Rapport sur le processus de calcul de la valeur moyenne de concentration de radon dans les bâtiments suisses

Office fédéral de la santé publique (OFSP)

Niels Lachat

6 janvier 2022

Table des matières

Chapitre 1

Introduction

1.1 But du document

Le but de ce document est de décrire le processus de traitement de données utilisé pour calculer la valeur moyenne de concentration de radon dans les bâtiments suisses à partir des données de la base de données nationale de mesures du radon.

Note : Le code source du programme servant à effectuer ce calcul doit être mis à disposition avec ce document pour que le lecteur puisse s'y référer si besoin.

1.2 Remarques

Veuillez prendre compte des remarques suivantes lors de la lecture de ce document :

— Sauf indication contraire, toutes les dates sont données au format ISO-8601 [1].

— Sauf indication contraire, les chemins d'accès faisant référence à des dossiers ou des fichiers du code source sont donnés depuis la racine du projet.

1.3 Abbréviation utilisées dans le document

Afin d'améliorer la lisibilité du document, nous utiliserons les abbréviations suivantes :

— **OFS** Office fédéral de la statistique

— **BDD** Base de données

1. `https://fr.wikipedia.org/wiki/ISO_8601`

1.4 Technologies utilisées pour le traitement de données

Le programme effectuant le traitement de données et autres calculs est écrit dans le langage de programmation Scala (2.12.15) [2]. La librairie utilisée pour faciliter les opérations sur des grands ensembles de données est Apache Spark (3.1.2) [3].

Pour simplifier l'importation des données dans le programme, l'outil OpenRefine (3.5.0) [4] a été utilisé pour prétraiter les données (voir chapitre 3).

1.5 Référence pour la méthode de calcul

Le calcul décrit dans le présent document est basé sur le calcul de la valeur moyenne de concentration du radon effectué en 2004. Le document de référence se nomme "Strahlenexposition Bevölkerung 2004" et la section d'intérêt est la section 2 "Beitrag von Radon".

Dans ce document, le calcul est effectué en 3 étapes.

1.5.1 Correction pour la saison

La moyenne de concentration doit être calculée en considérant une moyenne annuelle, en prenant en compte que les séjours à l'intérieur sont plus longs en hiver qu'en été. Cette correction a déjà été appliquée au moment où les mesures sont entrées dans la BDD radon.

Pour refléter cette correction dans la notation (et pour garder la même notation que dans le document de référence), nous considérerons que A_0 représente la valeur mesurée de concentration de radon en $[Bq/m^3]$, corrigée pour représenter une moyenne annuelle.

1.5.2 Correction pour l'étage

Pour mieux refléter la concentration de radon dans laquelle la population vit, une correction a été appliquée lors du calcul de 2004. Les valeurs mesurées au-dessous de l'étage moyen d'habitation des bâtiments du canton sont attenuées, et les valeurs mesurées au-dessus sont accentuées. La formule de correction est la suivante :

$$A_{0,St} = A_0 \cdot e^{-0.19 \cdot (St_m - St)} \tag{1.1}$$

où

— $A_{0,St}$ représente la concentration corrigée pour l'étage, en $[Bq/m^3]$

— A_0 représente la concentration avant correction, en $[Bq/m^3]$

2. `https://www.scala-lang.org/`
3. `https://spark.apache.org/docs/3.1.2/`
4. `https://openrefine.org/`

— e^x représente la fonction exponentielle évaluée en x
— St_m L'étage moyen d'habitation dans le canton où la mesure a été effectuée, $\in \mathbb{R}$
— St L'étage auquel la mesure a été effectuée, $\in \mathbb{Z}$

1.5.3 Calcul final de la concentration moyenne suisse

Le calcul final de la concentration moyenne de radon dans les locaux d'habitation et de séjour est calculée comme suit :

$$A = \frac{1}{\sum_{i=1}^{N} pop_i} \sum_{i=1}^{N} pop_i [\frac{1}{N_i} \sum_{j=1}^{N_i} [\frac{1}{N_{ij}} \sum_{k=1}^{N_{ij}} A_{0,St_{ijk}}]] \tag{1.2}$$

où
— A représente la concentration moyenne de radon finale, en $[Bq/m^3]$
— pop_i représente la population de la commune i
— N représente le nombre total de communes pour lesquelles des mesures existent
— N_i représente le nombre de bâtiments mesurés dans la commune i
— N_{ij} représente le nombre de mesures effectuées dans le bâtiment j de la commune i
— $A_{0,St_{ijk}}$ représente la k-ème mesure, corrigée pour l'étage, effectuée dans le bâtiment j de la commune i, en $[Bq/m^3]$

1.6 Vue d'ensemble du processus de traitement de données

Nous décrirons ici le processus du traitement des données, depuis les données brutes décrites dans le chapitre 2, jusqu'à l'obtention de la valeur moyenne de concentration de radon. Chaque étape du processus sera expliquée en détail dans les chapitres suivants.

1.6.1 Importation et prétraitement

Dans cette étape, les données brutes sont importées dans le programme et sont prétraitées pour les étapes suivantes.

1.6.2 Filtration

Dans cette étape, les données brutes prétraitées sont filtrées afin de supprimer les entrées invalides et les valeurs qu'on ne veut pas retenir pour le calcul final de la valeur moyenne (par exemple les mesures dans des locaux qui ne sont pas des locaux de séjour prolongé).

1.6.3 Corrections

Dans cette étape, les mesures brutes subissent une correction ou ajustement qui dépend de l'étage dans lequel la mesure a été prise.

1.6.4 Calcul de la valeur moyenne

Finalement les données traitées sont utilisées pour calculer la concentration moyenne de radon dans les bâtiments suisses selon la méthode appliquée en 2004.

Chapitre 2

Sources des données

Dans ce chapitre, nous décrirons les sources des données utilisées pour le calcul de la valeur moyenne de concentration de radon.

2.1 Base de données nationale du radon

La source principale des données est la BDD nationale du radon. Cette BDD contient 279'257 entrées, une pour chaque mesure de radon effectuée. Elle a été exportée initialement sous forme de fichier excel nommé "Messungen_020721.xlsx". C'est ce fichier que nous considèreront comme la source initiale des données.

— **Période de récolte des données** [1982]-[2021]
— **Date de publication** 2021-07-02
— **Lien vers les données** Données non disponibles publiquement

2.2 Nombre d'étages des bâtiments par canton

Afin d'effectuer la correction mentionnée au point 1.5.2 il est nécessaire d'avoir des données concernant l'étage d'habitation moyen par canton. Malheureusement, cette information n'est pas disponible directement sur le catalogue de données de l'OFS. Ce qui est par contre disponible, c'est le nombre de bâtiments qui ont un certain nombre d'étages, par canton. Pour obtenir le fichier source dans le format que le programme accepte, il faut aller sur ce lien `https://www.pxweb.bfs.admin.ch/pxweb/fr/px-x-0902010000_101/px-x-0902010000_101/px-x-0902010000_101.px` et sélectionner les options comme dans la figure 2.1. Il faut ensuite cliquer sur le bouton "Continuer" et enregistrer les données sous forme de fichier CSV.

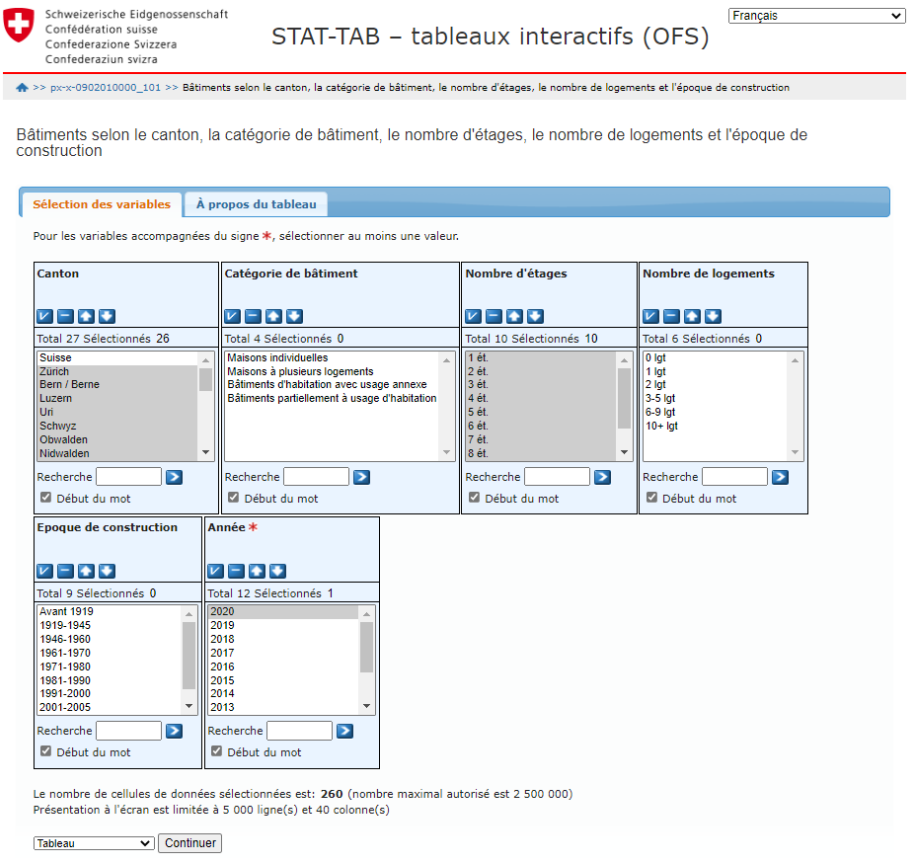

FIGURE 2.1 – Options à sélectionner pour exporter les données du nombre d'étage des bâtiments par canton

Dans le chapitre 3 nous expliquerons comment passer de ces données brutes à l'étage moyen d'habitation de la population, par canton.

- **Période de récolte des données** [2009]-[2020]
- **Date de publication** 2021-10-07
- **Lien vers les données** `https://www.bfs.admin.ch/bfs/fr/home/statistiques/construction-logement/batiments.assetdetail.19024696.html`

2.3 Nombre d'habitants par commune

La moyenne de concentration est pondérée pour chaque commune par le nombre d'habitants dans la commune. Il est donc nécessaire d'avoir la population de chaque commune suisse.

- **Période de récolte des données** [2004]-[2020]

— **Date de publication** 2021-03-26
— **Lien vers les données** `https://www.bfs.admin.ch/bfs/fr/home/statistiques/statistique-regions/portraits-regionaux-chiffres-cles/communes.assetdetail.15864461.html`

Chapitre 3

Importation et prétraitement (preprocessing)

3.1 BDD radon

3.1.1 Importation

Transformation du fichier xlsx en csv

A partir du fichier source au format *xlsx*, nous utilisons le programme Excel 2016 pour exporter les données dans un format plus facile à traiter par OpenRefine. Lorsque le fichier est ouvert avec Excel 2016, il faut suivre les étapes suivantes : Fichier > Exporter > Modifier le type de fichier > CSV (séparateur : point-virgule) (*.csv) > Enregistrer sous.

Nettoyage par OpenRefine

Nous devons à présent passer le fichier par OpenRefine afin d'éliminer des colonnes inutiles et de transformer les données dans un format acceptable par notre programme. Pour ce faire, il faut créer un nouveau projet OpenRefine à partir du fichier csv crée précedemment, puis l'exporter en appliquant les options d'exportation trouvable dans la source du programme sous "metadata/rn-db_openrefine-options.json". Une chose à ne pas oublier lors de cette étape est de cocher la case "Ignore facets and filters and export all rows" afin d'exporter toutes les lignes du fichier.

Le fichier ainsi généré doit être placé sous "data/rn-db/main.csv"

3.1.2 Prétraitement

La BDD radon ne nécessite pas de prétraitement avant de pouvoir être utilisée par le programme. Il faut cependant que les dates de début et de fin de la mesure du radon (colonnes START_TIME et END_TIME) soient transformées au format ISO 8601. La transformation de la date est faite par Spark en lui fournissant comme option d'importation "dateFormat" → "dd.MM.yyyy".

Pour connaître les champs de la BDD qui sont utilisés dans le programme, voir "src/main/scala/data/RadonDatabase.scala" (voir annexe 8.1).

3.2 Étage moyen d'habitation

3.2.1 Importation

Nous transformons le fichier téléchargé à la section 2.2 à l'aide d'OpenRefine. Les options par défault peuvent être gardées pour créer le projet OpenRefine. Il faut ensuite utiliser les options d'exportation trouvables dans le fichier suivant : "metadata/canton-floors-buildings_openrefine-options.json" pour exporter le fichier traité.

Le fichier ainsi généré doit être placé sous "data/avg-floor/2020/main.csv"

3.2.2 Prétraitement

Pour plus de détails sur les opérations décrites dans cette section, veuillez vous référer à "src/main/scala/data/AverageFloor2020.scala" (voir annexe 8.2).

Conversion des colonnes en format utile

Les colonnes "Canton" et "Nombre d'étages" doivent être traitées avant de pouvoir être utilisées car elles ont un format non compatible avec les autres sources de données.

- Pour la colonne "Canton", chaque nom de canton écrit en toutes lettres est converti dans le code à 2 lettres du canton (exemple "Bern / Berne" → "BE")
- Pour la colonne "Nombre d'étages", chaque description du nombre d'étages est converti en un nombre simple (exemple "4 ét." → 4, "10+ ét." → 10 [1])

Calcul de l'étage d'habitation moyen par canton

Pour calculer l'étage moyen d'habitation par canton, nous utilisons la moyenne pondérée du nombre d'étages par le nombre de bâtiments qui ont ce nombre d'étages :

1. On notera d'ailleurs ici qu'une partie du sens des données initiales est perdu comme on considère que tous les bâtiments qui ont **plus de** 10 étages ont **exactement** 10 étages. Cette simplification ne devrait cependant pas avoir un impact significatif sur le résultat comme la majorité des bâtiments en Suisse ont moins de 10 étages (ceci est visible facilement dans les données du nombre d'étage des bâtiments par canton).

$$E_c = \frac{1}{2} \cdot \frac{\sum_{e=1}^{10} n_{e,c} \cdot e}{\sum_{e=1}^{10} n_{e,c}} \tag{3.1}$$

où

— E_c représente l'étage moyen d'habitation dans le canton c

— $n_{e,c}$ représente le nombre de bâtiments qui ont e étages dans le canton c

Il est très important de noter ici qu'il faut distinguer le **nombre** d'étages, et l'étage moyen **d'habitation**. C'est pour cette raison que nous multiplions la moyenne pondérée du **nombre** d'étages par un facteur $\frac{1}{2}$. Nous pouvons en effet observer que par exemple, pour un bâtiment de 5 étages, les habitants résident en moyenne à l'étage 2.5 (en supposant que tous les étages sont uniformément occupés).

Ci-dessous, nous montrons les valeurs de l'étage moyen d'habitation par canton, en 2004, 2009 et 2020. Les données de 2004 sont issues du document cité à la section 1.5. Les données pour 2009 et 2020 sont calculées à partir du nombre d'étages par bâtiment (données de l'OFS). Les valeurs pour 2004 et 2009 sont données comme points de comparaison avec les valeurs de 2020 qui sont utilisées pour le calcul de la valeur moyenne de la concentration de radon. Il est important de noter ici, que les différences entre les valeurs de 2004 et 2009 sont plus importantes qu'entre 2009 et 2020. Un exemple frappant est la valeur pour le canton de Genève qui passe de 2.89 en 2004 à 1.67 en 2009 et reste à 1.67 en 2020. Ceci nous laisse à penser qu'il y a une différence dans la méthode de calcul ou dans la source des données entre ce qui avait été calculé en 2004 et ce que nous avons calculé pour 2009 et 2020. Cependant il nous est impossible d'être certain d'où vient la différence car le rapport de 2004 est imprécis sur la source de ces données. La seule indication qui nous est donnée est : "Quelle : Bundesamt für Statistik" (en français : "Source : Office fédéral de la statistique"). Nous considérons cependant que notre source de données et notre méthode de calcul semblent correctes et nous utiliserons donc les valeurs d'étage moyen d'habitation pour 2020 que nous avons calculées pour la suite des calculs.

Étage moyen d'habitation, en 2004, en 2009 et en 2020 respectivement

KANTON	ETAGE
AG	1.03
AI	0.57
AR	0.86
BE	1.11
BL	1.18
BS	1.94
FR	1.02
GE	2.89
GL	0.93
GR	1.17
JU	0.78
LU	1.4
NE	1.57
NW	1.31
OW	1.05
SG	1.08
SH	1.12
SO	0.92
SZ	1.2
TG	0.88
TI	1.29
UR	0.99
VD	1.52
VS	1.3
ZG	1.48
ZH	1.21

KANTON	ETAGE
AG	1.21
AI	1.26
AR	1.38
BE	1.27
BL	1.28
BS	1.89
FR	1.21
GE	1.67
GL	1.28
GR	1.32
JU	1.2
LU	1.35
NE	1.48
NW	1.46
OW	1.3
SG	1.33
SH	1.31
SO	1.18
SZ	1.37
TG	1.23
TI	1.13
UR	1.31
VD	1.32
VS	1.26
ZG	1.55
ZH	1.54

KANTON	ETAGE
AG	1.23
AI	1.16
AR	1.38
BE	1.27
BL	1.3
BS	1.92
FR	1.22
GE	1.67
GL	1.29
GR	1.3
JU	1.19
LU	1.37
NE	1.46
NW	1.48
OW	1.3
SG	1.33
SH	1.31
SO	1.18
SZ	1.4
TG	1.23
TI	1.17
UR	1.29
VD	1.33
VS	1.25
ZG	1.58
ZH	1.6

3.3 Nombre d'habitants par commune

3.3.1 Importation

À partir du fichier *xlsx* téléchargé à la section 2.3, nous utilisons OpenRefine pour transformer les données dans un format traitable par notre programme.

Les options d'importation pour OpenRefine doivent être ajustées pour traiter correctement le fichier *xlsx* qui a une structure peu régulière. Veuillez vous référer à la figure 3.1.

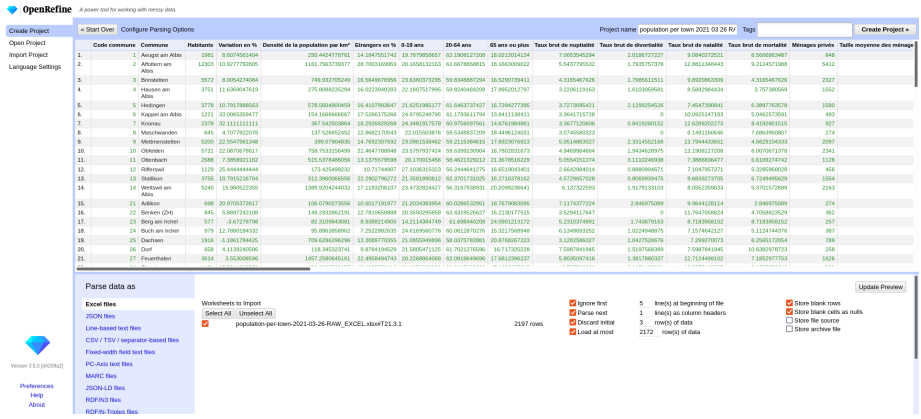

FIGURE 3.1 – Options à sélectionner pour créer le projet OpenRefine. Ignore first = 5, Parse next = 1, Discard initial = 3, Load at most = 2172

Il faut ensuite utiliser les options d'exportation trouvables dans le fichier suivant : "metadata/population-per-town_openrefine-options.json" pour exporter le fichier traité.

Le fichier ainsi généré doit être placé sous "data/population-per-town/main.csv".

3.3.2 Prétraitement

Le seul prétraitement qui doit être effectué est de convertir les nombres (numéro de commune et population correspondante) d'un type de nombre réel (avec des .0 en suffixe) en nombre entier. Ceci est fait dans "src/main/scala/-data/TownPopulations.scala".

Chapitre 4

Filtration

Dans ce chapitre nous décrirons les différents filtres qui sont appliqués à la BDD radon dans le but d'éliminer les mesures invalides et les mesures qui ne correspondent pas aux critères du calcul (locaux non habités).

Pour plus de détails concernant le processus de filtration, veuillez vous référer au fichier "src/main/scala/v01/Filtering.scala" (voir 8.2).

Note : Pour chaque filtre on compte le nombre d'entrées maximal que ce filtre peut supprimer, et quel pourcentage du nombre d'entrées initial cela représente. Ceci permet d'observer quels filtres auront le plus grand impact sur le calcul final. Ceci est montré comme suit :

Nombre maximal d'entrées supprimées	N
Pourcentage maximal d'entrées supprimées	$P\%$

Où
- N représente le nombre d'entrées maximal que ce filtre peut supprimer
- P représente le pourcentage du total initial d'entrées. $P = N/279'257$

4.1 Conserver uniquement les mesures valides

Ce filtre conserve uniquement les entrées où la colonne VALIDIERUNG vaut "Y".

Nombre maximal d'entrées supprimées	11'781
Pourcentage maximal d'entrées supprimées	4.22%

4.2 Conserver uniquement les locaux avec séjour de personnes

Ce filtre conserve uniquement les entrées où la colonne PERSONENAUFENTHALT vaut "YES_LONG" ou "YES_SHORT"

Nombre maximal d'entrées supprimées	79'358
Pourcentage maximal d'entrées supprimées	28.42%

4.3 Conserver uniquement les entrées qui contiennent une mesure

Ce filtre conserve uniquement les entrées qui contiennent une valeur dans la colonne RADONKONZENTRATION_BQ_M3.

Nombre maximal d'entrées supprimées	6'486
Pourcentage maximal d'entrées supprimées	2.32%

4.4 Exclure les mesures effectuées dans des communes qui n'existent plus

Après inspection des données, il s'est avéré que certaines mesures ont été effectuées dans des communes qui n'existent plus dans les données de population de l'OFS (voir 2.3). Cela est probablement dû principalement à des fusions de communes. Il serait techniquement possible de trouver les communes qui ont fusionné et remplacer les anciens numéros de communes par les nouveaux numéros de communes fusionnées, mais cela serait relativement couteux en terme de temps. C'est pourquoi il a été décidé d'ignorer les mesures effectuées dans ces communes "disparues".

Ce filtre fonctionne en plusieurs étapes :

4.4.1 Trouver les numéros des communes disparues

Pour ce faire, nous collectons l'ensemble des numéros de communes qui apparaissent dans la BDD radon (colonne GEMEINDENUMMER), nommons cet ensemble C_{Rn}. Nous collectons ensuite l'ensemble des numéros de communes qui apparaissent dans le tableau des populations des communes (voir 2.3), nommons cet ensemble C_{Pop}.

Les numéros des communes disparues sont donnés par la différence entre les ensembles :

$$D = C_{Rn} - C_{Pop} \tag{4.1}$$

Pour les données que nous utilisons, $|D| = 60$.

Pour plus de détails concernant cette étape, voir "src/main/scala/main/DataExploration.scala".

4.4.2 Conserver uniquement les mesures effectuées dans les communes non disparues

Nous conservons ensuite uniquement les mesures qui ont été effectuées dans une commune qui n'apparait pas dans D.

Nombre maximal d'entrées supprimées	5'078
Pourcentage maximal d'entrées supprimées	1.82%

4.5 Garder uniquement les mesures effectuées dans les étages entre 0 et 20

Afin d'inclure uniquement les locaux d'habitation dans le calcul de la valeur moyenne de concentration, nous gardons uniquement les entrées où

$$0 \leq ETAGE \leq 20 \tag{4.2}$$

Nombre maximal d'entrées supprimées	73'059
Pourcentage maximal d'entrées supprimées	26.16%

4.6 Exclure les mesures effectuées avant un assainissement radon

Pour les bâtiments qui ont subit un assainissement radon, des mesures ont été effectuées *avant* et *après* assainissement. Afin de représenter plus précisement la valeur de concentration de radon réelle, il faut exclure les mesures effectuées avant assainissement des mesures considérées pour le calcul de la valeur moyenne de concentration.

Ceci se fait en plusieurs étapes :

4.6.1 Trouver les identifiants des mesures effectuées avant assainissement

Dans cette étape, nous trouvons les identifiants des mesures (ID_MESSUNG) effectuées avant assainissement (dans les cas où il existe des mesures avant et après assainissement pour un bâtiment donné). Appelons cette liste d'identifiants L.

La méthode pour effectuer cette opération est la suivante :

1. Grouper les entrées par bâtiment (à l'aide de la colonne ID_HAUS)

2. Pour chaque bâtiment on trouve s'il y a eu un assainissement (si pour le bâtiment en question, une des entrées a MESSTYP = "Messung nach der Sanierung", on considère qu'il y a eu assainissement)
3. Si c'est le cas, on ajoute tous les identifiants des mesures effectuées avant assainissement dans le bâtiment à L (l'identifiant de toutes les entrées où MESSTYP = "Messung" pour le bâtiment). Autrement, L reste inchangée.

Pour plus de détails, voir "src/main/scala/v01/Filtering.scala :findIdsOfMeasurementsBeforeRemediation".

4.6.2 Exclure les mesures effectuées avant assainissement

Cette étape est relativement simple une fois qu'on a obtenu L. Il suffit de supprimer les mesures dont l'identifiant se trouve dans L.

Nombre maximal d'entrées supprimées	5'623
Pourcentage maximal d'entrées supprimées	2.01%

4.7 Exclure les mesures de concentration = 0 $[Bq/m^3]$

La BDD contient des entrées où RADONKONZENTRATION_BQ_M3 vaut 0, et il est impossible qu'un appareil de mesure correctement utilisé mesure une moyenne de 0 $[Bq/m^3]$. Il faut donc exclure ces mesures du calcul final. Par mesure de précaution, le filtre conserve uniquement les mesures > 0 $[Bq/m^3]$et supprime donc aussi d'éventuelles valeurs négatives de concentration.

Nombre maximal d'entrées supprimées	1'678
Pourcentage maximal d'entrées supprimées	0.60%

4.8 Conclusion

Après filtration, la BDD radon est réduite à 168'056 entrées. À l'origine, elle contenait 279'257 entrées. L'étape de filtration supprime donc 111'201 entrées.

Il faut noter que le nombre d'entrées supprimées ne correspond pas à la somme du nombre maximal d'entrées supprimées par chaque filtre, car certaines entrées sont exclues par plusieurs filtres à la fois (par exemple une mesure dans une cave effectuée avant un assainissement sera supprimée à cause des filtres décrits en 4.2, 4.5, 4.6).

Chapitre 5

Corrections

Après filtration, les mesures doivent encore être corrigées en fonction de l'étage auquel elles ont été mesurées (comme expliqué dans la section 1.5.2).

Nous décrirons ici les étapes qui permettent de corriger les mesures en fonction de l'étage. Pour chaque entrée, il faut :

1. Récupérer le canton dans lequel la mesure a été effectuée (colonne KANTON)
2. Trouver l'étage d'habitation moyen du canton grâce aux données importées à la section 3.2.2 (valeur St_m)
3. Récupérer l'étage auquel la mesure a été effectué (colonne ETAGE $\rightarrow$ valeur St)
4. Récupérer la valeur mesurée de concentration (colonne RADONKONZENTRATION_BQ_M3 $\rightarrow$ valeur A_0)
5. Calculer la valeur corrigée de concentration grâce à la formule 1.1

Pour plus de détail, veuillez vous référer à "src/main/scala/v01/Corrections.scala" (voir 8.4)

Chapitre 6

Calcul de la valeur moyenne

Dans ce chapitre, nous décrirons la façon dont est calculée la valeur moyenne à partir de la BDD radon filtrée et corrigée et des autres données nécessaires.

Le calcul s'effectue en plusieurs étapes que nous détaillerons ici. Chaque étape représente une partie de la formule 1.2.

Pour plus de détails, veuillez vous référer à "src/main/scala/v01/ComputeAverageVolumetricActivity.scala" (voir 8.5)

6.1 Calcul de la moyenne de concentration par bâtiment

Nous commençons par calculer la moyenne de concentration de radon par bâtiment, selon la formule suivante (moyenne arithmétique des valeurs mesurées dans le bâtiment j de la commune i) :

$$A_{ij} = \frac{1}{N_{ij}} \sum_{k=1}^{N_{ij}} A_{0,St_{ijk}} \tag{6.1}$$

où

— A_{ij} représente la concentration moyenne de radon dans le bâtiment j de la commune i, en $[Bq/m^3]$

— N_{ij} représente le nombre de mesures effectuées dans le bâtiment j de la commune i

— $A_{0,St_{ijk}}$ représente la k-ème mesure, corrigée pour l'étage, effectuée dans le bâtiment j de la commune i, en $[Bq/m^3]$

Concrètement, ce calcul se fait en groupant les entrées par ID_HAUS et en calculant la moyenne pour chaque groupe.

6.2 Calcul de la moyenne de concentration par commune

Cette étape est très similaire à l'étape précédente. Nous décrirons donc uniquement la formule nécessaire au calcul.

$$A_i = \frac{1}{N_i} \sum_{j=1}^{N_i} A_{ij} \tag{6.2}$$

où

— A_i représente la concentration moyenne de radon dans la commune i, en $[Bq/m^3]$

— N_i représente le nombre de bâtiments mesurés dans la commune i

— A_{ij} représente la concentration moyenne de radon dans le bâtiment j de la commune i, en $[Bq/m^3]$

6.3 Calcul final de la concentration moyenne suisse

Finalement, dans cette étape nous calculons la valeur moyenne de concentration de radon dans les bâtiments suisses. La formule est une moyenne pondérée des concentrations des communes par les populations des communes.

$$A = \frac{1}{\sum_{i=1}^{N} pop_i} \sum_{i=1}^{N} pop_i \cdot A_i \tag{6.3}$$

où

— A représente la valeur moyenne de concentration de radon dans les bâtiments suisses, en $[Bq/m^3]$

— pop_i représente la population de la commune i

— N représente le nombre total de communes pour lesquelles des mesures existent

— A_i représente la concentration moyenne de radon dans la commune i, en $[Bq/m^3]$

D'un point de vue du programme, on commence par calculer la somme du produit de la population par la concentration de la commune, puis on divise par la population totale **des communes dans lesquelles des mesures ont été effectuées.** (pour plus de détails voir 8.5)

Chapitre 7

Conclusion

En conclusion, nous avons pu reproduire le calcul de la valeur moyenne de radon dans le bâtiments suisses de la même façon qu'elle avait été calculée en 2004.

Il pourrait également être intéressant de comparer ce résultat avec d'autres méthodes de calcul (ajustement avec une distribution normale, utilisation d'autres fonctions que la moyenne pour calculer une valeur représentative d'un bâtiment (maximum des mesures, moyenne géométrique des mesures)). Ceci pourra être effectué dans un second temps.

Chapitre 8

Annexes

8.1 RadonDatabase.scala

```
/**
  * ch-avg-radon: Calcul de la moyenne de radon en suisse.
  * Copyright (C) 2022 Niels Lachat
  *
  * This program is free software: you can redistribute it and/or
    ↪  modify
  * it under the terms of the GNU General Public License as
    ↪ published by
  * the Free Software Foundation, either version 3 of the License
    ↪ , or
  * any later version.
  *
  * This program is distributed in the hope that it will be
    ↪ useful,
  * but WITHOUT ANY WARRANTY; without even the implied warranty
    ↪ of
  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
  * GNU General Public License for more details.
  *
  * You should have received a copy of the GNU General Public
    ↪ License
  * along with this program. If not, see <https://www.gnu.org/
    ↪ licenses/>.
  *
  * Author: Niels Lachat <niels.lachat@bag.admin.ch>
  *
  * For the full license, see the file 'COPYING' at the root of
    ↪ this repository.
```

```
  */

package data

import utils.RowUtils
import main.{Symbols => s}
import org.apache.spark.sql.DataFrame
import org.apache.spark.sql.Row
import org.apache.spark.sql.SparkSession
import org.apache.spark.sql.types._
import org.tinylog.scala.Logger

import DatasetInterface._

object RadonDatabase {

  /** to get this dataset from the raw Rn database, use
    * metadata/rn-db-UNIX_CSV_UTF8_PARTIAL-openrefine-options.json
      ↪ with
    * OpenRefine (https://openrefine.org/)
    */
  val datasetPath =
    f"${datasetsBasePath}/rn-db/main.csv"
  val schema = StructType(
    Array(
      StructField(s.canton, StringType, false),
      StructField(s.houseId, LongType, false),
      StructField("EGID", LongType, true),
      // StructField("PARZELLENNUMMER", StringType, true),
      // StructField("VORNAME", StringType, true),
      // StructField("NAME", StringType, true),
      // StructField("FUNKTION", StringType, true),
      // StructField("FIRMA", StringType, true),
      // StructField("ADDRESSTYP", StringType, true),
      // StructField("STRASSE_NR", StringType, true),
      // StructField("GEBAEUDEBEZEICHNUNG", StringType, true),
      // StructField("POSTLEITZAHL", StringType, true),
      // StructField("ORT", StringType, true),
      // StructField("ADRESSE_ALT", StringType, true),
      StructField(s.townCode, LongType, true),
      // StructField("GEMEINDE", StringType, true),
      // StructField("GEBAEUDEKATEGORIE", StringType, true),
      // StructField("BAUJAHR", StringType, true),
      // StructField("Anzahl_Etagen", IntegerType, true),
      // StructField("Fundament", StringType, true),
      // StructField("Struktur_des_Fundaments", StringType, true),
```

```
      // StructField("Untergeschoss vorhanden", StringType, true),
      // StructField("Hanglage", StringType, true),
      // StructField("Kontrollierte_Lftung", StringType, true),
      // StructField("KOORDINATENBEZUG", StringType, true),
      // StructField("LV95_ORDINATE", StringType, true),
      // StructField("LV95_ABSZISSE", StringType, true),
      // StructField("LV03_ORDINATE", StringType, true),
      // StructField("LV03_ABSZISSE", StringType, true),
      // StructField("BEMERKUNG_HAUS", StringType, true),
      StructField(s.measurementId, LongType, false),
      // StructField("MESSPROTOKOLL", StringType, true),
      StructField(s.measurementType, StringType, true),
      // StructField("DOSIMETER_NO", StringType, true),
      // StructField("IDENTIFIER", StringType, true),
      StructField(s.startDate, DateType, true),
      StructField(s.endDate, DateType, true),
      // StructField("KEYWORDS", StringType, true),
      // StructField("ID_Raum", StringType, true),
      StructField(s.roomType, StringType, true),
      // StructField("RAUMBEZEICHNUNG", StringType, true),
      StructField(s.peopleStay, StringType, true),
      StructField(s.floor, IntegerType, true),
      StructField("RADONEXPOSITION_KBQH_M3", StringType, true),
      StructField(s.rnVolConc, DoubleType, true),
      // StructField("MESSUNSICHERHEIT", StringType, true),
      StructField(s.validation, StringType, true)
      // StructField("SANIERUNGSFRIST_MESSUNG", StringType, true),
      // StructField("BEMERKUNG", StringType, true),
      // StructField("BENUTZER", StringType, true)
    )
  )
}

case class RadonDatabase(
    // allows inserting mock test data during testing
    providedDatasetPath: String = RadonDatabase.datasetPath,
    providedSchema: StructType = RadonDatabase.schema
)(implicit spark: SparkSession)
    extends DatasetInterface(
      datasetPath = providedDatasetPath,
      schema = providedSchema,
      options = Map(
        "sep" -> ",",
        "header" -> "true",
        "lineSep" -> "\n",
        "dateFormat" -> "dd.MM.yyyy",
```

```
        "maxColumns" -> providedSchema.fields.length.toString()
      )
    )

object MeasurementTypes extends Enumeration {
  type MeasurementType = Value

  val beforeRemediation = Value("Messung")
  val afterRemediation = Value("Messung nach der Sanierung")
}
```

8.2 AverageFloor2020.scala

```
/**
  * ch-avg-radon: Calcul de la moyenne de radon en suisse.
  * Copyright (C) 2022 Niels Lachat
  *
  * This program is free software: you can redistribute it and/or
      ↪  modify
  * it under the terms of the GNU General Public License as
      ↪ published by
  * the Free Software Foundation, either version 3 of the License
      ↪ , or
  * any later version.
  *
  * This program is distributed in the hope that it will be
      ↪ useful,
  * but WITHOUT ANY WARRANTY; without even the implied warranty
      ↪ of
  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
  * GNU General Public License for more details.
  *
  * You should have received a copy of the GNU General Public
      ↪ License
  * along with this program. If not, see <https://www.gnu.org/
      ↪ licenses/>.
  *
  * Author: Niels Lachat <niels.lachat@bag.admin.ch>
  *
  * For the full license, see the file 'COPYING' at the root of
      ↪ this repository.
  */

package data
```

```
import org.apache.spark.sql.SparkSession
case class AverageFloor2020(implicit spark: SparkSession)
    extends AverageFloorFromFSO(2020)
```

8.3 Filtering.scala

```
/**
  * ch-avg-radon: Calcul de la moyenne de radon en suisse.
  * Copyright (C) 2022 Niels Lachat
  *
  * This program is free software: you can redistribute it and/or
      ↪  modify
  * it under the terms of the GNU General Public License as
      ↪ published by
  * the Free Software Foundation, either version 3 of the License
      ↪ , or
  * any later version.
  *
  * This program is distributed in the hope that it will be
      ↪ useful,
  * but WITHOUT ANY WARRANTY; without even the implied warranty
      ↪ of
  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
  * GNU General Public License for more details.
  *
  * You should have received a copy of the GNU General Public
      ↪ License
  * along with this program. If not, see <https://www.gnu.org/
      ↪ licenses/>.
  *
  * Author: Niels Lachat <niels.lachat@bag.admin.ch>
  *
  * For the full license, see the file 'COPYING' at the root of
      ↪ this repository.
  */

package data_processing

import data.MeasurementTypes
import data.RadonDatabase
import data.TownPopulations
import main.CommandLineArgs
import main.DataExploration
```

```
import main.Flags
import utils.RowUtils.getAsOption
import main.{Symbols => s}
import org.apache.spark.sql.DataFrame
import org.apache.spark.sql.Row
import org.apache.spark.sql.SparkSession
import org.tinylog.scala.Logger

import java.time.LocalDate
import utils.RowUtils

case class Filter(descr: String, filterFunc: Row => Boolean)

/**
 * Wrapper for the filtered radon db.
 * This make the computation stage explicit.
 * @param df The underlying DataFrame for the radon db
 */
case class FilteredRnDb(df: DataFrame)

object Filtering {

  /** For buildings where there are measurements before and after
      ↪ radon
    * remediation (DE: Sanierung, FR: Assainissement), returns all
        ↪ the
    * measurement ids (ID_MESSUNG) of measurements that were made
        ↪ _before_
    * remediation.
    *
    * @param rnDb
    * The radon database
    * @return
    * The measurements ids of measurements made before remediation
    */
  def findIdsOfMeasurementsBeforeRemediation(
      rnDb: RadonDatabase
  )(implicit spark: SparkSession): Set[Long] = {
    import spark.implicits._
    rnDb.df
      .select(s.houseId, s.measurementId, s.measurementType)
      .groupByKey[Long]((row: Row) => row.getAs[Long](s.houseId))
      .flatMapGroups[Long](
        (houseId: Long, correspondingRows: Iterator[Row]) => {
          // need to convert to list because "one should never use
              ↪ an iterator after calling a method on it."
```

```
        // https://www.scala-lang.org/api/2.12.0/scala/
            ↪ collection/Iterator.html
        // and we need to use it multiple times
        val correspondingRowsList = correspondingRows.toList

        val hasMeasurementAfterRemediation =
            ↪ correspondingRowsList
          .exists(row =>
            row.getAs[String](
              s.measurementType
            ) == MeasurementTypes.afterRemediation.toString()
          )
        if (hasMeasurementAfterRemediation) {
          // need to return all ids of measurements before
              ↪ remediation
          correspondingRowsList
            .filter(row =>
              row.getAs[String](
                s.measurementType
              ) == MeasurementTypes.beforeRemediation.toString()
            )
            .map(row => row.getAs[Long](s.measurementId))
        } else {
          // there is no measurement to be removed
          List()
        }
      }
    )
    .collect()
    .toSet // make a set for fast lookup (to test)
}

def filterRnDb(
    rnDb: RadonDatabase,
    townPopulations: TownPopulations
)(implicit cliArgs: CommandLineArgs, spark: SparkSession):
    ↪ FilteredRnDb = {
  val invalidTownCodes =
    DataExploration.differenceOfTownCodes(rnDb.df,
        ↪ townPopulations.df)
  Logger.debug(f"Number␣of␣disappeared␣towns,␣|D|␣=␣${
      ↪ invalidTownCodes.size}")
  val idsOfMeasurementsBeforeRemediation =
    findIdsOfMeasurementsBeforeRemediation(rnDb)

  // 2021-10-22: MGR says that we don't include rooms below 0
```

```
val validFloorMin = 0
val validFloorMax = 20

val baseFilters = List[Filter](
  Filter(
    f"Keep␣only␣valid␣entries␣(${s.validation}␣==␣Y)",
    row => row.getAs[String](s.validation) == "Y"
  ),
  // 2021-12-03: Filtre probablement inutile car on a dj
  // PERSONENAUFENTHALT == YES_LONG/SHORT
  // Filter(
  // "Filter out any uninhabited locals",
  // row =>
  // row.getAs[String](s.roomType) != "Keller" &&
  // row.getAs[String](s.roomType) != "K" &&
  // row.getAs[String](s.roomType) != "?"
  // ),
  Filter(
    f"Keep␣only␣locals␣where␣people␣stay␣(${s.peopleStay}␣==␣
        ↪ YES_LONG/SHORT)",
    row => {
      val peopleStay = row.getAs[String](s.peopleStay)
      peopleStay == "YES_LONG" ||
      peopleStay == "YES_SHORT"
    }
  ),
  Filter(
    "Keep␣only␣rows␣with␣measurements",
    row => !row.isNullAt(row.fieldIndex(s.rnVolConc))
  ),
  Filter(
    "Keep␣only␣measurements␣done␣in␣towns␣with␣town␣codes␣that
        ↪ ␣are␣in␣the␣2021-07-02␣OFS␣data␣for␣population",
    row => !invalidTownCodes.contains(row.getAs[Long](s.
        ↪ townCode))
  ),
  Filter(
    f"Keep␣only␣valid␣floor␣values␣($validFloorMin␣<=␣${s.
        ↪ floor}␣<=␣$validFloorMax)",
    row =>
      getAsOption[Int](row, row.fieldIndex(s.floor)) match {
        case Some(floor: Int) =>
          validFloorMin <= floor && floor <= validFloorMax
        case None => true
      }
  ),
```

```
  Filter(
    "Keep␣only␣measurements␣after␣remediation␣(for␣houses␣
        ↪ where␣there␣was␣a␣remediation)",
    row => {
      val measurementId = row.getAs[Long](s.measurementId)
      !idsOfMeasurementsBeforeRemediation.contains(
          ↪ measurementId)
    }
  ),
  Filter(
    "Keep␣only␣rows␣where␣radon␣concentration␣is␣>␣0␣Bq/m",
    row => {
      val rnVolConc = RowUtils.getAsOption[Double](row, s.
          ↪ rnVolConc)
      rnVolConc match {
        case Some(value) => value > 0
        case None => true
      }
    }
  )
)
val keepOnlyBefore2004 = Filter(
  "Keep␣only␣measurements␣before␣2004",
  row => {
    val maybeEndDate =
      Option(row.getAs[java.sql.Date](s.endDate))
        .map(_.toLocalDate())
    maybeEndDate match {
      case Some(date) => date.isBefore(LocalDate.of(2004, 1,
          ↪ 1))
      case None => true
    }
  }
)
import Flags._
val filters = if (cliArgs.isFlagEnabled(
    ↪ Reproduce2004Computation)) {
  keepOnlyBefore2004 :: baseFilters
} else {
  baseFilters
}

if (!cliArgs.isFlagEnabled(RunFast)) {
  val filterToNbRowsRemoved = filters.map(filter =>
    (filter, rnDb.df.filter(row => !filter.filterFunc(row)).
        ↪ count())
```

```
      )

      filterToNbRowsRemoved.foreach { case (filter, count) =>
        Logger.debug(f"Filter␣'${filter.descr}'␣removed␣at␣most␣
            ↪ $count␣rows")
      }
    }

    val filteredDF = rnDb.df.filter((row: Row) => filters.forall(f
        ↪  => f.filterFunc(row)))
    FilteredRnDb(filteredDF)
  }
}
```

8.4 Corrections.scala

```
/**
  * ch-avg-radon: Calcul de la moyenne de radon en suisse.
  * Copyright (C) 2022 Niels Lachat
  *
  * This program is free software: you can redistribute it and/or
      ↪  modify
  * it under the terms of the GNU General Public License as
      ↪ published by
  * the Free Software Foundation, either version 3 of the License
      ↪ , or
  * any later version.
  *
  * This program is distributed in the hope that it will be
      ↪ useful,
  * but WITHOUT ANY WARRANTY; without even the implied warranty
      ↪ of
  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
  * GNU General Public License for more details.
  *
  * You should have received a copy of the GNU General Public
      ↪ License
  * along with this program. If not, see <https://www.gnu.org/
      ↪ licenses/>.
  *
  * Author: Niels Lachat <niels.lachat@bag.admin.ch>
  *
  * For the full license, see the file 'COPYING' at the root of
      ↪ this repository.
```

```
  */

package data_processing

import data.AverageFloor
import data.RadonDatabase
import utils.RowUtils
import main.Symbols
import org.apache.spark.sql.Column
import org.apache.spark.sql.DataFrame
import org.apache.spark.sql.Row
import org.apache.spark.sql.SparkSession

/**
 * Wrapper for the corrected radon db.
 * This make the computation stage explicit.
 * @param df The underlying DataFrame for the radon db
 */
case class CorrectedRnDb(df: DataFrame)

object Corrections {
  @deprecated(
    "This␣is␣not␣needed␣anymore,␣the␣values␣in␣the␣db␣are␣already␣
        ↪ corrected"
  )
  def correctForSeason = ???

  /** Applies the correction for the floor of the measurement
    *
    * @param rnDb
    * The original radon db
    * @param avgFloor
    * DataFrame containing the floor for each
    * @param spark
    * @return
    */
  def correctForFloor(rnDb: FilteredRnDb, avgFloor: AverageFloor)
      ↪ (implicit
      spark: SparkSession
  ): CorrectedRnDb = {
    import spark.implicits._
    def correctionFunction(
        measuredConcentration: Double,
        cantonAvgFloor: Double,
        floorOption: Option[Int]
    ): Double = {
```

```
      import Math.exp
      floorOption match {
        case Some(floor) => {
          val expConst = -0.19

          measuredConcentration * exp(
            expConst * (cantonAvgFloor - floor)
          ) // see README.md for reference

        }
        case None => measuredConcentration
      }
    }

    val avgFloorMap = avgFloor.toMap
    val floorCorrectedConc = rnDb.df
      .map[(Long, Double)]((row: Row) => {
        val canton = row.getAs[String](Symbols.canton)
        val cantonAvgFloor = avgFloorMap(canton)
        val floorOption =
          RowUtils.getAsOption[Int](row, row.fieldIndex(Symbols.
              ↪ floor))
        val measuredConcentration = row.getAs[Double](Symbols.
            ↪ rnVolConc)
        val measurementId = row.getAs[Long](Symbols.measurementId)
        (
          measurementId,
          correctionFunction(measuredConcentration, cantonAvgFloor
              ↪ , floorOption)
        )
      })
      .toDF(Symbols.measurementId, Symbols.floorCorrectedConc)

    val correctedDF = rnDb.df
      .drop(
        Symbols.rnVolConc
      ) // drop the uncorrected measurements to prevent use after
          ↪ this point
      .join(floorCorrectedConc, Symbols.measurementId)
    CorrectedRnDb(correctedDF)
  }
}
```

8.5 ComputeAverageVolumetricActivity.scala

```
/**
  * ch-avg-radon: Calcul de la moyenne de radon en suisse.
  * Copyright (C) 2022 Niels Lachat
  *
  * This program is free software: you can redistribute it and/or
      ↪  modify
  * it under the terms of the GNU General Public License as
      ↪ published by
  * the Free Software Foundation, either version 3 of the License
      ↪ , or
  * any later version.
  *
  * This program is distributed in the hope that it will be
      ↪ useful,
  * but WITHOUT ANY WARRANTY; without even the implied warranty
      ↪ of
  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
  * GNU General Public License for more details.
  *
  * You should have received a copy of the GNU General Public
      ↪ License
  * along with this program. If not, see <https://www.gnu.org/
      ↪ licenses/>.
  *
  * Author: Niels Lachat <niels.lachat@bag.admin.ch>
  *
  * For the full license, see the file 'COPYING' at the root of
      ↪ this repository.
  */

package v01

import main.DataExploration

import org.apache.spark.sql.DataFrame
import org.apache.spark.sql.Row
import org.apache.spark.sql.SparkSession
import main.{Symbols => s}
import data_processing.CorrectedRnDb

object ComputeAverageVolumetricActivity {
```

```
/** Compute the average radon concentration according to the
    ↪ 2004 method.
  *
  * @param rnDb
  * Radon database (filtered for invalid data)
  * @param townPopulations
  * Map from GEMEINDENUMMER to POPULATION
  * @return
  * The average concentration
  */
def computeA(corrRnDb: CorrectedRnDb, townPopulations:
    ↪ DataFrame)(implicit
    spark: SparkSession
): Double = {
  val buildingAvgs = computeBuildingAvg(corrRnDb)
  val townAvgs = computeTownAvg(buildingAvgs)
  computeAvgFromTownAvgs(townAvgs, townPopulations, corrRnDb)
}

/** Compute the average radon concentration for each building.
    ↪ The grouping is
  * done on ID_HAUS because EGID has some null values.
  *
  * @param rnDb
  * @return
  * The processed DataFrame with the average building
      ↪ concentrations.
  * RADONKONZENTRATION_BQ_M3 is the average for each building
  */
private def computeBuildingAvg(rnDb: CorrectedRnDb): DataFrame
    ↪ = {
  rnDb.df
    .groupBy(s.houseId, s.townCode)
    .avg(s.floorCorrectedConc)
}

/** Compute the average radon concentration for each town. The
    ↪ grouping is
  * done on GEMEINDENUMMER
  *
  * @param buildingAvgs
  * DataFrame containing the average for each building
  * @return
  * The processed DataFrame with the average town concentrations
      ↪ .
  * RADONKONZENTRATION_BQ_M3 is the average for each building
```

```
    */
  private def computeTownAvg(buildingAvgs: DataFrame): DataFrame
      ↪ = {
    buildingAvgs
      .groupBy(s.townCode)
      .avg(f"avg(${s.floorCorrectedConc})")
  }

  /** Compute the average concentration weighted by the town
      ↪ populations.
    *
    * @param townAvgs
    * @param townPopulations
    * @return
    */
  private def computeAvgFromTownAvgs(
      townAvgs: DataFrame,
      townPopulations: DataFrame,
      rnDb: CorrectedRnDb
  )(implicit spark: SparkSession): Double = {
    import spark.implicits._
    // sum of pop_i * town_avg
    val sumOfWeightedAvgs = townAvgs
      .join(
        townPopulations,
        s.townCode
      )
      .map((row: Row) => {
        val townAvg =
          row.getAs[Double](f"avg(avg(${s.floorCorrectedConc}))")
        val townPop = row.getAs[Long](s.population)
        townPop * townAvg
      })
      .reduce(_ + _)
    // compute sum of pop_i
    val townCodesInRnDb =
      DataExploration.valuesOfColumn[Long](rnDb.df, s.townCode)
    val sumOfPopulation = townPopulations
      .filter((row: Row) =>
        townCodesInRnDb.contains(row.getAs[Long](s.townCode))
      ) // only towns in the rnDb
      .map((row: Row) => row.getAs[Long](s.population))
      .reduce(_ + _)
    sumOfWeightedAvgs / sumOfPopulation
  }
}
```