

RAPPORT
PROJET CY-TRUCKS
FILIERE PREING2 • 2023-2024

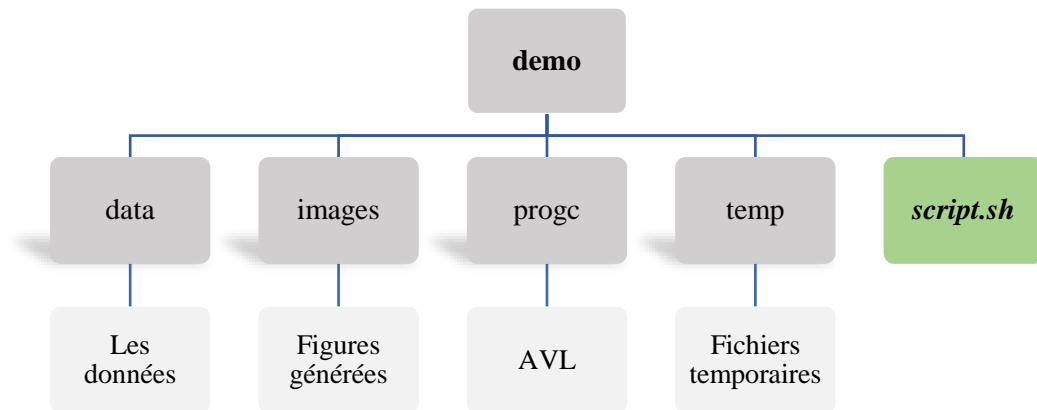


Réalisé par :

- **Dikra Bouhorma**
- **Bayane Benameur**

STRUCTURE DE PROJET

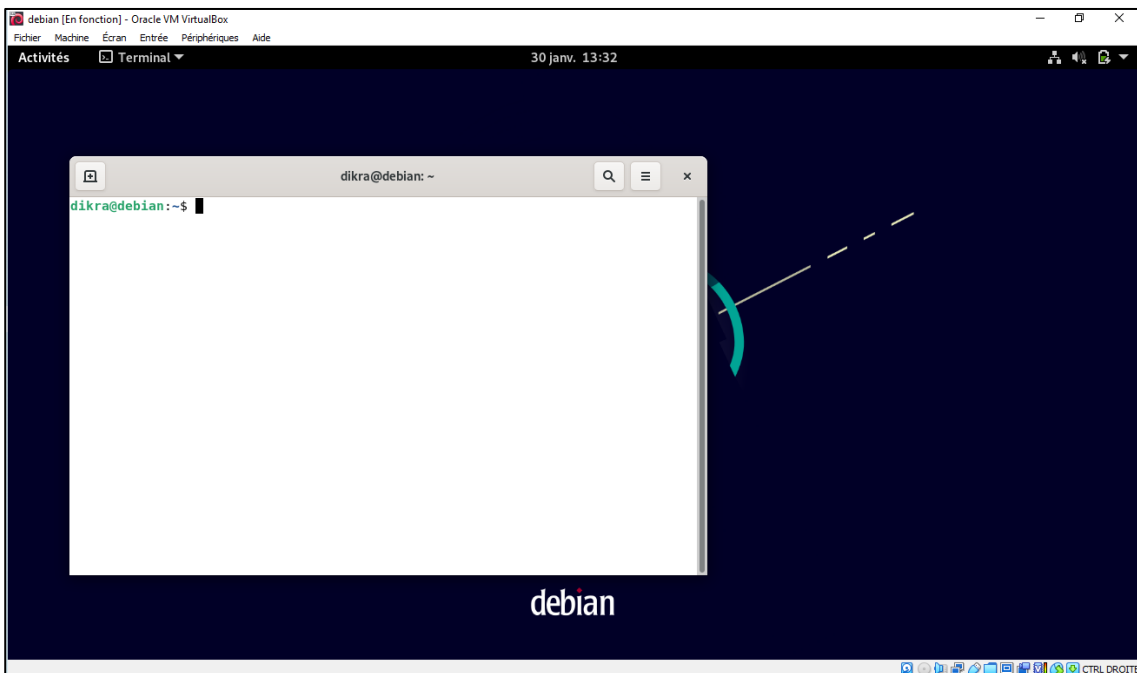
```
dikra@debian: ~/demo
dikra@debian:~/demo$ ls
data images progc script.sh temp
dikra@debian:~/demo$
```



ENVIRONNEMENT DE TRAVAIL

Machine Virtual VirtualBox : Debian

Linux debian 5.10.0-26-amd64 #1 SMP Debian 5.10.197-1 (2023-09-29) x86_64 GNU/Linux



SCRIPT SHELL

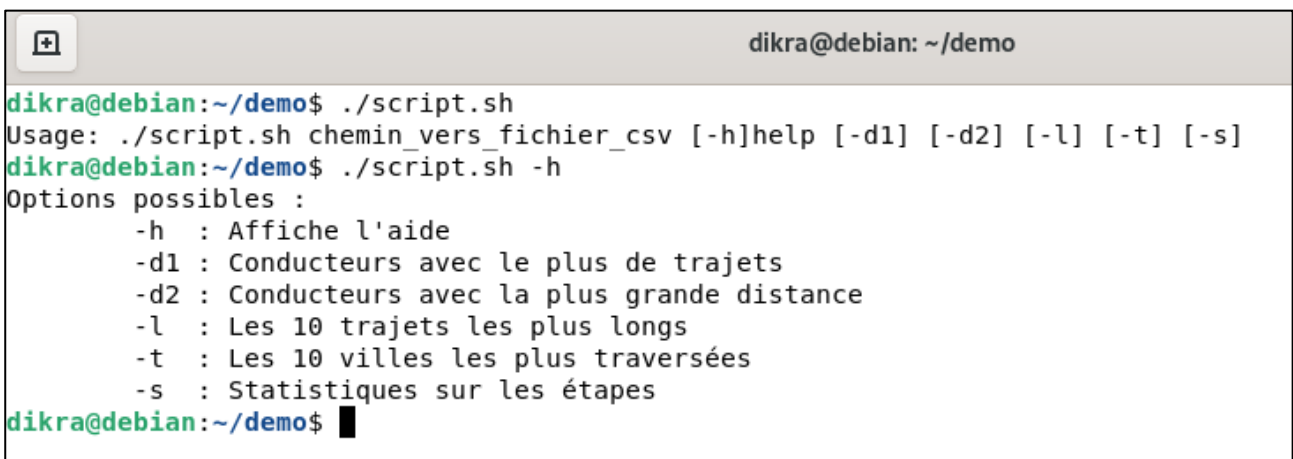
Le script Shell gère les arguments saisis par l'utilisateur. D'abord, il faut vérifier le nombre des arguments passé ensuite leur contenu.

```
# Vérification du nombre d'arguments
if [ "$#" -lt 2 ]; then
    # Si le premier argument est -h, afficher l'aide et ignorer les autres arguments
    if [ "$1" == "-h" ]; then
        echo "Options possibles :"
        echo "    -h : Affiche l'aide"
        echo "    -d1 : Conducteurs avec le plus de trajets"
        echo "    -d2 : Conducteurs avec la plus grande distance"
        echo "    -l : Les 10 trajets les plus longs"
        echo "    -t : Les 10 villes les plus traversées"
        echo "    -s : Statistiques sur les étapes"
        exit 0
    fi
    echo "Usage: $0 chemin_vers_fichier_csv [-h]help [-d1] [-d2] [-l] [-t] [-s]"
    exit 1
fi

# Récupération du chemin du fichier CSV
fichier_csv="$1"

# Récupération de l'argument passé au script
option="$2"
```

Exécution :



```
dikra@debian: ~/demo
dikra@debian:~/demo$ ./script.sh
Usage: ./script.sh chemin_vers_fichier_csv [-h]help [-d1] [-d2] [-l] [-t] [-s]
dikra@debian:~/demo$ ./script.sh -h
Options possibles :
    -h : Affiche l'aide
    -d1 : Conducteurs avec le plus de trajets
    -d2 : Conducteurs avec la plus grande distance
    -l : Les 10 trajets les plus longs
    -t : Les 10 villes les plus traversées
    -s : Statistiques sur les étapes
dikra@debian:~/demo$
```

Les fonctions **chrono_start()** et **chrono_stop()** permettent de lancer le chrono pour calculer la durée de chaque traitement. Les fonctions seront appelées au besoin pour chaque traitement.

La fonction **chrono_start()** enregistre le temps de départ du traitement en utilisant la commande « **date** » pour obtenir le nombre de secondes écoulées depuis l'époque (**epoch**) en nanosecondes. La fonction **chrono_stop()** enregistre le temps d'arrêt du traitement de la même manière. Ensuite, elle calcule la durée totale du traitement en soustrayant le temps de départ du temps d'arrêt, donnant le résultat en nanosecondes. Cette valeur est ensuite convertie en secondes avec deux décimales en divisant par un milliard (pour convertir de nanosecondes en secondes). Enfin, la durée calculée est affichée sous forme de message.

```
# Fonction pour démarrer le chrono qui va calculer la durée des traitements
chrono_start() {
    start_time=$(date +%s%N) # Enregistre le temps de départ en nanosecondes
}

# Fonction pour arrêter le chrono et afficher la durée
chrono_stop() {
    end_time=$(date +%s%N) # Enregistre le temps d'arrêt en nanosecondes
    duration=$((end_time - start_time)) # Calcule la durée en nanosecondes
    # Convertit la durée en secondes avec deux décimales
    duration_seconds=$(echo "scale=2; $duration / 1000000000" | bc)
    echo "Durée du traitement : ${duration_seconds} secondes"
}
```

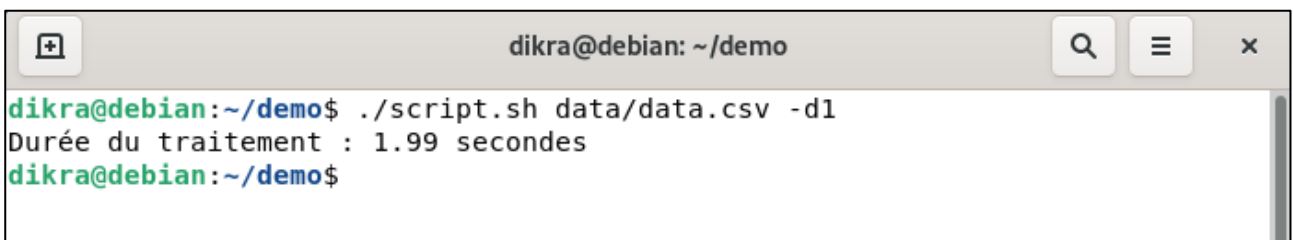
Ensuite, le reste du code est structuré sous forme de fonctions pour chaque traitement (d1, d2, l et t).
Expliquer en détail en bas un par un.

....

```
# ----- Les options selon le choix de l'utilisateur -----
if [ "$option" == "-d1" ]; then
    Option_d1
elif [ "$option" == "-d2" ]; then
    Option_d2
elif [ "$option" == "-l" ]; then
    Option_l
elif [ "$option" == "-t" ]; then
    # Makefile pour compiler et exécuter le programme C
    # -----
    # La partie AVL prend 21 heures !!! vous pouvez enlever le commentaire pour l'exécuter
    # -----
    # echo "Exécution du Makefile..."
    # Se déplacer dans le répertoire contenant le Makefile
    cd "./progC"
    # Exécution du Makefile
    make
    # Revenir au répertoire précédent
    cd ..
    # echo "Fin de l'exécution du Makefile."
    # -----
    # Utiliser le résultat du make pour créer le graphe
    Option_t
elif [ "$option" == "-s" ]; then
    echo "-s option ... [en cours] "
else
    echo "Option non reconnue !"
fi
```

Option -d1

Exécution :



```
dikra@debian: ~/demo
dikra@debian:~/demo$ ./script.sh data/data.csv -d1
Durée du traitement : 1.99 secondes
dikra@debian:~/demo$
```

Temps d'exécution : 2 secondes (en moyen)

Script :

```
# ----- Fonction pour le traitement d1 -----
Option_d1() {
    chrono_start
    #Compter le nombre de trajets pour chaque conducteur d1
    awk -F";" '{/;1;/ {compteur[$6] +=1} END {for (nom in compteur) print compteur[nom] ";" nom }}' data/data.csv | sort -nrk1,1 | head -
n 10 > temp/d1_t.csv

    awk -F";" '{print NR ";" $0}' temp/d1_t.csv > temp/d1.csv
    rm ./temp/d1_t.csv
    chrono_stop

    # Commandes GNUplot pour générer l'histogramme horizontal
    gnuplot <<- GNU_PLOT
        set term pngcairo size 500,500 enhanced font 'Arial,8'
        set output './images/d1.png'
        set datafile separator ";"
        set ylabel 'DRIVER NAMES'
        set xlabel 'NB ROUTES'
        set title 'Option -d1 : Nb routes = f(Driver)'
        binwidth = 0.5
        bin(x) = binwidth * floor(x/binwidth)
        set style fill solid 1
        set yrange reverse
        plot 'temp/d1.csv' u (\$2/2):1:(\$2/2):(binwidth/2.):yticlabels(3) w boxxy notitle linecolor rgb "green"
    GNU_PLOT
    # Nettoyage des fichiers temporaires
    # rm ./temp/d1.csv
}
```

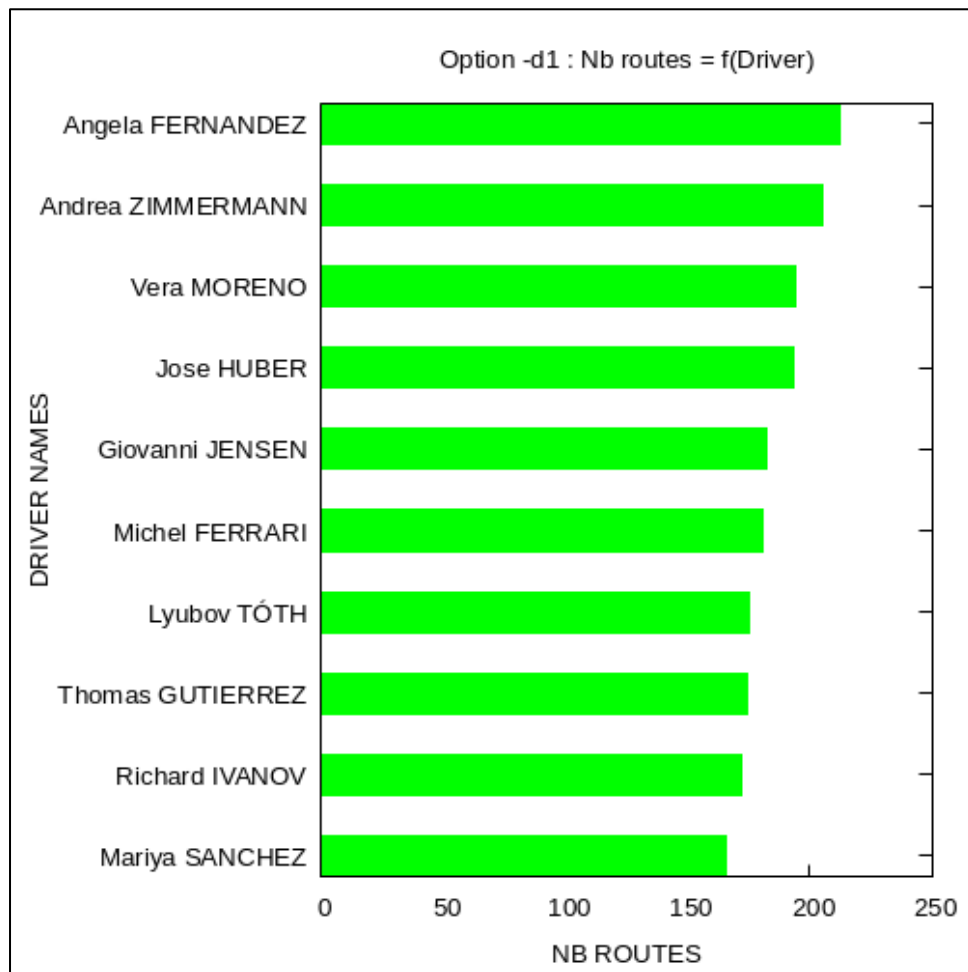
Pour réaliser le traitement d1, on a l'utilisation des commandes :

- **awk** : pour filtrer le fichier et trouver le résultat voulu
- **sort** : pour trier par ordre décroissant (-r) de la colonne 1
- **head** : pour sélectionner les 10 premières lignes de sortie

Le résultat de la commande va être utilisé pour la génération de l'histogramme demandé :

- ☐ **set term pngcairo size 500,500 enhanced font 'Arial,8'**
 - ≥ Définit le format de sortie du graphique comme PNG, spécifie la taille du graphique comme 500x500 pixels et utilise la police Arial avec une taille de 8 points.
- ☐ **set output './images/d1.png'**
 - ≥ Définit le fichier où sera enregistré le graphique généré.
- ☐ **set datafile separator ";"**
 - ≥ Spécifie le délimiteur de champ à utiliser lors de la lecture des données, dans notre cas, le point-virgule (;).
- ☐ **set ylabel 'DRIVER NAMES'**
 - ≥ Définit l'étiquette de l'axe des ordonnées comme "DRIVER NAMES".
- ☐ **set xlabel 'NB ROUTES'**
 - ≥ Définit l'étiquette de l'axe des abscisses comme "NB ROUTES".
- ☐ **set title 'Option -d1 : Nb routes = f(Driver)'**
 - ≥ Définit le titre du graphique comme "Option -d1 : Nb routes = f(Driver)".
- ☐ **binwidth = 0.5**
 - ≥ Définit la largeur des intervalles de données pour la création de l'histogramme.
- ☐ **bin(x) = binwidth * floor(x/binwidth)**
 - ≥ Cette fonction arrondit les valeurs des données à des intervalles spécifiés par 'binwidth'.
- ☐ **set style fill solid 1**
 - ≥ Définit le style de remplissage des boîtes de l'histogramme comme solide.
- ☐ **set yrange reverse**
 - ≥ Inverse l'axe des ordonnées pour afficher les noms des conducteurs dans l'ordre inverse.
- ☐ **plot 'temp/d1.csv' u (\\$2/2):1:(\\$2/2):(binwidth/2.):yticlabels(3) w boxxy notitle linecolor rgb "green"**
 - ≥ Trace le graphique de type histogramme horizontal (boxxy) en utilisant les données du fichier 'temp/d1.csv'. La colonne 1 est utilisée pour les noms des conducteurs et la colonne 2 est utilisée pour le nombre de routes. Les données sont transformées pour correspondre aux intervalles spécifiés par 'binwidth'. Les boîtes sont remplies en vert et les étiquettes des axes des ordonnées sont positionnées à l'extérieur des boîtes.

Résultat : (./images/d1.png) Histogramme horizontal qui présente le nombre de routes par conducteurs



En plus de l'histogramme créée dans le sous-dossier **./images**. Le résultat de la commande est aussi sauvegardé dans le fichier **./temp/d1.csv**. (On a laissé le fichier pour vérification, et on a commenté dans le script la partie nécessaire pour le nettoyage des fichier temporaire)

```
1;212;Angela FERNANDEZ
2;205;Andrea ZIMMERMANN
3;194;Vera MORENO
4;193;Jose HUBER
5;182;Giovanni JENSEN
6;181;Michel FERRARI
7;175;Lyubov TÓTH
8;174;Thomas GUTIERREZ
9;172;Richard IVANOV
10;166;Mariya SANCHEZ
```

1. La première colonne est utile pour dessiner l'histogramme avec GNUPLOT
2. La deuxième colonne est le nombre de routes par conducteurs
3. La troisième colonne est le nom du conducteur correspondant

Option -d2

Exécution :

```
dikra@debian: ~/demo
dikra@debian:~/demo$ ./script.sh data/data.csv -d2
Durée du traitement : 5.07 secondes
dikra@debian:~/demo$
```

Temps d'exécution moyen: 5 secondes

Script :

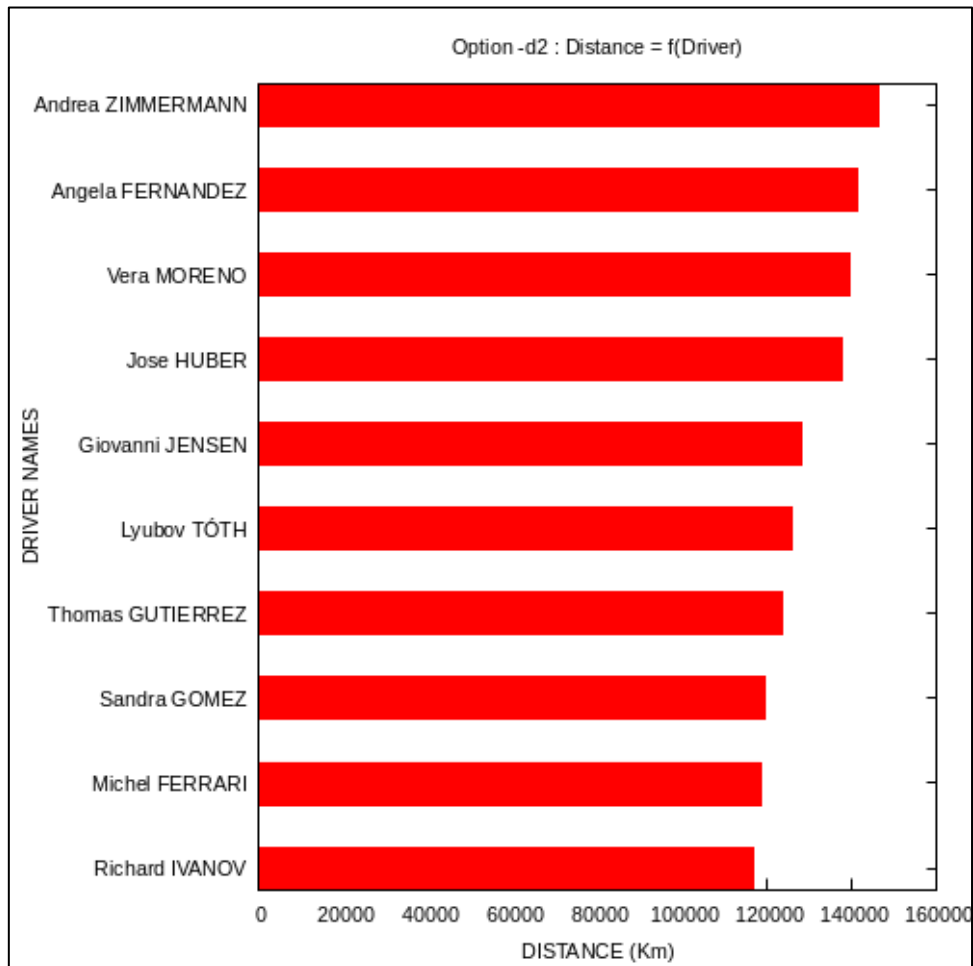
```
# ----- Fonction pour le traitement d2 -----
Option_d2() {
    chrono_start
    # Récupérer la distance totale parcourue par chaque conducteur
    awk -F";" '{compteur[$6] += $5} END {for (nom in compteur) print nom ";" compteur[nom]}' data/data.csv | sort -t";" -k2,2nr | head
-n 10 > temp/d2_t.csv

    awk -F";" '{print NR ";" $0}' temp/d2_t.csv > temp/d2.csv
    rm ./temp/d2_t.csv
    chrono_stop

    # Commandes GNUplot pour générer l'histogramme horizontal
    gnuplot <<- GNU_PLOT
        set term pngcairo size 500,500 enhanced font 'Arial,8'
        set output './images/d2.png'
        set datafile separator ";"
        set ylabel 'DRIVER NAMES'
        set xlabel 'DISTANCE (Km)'
        set title 'Option -d2 : Distance = f(Driver)'
        binwidth = 0.5
        bin(x) = binwidth * floor(x/binwidth)
        set style fill solid 1
        set yrange reverse
        plot 'temp/d2.csv' u (\$3/2):1:(\$3/2):(binwidth/2):yticlabels(2) w boxxy notitle linecolor rgb "red"
    GNU_PLOT
    # Nettoyage des fichiers temporaires
    # rm ./temp/d2.csv
}
```

Pour le traitement d2, la commande **AWK** traite les données du fichier CSV « **data/data.csv** », en accumulant la valeur de la cinquième colonne (**\$5** : distance) pour chaque entrée basée sur la sixième colonne (**\$6** : Nom du conducteurs). Une fois toutes les lignes traitées (**END**), les résultats sont triés en fonction de la somme des distances (**sort**), avec les dix premiers conducteurs en ordre décroissant (**head**). Le résultat est ensuite écrit dans le fichier « **temp/d2 t.csv** ».

Résultat : (./images/d2.png) Histogramme horizontal qui présente la distance parcourue par conducteurs



En plus de l'histogramme créée dans le sous-dossier `./images`. Le résultat de la commande est aussi sauvegardé dans le fichier `./temp/d2.csv`.

```
1;Andrea ZIMMERMANN;146260
2;Angela FERNANDEZ;141251
3;Vera MORENO;139635
4;Jose HUBER;137729
5;Giovanni JENSEN;127981
6;Lyubov TÓTH;126019
7;Thomas GUTIERREZ;123433
8;Sandra GOMEZ;119352
9;Michel FERRARI;118373
10;Richard IVANOV;116777
```

1. La première colonne est utile pour dessiner l'histogramme correctement avec GNUPLOT
2. Dans la deuxième colonne on a les noms des conducteurs
3. La troisième colonne contient la distance total par conducteurs

OPTION -l

Exécution :

```
dikra@debian:~/demo$ ./script.sh data/data.csv -l
Durée du traitement : 8.51 secondes
```

Temps d'exécution : environ 8 secondes

Script :

```
# ----- Fonction pour le traitement l -----
Option_l() {
  chrono_start
  # Calculer la distance total de chaque trajet
  awk -F';' '{journey[$1]+=$5} END {for (j in journey) print j, journey[j]}' $fichier_csv | sort -k2,2nr | head -10 | sort -k1,1n >
  ./temp/option_l.txt
  chrono_stop

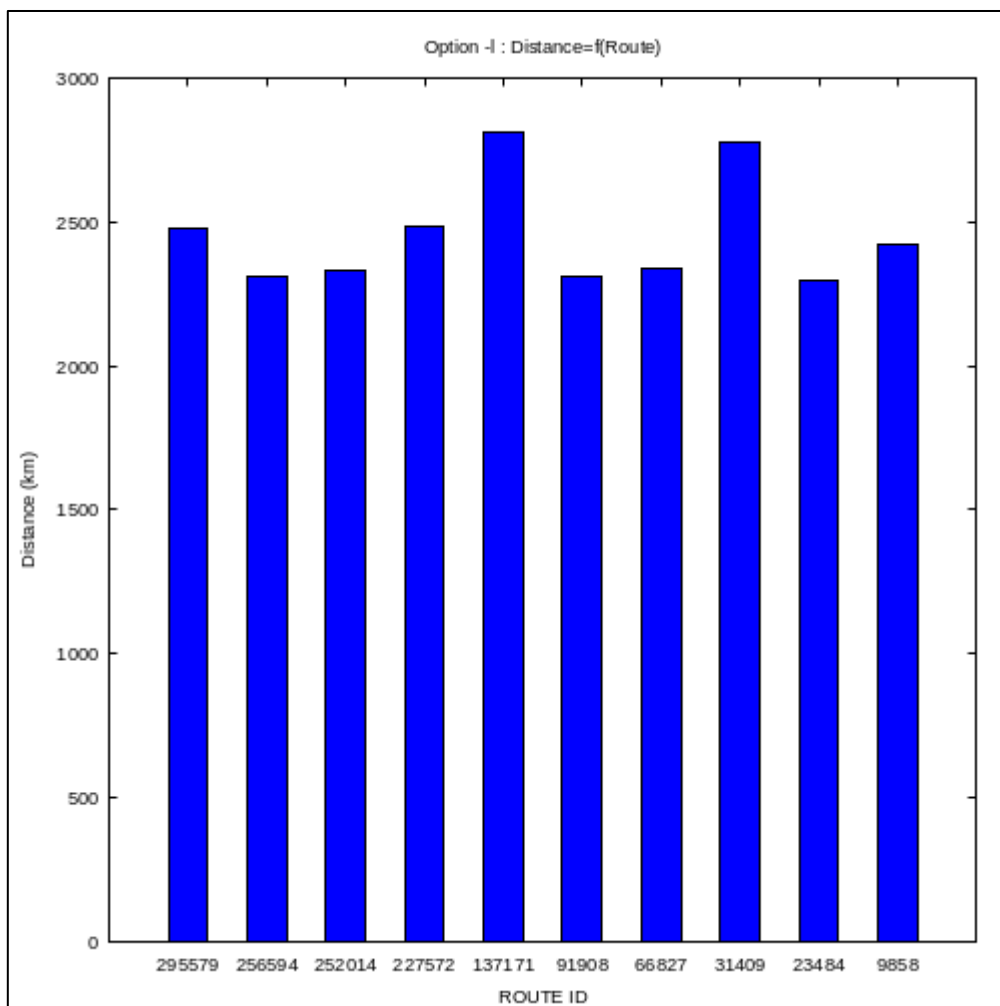
  gnuplot <<- GNU_PLOT
  set terminal pngcairo size 500,500 enhanced font 'Arial,8'
  set output './images/L.png'
  set xlabel 'ROUTE ID'
  set ylabel 'Distance (km)'
  set title 'Option -l : Distance=f(Route)'
  set style data histograms
  set boxwidth 1          # largeur de la barre
  set style histogram gap 1.5 # définir la distance vide entre les barres
  set yrange [0:3000]
  set style fill solid 1.00 border -1
  set xrange reverse
  plot './temp/option_l.txt' using 2:xtic(1) notitle linecolor rgb 'blue'

  GNU_PLOT
  # Nettoyage des fichiers temporaires
  #rm ./temp/option_l.txt
}
```

Là on calcule les distances totales pour chaque trajet à partir d'un fichier CSV donné, en utilisant AWK pour agréger les distances par ID de trajet. Ensuite, on trie les résultats par distance totale de manière décroissante (sort -r), sélectionne les 10 premiers trajets avec les distances les plus longues (head -10), et les enregistre dans un fichier temporaire.

En utilisant GNUPLOT, on crée un histogramme représentant ces trajets en fonction de leurs distances totales, avec les ID de trajet sur l'axe des x et les distances sur l'axe des y. Enfin, on nettoie les fichiers temporaires générés.

Résultat: (./images/L.png) les 10 trajets les plus longs



OPTION -t

Exécution :

./script.sh data/data.csv -t

Ce traitement, utilise le programme C avec AVL, à partir du fichier du script Shell en utilisant un Makefile. On va parler de chaque partie ci-dessous.

PARTIE 1

Dans ce programme, on veut lire les données du fichier csv et remplir un AVL selon les noms des villes.

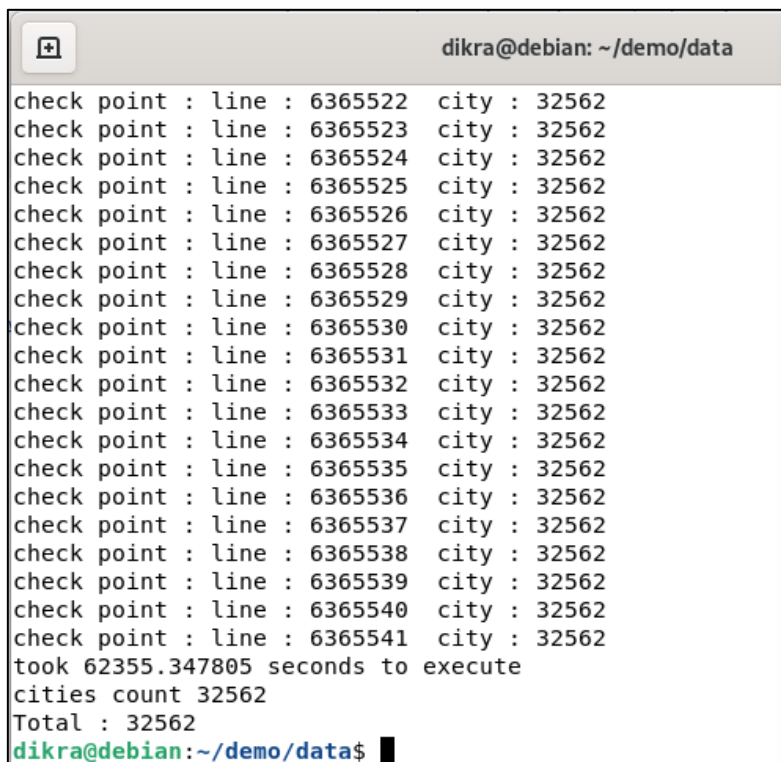
La structure **CityTrajets** permet de stocker le nom de la ville, le nombre de fois où cette ville a été parcouru dans un trajet (**count**), et le nombre de fois où cette ville a été une ville de départ (**departCount**).

La structure **AVLNode** c'est le nœud de notre AVL, et contient un élément de type **CityTrajets**, un fils gauche, un fils droit et le **height** qui est le facteur d'équilibre.

```
typedef struct CityTrajets {
    char *cityName;
    int count;
    int departCount; // Compter le nombre de fois où cette ville est une ville de départ
} CityTrajets;

// Definition de la structure pour un noeud de l'arbre AVL
typedef struct AVLNode {
    CityTrajets data;
    struct AVLNode *left;
    struct AVLNode *right;
    int height;
} AVLNode;
```

Ainsi, les fonctions et procédures nécessaires pour la gestion des AVL (insertion, équilibrage, etc...) sont aussi définies dans le programme C. Afin de construire l'AVL et compter le nombre de ville pour chaque trajet (sans redondances), l'exécution de cette partie de code prends environ **62355.347805 secondes** équivalent à **17 heures 19 minutes et 15 secondes**.



```
dikra@debian: ~/demo/data
check point : line : 6365522 city : 32562
check point : line : 6365523 city : 32562
check point : line : 6365524 city : 32562
check point : line : 6365525 city : 32562
check point : line : 6365526 city : 32562
check point : line : 6365527 city : 32562
check point : line : 6365528 city : 32562
check point : line : 6365529 city : 32562
check point : line : 6365530 city : 32562
check point : line : 6365531 city : 32562
check point : line : 6365532 city : 32562
check point : line : 6365533 city : 32562
check point : line : 6365534 city : 32562
check point : line : 6365535 city : 32562
check point : line : 6365536 city : 32562
check point : line : 6365537 city : 32562
check point : line : 6365538 city : 32562
check point : line : 6365539 city : 32562
check point : line : 6365540 city : 32562
check point : line : 6365541 city : 32562
took 62355.347805 seconds to execute
cities count 32562
Total : 32562
dikra@debian:~/demo/data$
```

Le résultat de cette exécution est sauvegarder dans le fichier [./demo/progc/trajets-ville.txt](#) , qui contient le parcours de l'AVL càd les noms des villes, le nombre de fois parcouru, ainsi que le nombre de fois où ces villes ont été des villes de départs. Le fichier contient toute l'AVL, voir 32560 lignes (32560 villes trouvées) voici un extrait :

```
1  AAST ; 201 ; 3
2  ABAINVILLE ; 133 ; 0
3  ABANCOURT ; 219 ; 6
4  ABAUCOURT HAUTECOURT ; 132 ; 1
5  ABAUCOURT SUR SEILLE ; 162 ; 1
6  ABBANS DESSOUS ; 82 ; 1
7  ABBANS DESSUS ; 83 ; 0
8  ABBARETZ ; 345 ; 17
9  ABBECOURT ; 238 ; 4
10 ABBENANS ; 169 ; 5
11 ABBEVILLE ; 69 ; 9
```

Pour calculer le nombre de fois une ville a été parcouru sans avoir des valeurs redondantes, on a essayer plusieurs approches, et celle qui a donnée un résultat similaire au prévus c'est la formule :

```
getNbrTrajetsByCity("../data/data.csv" , city.cityName , &countTrajetDeb ,&countTrajetFin ,&countCityDeb , count) ;

// probleme des villes dupliques dans un trajet
int intercitydebut = countTrajetDeb - countCityDeb ;
int intercityfin = countTrajetFin - intercitydebut ;
city.count = countCityDeb + intercitydebut + intercityfin ;
city.departCount = countCityDeb ;

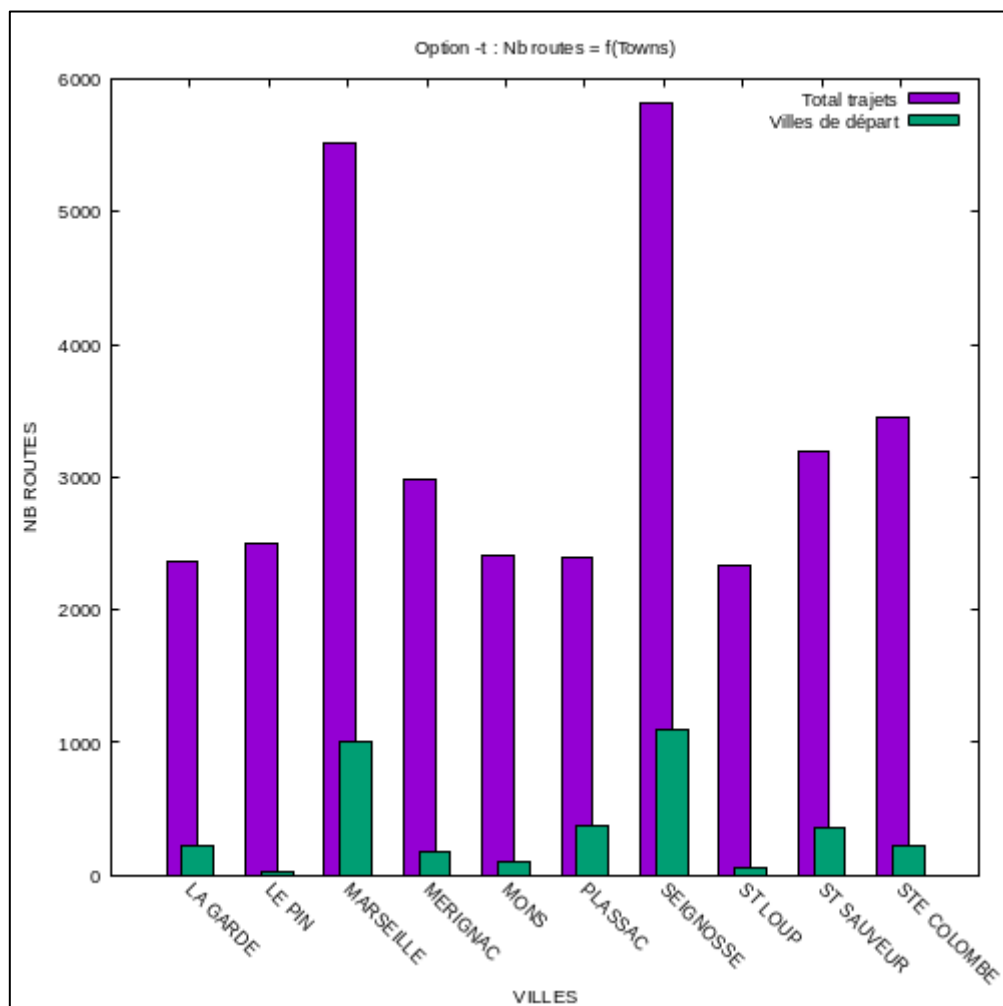
root = insert(root, city); // Insérer dans AVL
```

La fonction [getNbrTrajetsByCity\(...\)](#) nous donne le nombre avec la redondance, ensuite avec la formule présenter en haut on a pu avoir les bonnes valeurs.

PARTIE 2

Après avoir le résultat de l'AVL on va trier les données par le count réaliser avant, ensuite on gardera que les top10 qui va être ensuite trier encore une fois par ordre alphabétiques des villes. Pour ce, on a utiliser le trie à bulles pour le count et pour les noms. Le résultat final est sauvegarder dans [./demo/temp/option-t.txt](#) et sera utiliser par le script shell afin de générer la figure demander.

Résultat : Histogramme regroupé des villes en fonction du count et le count du départ.



Note 1 sur les résultats

Concernant les valeurs trouvées, ça correspond aux valeurs prévus. Sauf que, il y a des villes qui ne sont pas présent dans la figure prévue (donner par le prof), et qui ont été trouvé par nous-même. Or, les villes commun présent en histogramme ont exactement les mêmes valeurs totales.

Note 2 sur le Makefile

Il se peut que le Makefile ne marche pas. Lors des testes il nous semble instable. Vous pouvez utiliser directement le fichier [./demo/progc/main-all-save.c](#)

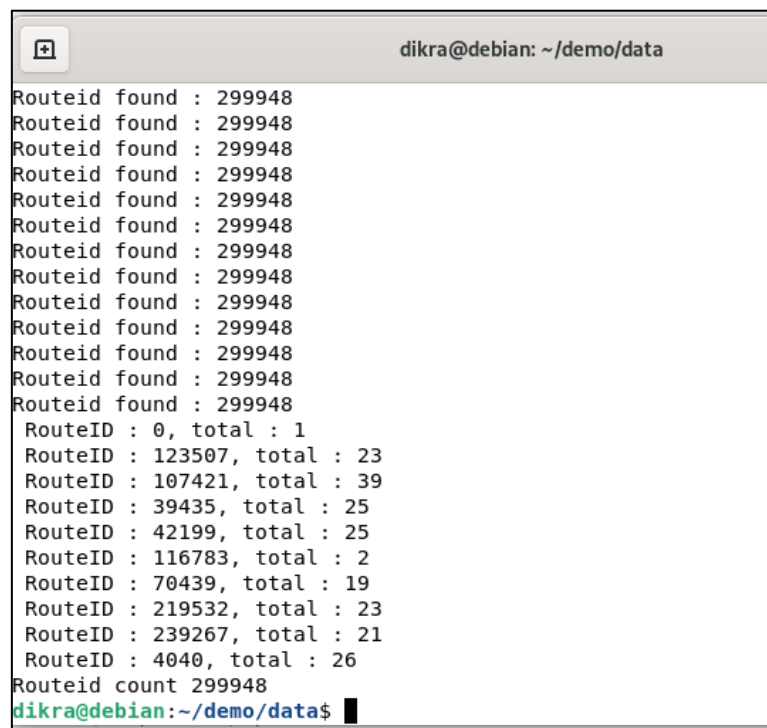
[en cours de réalisation]

Pour ce traitement on veut construire un AVL basé sur les RouteID. Avec une structure Trajet qui contient le RouteID, distance max, distance min et moyenne.

```
// Structure Trajets (routeid unique)
typedef struct Trajets {
    int routeid;
    int count; // combien de stepid
    int d_min;
    int d_max
    int moy
} Trajets;
```

On a commencé par réaliser les structures et fonctions nécessaires (voir *statistiques.c*)

Voici un extrait du résultat avec RouteID :



```
dikra@debian: ~/demo/data
Routeid found : 299948
Routeid found : 299948
Routeid found : 299948
Routeid found : 299948
Routeid found : 299948
Routeid found : 299948
Routeid found : 299948
Routeid found : 299948
Routeid found : 299948
Routeid found : 299948
Routeid found : 299948
Routeid found : 299948
Routeid found : 299948
RouteID : 0, total : 1
RouteID : 123507, total : 23
RouteID : 107421, total : 39
RouteID : 39435, total : 25
RouteID : 42199, total : 25
RouteID : 116783, total : 2
RouteID : 70439, total : 19
RouteID : 219532, total : 23
RouteID : 239267, total : 21
RouteID : 4040, total : 26
Routeid count 299948
dikra@debian:~/demo/data$
```

Travail en cours :

- Insérer dans un AVL (idem à traitement -t)
- Calculer la moyenne basée sur les distances min et max.
- Trier et récupérer uniquement les 20 premiers.
- Générer la figure demandée.

PROBLEMES RENCONTRES

Dans cette partie, on va discuter d'avantages le développement du traitement -t, qui nous a pris beaucoup de temps, et où on a rencontré plusieurs problèmes surtout dans la partie AVL.

Dans un premier temps, on a considéré chaque ligne comme un nœud dans la construction de l'AVL (voir *progc_faux.c*). L'arbre se crée correctement mais le problème c'est qu'en se basant sur le **RouteID**, qui n'est pas un identifiant unique, et lors de la création de l'AVL. Si on trouve un nœud existant avec même RouteID, le programme écrase l'ancien, pour en sauvegarder que le dernier RouteID trouver. Et du coup le résultat ne sera jamais correcte.

Et par conséquent, le résultat final des top 10, n'est pas comme prévu. Comme il est montré dans la figure ci-dessous :

