École nationale de la statistique

et de l'analyse de l'information

Projet de traitement de données (1AINF06)

Cahier des charges

Encadrant : Mme Njongwa-Yepnga

Groupe : Jérémie Charlotte, Rémi Malleville & Camille Navel

10 avril 2023

Table des matières

1 Présentation du projet

L'objectif est de créer un programme en python permettant à un voyageur de trouver le trajet ferroviaire le moins cher pour aller d'une gare de départ à une gare d'arrivée. L'utilisateur doit pouvoir sélectionner un certain nombre de critaires (type de train, classe, etc.). Il utilise des données publiques produites par la SNCF.

Fonctionnalités de l'application

L'application doit permettre à l'utilisateur de choisir un trajet selon ses critères. Il peut choisir :

- si le train est un OUIGO, OUIGO classique ou un TGV INOUI ;
  - de voyager en 1ere ou 2eme classe ;
  - son profil tarifaire (normal ou réglementé) ;
  - le prix qu'il est prêt à payer.
- si le train est un TER :
  - si il possède un abonnement jeune ou tout public ;
  - le prix qu'il est prêt à payer ;
  - proposer à l'utilisateur les trajets possibles entre deux gares, en particulier le moins onéreux.

2 Présentation des données

2.1 Table TGV

La table *TGV* contient les trajets à grande vitesse disponibles. Il est à noter que certaines gares proposées sont dans des pays limitrophes de la France.

Table 1 – Fichier détail de la table des trajets en TGV

Variable	Type	Description
Transporteur	str	Nom de la compagnie de transport
Gare origine	str	Nom de la gare d'origine
Gare origine - code UIC	str	Code international de la gare d'origine
Destination	str	Nom de la gare d'arrivée
Gare destination - code UIC	str	Code international de la gare d'arrivée
Classe	str	Entier valant 1 ou 2
Profil tarifaire	str	Chaîne de caractère valant Tarif Normal ou Tarif Réglementé
Prix minimum	float	Prix minimum observé
Prix maximum	float	Prix maximum observé

Nombres d'observations : 12 031

2.2 Table TER

La table *TER* contient les trajets en TER disponibles.

Table 2 – Fichier détail de la table des trajets en TER

Variable	Type	Description
Région	str	Nom de la région qui finance le TER
Origine	str	Nom de la gare d'origine
Origine - code UIC	str	Code international de la gare d'origine
Destination	str	Nom de la gare d'arrivée
Destination - code UIC	str	Code international de la gare d'arrivée
Libellé tarif	str	Description du tarif appliqué
Type tarif	str	Normal ou abonnement
Prix	float	Prix

Nombres d'observations : 92 042

2.3 Table Voyageurs

La table *Voyageurs* contient des informations sur les gares françaises (il n'y pas de gare etrangère). Elle permettra de faire des jointures lorsque nécessaire.

Table 3 – Fichier détail de la table Voyageurs

Variable	Type	Description
Code UIC	str	Code UIC de la gare
Code Commune	str	Code Insee de la commune de la gare
Code département	str	Code du département de la gare
...	...	Autres variables

Nombres d'observations : 3 220

2.4 Table Correspondances

La table *Correspondances* (à créer) permettra de relier les gares d'une même aglomération (les gares parisiennes entre elles, celles de Montpellier, Lille, etc.).

Table 4 – Fichier détail de la table Correspondances

Variable	Type	Description
Gare origine	str	Nom de la gare d'origine
Gare origine - code UIC	str	Code international de la gare d'origine
Destination	str	Nom de la gare d'arrivée
Gare destination - code UIC	str	Code international de la gare d'arrivée
Prix	float	Prix (à déterminer)

3 Présentation du programme

Le package python *ParcoursTrain* comprendra trois classes et des fonctions.

3.1 Diagramme de classes

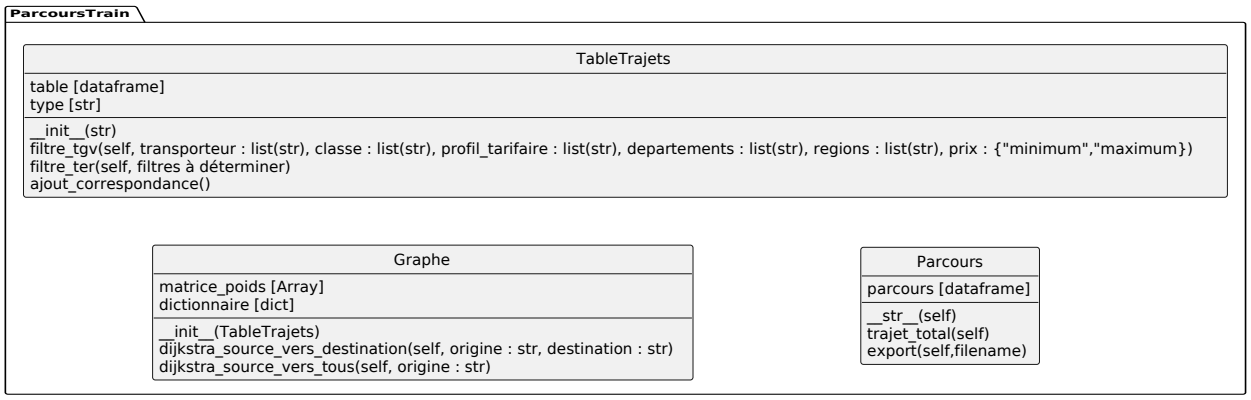

Figure 1 – Classes du package ParcoursTrain

3.2 Classe TableTrajets

```
Class TableTrajets
  """Définit une table de trajets.
```

```
    Attributs
      table : dataframe
        un dataframe avec les trajets.
      type : str
        Vaut "TGV" ou "TER"."""

  def filtre_tgv(self, transporteur=[], classe=[], profil_tarifaire=[], departements=[],
  regions=[], prix="minimum"):
    """Filtre la table TGV selon les critères choisis. La variable prix permet de choisir
    entre la colonne prix minimal et prix maximal.
        Arguments
            transporteur : list[str]
                liste des transporteurs retenus, si la liste est vide on ne filtre pas.
            classe : list[str]
              liste des classes retenues, si la liste est vide on ne filtre pas.
            profil_tarifaire : list[str]
                liste des profils tarifaires retenus, si liste vide on ne filtre pas.
            departements : list[str]
                liste des départements retenus, si la liste est vide on ne filtre pas.
            regions : list[str]
                liste des régions retenues, si la liste est vide on ne filtre pas.
            prix : str
                Vaut "minimum" ou "maximum".
      Renvoie
          La TableTrajets filtrée. La colonne du prix (min ou max) est renommée prix."""

  def filtre_ter():
      """À faire"""

  def ajout_correspondance(self):
      """Récupère la table des correspondances . La filtre  sur les UIC_origine et
      UIC_destination qui apparaissent dans self. L'ajoute à self.
          Renvoie
            un TableTrajets avec les tajets et les correspondances additionnelles."""
```

3.3 Classe Graphe

```
Class Graphe
    """Définit un graphe en utilisant une matrice de poids et un dictionnaire.
    Attributs
        matrice_poids : array float numpy
            matrice symétrique de nombres positifs à diagonale nulle.
        Dictionnaire : dict
            dictionnaire liant les numéros de ligne/colonne de matrice_poids aux codes
            UIC des gares correspondant"""

  def __init__(TableTrajets):
    """Transforme la table des trajets en un graphe.
      Arguments
        Prend un dataframe en argument.
      Renvoie
          Un Graphe."""

  def verifie_connexe(self):
    """ Prend un Graphe en argument et vérifie s'il est bien connexe
```

```
    Arguments
          graphe : un Graphe
      Renvoie
        un booléen : True si le graphe est connexe, False sinon."""

  def verifie_sommets_lies(self, origine, destination):
    """Vérifie qu'origine et destination sont des sommets reliables du graphe.
      Arguments
          graphe : Graphe
          origine : str
              Code UIC de la gare d'origine.
          Destination : str
              code UIC de la gare de destination.
        Renvoie
          un booléen : True si on peut relier l'origine à la destination, False sinon."""

  def dijkstra_source_vers_destination(self, origine, destination):
      """Applique l'algo de Dijkstra pour relier à moindre coût origine à destination.
    Arguments
          graphe : Graphe
              le graphe des trajets.
          origine : str
              Code UIC de la gare d'origine.
          Destination : str
              code UIC de la gare de destination.
      Renvoie
          Un parcours (cf classe parcours)."""

  def dijkstra_source_vers_tous(self, origine):
      """Applique l'algo de Dijkstra de l'origine aux sommets   du graphe.
    Arguments
          graphe : graphe
            le graphe des trajets.
        origine : str
            Code UIC de la gare d'origine.
    Renvoie
        Un dictionnaire qui à chaque sommet (str UIC) associe Parcours le moins cher."""
```

3.4 Classe Parcours

```
Class Parcours
    """Donne un parcours sous forme de dataframe. Une colonne UIC_origine, une colonne
    UIC_destination et une colonne prix. L'UIC_destination est égal à l'UIC_origine de
    la ligne    précédente."""
  def trajet_total(self):
    """Renvoie une seule ligne avec le premier UIC_origine, le dernier UIC_destination
           et le prix total (la somme des prix des trajets)"""

  def __str__(self):
        """Transforme le parcours en texte, ajoute une ligne d'ensemble au final (coût
        total)."""

  def export(self, filename):
        """Exporte le parcours dans un fichier csv ou txt (à déterminer).
        Arguments
```

```
        filename : str
          le nom du fichier de l'export (avec le chemin).
        Renvoie
          rien (mais écrit le fichier)"""
```

3.5 Fonctions hors classes

La fonction *___init___()* du package chargera les quatre tables *tableTgv*, *tableTer*, *tableVoyageurs* et *tableCorrespondances*. La fonction *creation_tableCorrespondances()* créra la table des correspondances. Une fonction permettra de lancer l'ensemble de l'algorithme avec les filtres désirés.

4 Déroulement du projet

4.1 Attendus et livrables

— Cahier des charges : pour le lundi 10 avril 2023 à 23h59. $\leq$ 5 pages.
— Note de suivi personnel : pour le 22 mai 2023 à 23h59. Travail individuel. $\leq$ 2 pages.
— Code : pour le 24 mai 2023 à 23h59.
— Rapport : pour le 24 mai 2023 à 23h59. $\leq$ 25 pages.
— Soutenance : du 31 mai au 2 juin 2023.

4.2 Planification

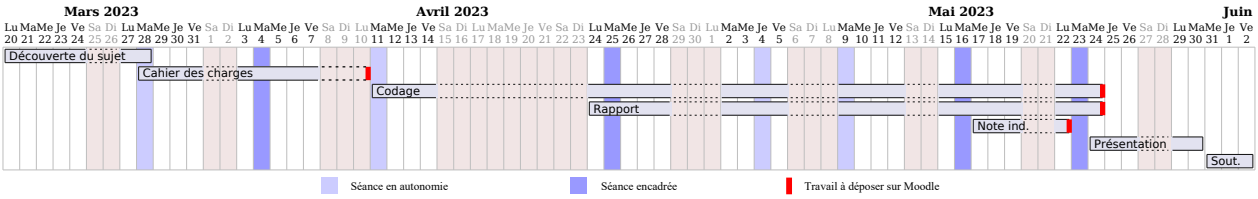

FIGURE 2 – Diagramme de Gantt du projet