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

15/04/2017

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 deux 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 trois 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 deux 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 représente une ligne du fichier CSV : ses éléments sont des sous-listes contenant les données des lignes du fichier.
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.

2. Fonctionnalité de description des données 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 dont chaque élément représente une colonne du fichier CSV : la clé représente le rôle de la colonne, et la variable correspondante est une liste contenant toutes les données de cette colonne.
Description : cette fonction prend en entrée le fichier CSV préalablement lu et stocké dans la liste `lignesCSV`. Elle crée un dictionnaire pour y sauvegarder le contenu de ce fichier, colonne par colonne. Elle lit la liste `lignesCSV` colonne par colonne, stocke en premier lieu le rôle de la colonne dans la clé puis stocke les données dans la variable (liste) correspondante.

Pour chaque colonne, une vérification de type sera effectuée sur chaque donnée. Les données manquantes ou erronées (dont le type ne correspond pas au type attendu) seront renseignées dans une liste de tuples dont la structure est `[("rôle colonne", numéro de ligne erronée)]`. Celle-ci sera stockée en dernier dans le dictionnaire avec comme clé `"erreurs"`.

2 Package Analyse descriptive des données

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

2.1 Module Analyse de données qualitatives

2.2 Module Analyse de données quantitatives discrètes

2.3 Module Analyse de données quantitatives continues

3 Package Interface web

3.1 Module Gestion des flux

3.2 Module Fenêtre choix fichier

3.3 Module Fenêtre rôle et choix colonne

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 definir l'experience utilisateur
Dernière phrase positive : - Decouverte nouveaux outils, nouvelle demarche pour le dev : devops, ux design, balsamiq