Étude de cas:

Analyse de marché du transport aérien canadien avec R

Atelier d'introduction à R

Beauchemin, David

Cabral Cruz, Samuel

Goulet, Vincent

Dans le cadre du colloque R à Québec

25 mai 2017

Table des matières

Préface

Dans le cadre du colloque "R à Québec" qui se tiendra le 25 et 26 mai 2017 sur le campus de l'Université Laval, une séance d'introduction au langage de programmation R sera offerte aux participants. Cette séance vise principalement la compréhension et la pratique permettant de maîtriser les rudiments de cet environnement de programmation. [4] Cette séance sera divisée en deux parties. En ce qui concerne la première partie, les fondements du langage seront visités d'une manière théorique sous la forme d'un exposé magistral. La deuxième partie, tant qu'à elle, se concentrera davantage sur la mise en pratique des notions abordées lors de la première partie grâce à la complétion d'une étude de cas cherchant à faire l'analyse de marché du transport aérien canadien. Ce document correspond en fait à la documentation complète de cette deuxième partie de formation.

Étant donné qu'il s'agit tout de même d'une formation pour débutants, la majorité du code sera déjà fournie, mais il n'en vaut pas moins la peine de parcourir ce projet si ce n'est que pour constater la puissance et la simplicité du langage. De plus, il est souvent difficile de mettre en perspective les innombrables fonctionnalités d'un langage lorsque nous commençons à l'utiliser. Cet étude de cas nous fournit ainsi un bel exemple d'enchaînement de traitements jusqu'à l'aboutissement ultime qui consiste à répondre aux questions que nous nous posions avant même d'amorcer l'analyse.

D'autre part, il est important de préciser que le code qui sera présenté ne correspond pas toujours à la manière la plus efficiente d'accomplir une tâche donnée. L'objectif principal étant ici la transmission de connaissances dans un dessin éducatif plutôt que d'une réelle analyse de marché. Nous tenons aussi à mentionner que bien qu'il s'agisse d'une formation s'adressant à des débutants, plusieurs notions qui seront mises en valeur font plutôt état de niveau intermédiaire et avancé, mais apportées toujours de manière simplifiée et accessible à quiconque qui n'aurait jamais travaillé avec R.

Nous tenons à remercier Vincent Goulet de nous avoir fait confiance dans l'élaboration de cette partie de la formation ainsi que tous les membres du comité organisationnel de l'évènement. Nous croyons sincèrement que R est un langage d'actualité qui se révèle un atout à quiconque oeuvrant dans un domaine relié de près ou de loin aux mathématiques.

Introduction

Dans le cadre de cette étude de cas, nous nous placerons dans la peau d'un analyste du département de la tarification oeuvrant au sein d'une compagnie canadienne se spécilisant dans le transport de colis par voies aériennes en mettant à profit le jeu de données d'*OpenFlights*. [3]

FIGURE 1 – Interface de l'outil OpenFlights

Parmi les bases de données disponibles, nous retrouvons :

- airports.dat - les données relatives à tous les aéroports du monde
- routes.dat - les données relatives à tous les trajets possibles dans le monde
- airlines.dat - les données relatives à toutes les compagnies aériennes du monde

Ainsi, notre mandat consistera, dans un premier temps, à analyser les bases de données mises à notre disposition afin de créer des fonctions utilitaires qui permettront de facilement intégrer les informations qu'elles contiennent lors de la tarification d'une livraison spécifique. Une fois cette tarification complétée, nous devrons fournir des chartes pour facilement estimer les prix d'une livraison qui s'avèreront être des outils indispensables au département de marketing et au reste de la direction. Après avoir transmis les documents en question, votre gestionnaire voulant s'assurer que la nouvelle tarification sera efficiente et profitable vous demandera d'analyser les prix de la concurrence pour en extrapoler

leur tarification. Finalement, vous serez appelé à comparer ces deux tarifications et la compétitivité de votre nouvelle tarification comparativement au reste du marché en procédant à une analyse stochastique.

OpenFlights

OpenFlights est un outil en ligne permettant de visualiser, chercher et filtrer tous les vols aériens dans le monde. Il s'agit d'un projet libre entretenu par la communauté via GitHub. [2] L'information rendu disponible y est étonnament très complète et facile d'approche ce qui en rend ce jeu de données très intéressant pour quiconque qui désire s'initié à l'analyse statistique.
https://openflights.org/

Bien qu'on n'en soit toujours qu'à l'introduction, nous tenons dès lors introduire des notions de programmation qui comparativement à celles qui suivront sont d'autre un peu plus général. Tout d'abord, afin de maximiser la portabilité des scripts que vous créerez dans le futur, il est important de rendre votre environnement de travail indépendant de la structure de dossier dans laquelle il se trouve. Pour ce faire, nous devons donc utiliser le principe de liens relatifs plutôt qu'absolus. En R, deux fonctions bien spécifiques nous fournient les outils afin de rendre cette tâche possible. Il s'agit de *getwd* et *setwd*. Comme leurs noms l'indiquent, elles servent respectivement à extraire le chemin vers l'environnement de travail et à le modifier.

De manière similaire qu'au sein d'un invité de commandes traditionnel, il est possible d'utiliser ".." (cd ..) afin de revenir à un niveau supérieur dans la structure de dossiers. Dans la plupart des cas, le code source d'un projet sera souvent isolé du reste du projet en le plaçant dans un sous-dossier dédié. [1]

Bref, comme le code source du présent projet se retrouve à l'intérieur du sous-dossier *dev* et que nous pourrions vouloir avoir accès à d'autres parties du répertoire au sein du code, le code suivant nous permettra de placer notre racine de projet à un niveau de dossier supérieur et de stocker ce chemin dans la variable *path*. Avec cette variable, tous les appels subséquents à des portions du répertoire pourront donc ce faire de manière relative puisque c'est cette variable path qui changera d'une architecture à un autre, tandis que la structure du répertoire restera toujours la même.

La deuxième notion que nous tenons à introduire immédiatement est celle de reproductibilité d'une analyse statistique. Comme vous le savez probablement, l'aléatoire pur n'existe pas en informatique, d'où la raison pour laquelle nous utiliserons plutôt le terme de nombres pseudo-aléatoires. Bien que cela peut

1. Il s'agit ici d'une excellente pratique de programmation et je dirais même indispensable si vous utilisez un gestionnaire de versions.

sembler étrange à première vue, il existe tout de même un point positif à tout ceci, soit la possibilité de fixer une racine au générateur de nombre pseudo-aléatoire (GNPA) ce qui aura comme impact de toujours produire les mêmes résultats pour autant que le GNPA utilisé soit le même. Comme nous pouvons le voir dans le code ci-dessous, l'instruction *set.seed* se chargera de fournir une valeur de départ aux calculs du GNPA.

Code Source 1 – Environnement de travail

```
#### Setting working directory properly ####
setwd('..')
path <- getwd()
set.seed(31459)
```

Code source du projet

Le code source du projet se retrouve dans son intégrité en annexe à ce document. N'hésitez pas à vous y référer au besoin.

Étude de cas

1.1 Extraction, traitement, visualisation et analyse des données

Cette section est certainement la plus importante de toutes, elle vise à faire un traitement adéquat et pertinent des données afin de pouvoir les réutiliser facilement dans les sections suivantes. Une mauvaise application des concepts d'extraction, de traitement et de visualisation des données peut entraîner des interprétations abérantes des phénomènes que nous cherchons à analyser.

1.1.1 Extraction

Les données d'OpenFlights possèdent l'avantage d'être téléchargeables directement via le web pour les rendre disponibles à notre environnement de travail. Pour ce faire, nous mettons à profit la fonction *read.csv*. Bien que le nom de la fontion indique qu'elle permet de lire un fichier présenté dans un format *.csv*, nous pouvons tout aussi bien utiliser cette fonction pour extraire des fichiers *.dat*. La différence principale entre ces deux types de fichiers et que les fichiers *.csv* utilisent un caractère d'encadrement des informations qui se trouve à être les doubles guillemets dans la majorité des cas. De plus, les fichiers *.csv* utiliseront comme leur nom l'indique la virgule à titre de séparateur bien que celui-ci puisse être modifié pour un symbol différent.[1] Lorsque nous jetons un coup d'oeil à la structure des fichiers *.dat* disponibles à la Figure 1.2, nous constatons que ceux-ci respectent à la fois les deux caractéristiques que nous venons de mentionner rendant ainsi l'utilisation de la fonction *read.csv* si naturelle.
Dans la même figure, on constate aussi l'absence d'une ligne servant à présenter les en-têtes de colonnes. Ceci pourra dans certains cas vous jouer de mauvais tours en ignorant la première ligne de données ou encore de considérer les titres comme étant des entrées en soi. [2] Bien qu'il serait possible de travailler avec des données sans nom, il s'agit ici d'une trèes mauvaise pratique à proscrire. Pour remédier à la situation, nous assignerons donc des noms aux colonnes grâce à l'attribut *colnames* d'un objet *data.frame* en lui passant un vecteur de noms.

2. La deuxième situation étant bien moins dramatique et plus facilement identifiable.

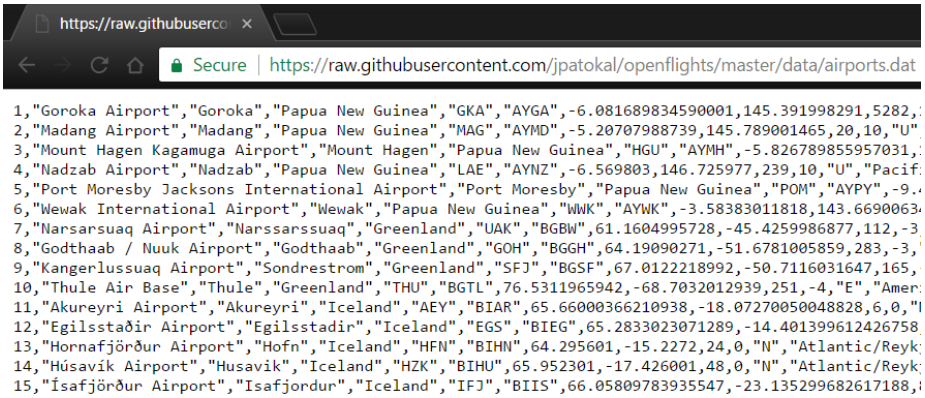

FIGURE 1.2 – Extrait du fichier airports.dat

Par défaut, lors de l'importation, la fonction *read.csv* retournera un *data.frame* en transformant les chaînes de caractères sous la forme de facteurs (*factors*). Cette action sera complètement transparente à l'utilisateur puisque l'affichage des variables ne sera pas impacté étant donné que R aura créer des formats d'affichage qui associe à chaque facteur la valeur unique corrspondante. Le seul impact réel réside dans la possibilité d'utiliser des fonctions à caractères mathématiques sur les données peu importe si ces dernières sont numériques ou non. Parmi ce genre de fonctions, nous pouvons penser à des fonctions d'agrégation (*clustering*) ou tout simplement à l'utilisation de la fonction *summary* permettant d'afficher des informations génériques sur le contenu d'un objet. Il est important de comprendre que les données ne sont toutefois plus représentées comme des chaînes de caractères, mais bien pas un indice référant à la valeur textuelle correspondante.

La manière de représenter des valeurs manquantes varira souvent d'une base de données à une autre. Une fonctionnalité très intéressante de la fonction *read.csv* est de pouvoir automatiquement convertir ces chaînes de caractères symboliques en *NA* ayant une signification particulaire dans R. Dans le cas présent, les valeurs manquantes sont représentées par \\n ou " " correspondant à un simple retour de chariot et un espace vide respectivement. Il suffit donc de passer cette liste de valeurs à l'argument *na.strings*.

> ***read.csv***
>
> La fonction *read.csv* possède plusieurs autres arguments très intéressants dans des situations plus pointue. Pour en savoir plus, nous vous invitons à consulter la documentation officielle.
> https://stat.ethz.ch/R-manual/R-devel/library/utils/html/read.table.html

Comme nous venons de le démontrer, l'extraction des données peut facilement devenir une tâche ingrate si nous n'avons aucune connaissance sur la manière dont l'information y a été entreposée. La rèegle d'or est donc de toujours avoir une idée globale de ce que nous cherchons à importer afin de bien paramétriser les fonctions. Si nous assemblons les différents aspects que nous venons d'aborder, nous aboutissons donc au code suivant :

Code Source 1.2 – Extraction des données

```
1  airports <- read.csv("https://raw.githubusercontent.com/jpatokal/
       openflights/master/data/airports.dat", header = FALSE, na.
       strings=c('\\N',''))
```

1.1.2 Traitement

Une fois en possession du jeu de données, il fut nécessaire de nettoyer ce dernier pour en rendre son utilisation plus simple selon nos besoins. Parmi les différentes modifications apportées nous retrouvons :

- Conserver que les observations relatives au aéroports Canadiens
- Filtrer les variables qui seront pertinentes dans le cadre de l'analyse que nous menons. [3]
- Alimentation des valeurs manquantes avec des sources de données externes (si possible) ou appliquer un traitement approximatif justifiable en documentant les impacts possibles sur le restant de l'analyse.

Nous considérons pertinent d'apporter quelques précisions sur le fonctionnement de R avant d'expliciter les traitements sus-mentionnés. Tout d'abord, R est un langage interprèté orienté objet à caractère fonctionnel optimisé pour le traitement vectoriel. Ces simples mots ne sont pas à prendre à la légère puisque ce n'est qu'en s'appropriant ce mode de penser que les futurs développeurs que vous êtes parviendront à utiliser R dans toute sa puissance, sa simplicité et son élégance. Par sa sémantique objet, R permet de définir des attributs aux objets créés. Comme il sera possible de le voir plus loin, l'accès à ces attributs se fera à l'aide de l'opérateur `$`. Vous vous demandez probablement : Comment

3. On ne devrait jamais travailler avec des informations superflues. Faire une pré-sélection de l'information ne fait qu'alléger les traitements et augmente de manière significative la compréhensibilité du programme.

savoir si nous sommes en présence d'un objet ? C'est simple, tout dans R est un objet ! Le langage R permet aussi de mimer le paradigme fonctionnel puisque les fonctions (qui sont en fait des objets) sont des valeurs à part entière qui peuvent être argument ou valeur d'une autre fonction. De plus, il est possible de définir des fonctions dites anonymes qui se révèleront très pratiques. Finalement, par son caractère vectoriel, la notion de scalaire n'existe tout simplement pas en R. C'est pour cette raison que l'utilisation de boucles est à proscrire (ou du moins à minimiser le plus possible). En effet, l'utilisation d'une boucle revient en quelque sorte à la création d'un nouveau vecteur et à la mise en place de processus itératifs afin d'exécuter la tâche demandée. Heureusement, par un raisonnement vectoriel, il est très simple de convertir ces traitements sous une forme vectorielle dans la plupart des cas. [**?**] Pour accéder à une valeur précise d'un vecteur, nous utiliserons l'opérateur `[]` en spécifiant les indices correspondants aux valeurs désirées, un vecteur booléen d'inclusion/exclusion ou encore un vecteur contenant les noms des attributs nommés qui nous intéressent.

Avec ces outils en mains, il devient ainsi très facile de filtrer les aéroports canadiens à l'aide de l'attribut que nous avons nommé *country* du data.frame *aiports*. Par un raisonnement connexe, la fonction *subset* nous offre aussi la possibilité de conserver que certaines variables contenues dans une table tout en appliquant des contraintes sur les observations à conserver. Le **??** dévoile au grand jour la dualité qui peut exister entre la multitude des fonctionnalités présentent en R.

Code Source 1.3 – Filtrer les données

```
1  airportsCanada <- airports[airports$country=='Canada',]
2  airportsCanada2 <- subset(airports,country == 'Canada')
3  all.equal(airportsCanada,airportsCanada2)
4
5  airportsCanada[is.na(airportsCanada$IATA),c("airportID","name","
       IATA","ICAO")]
6  subset(airportsCanada, is.na(IATA), select = c("airportID","name"
       ,"IATA","ICAO"))
```

Nous ne devons pas être surpris qu'il y ait autant de possibilités différentes de parvenir au même résultat, il s'agit là d'une des principales caractéristiques d'un logiciel libre puisque la responsabilité du développement continu ne dépend plus que d'une seule personne ou entité, mais bien de la communauté d'utilisateurs au complet. Ceci peut toutefois sembler mélangeant pour des nouveaux utilisateurs et la question suivante arrivera assez rapidement lorsque vous commencerez à developper vos propres applications : Quelle est la meilleure manière d'accomplir une tâche X ? La bonne réponse est tout aussi décevante que la premisse étant donné que chaque fonction aura été devéloppée dans un besoin précis si ce n'est que de rendre l'utilisation de fonctionnalité de base plus aisée et facile d'approche... C'est pourquoi nous conseillons plutôt d'adopter un mode de penser itératif, créatif et ouvert qui consiste à utiliser les fonctions qui vous semblent,

à la fois, les plus simples, les plus versatiles et les plus efficientes. À partir du moment où vous constaterez qu'une de ces trois caractéristiques n'est plus au rendez-vous, il suffira d'amorcer des recherches pour bonifier vos connaissances et améliorer vos techniques. C'est un peu le but de ce document de vous faire faire une visite guidée pour vous offrir un coffre d'outil qui facilitera vos premiers pas en R.

subset

Bien que la fonction *subset* simplifie énormément l'écriture de requêtes afin de manipuler des bases de données, celle-ci souffre par le fait même de devenir rapidement innefficiente lors de traitements plus complexes. D'autres packages tels que *dplyr* et *sqldf* divendront dans ces situations des meilleures alternatives.
https://www.rdocumentation.org/packages/raster/versions/2.5-8/topics/subset

Après avoir fait une présélection des données qui nous seront utiles dans le reste de l'analyse, nous avons constater que certaines variables n'étaient pas toujours totalement alimentées. Tout d'abord, la variable IATA n'était pas toujours définie pour tous les aéroports canadiens contrairement à ICAO. Étant donné la faible proportion des valeurs manquantes et du fait qu'une valeur fictive n'aurait qu'un impact minimal dans le cas de l'analyse, nous avons décider de remplacer les valeurs manquantes par les 3 dernières lettres du code ICAO. En regardant les aéroports canadiens possèdant les deux codes, nous observons que cette relation est respectée dans plus de 80% des cas. Une autre alternative aurait été de simplement prendre le code ICAO, mais le code IATA semblait beaucoup plus facile d'approche puisqu'il s'agit du code communément utiliser pour le transport des particuliers.

Les vrais problèmes au niveau des données résidaient davantage dans l'absence d'informations sur les zones temporelles de certains aéroports ainsi qu'un accès indirect à la province de correspondance de tous les aéroports. Heureusement, ce genre d'information ne dépend que de l'emplacement de l'entité dans le monde, ce qui rend la tâche beaucoup plus simple lorsque nous avons accès aux coordonnées géospatiales.

Adresse et coordonnées géospatiales

Dans la situation où seule l'adresse de l'entité aurait été disponible, nous aurions été contraint d'utiliser des techniques de géocodage qui permettent de transfomer une adresse en coordonnées longitude/latitude et parfois même altitude. Ce genre de transformation est devenu beaucoup plus accessible avec l'avancement de la technologie et plusieurs APIs sont disponibles gratuitement sur le web pour procéder à ce genre de transformation. Encore une fois, il vaut mieux bien se renseigner pour identifier l'interface qui répondra le mieux à nos besoins en considérant notamment :

- Format de l'entrant
- Format de retour
- Limitation du nombre de requêtes sur une période de temps donnée
- Efficacité de l'outil
- Méthode d'interpolation
- Précision des valeurs

https://www.programmableweb.com/news/7-free-geocoding-apis-google-bing-yahoo-and-mapquest/2012/06/21

La visualisation des données est une étape cruciale dans l'interprétation de celle-ci. De nombreuses fonctions R permettent de sortir plusieurs informations pertinentes sur les données. Tel que

- *View*[**?**] qui permet de visualiser le *data frame* R dans un onglet à part ;
- *head* [**?**] qui permet de visualiser dans la console les premières entrées ;
- *summary*[**?**] qui permet de visualiser différentes informations sur les données quantitatives et qualitatives telles que les quartiles, la moyenne, le maximum et le minimum pour les données quantitatives et la fréquence des observations de chacune des données qualitatives.

Par la suite, il devient possible de sortir différentes informations spécifiques de chaque variable. Par exemple, le nombre d'aéroports par ville a été extrait et présenter à l'aide de la fonction R *table* [**?**]

La prochaine étape consiste à nettoyer la pertinence des données ainsi que le remplissage des données absentes. On observe que les relations (colonne) *typeAirport*, *country* et *Source* ne sont pas pertinentes à notre situation puisque nous observons uniquement les aéroports canadiens. Elles seront retirées à l'aide du code suivant,

On cherche maintenant les données absentes, on observe à l'aide de la fonction *summary* que 27 aéroports ne comportent pas leur indicatif IATA, que l'on peut visualiser ainsi

Deux solutions sont possibles concernant cette situation, étant donné que seulement 18% des IATA sont manquants, il pourrait être possible d'ignorer et de retirer les données. Par contre, à l'aide de l'indicatif ICAO il est possible de

déterminer l'indicatif IATA. En effet, le ICAO correspond à un caractère unique par pays concaténer avec le IATA. À l'aide de cette information, il est possible de retrouver les informations manquantes facilement

La fonction R *substr* [**?**] permet de faire un découpage de la chaîne de caractère.

Finalement, on peut aussi observer que 52 aéroports ne comportent pas de fuseau horaire, deux options sont envisageables pour résoudre la problématique.

1. Étant donné que les fuseaux horaires sont déterminés par des positions géographiques, il est possible de déduire les informations manquantes à partir des aéroports à proximité
2. Utiliser des outils de cartographie pour retrouver les vrais fuseaux horaires.

La seconde solution a été adoptée, elle peut sembler complexe, mais avec les bons outils elle s'avère beaucoup plus simple et efficace. Cette partie nécessite l'installation de deux paquetages R soit *sp* [**?**] et *rgdal* [**?**].

On constate qu'il y a absence de la province de chacun des aéroports, alors pour le calcul des taxes cette information est nécessaire. Il est donc possible à l'aide des méthodes de cartographie vue précédemment ainsi qu'avec les données sur les frontières des provinces [**?**] de déterminer la province de l'aéroport. L'installation du paquetage R *sqldf* [**?**] est nécessaire pour l'exécution de cette partie. En effet, on applique les mêmes concepts de cartographie au territoire canadien afin de *quadriller* les provinces. Par la suite, à l'aide du langage *sql* [**?**] on [...]

Ainsi, on obtient des données complètes pour l'ensemble des aéroports canadiens. Certaines informations sont toutefois devenues obsolètes pour la suite de l'étude de cas, en effet les relations (colonnes) *timezone*, *DST* et *city* ne sont plus pertinentes. De plus, la relation *tzformat* doit être retirée car elle sera remplacé par la relation *tzmerge* créée par la requête *sql* précédente. Afin de renommer les nouvelles colonnes ajouter par la requête, on utilise la fonction R *rename* [**?**, **?**] du paquetage *plyr* [**?**].

On s'intéresse maintenant aux deux dernières bases données pertinentes pour l'étude de cas, les voies aériennes et les compagnies aériennes. Tout d'abord, on sélectionne à l'aide d'une requête *sql* les voies aériennes canadiennes. On analyse les informations présentes pour les voies aériennes et on observe que seulement 2 trajets ne sont pas des vols directs. Pour des fins de simplifications, tous les vols seront considérés comme des vols directs. Ainsi, les colonnes *codeshare* et *stops* sont inutiles.

On s'intéresse maintenant à créer un indice d'achalandage des aéroports en fonction des routes. Cet indice nous sera utile plus tard dans la simulation des aéroports d'arrivée. On observe d'abord les faits marquants des voies aériennes, valeurs maximum, moyenne et variance. On représente graphiquement la répartition de l'index qui correspond au nombre de voies aériennes de l'aéroport diviser par le maximum de nombre de voies aériennes. Cet index nous permet ainsi de représenter sur un graphique à bulles la densité des réseaux aériens pour chacun des aéroports.

1.2 Création de fonctions utilitaires

1.3 Communication des résultats

1.4 Analyse de la compétition

1.5 Ajustement de distribution statistiques sur données empiriques

1.6 Simulation et analyse de rentabilité

Conclusion

Bibliographie

[1] CSV vs. Delimited Flat Files : How to Choose. http://www.thoughtspot.com/blog/csv-vs-delimited-flat-files-how-choose.

[2] GitHub. https://github.com/.

[3] OpenFlights. https://openflights.org/data.html.

[4] R à Québec 2017. http://raquebec.ulaval.ca/2017/programme-r-a-quebec-2017.

Annexe A

Code Source

Cette annexe présente l'ensemble des codes sources constituant l'ensemble du projet. Ceux-ci se divisent sous la forme de 6 parties correspondantes aux différents thèmes abordés dans le présent document.

Code Source A.1 – Benchmark.R

```
# Source code for the creation of the benchmark.csv file

# Parameters of the simulation
n <- 100000
x <- matrix(c(runif(2*n)),ncol = 2,byrow = TRUE)

# Generate weights with a LogNormal distribution
mu1 <- log(3000)
sigma1 <- log(1.8)
exp(mu1+sigma1**2/2)
exp(2*mu1+4*sigma1**2/2)-exp(mu1+sigma1**2/2)**2
weights <- round(qlnorm(x[,1],mu1,sigma1)/1000,3)
hist(weights,breaks = 100,freq=FALSE)
mean(weights)
max(weights)

# Generate the errors on the weights
weightsTarifParam <- 0.7
weightsCost <- weights*weightsTarifParam
weightsError <- rnorm(n,mean(weightsCost),sd(weightsCost))
max(weightsError)
min(weightsError)
weightsFinalPrice <- weightsCost+weightsError
mean(weightsFinalPrice)
min(weightsFinalPrice)
var(weightsFinalPrice)

# Generate the distance with a LogNormal distribution
# routesCanada
# routesIATA <- cbind(paste(routesCanada$sourceAirport),paste(
    routesCanada$destinationAirport))
```

```
# routesDistance <- apply(routesIATA, 1, function(x) airportsDist(x[1],x[2])$value)
# max(routesDistance)
# mean(routesDistance)
mu2 <- log(650)
sigma2 <- log(1.4)
(distances <- round(qlnorm(x[,2],mu2,sigma2)))
hist(distances,breaks = 100,freq=FALSE)
mean(distances)
max(distances)

# Generate the errors on the distances
distancesTarifParam <- 0.0275
distancesCost <- distances*distancesTarifParam
distancesError <- rnorm(n,mean(distancesCost),sd(distancesCost))
distancesFinalPrice <- distancesCost+distancesError
mean(distancesFinalPrice)
var(distancesFinalPrice)
max(distancesFinalPrice)
min(distancesFinalPrice)

# Generate total price
baseCost <- 10
#taxRate <- sum(table(airportsCanada$province)*as.numeric(paste(taxRates$taxRate)))/length(airportsCanada$province)
taxRate <- 1.082408
profitMargin <- 1.15
(totalCost <- round((baseCost + weightsFinalPrice + distancesFinalPrice)*profitMargin*taxRate,2))
mean(totalCost)
var(totalCost)
max(totalCost)
min(totalCost)

# Export to csv format
(dataExport <- cbind(weights,distances,totalCost))
colnames(dataExport) <- c("Poids (Kg)","Distance (Km)","Prix (CAD $)")
write.csv(dataExport,paste(path,"/Reference/benchmark.csv",sep=''),row.names = FALSE)
```