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

16/04/2017

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. On utilisera d'une part la programmation fonctionnelle pour les calculs, et d'autre part la programmation objet pour le développement des interfaces graphiques. 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 fonctions et les structures des classes. 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, puis ceux du package d'analyse descriptive des données et ensuite ceux du package de l'interface web. Les types Python utilisés seront décrits dans la dernière partie, le glossaire des types.

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

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

```
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 entier signalant un succès (0) ou une erreur (1, 2, ou 3).
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 le code de la fonction qui fait échouer l'ouverture (l'entier 1, 2 ou 3), ou bien un code signalant le succès de l'ouverture (l'entier 0).

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 un dictionnaire 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 un dictionnaire qui sera finalement stocké dans la liste englobante. L'utilisation d'un dictionnaire permet d'indexer les données de la ligne par le nom de la colonne :

```
[
   {"nomColonne1" : donnée1Ligne1, "nomColonne2" donnée2Ligne1},
   {"nomColonne1" : donnée1Ligne2, "nomColonne2" donnée2Ligne2}
]
```

Ces noms de colonnes proviennent du fichier CSV.

2. Fonctionnalité de description 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 2 dictionnaires indexés par "types" et "erreurs".
Description : cette fonction prend en entrée le fichier CSV préalablement lu et stocké dans la liste lignesCSV. Elle crée un dictionnaire de 2 éléments de type dictionnaire également, pour y renseigner le type et les erreurs de chaque colonne de ce fichier.
Elle remplit d'abord le champ types du dictionnaire par les types attendus. Ensuite, 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 numéro de cette ligne dans une liste dans le champ erreurs du dictionnaire.

```
{
   "types" : {"nomColonne1" : typeColonne1, "nomColonne2"
   typeColonne2},
   "erreurs" :
   {
      "nomColonne1" : [liste des numéros de lignes des données
   erronées de la colonne 1],
      "nomColonne2" : [liste des numéros de lignes des données
   erronées de la colonne 2]
   }
}
```

2 Package Analyse descriptive des données

package à décrire Sonny stp
Ce package va être livré au client pour une intégration externe. On pourra donc parler de API.

2.1 Module Analyse de données qualitatives

Ce module va effectuer les calculs d'effectifs, d'effectifs cumulés, de fréquences et de fréquences cumulés. Il s'occupera également de fournir les informations nécessaire pour la construction d'un diagramme de secteur 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 calculer de l'effectif cumulé :

```
calculEffectifCumule(listeEffectif)
```

Paramètres :

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

Retour : `listeEffectif` : **`list`** : copie de `listeEffectif` mais dont les effectifs seront remplacé 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 la fréquence :

```
calculFrequence(donneeQualitative)
```

Paramètres :

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

Retour : variable de type **`list`** - correspondant 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 la fréquence cumulé :

```
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 remplacé par les fréquences cumulés.
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 cumulés. La fonction retournera cette nouvelle liste.

5. Fonctionnalité pour les données du diagramme en secteur :

```
infoSecteur(frequence)
```

Paramètres :

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

Retour : fichier au format `.json` - contenant les informations nécessaire pour la construction d'un diagramme de secteur.
Description : cette fonction prend en entrée le résultat du calcul des fréquences préalablement stocké dans une liste `frequence`. Elle va créer un fichier `.json` pour y stocker les données pour construire le diagramme de secteur. Pour chaque couple (fréquence-valeur) elle va associer un angle compris entre 0 ° et 360 ° tel que la somme de tout les angles soit égale à 360 ° .

6. Fonctionnalité pour les données de l'histogramme :

```
infoHistogramme(effectif)
```

Paramètres :

`effectif` : **list** - correspondant à la sortie du calcul des effectifs.

Retour : fichier au format `.json` - contenant les informations nécessaire pour la construction d'un histogramme.
Description : cette fonction prend en entrée le résultat du calcul des effectifs préalablement stocké dans une liste `effectif`. Elle va créer un fichier `.json` pour y stocker les données pour construire l'histogramme.

2.2 Module Analyse de données quantitatives discrètes

2.3 Module Analyse de données quantitatives continues

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 communiquer les informations entre les différents modules et arrêtera le programme en cas d'erreur. Il aura ?? fonctionnalités ainsi que le `main` de l'application :

1. La fonction main.

```
main()
```

Paramètres :

Ne prend aucun paramètre.

Retour : Ne retourne rien.
Description : la fonction `main()` 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. Elle s'occupe également de gérer les erreurs et donc de l'arrêt du programme.

2. Fonctionnalité pour générer la fenêtre choix fichier.

```
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 au client.
Description : cette fonction correspond à la requête HTTP (c'est à dire le chemin de la page demandée) envoyée par le client 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 utilisé dans le cas où l'on souhaiterait afficher des valeurs.

3. Fonctionnalité pour générer la fenêtre rôle et choix colonne.

```
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 au client.
Description : cette fonction correspond à la requête HTTP (c'est à dire le chemin de la page demandée) envoyée par le client 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 utilisé dans le cas où l'on souhaiterait afficher des valeurs.

4. Fonctionnalité pour générer la fenêtre résultats ADD.

```
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 au client.
Description : cette fonction correspond à la requête HTTP (c'est à dire le chemin de la page demandée) envoyée par le client 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. Des variables en tant qu'arguments optionnels peuvent-être utilisé dans le cas où l'on souhaiterait afficher des valeurs.

5. Fonctionnalité pour la validation du choix colonne. En attente de Malek

```
validChoixColonne()
```

Paramètres :

Retour :
Description :

6. Fonctionnalité pour la demande d'exportation.

```
sauvResultats()
```

Paramètres :

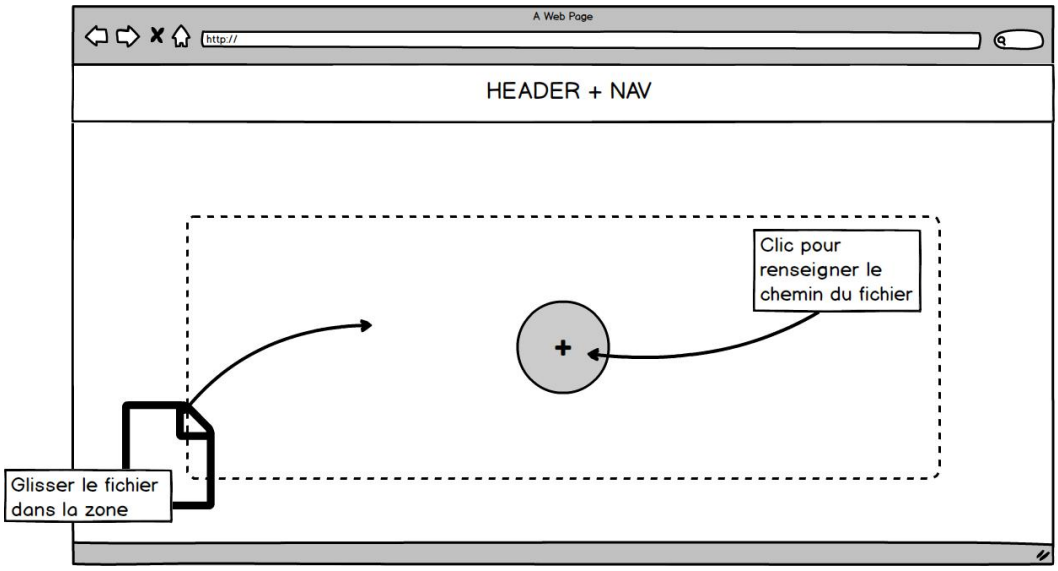
Ne prend pas de paramètre.

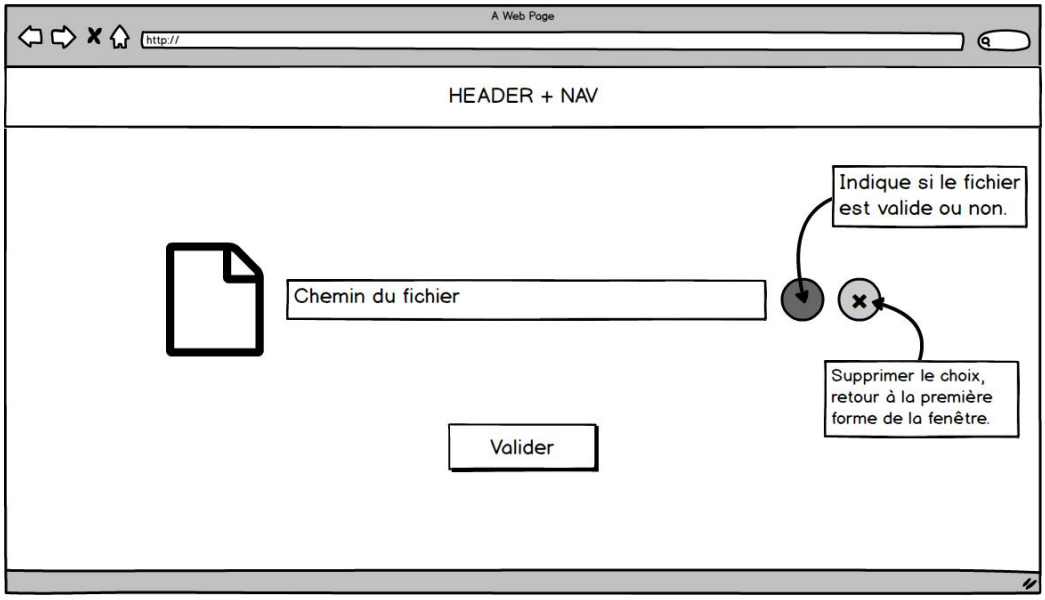
Retour : Ne retourne rien.

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

3.2 Module Fenêtre choix fichier

Ce module est une fenêtre graphique web qui va permettre à l'utilisateur de charger un fichier CSV. Ce module aura 2 fonctionnalités et peut être illustré par les deux croquis suivants :





1. Fonctionnalité pour ouvrir le fichier avec le Système de Gestion de fichier :

```
FileWithSGF()
```

Paramètres :

Ne demande aucun paramètre en entrée.

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.

2. Fonctionnalité pour ouvrir le fichier avec la technique du Drag&Drop.

```
FileWithDragDrop()
```

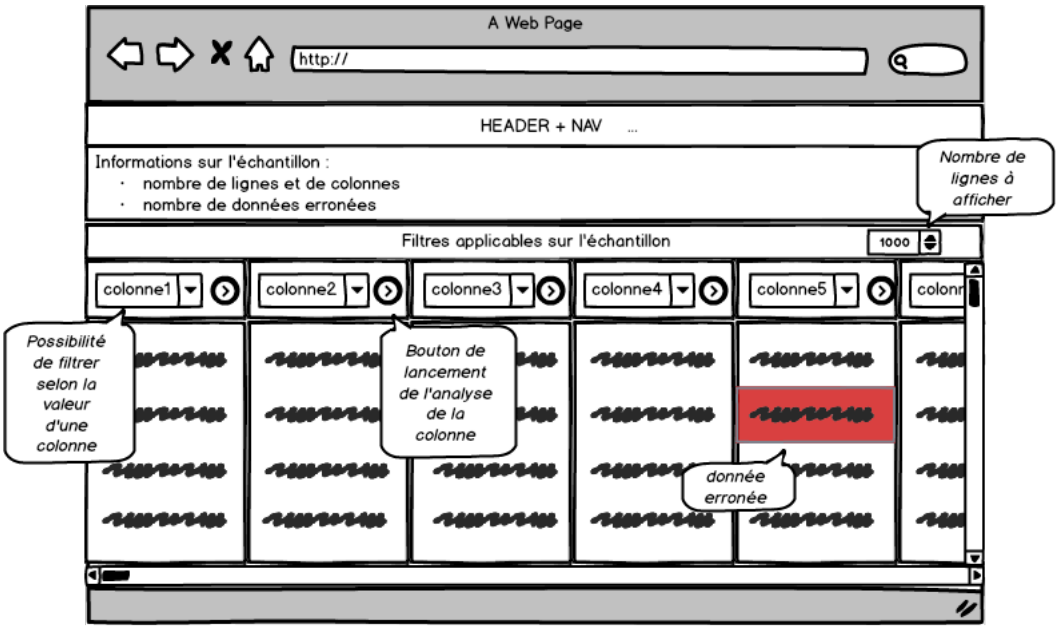
Paramètres :

Ne demande aucun paramètre en entrée.

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.

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 3 fonctionnalités qui peuvent être illustrées par ce croquis :



1. Fonctionnalité d'affichage de l'échantillon :

2. Fonctionnalité de filtrage de l'échantillon :

3. Fonctionnalité de choix de colonne à analyser :

3.4 Module Fenêtre résultats ADD

4 Glossaire des types

Dans ce glossaire, on va préciser la définition en Python 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.

— **`list`** : classe représentant des séquences de variables de type quelconque indexés par des entier. Elle définit des méthodes de manipulation de ces séquences.

— **`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).

Conclusion

Bilan : rappel ce qu'on a fait et ce que ça a apporté

Ouvertures :

- Mise en place des tests d'acceptations (comment on va mesurer, il faut définir une mesure et après la calculer)
- 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