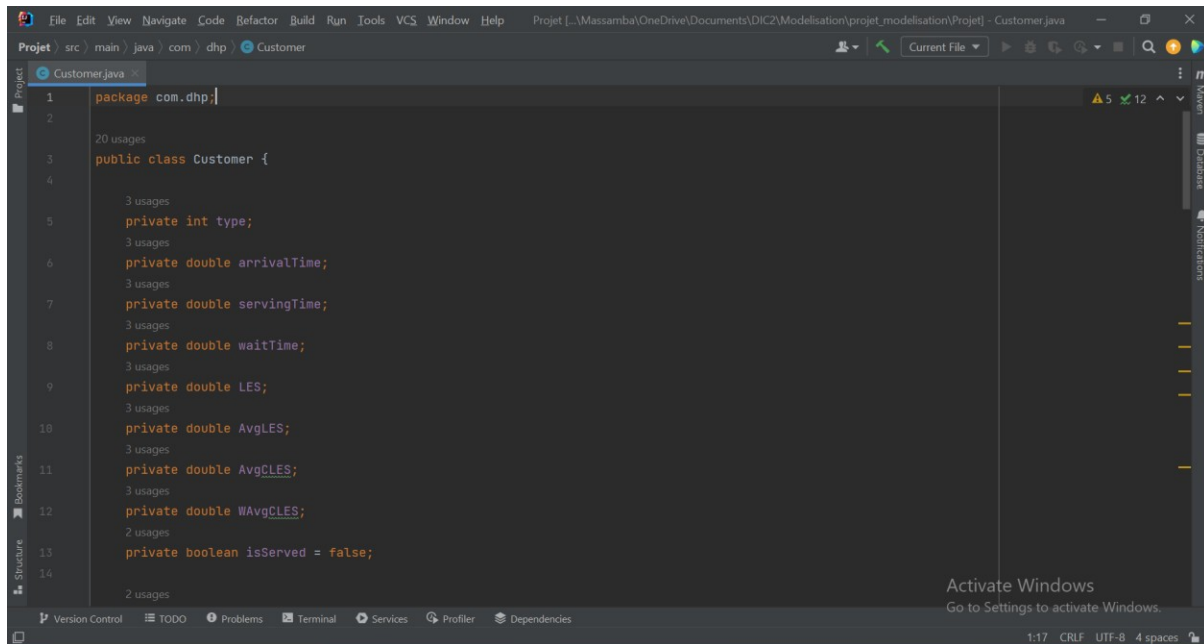
Donc on commence par enregistrer nos données dans le répertoire data au niveau du projet



On crée ensuite les classes dont nous avons besoin pour rejouer l'activité :

➤ **Customer**

Nous commençons donc par créer la classe Customer qui représente les clients ayant passé des appels au niveau du centre. Nous ajoutons tous les attributs nécessaires type, LES, AvgLES, AvgCLES, ... ainsi que les getters et setters et redéfinissons la méthode toString().



```
1 package com.dhp;
2
3 public class Customer {
4
5     private int type;
6     private double arrivalTime;
7     private double servingTime;
8     private double waitTime;
9     private double LES;
10    private double AvgLES;
11    private double AvgCLES;
12    private double WAvgCLES;
13    private boolean isServed = false;
14 }
```

➤ ReplayOneDay

Cette classe nous permet de rejouer l'activité d'une journée au niveau du centre. Pour cela on ajoute d'abord les attributs nous permettant de suivre l'activité comme waitList, servedCustomers, abandonedCustomers qui contiennent l'ensemble des clients en attente, servis et ayant abandonnés. Nous avons également nbServeurs pour le nombre de serveurs occupés, lengthsQtype qui contient la taille de la file pour chaque type. Nous avons également arrayLES, arrayAvgLES, arrayAvgCLES et arrayWAvgCLES qui sont des listes contenant pour chaque type d'appels le LES du dernier client, les LES des 10 derniers clients, les LES des 100 derniers clients ayant trouvé la même longueur de queue et le WAvgCLES du dernier client ayant trouvé la même longueur de queue respectivement.

On ajoute également des variables pour le temps de la simulation TIME, PERIODE et HOUR.

On crée le constructeur en initialisant les listes chaînées LinkedList() au niveau arrayAvgLES et arrayAvgCLES ainsi que les valeurs de WAvgCLES à -1 pour les premiers clients.

```

11
12 public class ReplayOneDay {
13     static final double HOUR = 3600;
14     static final int PERIODE = 13;
15     static final double TIME = PERIODE * HOUR;
16     private LinkedList<Customer> waitList = new LinkedList<Customer>();
17     private ArrayList<Customer> servedCustomers = new ArrayList<Customer>();
18     private ArrayList<Customer> abandonedCustomers = new ArrayList<Customer>();
19     private int nbServeurs;
20     private int[] lenghtsQtype = new int[27];
21     private double[] arrayLES = new double[27];
22     private LinkedList[] arrayAvgLES = new LinkedList[27];
23     private LinkedList[][] arrayAvgCLES = new LinkedList[27][100];

```

```

17     private ArrayList<Customer> servedCustomers = new ArrayList<Customer>();
18     private ArrayList<Customer> abandonedCustomers = new ArrayList<Customer>();
19     private int nbServeurs;
20     private int[] lenghtsQtype = new int[27];
21     private double[] arrayLES = new double[27];
22     private LinkedList[] arrayAvgLES = new LinkedList[27];
23     private LinkedList[][] arrayAvgCLES = new LinkedList[27][100];
24     private double[][] arrayWAvGCLES = new double[27][100];
25
26     public ReplayOneDay() {
27         for (int i=0; i<27; i++) {
28             arrayAvgLES[i] = new LinkedList<Double>();
29             for (int j=0; j<100; j++) {
30                 arrayAvgCLES[i][j] = new LinkedList<Double>();
31                 arrayWAvGCLES[i][j] = -1.0;
32             }
33         }
34     }

```

Nous créons également des méthodes qui vont nous permettre d'effectuer des calculs et qui seront réutiliser dans nos autres méthodes :

➤ **getTime()**

Etant donné que les dates dans le csv sont au format « yyyy-dd-mm HH :MM :SS », il nous faut convertir l'heure en secondes. Pour cela on utilise cette méthode qui va effectuer la conversion en séparant d'abord la date et l'heure. On effectue le calcul en utilisant 7 heures comme repère étant donné qu'il y a des enregistrements qui commencent à 7h30 et on renvoie le résultat.

```

9 usages
public double getTime(String s) {
    String s1 = s.split( regex: " ")[1];
    String[] time = s1.split( regex: ":");
    return (Integer.parseInt(time[0])-7)*3600+Integer.parseInt(time[1])*60+Integer.parseInt(time[2]);
}

```

➤ **getWaitingTime()**

On calcule le temps d'attente du Customer c. Si l'appel ait été pris, on récupère la différence entre le temps le temps où l'appel a été pris et le temps où il a été reçu. Sinon on récupère la différence entre le temps où le client à raccrocher et le temps où le client a été reçu.

11

```

1 usage
public double getWaitingTime(String arrival, String answered, String hangup) {
    if (!answered.equals("NULL")) {
        return getTime(answered) - getTime(arrival);
    }
    return getTime(hangup) - getTime(arrival);
}

```

➤ **getAvgLES()**

Cette méthode nous permet de retourner l'avgLES du Customer c en calculant la moyenne des valeurs contenues dans arrayAvgLES qui correspond ici à la liste chaînée contenant les 10 derniers LES des clients pour un type donné.

```

2 usages
public double getAvgLES(LinkedList<Double> arrayAvgLES) {
    double sum = 0;
    for (double val: arrayAvgLES) {
        sum += val;
    }
    return sum/arrayAvgLES.size();
}

```

➤ **getAvgCLES()**

Cette méthode nous permet de retourner l'avgCLES du Customer c en calculant la moyenne des valeurs contenues dans array_avgCLES qui correspond ici à la liste chaînée contenant les 100 derniers LES des clients pour un type et une longueur de queue donnée.

```

1 usage
public double getAvgCLES(LinkedList[] arrayAvgCLES, int queueLength) {
    LinkedList<Double> array_avgCLES = arrayAvgCLES[queueLength];
    return getAvgLES(array_avgCLES);
}

```

➤ **getWAvGCLES()**

Cette méthode nous permet de retourner le wAvGCLES du Customer c qui correspond à la valeur contenue dans la liste des WAvGCLES pour un type et une longueur de queue donnée

```

public double getWAvGCLES(int type, int queueLength) { return arrayWAvGCLES[type][queueLength]; }

```

➤ **setAvgLES()**

Cette méthode nous permet de mettre à jour arrayAvgLES. Etant donné que l'on garde en mémoire les 10 derniers, lorsque la taille est inférieure à 10 on se contente d'ajouter le waitTime du Customer c à la liste chaînée du type donnée. Si ce n'est pas le cas, on supprime la valeur en première position et on ajoute le waitTime du Customer c en dernière position.

```

public void setAvgLES(double waitTime, int type) {
    if (arrayAvgLES[type].size() < 10) {
        arrayAvgLES[type].addLast(waitTime);
    } else {
        arrayAvgLES[type].removeFirst();
        arrayAvgLES[type].addLast(waitTime);
    }
}

```

➤ **setAvgCLES()**

Cette méthode nous permet de mettre à jour arrayAvgCLES. Etant donnée que l'on garde en mémoire les 100 derniers, lorsque la taille est inférieure à 100 on se contente d'ajouter le waitTime du Customer c à la liste chaînée pour le type et queueLength donnés. Si ce n'est pas le cas, on supprime la valeur en première position et on ajoute le waitTime du Customer c en dernière position

```

public void setAvgCLES(double waitTime, int type, int queueLength) {
    if (arrayAvgCLES[type][queueLength].size() < 100) {
        this.arrayAvgCLES[type][queueLength].addLast(waitTime);
    } else {
        this.arrayAvgCLES[type][queueLength].removeFirst();
        this.arrayAvgCLES[type][queueLength].addLast(waitTime);
    }
}
}

```

➤ **setWAvgCLES()**

Cette méthode nous permet de mettre à jour arrayWAvgCLES. Si la valeur est -1 alors on la met à jour avec le temps d'attente du client. A défaut, on applique la formule de calcul à la valeur se trouvant actuellement dans la liste pour le type et queueLength donnés. Puis on la met à jour avec la valeur nouvellement calculée

```

1 usage
179 public void setWAvgCLES(int type, int queueLength, double waitTime) {
180     double alpha = 0.2;
181     if (this.arrayWAvgCLES[type][queueLength] == -1.0) {
182         arrayWAvgCLES[type][queueLength] = waitTime;
183     } else {
184         double wavgCLES = (1-alpha)*arrayWAvgCLES[type][queueLength] + alpha*waitTime;
185         this.arrayWAvgCLES[type][queueLength] = wavgCLES;
186     }
187 }

```

➤ **getServedCustomer()**

Cette méthode permet à la fin de la journée de récupérer une liste contenant tous les clients ayant été servis en retournant servedCustomers.

```

public ArrayList<Customer> getServedCustomers() { return servedCustomers; }

```

On crée ensuite la méthode **createCustomerOfDay()** qui prend en entrée le fichier csv contenant les données sur la journée et permet de créer les clients de la journée ainsi que de programmer leur arrivée.

Pour cela on parcourt le fichier et pour chaque ligne on crée un nouveau Customer c, on sépare les informations contenues dans la ligne que l'on place dans une liste. Ensuite en utilisant ces informations, on calcule et met à jour le type et l'arrivalTime et waitTime de c. Si le client a été servi on met à jour sa variable isServed et son servingTime.

Pour finir on programme son arrivée après une durée correspondant à son arrivalTime.


```

public void createCustomerOfDay(String file) throws IOException {
    BufferedReader br = new BufferedReader(new FileReader(file));
    br.readLine();
    String line = br.readLine();
    while (line != null) {
        Customer c = new Customer();
        String[] tab = line.split(regex: ",");
        c.setArrivalTime(getTime(tab[0]));
        c.setType(Integer.parseInt(tab[1]));
        c.setWaitTime(getWaitingTime(tab[0], tab[3], tab[6]));
        if (!tab[3].equals("NULL")) {
            c.setIsServed(true);
            try {
                c.setServingTime(getTime(tab[6]) - getTime(tab[3]));
            }
            catch (ArrayIndexOutOfBoundsException e) {
                c.setServingTime(getTime(tab[3]) - getTime(tab[0]));
            }
        }
        new Arrival(c).schedule(c.getArrivalTime());
        line = br.readLine();
    }
    br.close();
}

```

On crée une classe **Arrival** qui nous permet en héritant d'Event et en implémentant la méthode actions() de définir les décisions à prendre lorsqu'un nouveau Customer c arrive. On crée donc un attribut pour le customer et on l'initialise dans le constructeur.

Au niveau de la méthode actions(), lorsque c arrive on commence par mettre à jour ses attributs LES, avgLES,... en utilisant les getters que nous avons précédemment créé ainsi que le nombre de serveurs et la taille des files.

On ajoute ensuite le c à la file d'attente et on met à jour la taille de la file pour le type de c.

Pour finir on programme son départ de la file après une durée correspondant au waitTime de c.


```

public class Arrival extends Event {
    17 usages
    private Customer c;

    1 usage
    public Arrival(Customer cust) { this.c = cust; }

    public void actions() {
        c.setLES(arrayLES[c.getType()]);
        c.setAvgLES(getAvgLES(arrayAvgLES[c.getType()]));
        c.setAvgCLES(getAvgCLES(arrayAvgCLES[c.getType()], lenghtsQtype[c.getType()]));
        c.setWAvgCLES(getWAvgCLES(c.getType(), lenghtsQtype[c.getType()]));
        c.setServeurs(nbServeurs);
        c.setQueueLengths(lenghtsQtype);
        waitList.push(c);
        lenghtsQtype[c.getType()] += 1;
        new Departure(c).schedule(c.getWaitTime());
    }
}

```

On crée une classe **Departure** qui nous permet en héritant d'Event et en implémentant la méthode actions() de définir les décisions à prendre lorsqu'un nouveau Customer c quitte la file. On crée donc un attribut pour le customer et on l'initialise dans le constructeur.

Au niveau de la méthode actions(), on retire donc c de la file d'attente et on met à jour la taille de la file. Ensuite on met à jour les variables arrayLES, arrayAvgLES,... en utilisant les setters que nous avons créés précédemment.

Pour finir si l'appel de c n'avait pas été pris i.e l'attribut isServed de c est false alors on l'ajoute à la liste des clients ayant abandonné sinon on augmente le nombre de serveurs occupés car il passe en service et on programme la fin de service pour une durée correspondant à son servingTime.

```

public class Departure extends Event {
    19 usages
    private Customer c;

    1 usage
    public Departure(Customer cust) { this.c = cust; }

    public void actions() {
        waitList.remove(c);
        lengthsQtype[c.getType()] -= 1;
        arrayLES[c.getType()] = c.getWaitTime();
        setAvgLES(c.getWaitTime(), c.getType());
        setAvgCLES(c.getWaitTime(), c.getType(), c.getQueueLengths()[c.getType()]);
        setWAVgCLES(c.getType(), c.getQueueLengths()[c.getType()], c.getWaitTime());
        if (!c.isServed()) {
            abandonedCustomers.add(c);
        } else {
            nbServeurs += 1;
            new EndOfSim(c).schedule(c.getServingTime());
        }
    }
}

```

On crée une classe **EndOfSim** qui nous permet en héritant d'Event et en implémentant la méthode actions() de définir les décisions à prendre lorsqu'unCustomer c est en fin de service. On crée donc un attribut pour le customer et on l'initialise dans le constructeur.

Au niveau de la méthode actions(), on ajoute c à la liste des clients qui ont été servis et on décrémente le nombre de serveurs occupés.

```

public class EndOfSim extends Event {
    2 usages
    private Customer c;

    1 usage
    public EndOfSim(Customer cust) { this.c = cust; }

    public void actions() {
        servedCustomers.add(c);
        nbServeurs -= 1;
    }
}

```

Pour effectuer la simulation on crée une méthode **simulateOneDay()** et une classe **EndOfDaySim**.

Dans cette classe on arrête l'objet Sim qui est chargé de la simulation et dans la méthode, on initialise cet objet puis on appelle la méthode `createCustomerOfDay` qui va créer l'ensemble des clients. On programme la fin de la simulation pour une durée correspondant à celle de la journée de travail `TIME` que l'on a créé précédemment et on démarre la simulation

```

public class EndOfDaySim extends Event {
    public void actions() { Sim.stop(); }
}

1 usage
public void simulateOneDay(String file) throws IOException {
    Sim.init();
    createCustomerOfDay(file);
    new EndOfDaySim().schedule(TIME);
    Sim.start();
}

```

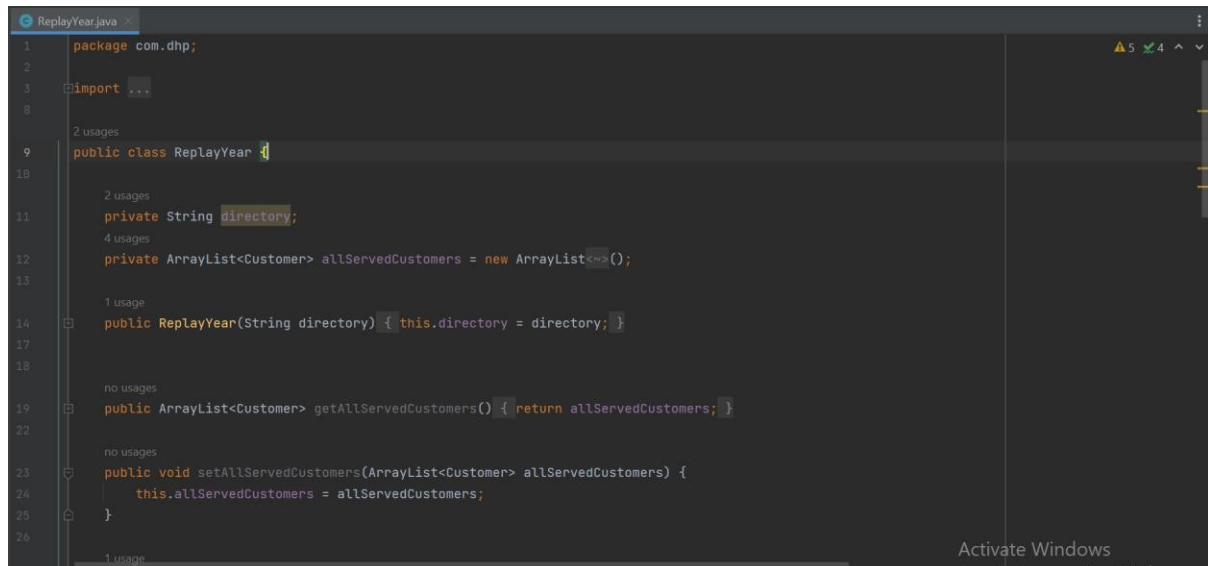
➤ **ReplayOneYear**

Dans cette classe nous permet de rejouer entièrement l'activité du centre d'appel.

On crée les attributs `directory` correspondant au répertoire où se trouve nos données et `allServedCustomers` correspondant à la liste de l'ensemble des clients servis. On crée également un constructeur dans lequel on initialise le répertoire.

18

On crée deux méthodes :

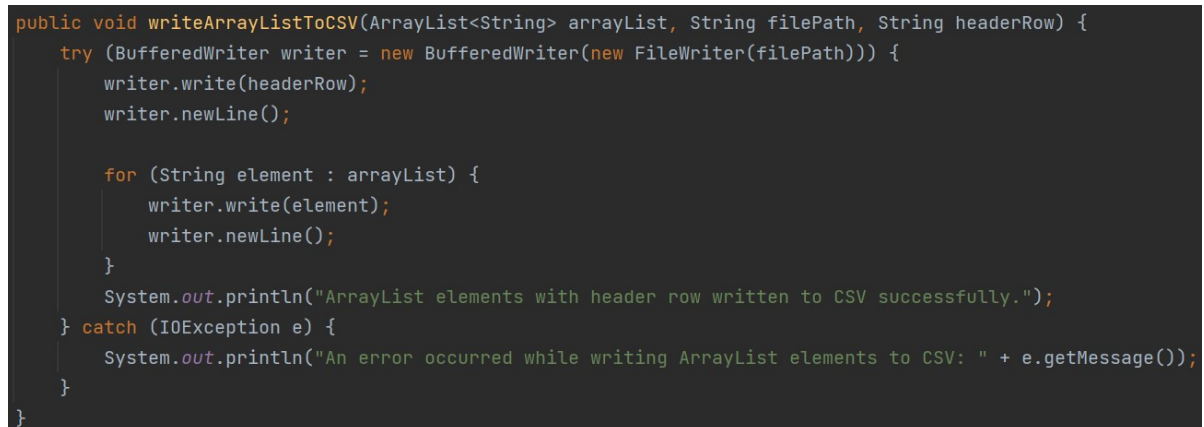


```
1 package com.dhp;
2
3 import ...
4
5
6
7
8
9 public class ReplayYear {
10
11     2 usages
12     private String directory;
13     4 usages
14     private ArrayList<Customer> allServedCustomers = new ArrayList<>();
15
16     1 usage
17     public ReplayYear(String directory) { this.directory = directory; }
18
19     no usages
20     public ArrayList<Customer> getAllServedCustomers() { return allServedCustomers; }
21
22     no usages
23     public void setAllServedCustomers(ArrayList<Customer> allServedCustomers) {
24         this.allServedCustomers = allServedCustomers;
25     }
26 }
```

➤ **writeArrayListToCSV()**

Cette méthode prend en entrée une liste des données à enregistrer, le chemin du fichier où sauvegarder le csv résultat et la ligne d'entête.

On crée un `writer` puis on ajoute l'entête et ensuite on parcourt la liste en créant une nouvelle ligne pour chaque élément de la liste. Une fois l'écriture achevée, le fichier est créé.



```
public void writeArrayListToCSV(ArrayList<String> arrayList, String filePath, String headerRow) {
    try (BufferedWriter writer = new BufferedWriter(new FileWriter(filePath))) {
        writer.write(headerRow);
        writer.newLine();

        for (String element : arrayList) {
            writer.write(element);
            writer.newLine();
        }

        System.out.println("ArrayList elements with header row written to CSV successfully.");
    } catch (IOException e) {
        System.out.println("An error occurred while writing ArrayList elements to CSV: " + e.getMessage());
    }
}
```

➤ **simulateYear()**

Nous avons précédemment organisé nos données de sorte à obtenir un dossier pour chacun des mois avec dans chaque dossier un fichier par journée de travail. Ainsi cette méthode parcourt `directory`, en entrant dans chaque dossier puis une fois dans un dossier il entre dans les fichiers un à un.

On crée d'abord les variables `headerRow` contenant l'en-tête de notre CSV et `csvList` contenant les données à enregistrer dans le csv. Lors du parcours lorsqu'on atteint un fichier cela signifie qu'on a les données d'une journée ainsi on crée une instance de `ReplayOneDay` et on rejoue la journée en utilisant la méthode `simulateOneDay`. Une fois la simulation terminée, on récupère la liste des clients servis que l'on ajoute dans la liste `allServedCustomers`.

Une fois que tous les fichiers ont été parcourus, on parcourt `allServedCustomers` et pour chaque élément on l'ajoute sa représentation sous string à `csvList` en utilisant la méthode `toString()`.

Enfin on appelle la méthode `writeArrayListToCSV()` en lui passant les paramètres pour créer le fichier `result.csv` final.

```
public void simulateYear() throws IOException {
    String headerRow = "TYPE,ARRIVAL_TIME,SERVING_TIME,WAIT_TIME,WAVEC_LES,LES,AVGLES,AVGCLES,SERVEURS,Q0,Q1,Q2,Q3,Q4,Q5,Q6,Q7,Q8,Q9,Q10,Q11,Q12,Q13,Q14,Q15,Q16,Q17,Q18,Q19,Q20,Q21,Q22,Q23,Q24,Q25,Q26";
    ArrayList<String> csvList = new ArrayList<>();
    File directory = new File(this.directory);
    if (directory.isDirectory()) {
        File[] subDirectories = directory.listFiles();
        if (subDirectories != null) {
            for (File subDirectory : subDirectories) {
                if (subDirectory.isDirectory()) {
                    File[] files = subDirectory.listFiles();
                    if (files != null) {
                        for (File file : files) {
                            if (file.isFile()) {
                                try {
                                    ReplayOneDay replayOneDay = new ReplayOneDay();
                                    replayOneDay.simulateOneDay(file.getAbsolutePath());
                                    this.allServedCustomers.addAll(replayOneDay.getServedCustomers());
                                } catch (IllegalArgumentException e) {
                                    System.out.println(file.getAbsolutePath());
                                }
                            }
                        }
                    }
                }
            }
        }
    }
    for (Customer c : this.allServedCustomers) {
        csvList.add(c.toString());
    }
    writeArrayListToCSV(csvList, filePath: "data/result.csv", headerRow);
}
```

➤ Test

Pour finir on crée une classe `Test` pour faire rejouer l'activité du centre d'appel. Il nous suffit de créer une instance de `ReplayOneYear` et d'appeler la méthode `simulateYear()`.

```
Test.java
1 package com.dhp;
2
3 import ...
4
5 no usages
6
7 public class Test {
8
9     no usages
10
11     public static void main(String[] args) throws IOException {
12         ReplayYear replayYear = new ReplayYear( directory: "data");
13         replayYear.simulateYear();
14     }
15 }
16
17
18
```

Comme nous pouvons le constater, une fois que l'exécution se termine nous obtenons le fichier csv suivant avec toutes les métriques.

```
Test.java
1 package com.dhp;
2
3 import ...
4
5 no usages
6
7 public class Test {
8
9     no usages
10
11     public static void main(String[] args) throws IOException {
12         ReplayYear replayYear = new ReplayYear( directory: "data");
13     }
14 }
15
```

```
Run: Test
1 "C:\Program Files\Java\jdk-18.0.2.1\bin\java.exe" ...
2 ArrayList elements with header row written to CSV successfully.
3
4 Process finished with exit code 0
```

result.csv

TYPE

ARRIVAL_TIME

SERVING_TIME

WAIT_TIME

WAVGC_LES

LES

AVGLES

AVGCLES

SERVEURS_Q0

Q1

Q2

Q3

Q4

Q5

Q6

Q7

Q8

Q9

Q10

Q11

Q12

Q13

Q14

Q15

Q16

Q17

Q18

Q19

Q20

Q21

Q22

Q23

Q24

5

3801.0

152.0

1.0

-1.0

0.0

NaN

NaN

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

20

3877.0

311.0

1.0

-1.0

0.0

NaN

NaN

1

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

5

4087.0

149.0

1.0

1.0

1.0

1.0

1.0

2

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

5

4158.0

269.0

1.0

1.0

1.0

1.0

1.0

5

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

5

4150.0

303.0

1.0

1.0

1.0

1.0

1.0

4

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

2

4193.0

283.0

1.0

1.0

1.0

1.0

1.0

5

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

2

4106.0

392.0

1.0

1.0

1.0

1.0

1.0

3

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

20

4469.0

466.0

1.0

1.0

1.0

1.0

1.0

5

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

2

3986.0

530.0

1.0

-1.0

0.0

NaN

NaN

1

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

2

4416.0

118.0

1.0

1.0

1.0

1.0

1.0

6

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

5

4373.0

200.0

1.0

1.0

1.0

1.0

1.0

5

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

5

4583.0

18.0

1.0

1.0

1.0

1.0

1.0

1

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

4549.0

167.0

1.0

-1.0

0.0

NaN

NaN

1

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

6

4586.0

207.0

1.0

-1.0

0.0

NaN

NaN

2

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

20

4694.0

145.0

1.0

1.0

1.0

1.0

1.0

4

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

2

4683.0

191.0

1.0

1.0

1.0

1.0

1.0

3

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

3

4902.0

12.0

1.0

-1.0

0.0

NaN

NaN

6

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

5

4824.0

136.0

1.0

1.0

1.0

1.0

1.0

4

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

3

4929.0

31.0

1.0

1.0

1.0

1.0

1.0

6

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

5

4825.0

223.0

1.0

-1.0

1.0

1.0

1.0

NaN

4

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

9

4836.0

216.0

1.0

-1.0

0.0

NaN

NaN

6

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

9

4982.0

90.0

1.0

1.0

1.0

1.0

1.0

6

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

20

4620.0

458.0

1.0

1.0

1.0

1.0

1.0

2

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

4

4780.0

704.0

1.0

1.0

1.0

1.0

1.0

6

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

File Explorer window showing the directory structure of 'C:\Users\Massamba\OneDrive\Projet...'. The selected file is 'result.csv'.

Excel spreadsheet showing data from 'result.csv'. The spreadsheet has columns A through N. The data includes various numerical values and some text labels like 'TYPE', 'ARRIVAL_TIME', 'SERVING_TIME', 'WAIT_TIME', 'WAVG', 'LES', 'AVGLES', 'AVGLES', 'SERVEURS', 'Q0', 'Q1', 'Q2', 'Q3', 'Q4'.

TYPE	ARRIVAL_TIME	SERVING_TIME	WAIT_TIME	WAVG	LES	AVGLES	AVGLES	SERVEURS	Q0	Q1	Q2	Q3	Q4
1	5	3805	152	1	-1	0	NaN	NaN	0	0	0	0	0
3	20	3877	311	1	-1	0	NaN	NaN	1	0	0	0	0
4	5	4087	149	1	1	1	1	1	2	0	0	0	0
5	5	4158	289	1	1	1	1	1	5	0	0	0	0
6	5	4150	303	1	1	1	1	1	4	0	0	0	0
7	2	4193	283	1	1	1	1	1	5	0	0	0	0
8	2	4126	392	1	1	1	1	1	3	0	0	0	0
9	20	4469	46	1	1	1	1	1	5	0	0	0	0
10	2	3986	530	1	-1	0	NaN	NaN	1	0	0	0	0
11	2	4416	118	1	1	1	1	1	6	0	0	0	0
12	5	4373	200	1	1	1	1	1	5	0	0	0	0
13	5	4183	18	1	1	1	1	1	1	0	0	0	0
14	0	4549	587	1	-1	0	NaN	NaN	1	0	0	0	0
15	6	4386	207	1	-1	0	NaN	NaN	2	0	0	0	0
16	20	4694	145	1	1	1	1	1	4	0	0	0	0
17	2	4683	181	1	1	1	1	1	3	0	0	0	0
18	5	4902	12	1	-1	0	NaN	NaN	6	0	0	0	0
19	5	4824	136	1	1	1	1	1	4	0	0	0	0
20	3	4929	31	1	1	1	1	1	6	0	0	0	0
21	5	4625	223	1	-1	1	1	1	4	0	0	0	0
22	9	4836	218	1	-1	0	NaN	NaN	6	0	0	0	0
23	9	4982	90	1	1	1	1	1	6	0	0	0	0
24	20	4620	458	1	1	1	1	1	2	0	0	0	0
25	5	4789	295	1	1	1	1	1	4	0	0	0	0
26	2	4837	247	1	1	1	1	1	6	0	0	0	0
27	9	5078	157	1	1	1	1	1	5	0	0	0	0
28	5	5036	313	1	1	1	1	1	7	0	0	0	0
29	9	5288	108	1	1	1	1	1	7	0	0	0	0
30	5	5238	199	1	1	1	1	1	4	0	0	0	0
31	3	5280	157	1	1	1	1	1	6	0	0	0	0
32	9	4934	500	1	1	1	1	1	7	0	0	0	0
33	20	5420	199	1	1	1	1	1	7	0	0	0	0
34	20	5547	74	1	1	1	1	1	10	0	0	0	0
35	20	5551	82	1	1	1	1	1	11	0	0	0	0
36	9	5614	28	13	0.8	0	0.83333333	0.83333333	14	0	0	0	0
37	5	5489	286	1	1	1	1	1	9	0	0	0	0
38	17	5648	23	13	-1	0	NaN	NaN	1	13	0	0	0
39	5	5573	114	1	1	1	1	1	1	13	0	0	0

III. Création du modèle ANN

Une fois le fichier CSV obtenu, nous l'utilisons pour créer des modèles basés sur les ANN pour les 5 types de services les plus fréquents. Pour effectuer cela on procède comme suit :

➤ Charger le CSV dans un dataframe

```
In [1]: import pandas as pd
import numpy as np
import tensorflow as tf
from keras.models import Sequential
from keras.layers import Dense, Dropout
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
```

Nous commençons par charger le dataset créé

```
In [2]: df = pd.read_csv("result.csv")
```

```
In [3]: df["AVGLES"] = df["AVGLES"].replace(np.nan, 0)
df["AVGCLES"] = df["AVGCLES"].replace(np.nan, 0)
df
```

Out[3]:

	TYPE	ARRIVAL_TIME	SERVING_TIME	WAIT_TIME	WAVGC_LES	LES	AVGLES	AVGCLES	SERVEURS	Q0	...	Q17	Q18	Q19	Q20	Q21	Q22
0	5	3801.0	152.0	1.0	-1.000000	0.0	0.0	0.00	0	0	...	0	0	0	0	0	0
1	20	3877.0	311.0	1.0	-1.000000	0.0	0.0	0.00	1	0	...	0	0	0	0	0	0
2	5	4087.0	149.0	1.0	1.000000	1.0	1.0	1.00	2	0	...	0	0	0	0	0	0
3	5	4158.0	269.0	1.0	1.000000	1.0	1.0	1.00	5	0	...	0	0	0	0	0	0
4	5	4150.0	303.0	1.0	1.000000	1.0	1.0	1.00	4	0	...	0	0	0	0	0	0
...
1462939	5	46656.0	6.0	1.0	10.787741	50.0	5.9	3.21	2	0	...	0	0	0	0	0	0
1462940	2	46545.0	81.0	109.0	10.337674	1.0	69.4	4.06	5	0	...	0	0	0	0	0	0
1462941	2	46653.0	88.0	1.0	67.700640	1.0	69.4	32.04	2	0	...	0	0	0	0	0	0
1462942	20	46741.0	3.0	1.0	8.597853	38.0	4.7	4.09	2	0	...	0	0	0	0	0	0
1462943	5	46787.0	3.0	1.0	6.011323	1.0	5.9	3.22	2	0	...	0	0	0	0	0	0

1462944 rows × 36 columns

Nous voyons donc qu'on se retrouve avec un dataset de presque 1,5 millions de lignes

➤ Déterminer les 5 types les plus fréquents et créer un dataframe pour chacun

```
In [6]: type_counts = df['TYPE'].value_counts()
top_5_types = type_counts.nlargest(5).index
new_df = df[df['TYPE'].isin(top_5_types)].copy()
```

```
In [7]: top_5_types
```

```
Out[7]: Int64Index([5, 20, 2, 11, 9], dtype='int64')
```

```
In [8]: grouped_df = new_df.groupby("TYPE")
for group, subset_df in grouped_df:
    print(f"Dataframe pour le type {group}")
    print(subset_df)
```

```
Dataframe pour le type 2
  TYPE  ARRIVAL_TIME  SERVING_TIME  WAIT_TIME  WAVGC_LES  LES  AVGLES  \
5      2      4193.0      283.0      1.0      1.000000      1.0      1.0
6      2      4106.0      392.0      1.0      1.000000      1.0      1.0
8      2      3986.0      530.0      1.0     -1.000000      0.0      0.0
9      2      4416.0      118.0      1.0      1.000000      1.0      1.0
15     2      4683.0      191.0      1.0      1.000000      1.0      1.0
...    ...          ...          ...          ...          ...          ...
1462926  2      44975.0      77.0      1.0     19.237645      1.0     95.5
1462931  2      45949.0     185.0      1.0     15.590116      1.0     95.5
1462932  2      46110.0      70.0      1.0     12.672093      1.0     95.5
1462940  2      46545.0      81.0     109.0     10.337674      1.0     69.4
1462941  2      46653.0      88.0      1.0     67.700640      1.0     69.4
```

```
  AVGLES  SERVEURS  Q0  ...  Q17  Q18  Q19  Q20  Q21  Q22  Q23  Q24  \
5      1.00      5  0  ...    0    0    0    0    0    0    0    0
6      1.00      3  0  ...    0    0    0    0    0    0    0    0
8      0.00      1  0  ...    0    0    0    0    0    0    0    0
9      1.00      6  0  ...    0    0    0    0    0    0    0    0
15     1.00      7  0  ...    0    0    0    0    0    0    0    0
```

- Créer les dataframes pour les 5 services, les transformer en matrices, normaliser et séparer en données de test et d'entraînement

Creation des ANN pour chaque type

Nous commençons par créer un dataframe pour chacun des 5 services, on le convertit en numpy array, on normalise les données et on sépare les données en train et test

```
In [9]: X1,y1 = df[df['TYPE']==top_5_types[0]].copy().drop(columns=["WAIT_TIME", "TYPE"]).to_numpy(), df[df['TYPE']==top_5_types[0]].cc
X2,y2 = df[df['TYPE']==top_5_types[1]].copy().drop(columns=["WAIT_TIME", "TYPE"]).to_numpy(), df[df['TYPE']==top_5_types[1]].cc
X3,y3 = df[df['TYPE']==top_5_types[2]].copy().drop(columns=["WAIT_TIME", "TYPE"]).to_numpy(), df[df['TYPE']==top_5_types[2]].cc
X4,y4 = df[df['TYPE']==top_5_types[3]].copy().drop(columns=["WAIT_TIME", "TYPE"]).to_numpy(), df[df['TYPE']==top_5_types[3]].cc
X5,y5 = df[df['TYPE']==top_5_types[4]].copy().drop(columns=["WAIT_TIME", "TYPE"]).to_numpy(), df[df['TYPE']==top_5_types[4]].cc
```

```
In [10]: sc = StandardScaler()
X1 = sc.fit_transform(X1)
X2 = sc.fit_transform(X2)
X3 = sc.fit_transform(X3)
X4 = sc.fit_transform(X4)
X5 = sc.fit_transform(X5)
```

```
In [11]: X1_train, X1_test, y1_train, y1_test = train_test_split(X1,y1,test_size=0.2,random_state=42)
X2_train, X2_test, y2_train, y2_test = train_test_split(X2,y2,test_size=0.2,random_state=42)
X3_train, X3_test, y3_train, y3_test = train_test_split(X3,y3,test_size=0.2,random_state=42)
X4_train, X4_test, y4_train, y4_test = train_test_split(X4,y4,test_size=0.2,random_state=42)
X5_train, X5_test, y5_train, y5_test = train_test_split(X5,y5,test_size=0.2,random_state=42)
```

- Créer l'architecture de l'ANN et entrainer un pour chaque type de serviceTYPE

Type 5

```
In [12]: M ann = Sequential([
    Dense(units=32,activation="relu",kernel_initializer="normal"),
    Dense(units=16,activation="relu",kernel_initializer="normal"),
    Dense(units=8,activation="relu",kernel_initializer="normal"),
    Dense(units=1,activation="linear")
])

adam = tf.keras.optimizers.Adam(learning_rate=0.01)

ann.compile(optimizer=adam,loss="mean_squared_error", metrics="mse")

In [13]: M ann.fit(X1_train,y1_train,batch_size=32,epochs=20)

Epoch 12/20
14498/14498 [=====] - 38s 3ms/step - loss: 632.1505 - mse: 632.1505
Epoch 13/20
14498/14498 [=====] - 38s 3ms/step - loss: 636.9327 - mse: 636.9327
Epoch 14/20
14498/14498 [=====] - 39s 3ms/step - loss: 633.0061 - mse: 633.0061
Epoch 15/20
14498/14498 [=====] - 39s 3ms/step - loss: 629.2704 - mse: 629.2704
Epoch 16/20
14498/14498 [=====] - 36s 2ms/step - loss: 630.0024 - mse: 630.0024
Epoch 17/20
14498/14498 [=====] - 34s 2ms/step - loss: 628.9665 - mse: 628.9665
Epoch 18/20
14498/14498 [=====] - 34s 2ms/step - loss: 630.5579 - mse: 630.5579
Epoch 19/20
14498/14498 [=====] - 40s 3ms/step - loss: 628.0359 - mse: 628.0359
Epoch 20/20
14498/14498 [=====] - 42s 3ms/step - loss: 624.7079 - mse: 624.7079

Out[13]: <keras.src.callbacks.History at 0x264d58d5e0>

In [14]: M score5 = ann.evaluate(X1_test, y1_test)

3625/3625 [=====] - 9s 2ms/step - loss: 594.5507 - mse: 594.5507

In [15]: M print(f"Le RMSE obtenu avec ANN pour le service de type 5 est {score5[1]**0.5}")

Le RMSE obtenu avec ANN pour le service de type 5 est 24.383410758440743

In [16]: M ann.save("models/service_type_5.h5")
```

TYPE 20

Type 20

```
In [17]: M ann = Sequential([
    Dense(units=32,activation="relu",kernel_initializer="normal"),
    Dense(units=16,activation="relu",kernel_initializer="normal"),
    Dense(units=8,activation="relu",kernel_initializer="normal"),
    Dense(units=1,activation="linear")
])

adam = tf.keras.optimizers.Adam(learning_rate=0.01)

ann.compile(optimizer=adam,loss="mean_squared_error", metrics="mse")

In [18]: M ann.fit(X2_train,y2_train,batch_size=32,epochs=20)

Epoch 12/20
7819/7819 [=====] - 21s 3ms/step - loss: 959.0083 - mse: 959.0083
Epoch 13/20
7819/7819 [=====] - 22s 3ms/step - loss: 956.4084 - mse: 956.4084
Epoch 14/20
7819/7819 [=====] - 21s 3ms/step - loss: 954.6839 - mse: 954.6839
Epoch 15/20
7819/7819 [=====] - 22s 3ms/step - loss: 951.7099 - mse: 951.7099
Epoch 16/20
7819/7819 [=====] - 22s 3ms/step - loss: 947.9747 - mse: 947.9747
Epoch 17/20
7819/7819 [=====] - 22s 3ms/step - loss: 946.8612 - mse: 946.8612
Epoch 18/20
7819/7819 [=====] - 23s 3ms/step - loss: 944.0894 - mse: 944.0894
Epoch 19/20
7819/7819 [=====] - 24s 3ms/step - loss: 947.1210 - mse: 947.1210
Epoch 20/20
7819/7819 [=====] - 25s 3ms/step - loss: 944.8376 - mse: 944.8376

Out[18]: <keras.src.callbacks.History at 0x265134f7e50>

In [19]: M score20=ann.evaluate(X2_test, y2_test)

1955/1955 [=====] - 6s 3ms/step - loss: 964.1425 - mse: 964.1425

In [20]: M print(f"Le RMSE obtenu avec ANN pour le service de type 20 est {score20[1]**0.5}")

Le RMSE obtenu avec ANN pour le service de type 20 est 31.05064340806302

In [20]: M ann.save("models/service_type_20.h5")
```

TYPE 2

Type 2

```
In [21]: M ann = Sequential([
        Dense(units=32,activation="relu",kernel_initializer="normal"),
        Dense(units=16,activation="relu",kernel_initializer="normal"),
        Dense(units=8,activation="relu",kernel_initializer="normal"),
        Dense(units=1,activation="linear")
    ])

adam = tf.keras.optimizers.Adam(learning_rate=0.01)

ann.compile(optimizer=adam,loss="mean_squared_error", metrics="mse")

In [22]: M ann.fit(X3_train,y3_train,batch_size=32,epochs=20)

6656/6656 [=====] - 18s 3ms/step - loss: 1209.8718 - mse: 1209.8718
Epoch 12/20
6656/6656 [=====] - 17s 3ms/step - loss: 1205.1709 - mse: 1205.1709
Epoch 13/20
6656/6656 [=====] - 17s 3ms/step - loss: 1203.2120 - mse: 1203.2120
Epoch 14/20
6656/6656 [=====] - 17s 3ms/step - loss: 1195.6145 - mse: 1195.6145
Epoch 15/20
6656/6656 [=====] - 18s 3ms/step - loss: 1197.4598 - mse: 1197.4598
Epoch 16/20
6656/6656 [=====] - 17s 3ms/step - loss: 1198.2201 - mse: 1198.2201
Epoch 17/20
6656/6656 [=====] - 18s 3ms/step - loss: 1194.4625 - mse: 1194.4625
Epoch 18/20
6656/6656 [=====] - 17s 3ms/step - loss: 1192.3406 - mse: 1192.3406
Epoch 19/20
6656/6656 [=====] - 18s 3ms/step - loss: 1189.0452 - mse: 1189.0452
Epoch 20/20
6656/6656 [=====] - 21s 3ms/step - loss: 1188.9938 - mse: 1188.9938

In [23]: M score2=ann.evaluate(X3_test, y3_test)

1664/1664 [=====] - 4s 2ms/step - loss: 1157.4689 - mse: 1157.4689

In [24]: M print(f"Le RMSE obtenu avec ANN pour le service de type 2 est {score2[1]**0.5}")

Le RMSE obtenu avec ANN pour le service de type 2 est 34.02159420236378

In [25]: M ann.save("models/service_type_2.h5")
```

TYPE 11

Type 11

```
In [26]: M ann = Sequential([
        Dense(units=32,activation="relu",kernel_initializer="normal"),
        Dense(units=16,activation="relu",kernel_initializer="normal"),
        Dense(units=8,activation="relu",kernel_initializer="normal"),
        Dense(units=1,activation="linear")
    ])

adam = tf.keras.optimizers.Adam(learning_rate=0.01)

ann.compile(optimizer=adam,loss="mean_squared_error", metrics="mse")

In [27]: M ann.fit(X4_train,y4_train,batch_size=32,epochs=20)

Epoch 12/20
3702/3702 [=====] - 10s 3ms/step - loss: 735.9631 - mse: 735.9631
Epoch 13/20
3702/3702 [=====] - 11s 3ms/step - loss: 728.3000 - mse: 728.3000
Epoch 14/20
3702/3702 [=====] - 11s 3ms/step - loss: 727.5033 - mse: 727.5033
Epoch 15/20
3702/3702 [=====] - 11s 3ms/step - loss: 723.1284 - mse: 723.1284
Epoch 16/20
3702/3702 [=====] - 10s 3ms/step - loss: 723.7409 - mse: 723.7409
Epoch 17/20
3702/3702 [=====] - 11s 3ms/step - loss: 718.6819 - mse: 718.6819
Epoch 18/20
3702/3702 [=====] - 11s 3ms/step - loss: 717.0871 - mse: 717.0871
Epoch 19/20
3702/3702 [=====] - 11s 3ms/step - loss: 712.5883 - mse: 712.5883
Epoch 20/20
3702/3702 [=====] - 10s 3ms/step - loss: 714.8467 - mse: 714.8467

Out[27]: <keras.src.callbacks.History at 0x2656eb6ae80>

In [28]: M score11=ann.evaluate(X4_test, y4_test)

926/926 [=====] - 2s 2ms/step - loss: 773.4893 - mse: 773.4893

In [29]: M print(f"Le RMSE obtenu avec ANN pour le service de type 11 est {score11[1]**0.5}")

Le RMSE obtenu avec ANN pour le service de type 11 est 27.811674847310076

In [30]: M ann.save("models/service_type_11.h5")
```

TYPE 9

```

Type 9

In [31]: M ann = Sequential([
          Dense(units=32,activation="relu",kernel_initializer="normal"),
          Dense(units=16,activation="relu",kernel_initializer="normal"),
          Dense(units=8,activation="relu",kernel_initializer="normal"),
          Dense(units=1,activation="linear")
        ])

adam = tf.keras.optimizers.Adam(learning_rate=0.01)

ann.compile(optimizer=adam,loss="mean_squared_error", metrics="mse")

In [32]: M ann.fit(X5_train,y5_train,batch_size=32,epochs=20)

Epoch 12/20
1615/1615 [=====] - 4s 3ms/step - loss: 1285.2109 - mse: 1285.2109
Epoch 13/20
1615/1615 [=====] - 4s 3ms/step - loss: 1273.3555 - mse: 1273.3555
Epoch 14/20
1615/1615 [=====] - 4s 3ms/step - loss: 1265.9307 - mse: 1265.9307
Epoch 15/20
1615/1615 [=====] - 4s 3ms/step - loss: 1269.7493 - mse: 1269.7493
Epoch 16/20
1615/1615 [=====] - 4s 3ms/step - loss: 1258.1716 - mse: 1258.1716
Epoch 17/20
1615/1615 [=====] - 4s 3ms/step - loss: 1263.1052 - mse: 1263.1052
Epoch 18/20
1615/1615 [=====] - 4s 3ms/step - loss: 1254.6115 - mse: 1254.6115
Epoch 19/20
1615/1615 [=====] - 4s 3ms/step - loss: 1255.0640 - mse: 1255.0640
Epoch 20/20
1615/1615 [=====] - 4s 3ms/step - loss: 1248.4398 - mse: 1248.4398

Out[32]: <keras.src.callbacks.History at 0x2657cdd1af0>

In [33]: M score9=ann.evaluate(X5_test, y5_test)

404/404 [=====] - 1s 2ms/step - loss: 1199.9482 - mse: 1199.9482

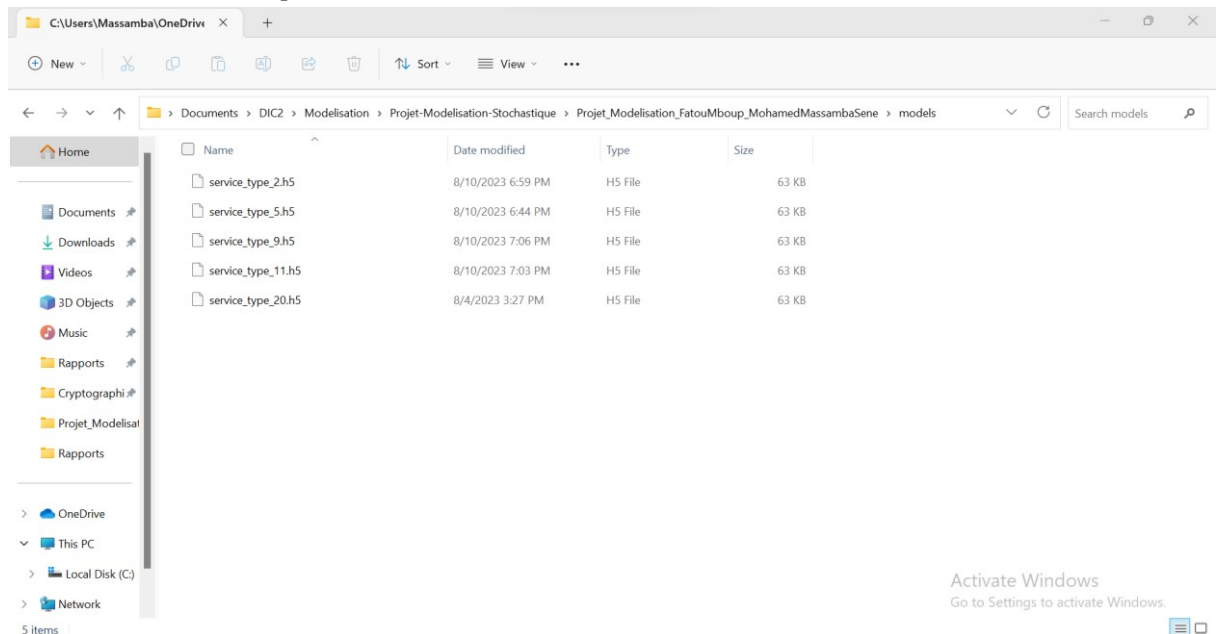
In [34]: M print(f"Le RMSE obtenu avec ANN pour le service de type 9 est {score9[1]**0.5}")

Le RMSE obtenu avec ANN pour le service de type 9 est 34.64026908364743

In [35]: M ann.save("models/service_type_9.h5")

```

Une fois l'entraînement et l'évaluation effectuée on enregistre également le modèle afin qu'il puisse être réutilisé plus tard. Nous pouvons ainsi voir les différents modèles que nous avons eu à entraîner



➤ Résultats finaux

Pour finir on conserve tous les scores dans un dictionnaire et on affiche le RMSE pour chaque type de service


```
In [35]: ann.save("models/service_type_9.h5")
```

```
In [37]: scores = {  
    "2": score2[1]**0.5,  
    "5": score5[1]**0.5,  
    "9": score9[1]**0.5,  
    "11": score11[1]**0.5,  
    "20": score20[1]**0.5  
}
```

Résultats finaux

```
In [38]: for group, subset_df in grouped_df:  
    print(f"Pour le service {group}, le RMSE pour l'ANN est {scores[str(group)]}\n")
```

Pour le service 2, le RMSE pour l'ANN est 34.02159420236378

Pour le service 5, le RMSE pour l'ANN est 24.383410758440743

Pour le service 9, le RMSE pour l'ANN est 34.64026908364743

Pour le service 11, le RMSE pour l'ANN est 27.811674847310076

Pour le service 20, le RMSE pour l'ANN est 31.05064340806302