Spécifications

Sonny Klotz - Jean-Didier Pailleux - Malek Zemni

Interface de chargement, de contrôle
et d'analyse statistique des données
pour la constitution d'un graphe de flux

17/04/2017

Module *Projet*

Table des matières

Introduction

Ce document va décrire l'ensemble des exigences fonctionnelles que doit satisfaire notre produit final, c'est-à-dire les différentes fonctionnalités que notre application va fournir. Cette description va prendre en compte les caractéristiques des outils de développement choisis.

Notre outil, Python, est un langage de programmation hybride alliant programmation fonctionnelle et objet. Dans Python, le type de données n'est connu qu'à l'exécution (typage dynamique), par conséquent, ces types ne seront pas indiqués dans les signatures des fonctionnalités. Ils seront précisés dans des paragraphes explicatifs.

Pour les parties qui s'appuient sur une interaction avec l'utilisateur, notre démarche de description des fonctionnalités va essentiellement prendre en compte l'*expérience utilisateur*[1]. Cette description sera donc axée sur la qualification du résultat et du ressenti de l'utilisateur lors de la manipulation de l'interface fournie (une illustration à l'aide de croquis), plutôt que sur les points techniques de l'application (fonctions et classes).

Les fonctionnalités de notre application seront présentées selon les modules de l'organigramme établi dans le cahier des charges. Ces modules eux-mêmes seront regroupés en packages. Ce document va donc décrire, pour chaque package de l'organigramme, les fonctionnalités de ses modules : d'abord ceux du package de chargement des données, ensuite ceux du package d'analyse descriptive des données et enfin ceux du package de l'interface web. Dans une dernière partie, un glossaire sera consacré à la définition des types de données utilisés.

1 Package Chargement des données

Ce package est composé de 2 modules qui ont pour fonction de traiter le fichier de données fourni : une vérification de son format et une analyse de son contenu. On pourra aussi parler de API puisque ce package peut être éventuellement livré en sortie.

1.1 Module Vérification format fichier

Ce module va vérifier le format du fichier de données fourni en entrée en 3 points. Il aura donc 3 fonctionnalités de vérification et une fonctionnalité globale d'ouverture du fichier.

1. Fonctionnalité de vérification de l'ouverture du fichier :

1. http ://uxdesign.com/ux-defined

```
verifOuverture(fichierCSV)
```

Paramètres :

`fichierCSV` : `TextIoWrapper` - représente le fichier CSV fourni.

Retour : variable de type entier signalant un succès (0) ou une erreur (1).
Description : cette fonction prend en entrée le fichier CSV ouvert. Elle vérifie que le paramètre `fichierCSV` contient bien des informations représentant un fichier quelconque, et renvoie l'entier 0 si c'est le cas, l'entier 1 sinon.

2. Fonctionnalité de vérification de l'extension du fichier ouvert :

```
verifExtenstion(fichierCSV)
```

Paramètres :

`fichierCSV` : `TextIoWrapper` - représente le fichier CSV fourni.

Retour : variable de type entier signalant un succès (0) ou une erreur (2).
Description : cette fonction prend en entrée le fichier CSV ouvert. Elle vérifie que l'information décrivant l'extension du fichier dans le paramètre `fichierCSV` correspond bien au format CSV, et renvoie l'entier 0 si c'est le cas, l'entier 2 sinon.

3. Fonctionnalité de vérification de l'accessibilité en lecture du fichier :

```
verifLecture(fichierCSV)
```

Paramètres :

`fichierCSV` : `TextIoWrapper` - représente le fichier CSV fourni.

Retour : variable de type entier signalant un succès (0) ou une erreur (3).
Description : cette fonction prend en entrée le fichier CSV ouvert. Elle vérifie que le paramètre `fichierCSV` représentant le fichier possède bien la propriété d'accès en lecture. Un test de lecture sera aussi effectué sur le fichier. La fonction renvoie l'entier 0 si l'accès en lecture est permis, l'entier 3 sinon.

4. Fonctionnalité globale d'ouverture du fichier CSV :

```
ouvrir(chemin)
```

Paramètres :

`chemin` : **`str`** - chaîne de caractères représentant le chemin relatif ou absolu du fichier CSV fourni.

Retour : variable de type chaîne de caractères signalant le succès ou la description d'une erreur.
Description : cette fonction prend en entrée le chemin du fichier CSV fourni. Elle ouvre ce fichier en lecture à l'aide de la fonction `open(chemin,"r")` et le stocke dans une variable de type `TextIoWrapper`. Elle applique ensuite dans l'ordre les trois fonctions de vérifications du module et renvoie la description de la fonction qui fait échouer l'ouverture (texte correspondant au code 1, 2 ou 3), ou bien la chaîne `"success"` signalant le succès de l'ouverture.

1.2 Module Analyse contenu fichier

Ce module va analyser le contenu du fichier fourni en lisant une à une les données de ce fichier et en repérant les données erronées. Il aura donc 2 fonctionnalités :

1. Fonctionnalité de lecture du contenu du fichier CSV :

```
lecture(fichierCSV)
```

Paramètres :

`fichierCSV` : `TextIoWrapper` - représente le fichier CSV fourni.

Retour : une variable de type **`list`** : une liste dont chaque élément est une sous-liste contenant les données d'une ligne du fichier CSV.
Description : cette fonction prend en entrée le fichier CSV ouvert. Elle crée une liste pour y sauvegarder le contenu de ce fichier, ligne par ligne. Elle lit les caractères du fichier en entier à l'aide de la fonction `read()`. Ensuite, pour chaque ligne du fichier, elle applique la fonction `split(',')` pour séparer les caractères de cette ligne selon la virgule et les stocker dans une sous-liste qui sera finalement stockée dans la liste englobante. L'utilisation d'une liste de listes permet d'indexer les lignes et les colonnes du fichier par leurs numéros.

```
#Exemple d'un fichier CSV de 2 lignes à 2 données
[
   [donnéeColonne1Ligne1, donnéeColonne2Ligne1],
   [donnéeColonne1Ligne2, donnéeColonne2Ligne2]
]
```

2. Fonctionnalité de description du nom, du type et des erreurs des colonnes du fichier CSV :

```
descriptionColonnes(lignesCSV)
```

Paramètres :

`lignesCSV` : **`list`** - liste représentant les données du fichier CSV ligne par ligne.

Retour : une variable de type **dict** : un dictionnaire de 3 sous-listes ayant pour clés : "nom", "type" et "erreurs".
Description : cette fonction prend en entrée le fichier CSV préalablement lu et stocké dans la liste lignesCSV. Elle crée un dictionnaire de 3 éléments de type liste, pour y renseigner le nom, le type et les lignes erronées de chaque colonne de ce fichier :
1 - Elle remplit le champ "nom" du dictionnaire par le nom de chaque colonne, lu à partir du fichier CSV.
2 - Elle remplit le champ "type" du dictionnaire par le type attendu pour chaque colonne.
3 - Elle lit la liste lignesCSV colonne par colonne. Pour chaque donnée d'une colonne, elle compare son type au type attendu et :
- en cas d'erreur, renseigne le descriptif de cette erreur dans la case correspondante à cette colonne dans une sous-sous-liste dans le champ "erreurs" du dictionnaire,
- en cas de succès, inscrit la chaîne de caractères "correct" dans la case correspondante de la sous-sous-liste du dictionnaire.

```
#Exemple de description d'un fichier CSV de 2 lignes à 2 colonnes
{
   "nom" : ["nomColonne1", "nomColonne2"],
   "type" : [typeColonne1, typeColonne2],
   "erreurs" :
   [
      ["correct","description d'une erreur dans la colonne 2 de la ligne 1"],
      ["correct","correct"]
   ]
}
```

L'utilisation d'une telle structure pour le renseignement des erreurs permet un accès direct (indexé) lors de l'affichage des erreurs, et donc un gain de temps de calcul.

2 Package Analyse descriptive des données

Ce package est composé de trois modules d'ADD [2]. Ces informations pourront être utilisées pour le filtrage des valeurs dans l'échantillon de la fenêtre rôle et choix colonne, mais aussi servir pour les compte-rendu statistiques des colonnes. Ce package va aussi être livré au client pour une intégration externe. On pourra donc parler de API.

2. ADD : analyse descriptive de données

2.1 Module Analyse de données qualitatives

Ce module va effectuer les calculs d'effectifs, d'effectifs cumulés, de fréquences et de fréquences cumulées. Il s'occupera également de fournir les informations nécessaires pour la construction d'un diagramme en secteurs et d'un histogramme. Il aura donc 6 fonctionnalités :

1. Fonctionnalité de calcul de l'effectif :

```
calculEffectif(donneeQualitative)
```

Paramètres :

`donneeQualitative` : **`list`** - structure contenant les données de type qualitatif de la colonne.

Retour : variable de type **`list`** : représente la sortie du calcul des effectifs de chaque valeur de la colonne.
Description : cette fonction prendra en paramètre une liste qui contiendra les données de type qualitatif pour le calcul. Elle calculera les effectifs pour chaque valeur et effectuera un tris pour ordonner ces valeurs.

2. Fonctionnalité de calcul de l'effectif cumulé :

```
calculEffectifCumule(listeEffectif)
```

Paramètres :

`listeEffectif` : **`list`** - correspond au résultat de la fonction `calculEffectif`.

Retour : variable de type **`list`** : copie du paramètre `listeEffectif` dont les effectifs seront remplacés par les effectifs cumulés.
Description : cette fonction prend en entrée les résultats de la fonction `calculEffectif` préalablement stocké dans une liste. Elle va copier dans une nouvelle liste celle rentrée en paramètre, et remplacera les effectifs par les effectifs cumulés. La fonction retournera cette nouvelle liste.

3. Fonctionnalité pour calculer les fréquences d'apparition :

```
calculFrequence(donneeQualitative)
```

Paramètres :

`donneeQualitative` : **`list`** - structure contenant les données de type qualitatif de la colonne.

Retour : variable de type **`list`** : correspond à la sortie du calcul des fréquences de chaque valeur de la colonne.

Description : cette fonction prendra en paramètre une liste qui contiendra les données de type qualitatif pour le calcul. Elle calculera les fréquences pour chaque valeur et effectuera un tris pour ordonner ces valeurs.

4. Fonctionnalité pour calculer les fréquences cumulées :

```
calculFrequenceCumule(listeFrequence)
```

Paramètres :

`listeFrequence` : **`list`** - correspond au résultat de la fonction `calculFrequence`.

Retour : variable de type **`list`** - copie de `listFréquence` mais dont les fréquences seront remplacés par les fréquences cumulées.
Description : cette fonction prend en entrée les résultats de la fonction `calculFrequence` préalablement stocké dans une liste. Elle va copier dans une nouvelle liste celle rentrée en paramètre, et remplacera les fréquences par les fréquences cumulées. La fonction retournera cette nouvelle liste.

5. Fonctionnalité fournissant les données du diagramme en secteurs :

```
infoSecteurs(listeFrequences)
```

Paramètres :

`listeFrequences` : **`list`** - correspondant à la sortie du calcul des fréquences.

Retour : sans retour (vide).
Description : cette fonction prend en entrée le résultat du calcul des fréquences préalablement stocké dans une liste `listeFrequences`. Elle va créer un fichier `.json` pour y stocker (écrire) les données nécessaires à la construction du diagramme en secteurs. Pour chaque couple (fréquence, valeur) elle va associer un angle compris entre 0 ° et 360 ° telle que la somme de tous les angles soit égale à 360 °. Ne retourne aucune valeur.

6. Fonctionnalité fournissant les données de l'histogramme :

```
infoHistogramme(listeEffectifs)
```

Paramètres :

`listeEffectifs` : **`list`** - correspond à la sortie du calcul des effectifs.

Retour : sans retour (vide).
Description : cette fonction prend en entrée le résultat du calcul des effectifs préalablement stocké dans une liste `listeEffectifs`. Elle va créer un fichier `.json` pour y stocker (écrire) les données nécessaires à la construction de l'histogramme. Ne retourne aucune valeur.

2.2 Module Analyse de données quantitatives discrètes

1. Calcul de la moyenne

```
moyenne(listeEffectifs)
```

Paramètres :

`listeEffectifs` : **`list`** - structure contenant les différents effectifs des données de l'ensemble analysé.

Type retour : **`float`** : moyenne arithmétique de l'ensemble de données
Description : Calcul de la moyenne arithmétique en fonction des données et de leur occurrences dans la liste.

2. Calcul des quantiles (cas discret)

```
quantileDiscret(ordre, listeFrequencesCumulees)
```

Paramètres :

`ordre` : **`float`** - Nombre flottant compris entre 0 et 1

`listeFrequencesCumulees` : **`list`** - structure contenant les différentes fréquences cumulées des données de la colonne analysées

Type retour :**`float`**
Description : Valeur de la liste séparant les données en deux ensembles, le premier contenant la fraction `ordre` des données, le deuxième contenant le reste. La médiane est le quantile d'ordre 1/2.

3. Calcul de la variance

```
variance(listeEffectifs)
```

Paramètres :

`listeEffectif` : **`list`** - structure contenant les différents effectifs des données de l'ensemble analysé.

Type retour : **`float`**
Description : Variance de l'ensemble de données analysé

4. Calcul de l'écart-type

```
ecartType(variance)
```

Paramètres :

`variance` : **float**

Type retour : **float**
Description : Racine carrée de la variance.

5. Détection des anomalies

```
anomaliesTukey(listeEffectifs)
```

Paramètres :

`listeEffectifs` : **list** - structure contenant les différents effectifs des données de l'ensemble analysé.

Type retour :**list** : collection contenant les données considérées anormales pour la distribution des valeurs
Description : Selon la méthode de Tukey, toute donnée n'étant pas contenue dans l'intervalle `[Q1 - 1.5*(Q3 - Q1) ; Q3 + 1.5*(Q3 - Q1)]` , avec `Q1, Q3` les quartiles, est considérée comme une anomalie statistique. [3]

6. Calcul du coefficient de symétrie

```
symetrie(listeEffectifs)
```

Paramètres :

`listeEffectifs` : **list** - structure contenant les différents effectifs des données de l'ensemble analysé.

Type retour : **float** - Nombre flottant compris entre -1 et 1.
Description : Coefficient d'asymétrie de Pearson calculé avec la moyenne, la médiane et l'écart-type : s'il est nul la distributione est symétrique, s'il est positif, la série est étalée à droite, et s'il est négatif, la série est étalée à gauche.

7. Calcul du coefficient d'aplatissment

```
aplatissement(listeEffectifs)
```

`listeEffectifs` : **list** - structure contenant les différents effectifs des données de l'ensemble analysé. Paramètres :

Type retour : **float**
Description : Coefficient d'aplatissement de Fisher, calculé à partir de la moyenne et de l'écart-type. Si le coefficient est égal à 3, la série suit une loi normale centrée réduite, s'il

3. https ://hal.archives-ouvertes.fr/halshs-00287751/document

est inférieur la distribution des valeurs est aplatie, et s'il est supérieur, les valeurs sont concentrées autour de la moyenne.

8. Fonctionnalité pour les données de la distribution des valeurs

```
infoDistribution(listeEffectifs)
```

Paramètres :

`listeEffectifs` : **`list`** - structure contenant les différents effectifs des données de l'ensemble analysé.

Retour : fichier au format `.json` - contenant les couples (valeur, effectif)
Description : Préparation de l'objet contenant les informations nécessaires à l'affichage de la distribution des valeurs.

9. Fonctionnalité pour les données de la distribution cumulative des valeurs (cas discret)

```
infoDistributionCumulativeDiscrete(listeEffectifsCumules)
```

Paramètres :

`listeEffectifsCumules` : **`list`** - structure contenant les différents effectifs cumulés des données de l'ensemble analysé.

Retour : fichier au format `.json` - contenant les couples (valeur, effectif cumulé)
Description : Préparation de l'objet contenant les informations nécessaires à l'affichage de la distribution cumulative des valeurs (diagramme en escaliers)

10. Fonctionnalité pour les données de la boîte à moustaches de Tukey

```
infoBoiteTukey(listeEffectifs)
```

Paramètres :

`listeEffectifs` : **`list`** - structure contenant les différents effectifs des données de l'ensemble analysé.

Retour : fichier au format `.csv` - contenant les données du paramètre.
Description : La représentation graphique met en valeur les quartiles et la médiane, ainsi que les anomalies de Tukey.

11. Fonctionnalité pour les données de la série temporelle

```
infoSerieTemporelle(listeSerieTemporelle)
```

Paramètres :

`listeSerieTemporelle` : **`list`** - liste des couples (Timestamp, valeur) des données analysées.

Retour : fichier au format `.json` - contenant les couples (Timestamp, valeur)

Description : La représentation graphique associée consiste à afficher l'évolution des valeurs dans le temps.

2.3 Module Analyse de données quantitatives continues

1. Fonctionnalité pour la discrétisation des valeurs

```
discretisation(nombreClasses, donneesContinues)
```

Paramètres :

`nombreClasses` : **`int`** - indique en combien de classes d'intervalle l'étendue des données va être divisée.

`donnesContinues` : **`list`** - liste de flottants représentant les mesures analysées.

Type retour : **`list`** - collection d'`Intervalle` (voir glossaire des types)
Description : La fonction se charge de décomposer l'étendue [min ; max] de l'ensemble de données en `nombreClasses` intervalles de même étendue, puis de remplacer les occurrences des données par l'intervalle auquel la donnée appartient.

2. Calcul du nombre de classes optimal pour la discrétisation des valeurs

```
calculNombreClasses(donneesContinues)
```

Paramètres :

`donneesContinues` : **`list`** - liste de flottants représentant les mesures analysées.

Type retour : **`int`**
Description : Calcul le nombre de classes selon la règle de Sturges.

3. Préparation des données pour l'utilisation des éléments de calcul du module ADD quantitatives discrètes

```
preparationIntervallesAnalyse(listeIntervalles)
```

Paramètres :

`listeIntervalles` : **`list`** - issue de la discrétisation des valeurs.

Type retour : **list** - liste de flottants
Description : Pour effectuer les analyses descriptives dans le cas continu, la démarche est la même (sauf quantiles) que pour le cas discret en utilisant comme données les centres des intervalles.

4. Calcul des quantiles (cas continu)

```
quanileContinu(ordre, listeFrequencesCumulees)
```

Paramètres :

`ordre` : **float** - Nombre flottant compris entre 0 et 1

`listeFrequencesCumulees` : **list** - structure contenant les différentes fréquences cumulées des données de la colonne analysées

Type retour : **float**
Description : Les quantiles dans le cas continu sont calculés tout d'abord en identifiant la classe d'intervalle dans laquelle le quantile se trouve à l'aide des fréquences cumulées, puis par interpolation linéaire.

5. Calcul d'abscisse par interpolation linéaire

```
interpolationLineaire(p1, p2, y)
```

Paramètres :

`p1` : **float** Premier point de la droite

`p2` : **float** Second point de la droite

`y` : **float** ordonnée du point dont on veut retrouver l'abscisse

Type retour : **float** - abscisse
Description : Les points `p1, p2` nous permettent d'identifier une fonction linéaire. On retrouve ensuite l'abscisse du point d'ordonnée `y` se trouvant sur la courbe de la fonction.

6. Fonctionnalité pour les données de la distribution cumulative des valeurs (cas continu)

```
infoDistributionCumulativeContinue(listeFrequencesCumulees)
```

Paramètres :

`listeFrequencesCumulees` : **list** - structure contenant les différentes fréquences cumulées des données de l'ensemble analysé.

Retour : fichier au format `.json` - contenant les couples (valeur, fréquence cumulée)
Description : Préparation de l'objet contenant les informations nécessaires à l'affichage de la distribution cumulative des valeurs. L'objet de retour est une collection de points que l'on va relier avec des droites.

3 Package Interface web

Ce package est composé de 3 modules de fenêtre graphique web qui assurent l'interaction avec l'utilisateur, et d'un module de gestion des flux qui s'occupe du lancement de l'application et de la transition des informations entre ses différents packages.

3.1 Module Gestion des flux

Ce module va lancer l'application et sera au centre du programme. Il s'occupera de gérer les branchements pour l'exécution, de faire transiter les informations entre les différents modules et de faire le traitement nécessaire en cas d'erreur. Il aura 4 fonctionnalités qui se chargeront de l'affichage des pages web dans lesquels les branchements seront gérés, ainsi que la fonctionnalité principale `main` de l'application :

1. Fonctionnalité principale de l'application :

```
main()
```

   Paramètres :

   Ne prend aucun paramètre.

   Retour : sans retour (vide).
   Description : cette fonction ne prend aucun paramètre en entrée et ne retourne rien en sortie. Elle s'occupe de lancer l'application en faisant appel à la première fenêtre choix fichier, et de faire tourner l'application. Également la gestion des erreurs sera traité ainsi que de l'arrêt du programme.

2. Fonctionnalité générant la fenêtre choix fichier :

```
fenetre_choix_fichier()
```

   Paramètres :

   Ne prend aucun paramètre.

   Retour : `render_template()` : fonction qui renvoie le contenu de la page web mis en paramètre qui va être envoyée à l'utilisateur.
   Description : cette fonction correspond à la requête HTTP (c'est-à-dire le chemin de la page demandée) envoyée par le client (l'utilisateur) et reçue par le serveur qui sera soit un GET soit un POST. Pour cela il faudra indiquer avant de déclarer la fonction la ligne suivante : `@app.route('/choix_fichier/', methods=['GET', 'POST'])`. Elle retournera le contenu de la page web mis en paramètre de la fonction `render_template()` associé

au code HTML de la fenêtre choix fichier. Des variables en tant qu'arguments optionnels peuvent être utilisées dans le cas où l'on souhaiterait afficher des valeurs.

3. Fonctionnalité générant la fenêtre rôle et choix colonne :

```
fenetre_role_choix_colonne()
```

Paramètres :

Ne prend aucun paramètre.

Retour : `render_template()` : fonction qui renvoie le contenu de la page web mis en paramètre qui va être envoyée à l'utilisateur.
Description : cette fonction correspond à la requête HTTP (c'est à dire le chemin de la page demandée) envoyée par le client (l'utilisateur) et reçue par le serveur qui sera soit un GET soit un POST. Pour cela il faudra indiquer avant de déclarer la fonction la ligne suivante : `@app.route('/role_choix_colonne/', methods=['GET', 'POST'])`. Elle retournera le contenu de la page web mis en paramètre de la fonction `render_template()` associé au code HTML de la fenêtre rôle et choix colonne. Des variables en tant qu'arguments optionnels peuvent être utilisées dans le cas où l'on souhaiterait afficher des valeurs.

4. Fonctionnalité générant la fenêtre résultats ADD :

```
fenetre_resultat_ADD()
```

Paramètres :

Ne prend aucun paramètre.

Retour : `render_template()` : fonction qui renvoie le contenu de la page web mis en paramètre qui va être envoyée à l'utilisateur.
Description : cette fonction correspond à la requête HTTP (c'est à dire le chemin de la page demandée) envoyée par le client (l'utilisateur) et reçue par le serveur qui sera soit un GET soit un POST. Pour cela il faudra indiquer avant de déclarer la fonction la ligne suivante : `@app.route('/resultat_ADD/', methods=['GET', 'POST'])`. Elle retournera le contenu de la page web mis en paramètre de la fonction `render_template()` associé au code HTML de la fenêtre résultat ADD. Le deuxième argument qui sera mis dans `render_template()` sera la liste contenant les données de la colonne choisie. Cette fonction sera appelé dès que le choix sera validé dans la fenêtre rôle et choix colonne pour accéder à la fenêtre suivante.

5. Fonctionnalité de demande d'exportation :

```
sauvResultats()
```

Paramètres :

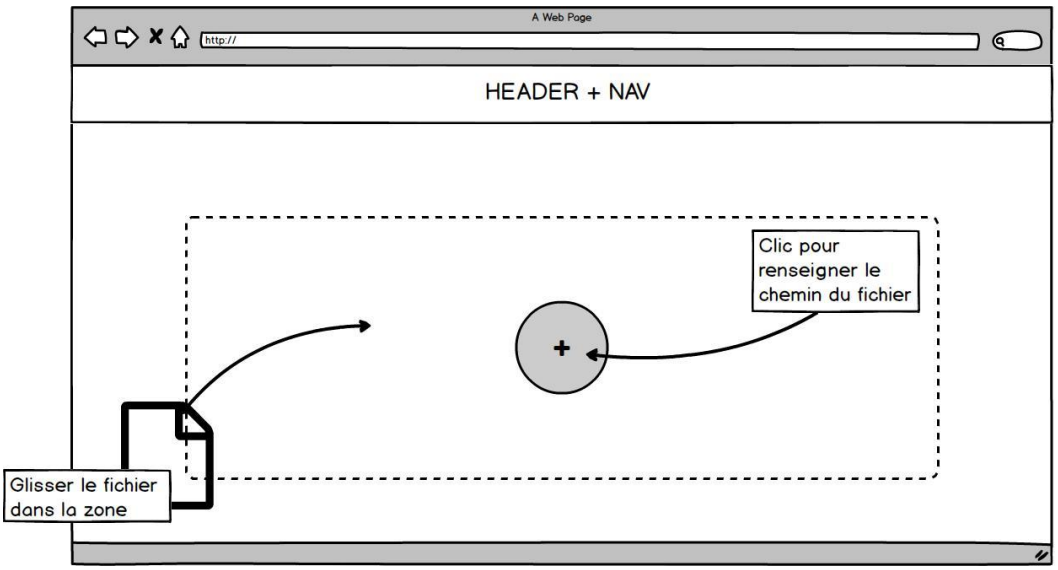
Ne prend aucun paramètre.

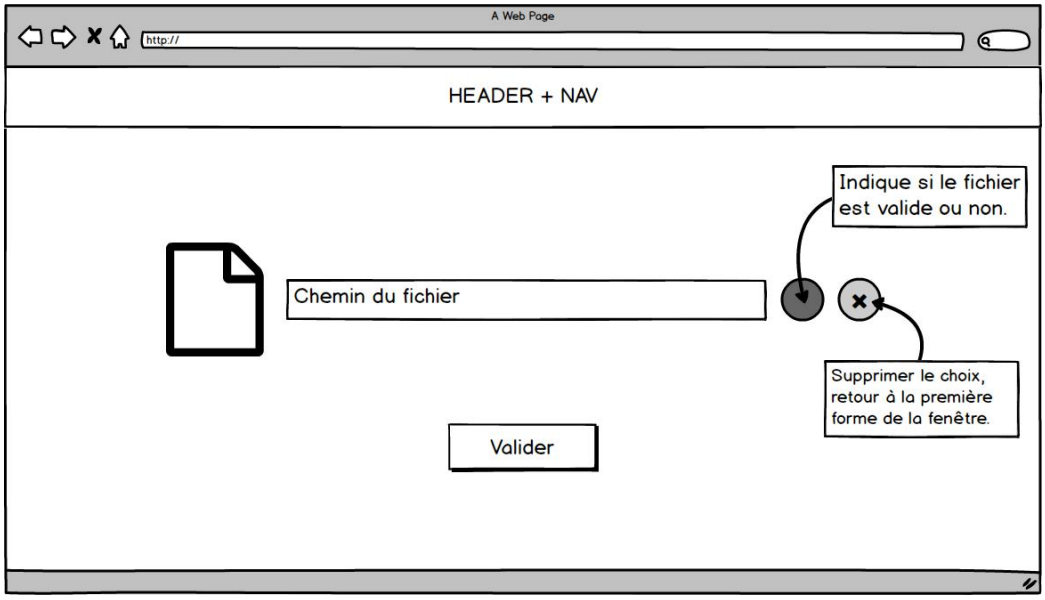
Retour : sans retour (vide).

Description : cette fonction ne prend aucun paramètre. Elle s'active lorsque l'utilisateur fait la demande d'exporter les résultats depuis la fenêtre résultats ADD. Le téléchargement sera alors lancé par le navigateur web.

3.2 Module Fenêtre choix fichier

Ce module est une fenêtre graphique web qui permet à l'utilisateur de charger un fichier CSV. Ce module aura 2 fonctionnalités qui peuvent être illustrées par les croquis suivants :





1. Fonctionnalité d'ouverture du fichier avec le système de gestion de fichiers :

```
FileWithSGF()
```

Paramètres :

Ne prend aucun paramètre.

Retour : variable de type **str** : chaîne de caractères représentant le chemin relatif ou absolu du fichier CSV fourni.
Description : cette fonction ne prend pas de paramètre en entrée. Elle permet de récupérer l'emplacement du fichier CSV en parcourant le système de gestion de fichiers. Elle renvoie donc une chaîne de caractères correspondant à ce chemin.

2. Fonctionnalité d'ouverture du fichier avec la technique du Drag&Drop :

```
FileWithDragDrop()
```

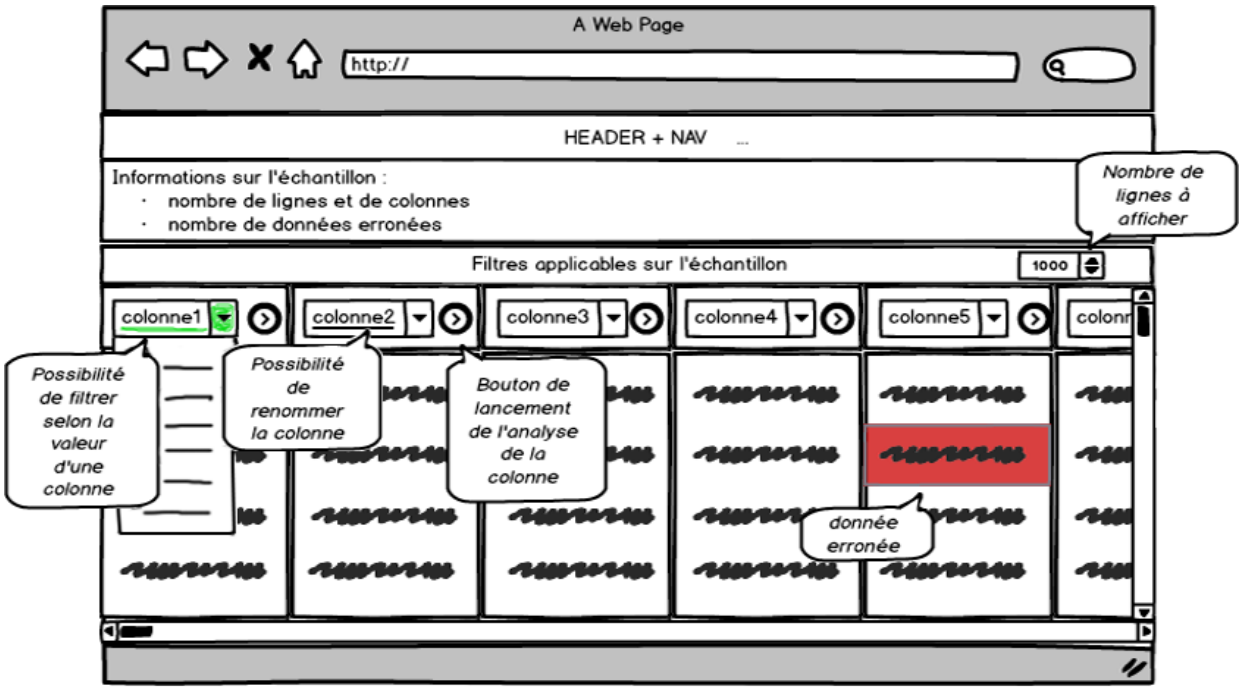
Paramètres :

Ne prend aucun paramètre.

Retour : variable de type **str** : chaîne de caractères représentant le chemin relatif ou absolu du fichier CSV fourni.
Description : cette fonction ne prend pas de paramètre en entrée. Elle permet de récupérer l'emplacement du fichier CSV avec la technique du Drag&Drop. Elle renvoie donc une chaîne de caractères correspondant à ce chemin.

3.3 Module Fenêtre rôle et choix colonne

Ce module est une fenêtre graphique web qui permet d'une part d'afficher un échantillon filtré des données du fichier et ses informations relatives, et d'autre part, de choisir une colonne de cet échantillon pour l'analyser. Ce module aura 5 fonctionnalités qui peuvent être illustrées par ce croquis :



1. Fonctionnalité d'affichage de l'échantillon :

```
1 afficherEchantillon(lignesCSV, descCSV)
```

Paramètres :

`lignesCSV` : **`list`** - liste filtrée représentant les données du fichier CSV ligne par ligne.

`descCSV` : **`dict`** - dictionnaire contenant 3 champs d'informations sur les colonnes du fichier CSV.

Retour : sans retour (vide).
Description : cette fonction prend deux paramètres représentant l'échantillon pour en afficher le contenu sous forme de tableau.
1 - Elle lit les champs `"nom"` et `"type"` du dictionnaire descriptif de l'échantillon et les affiche en en tête du tableau.

2 - Elle lit simultanément, case par case, la liste des données l'échantillon filtré `lignesCSV` et le champ `"erreurs"` dictionnaire descriptif `descCSV` et affiche la valeur de chaque donnée. Si une donnée est erronée, elle sera surlignée en rouge et un survol du curseur permettra d'afficher les informations sur cette erreur.

2. Fonctionnalité de filtrage de l'échantillon selon la valeur exacte d'une colonne :

```
filtreValeur(lignesCSV, nomColonne, valeurColonne)
```

Paramètres :

`lignesCSV` : **`list`** - liste représentant les données du fichier CSV en entier, ligne par ligne.

`nomColonne` : **`str`** - nom de la colonne filtrante.

`valeurColonne` : type de la colonne - valeur de la colonne filtrante.

Retour : une variable de type **`list`** : une liste représentant les données du fichier CSV sur lesquelles le filtre a été appliqué.
Description : cette fonction prend en entrée la liste `lignesCSV` représentant les données fichier CSV en entier, ligne par ligne. Elle renvoie ensuite une nouvelle liste filtrée représentant les données du fichier dont les valeurs de la colonne `nomColonne` valent `valeurColonne`.

3. Fonctionnalité de filtrage de l'échantillon selon une plage de valeurs d'une colonne :

```
filtrePlageValeurs(lignesCSV, nomColonne, valeurColonneMin,
   ValeurColonneMax)
```

Paramètres :

`lignesCSV` : **`list`** - liste représentant les données du fichier CSV en entier, ligne par ligne.

`nomColonne` : **`str`** - nom de la colonne filtrante.

`valeurColonneMin` : type de la colonne - valeur minimale de la colonne filtrante.

`valeurColonneMax` : type de la colonne - valeur maximale de la colonne filtrante.

Retour : une variable de type **`list`** : une liste représentant les données du fichier CSV sur lesquelles le filtre a été appliqué.
Description : cette fonction prend en entrée la liste `lignesCSV` représentant les données fichier CSV en entier, ligne par ligne. Elle renvoie ensuite une nouvelle liste filtrée représentant les données du fichier dont les valeurs de la colonne `nomColonne` sont comprises entre `valeurColonneMin` et `valeurColonneMax`.

4. Fonctionnalité de filtrage de l'échantillon selon l'occurrence d'un arc du fichier CSV :

```
filtreArc(lignesCSV, nomColonneDepart, nomColonneArrivee,
    valeurColonneDepart, ValeurColonneArrivee)
```

Paramètres :

`lignesCSV` : **`list`** - liste représentant les données du fichier CSV en entier, ligne par ligne.

`nomColonneDepart` : **`str`** - nom de la colonne du nœud de départ l'arc.

`nomColonneArrivee` : **`str`** - nom de la colonne du nœud d'arrivée de l'arc.

`valeurDepart` : type de la colonne - valeur de la colonne du nœud de départ l'arc.

`valeurArrivee` : type de la colonne - valeur de la colonne du nœud d'arrivée de l'arc.

Retour : une variable de type **`list`** : une liste représentant les données du fichier CSV sur lesquelles le filtre a été appliqué.
Description : cette fonction prend en entrée la liste `lignesCSV` représentant les données fichier CSV en entier, ligne par ligne. Elle renvoie ensuite une nouvelle liste filtrée représentant les données du fichier dont les valeurs de la colonne `nomColonneDepart` valent `valeurDepart` et ceux de la colonne `nomColonneArrivee` valent `valeurArrivee`.

5. Fonctionnalité de renommage de colonne :

```
renommerColonne(descCSV, indiceColonne, nouveauNom)
```

Paramètres :

`descCSV` : **`dict`** - dictionnaire contenant 3 champs d'informations sur les colonnes du fichier CSV, à modifier.

`indiceColonne` : **`int`** - numéro de la colonne à renommer.

`nouveauNom` : **`str`** - nouveau nom à donner à la colonne.

Retour : sans retour (vide).
Description : cette fonction prend en entrée le dictionnaire descriptif des colonnes `descCSV`. Elle accède à la colonne d'indice `indiceColonne` du champ `"nom"` de ce dictionnaire, et remplace sa valeur par `"nouveauNom"`.

6. Fonctionnalité de choix de colonne à analyser :

```
choixColonne(lignesCSV, indiceColonne)
```

Paramètres :

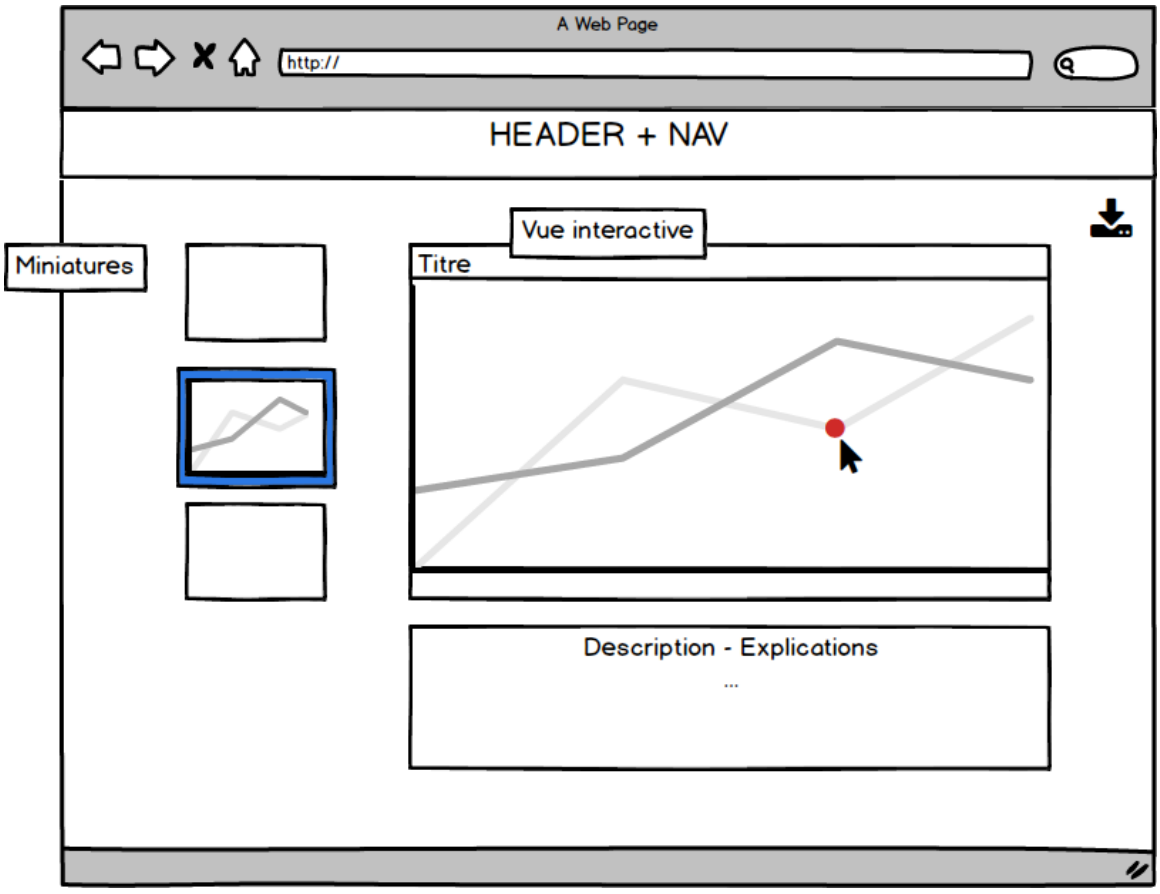
`lignesCSV` : **`list`** - liste représentant les données filtrées du fichier CSV en entier, ligne par ligne.

Retour : variable de type **`list`** : liste simple contenant les données d'une colonne choisie.
Description : cette fonction prend en entrée la liste `lignesCSV` représentant les données filtrées du fichier CSV ligne par ligne. Elle crée une nouvelle liste pour y stocker une colonne unique. Elle parcourt le liste initiale ligne par ligne, et pour chaque ligne, elle stocke la valeur de la colonne d'indice `indiceColonne` dans la nouvelle liste.

3.4 Module Fenêtre résultats ADD

Ce module est une fenêtre graphique web qui permet à l'utilisateur de consulter les résultats d'analyse des colonnes du fichier importé.

1. Aspect visuel :



Les résultats doivent se présenter de manière simple et sans surcharge. Les utilisateurs de

ce produit doivent pouvoir prendre en main et comprendre les données sans pour autant être formés.

2. Intéractions avec l'utilisateur :

   — Miniatures : Un clic sur une miniature permet de la faire basculer en vue précise.

   — Vue précise : Un affichage interactif des valeurs rend la consultation plus intuitive et la charge de la vue plus légère. (une valeur non-demandée n'est pas affichée)

   — Sauvegarde des résultats : Une simple icône de téléchargement permettra de lancer la sauvegarde des résultats en local.

3. Vues précises :
   Ci-dessous, une liste des représentations graphiques disponibles, ainsi que le type d'analyse qui permet de les afficher :

   — Histogramme : ADD qualitatives

   — Diagramme en secteur : ADD qualitatives

   — Série temporelle : ADD quantitatives

   — Distribution des valeurs : ADD quantitatives

   — Distribution cumulative des valeurs : ADD quantitatives

   — Diagramme à moustaches : ADD quantitatives

4 Glossaire des types

Dans ce glossaire, on va préciser la définition des types des données qu'on a utilisé dans la description des fonctionnalités.

— `TextIoWrapper` : classe représentant les flux de texte bufferisés (entrées/sorties de texte avec une sauvegarde dans une mémoire tampon). Elle permet donc de représenter les fichiers de texte brut en particulier. Elle définit des méthodes de manipulation de ces flux.

— **`str`** : classe représentant des séquences de caractères (donc des chaînes de caractères) en Python. Elle définit des méthodes de manipulation de ces chaînes.

— **`float`** : classe représentant les nombres flottants à double précision en Python.

— **`list`** : classe représentant des séquences de variables de type quelconque indexés par des entier. Elle définit des méthodes de manipulation de ces séquences.
L'utilisation de ce type de données pour la manipulation des colonnes du fichier `.csv` est préconisée car, tout d'abord, le type et les fonctions de manipulations sont disponibles

directement avec Python, ensuite, ces listes sont implémentées sous forme de tableau ce qui garantie l'accès rapide à chacun de ses éléments, et enfin, leur taille maximale est définie par l'entier `sys.maxsize`[4] qui dépend des machines mais dépasse largement le million sur des PC bureautiques standards. Le million correspond à une borne maximale du nombre de lignes du fichier `.csv` importé, la manipulation des listes peut donc se faire sans risque de crash mémoire.

— **`dict`** : classe représentant des séquences de couple clés, variables (indexés par cette clé). Elle définit des méthodes de manipulation de ces séquences.

— **`tuple`** : classe représentant des listes immuables (non modifiables).

— `.json` : (JavaScript Object Notation) format de données textuelles basé sur la notation des objets JavaScript. C'est un moyen simple d'accéder à des données car un flux JSON est une sérialisation (une traduction) d'un objet JavaScript.

— `Intervalle` : classe à implémenter

```
class Intervalle:
```

Attributs :

- `borneInf : float` - limite inférieure de l'intervalle
- `borneSup : float` limite supérieure de lintervalle
- `infInclus : boolean` - vrai si l'intervalle est fermé à gauche
- `supInclus : boolean` - vrai si l'intervalle est fermé à droite
- `centre : float` - centre de l'intervalle calculé dans le constructeur

Méthodes :

— `__init__(self, borneInf, borneSup, infInclus, supInclus)`

Description : Constructeur

— `contient(nombre)`

Paramètres :

`nombre : float`

Retour : `boolean`

Description : Renvoie vrai si `nombre` est contenu dans l'intervalle, faux sinon.

4. https ://docs.python.org/2/library/sys.html#sys.maxsize

Conclusion

Bilan : rappel ce qu'on a fait et ce que ça a apporté
Ouvertures :
- Difficultés
- Limites des spécifications dans notre cas : nécessité de tester une 1ere version de l'appli pour bien définir l'expérience utilisateur
Dernière phrase positive :
- Découverte nouveaux outils, nouvelle démarche pour le dev : devops, ux design, balsamiq, les app web (javascript c'est assez bien en vrai)

Lors de la rédaction de ce document, plusieurs difficultés ont été rencontrées, notamment sur la spécification des aspects visuels du produit. En effet, ce produit étant une application web de traitement de données, on a été contraint d'adopter des outils de développement qui nous sont peu familiers : le langage Python et ses frameworks d'une part, et JavaScript d'autre part. Ces outils seront utilisés pour la réalisation des interfaces d'interaction avec l'utilisateur, ainsi que des rendus graphiques d'analyses. Par conséquent, il est possible que certaines de ces fonctionnalités s'avèrent non spécifiées.
Néanmoins, et dans un cadre pédagogique, cette épreuve ne sera qu'enrichissante pour nous, non seulement sur le plan compétence, mais aussi expérience.