Programmation et Applications Interactives

Introduction à Pandas

Philippe Blasi

Pandas : Panel Data Analysis (pandas.pydata.org)

- Manipulation de tableaux de données hétérogènes
- Collaboration avec NumPy

Différences Panda ↔ NumPy

Panda

- **3 types de conteneurs**
  - 1D : series
  - 2D : dataframe (table)
  - 3D : ~~panel (classeur)~~ ⇒ multiIndex
- **Nombre quelconque de type de données**

NumPy

- **1 seul conteneur : array**
  - Nombre de dimensions quelconque
- **Données du même type**

Différences Panda ↔ NumPy

Panda

- **Type d'index libre**
  - Comme dictionnaire Python
- **Gère bien les données partielles**
  - Outils pour données manquantes

NumPy

- **Dimensions indexées par des entiers**
  - Démarrage à 0
- **Cases vides possibles**
  - NaN pour Not a Number
  - Problème pour les traitements

Démarrer avec Panda

- **Installer la bibliothèque**
  - `pip install pandas`
- **Plus simple, installer Anaconda**
- **Importer la bibliothèque**
  - `import panda as pd`
- **Utiliser la bibliothèque**
  - `fr = pd.DataFrame({'A': [1, 2, 3],'B': [4, 5, 6]})`

Création de séries et accès

- **1-D matrice = série**
  - Création de base
    - `vec = [1,2,3,4,5]`
    - `ser = pd.Series(vec) # à partir d'une liste`
    - `Indices de 0 à 4`
    - `Se comporte comme un vecteur NumPy`
  - Création de base en spécifiant un index de str (labels)
    - `ser = pd.Series(vec,index=['A','B','C','D','E'])`
  - Contenu de la série
    - `ser.values ⇒ [1,2,3,4,5]`
    - `ser.index ⇒ Index(['A', 'B', 'C', 'D', 'E'], dtype='object')`
    - `ser.index.values ⇒ ['A' 'B' 'C' 'D' 'E']`

Création de séries et accès

- **1-D matrice = série**
  - Accès comme en NumPy

```
ser[1]                # accès par position
ser['D']              # accès par label
ser.D                 # accès par attribut
ser[1:4]              # accès par tranche de positions
ser['B':'D']          # accès par tranche de labels (inclus)
ser[[4,2,1]]          # accès par énumération de positions
ser[['E','C','B']] # accès par énumération de labels
ser[ser > 3]          # accès par prédicat
```

Création de séries et accès

- **1-D matrice = série**
  - Création à partir de vecteur NumPy
    - `ser = pd.Series(np.random.randint(0, 1000, 11))`
    - `ser.values ⇒ [243 874 361 102 327 248 263 928  80 789 163]`
    - `ser.index.values ⇒ [ 0  1  2  3  4  5  6  7  8  9 10]`
    - `Se comporte comme un vecteur NumPy`
  - Création NP avec un IntervalIndex (index d'intervalles) de flottants
    - ```
      intervals = pd.interval_range(0, 10, freq=2.5, closed='left')
      # closed : left|right|both|neither
      ```
    - `ser = pd.Series(np.random.rand(intervals.size), index=intervals)`
    - ```
      ser.index ⇒ IntervalIndex([[0.0, 2.5), [2.5, 5.0), [5.0, 7.5), [7.5, 10.0)],
                                 dtype='interval[float64, left]')
      ```
    - `ser[2.0] ⇒ première valeur`
    - `Ser[2.5] ⇒ seconde valeur`

Création de séries et accès

- **1-D matrice = série**
  - Création NP avec un DateTimeIndex
    - `index = pd.date_range('2000-01-01', periods=6, freq='MS')`
      `# création d'un index avec fréquence mensuelle`
    - `D = jour, W ou W-day = semaine, M ou MS = mois,`
      `Q ou QS = trimestre, Y ou YS = année`
    - `ser = pd.Series(np.random.rand(index.size), index=index)`
    - `ser.index ⇒ DatetimeIndex(['2000-01-01', '2000-02-01',`
      `'2000-03-01', '2000-04-01','2000-05-01', '2000-06-01'],`
      `dtype='datetime64[ns]', freq='MS')`

Création de séries et accès

- **1-D matrice = série**
  - Création NP avec un TimedeltaIndex
    - `index = pd.timedelta_range('15H', periods=9, freq='1H30T').round('T') # intervalle 1H30`
    - `N = nanoseconde, U = microseconde, L = milliseconde, S = seconde, T = minute, H = heure`
    - `ser = pd.Series(np.random.rand(index.size), index=index)`
    - `ser.index ⇒ TimedeltaIndex( ['0 days 15:00:00', '0 days 16:30:00', '0 days 18:00:00','0 days 19:30:00', '0 days 21:00:00', '0 days 22:30:00','1 days 00:00:00', '1 days 01:30:00', '1 days 03:00:00'], dtype='timedelta64[ns]', freq=None)`

Création de dataframe (table)

- **2-D matrice = dataframe**
  - Création de base
    - `mat = [[1,-2,3,-4], [-5,6,-7,8], [9,-10,11,-12]]`
    - `df = pd.DataFrame(mat) # création d'une table à partir de la matrice de test`
  - Contenu de la table
    - `df.values` ⇒ `matrice numpy`
    - `df.shape` ⇒ `(3,4)` `lignes et colonnes`
    - `df.size` ⇒ `12` `nombre d'éléments`
    - `df.index.values` ⇒ `[0 1 2]` `index des lignes`
    - `df.columns.values` ⇒ `[0 1 2 3]` `index des colonnes`

Création de dataframe (table)

- **2-D matrice = dataframe**
  - Accès : **indices pour les colonnes, tranches d'indices pour les lignes**
    - `df[1]` ⇒ accès à une colonne
    - `df.T[1]` ⇒ accès à une ligne
    - `df.iloc[1]` ⇒ idem avec 'iloc' : df.T[1] <=> df.iloc[1]
    - `df[1:3]` ⇒ accès à une tranche de lignes
    - `df.T[1:3].T` ⇒ accès à une tranche de colonnes
    - `df.iloc[:,1:3]` ⇒ idem avec 'iloc': df.T[1:3].T <=> df.iloc[:,1:3]

Création de dataframe (table)

- **2-D matrice = dataframe**
  - Accès par énumération
    - Liste d'indices
    - **`df[[3,1]]`** ⇒ **accès par énumération de colonnes 3 et 1**
    - **`df.iloc[[2,0]]`** ⇒ **accès par énumération de lignes 2 et 0**
  - Accès par prédicat
    - Données ne vérifiant pas le prédicat remplacées par NaN (Not-A-Number)
    - Données de la série (colonne) contenant un NaN convertis en **`float`**

Création de dataframe (table)

- **2-D matrice = dataframe**
  - Accès par énumération sur index non entier (caractère ici)
    - `df = pd.DataFrame(mat, columns=list('ABCD'))`
    - `df['B']` ⇒ accès à une colonne par label
    - `df.B` ⇒ idem à une colonne par attribut
    - `df.T[1]` ⇒ accès à une ligne par position <=> `df.iloc[1]`
    - `df[1:3]` ⇒ accès à une tranche de ligne par positions
    - `df.T['B':'D'].T` ⇒ accès à une tranche de colonne par labels <=> `df.loc[:,'B':'D']`

Création de dataframe (table)

- **2-D matrice = dataframe**
  - Index plus évolués
    - Exemple : dates pour les lignes et lettres pour les colonnes
    - `dates = pd.date_range('2000-01-01', periods=6, freq='MS')`
    - `labels = list('ABCDEF')`
    - `data = ...`
    - `df = pd.DataFrame(data, index=dates, columns=labels)`
  - Accès aux lignes
    - `df.T['2000-03-01']` ⇒ `ligne par date  df.loc['2000-03-01']`
    - `df.iloc[2]` ⇒ `ligne par position`
    - `df['2000-03-01':'2000-05-01']` ⇒ `tranche de lignes par dates`
    - `df[2:5]` ⇒ `tranche de lignes par positions`

Création de dataframe (table)

- **2-D matrice = dataframe**
  - Création plus évoluées : données hétérogènes
    - Dictionnaire : clé = nom de la colonne, valeur = valeur de la colonne
    - `dic = {}`
    - `dic["Name"]  = [3*c for c in 'ABCDEF']`
    - `dic["Model"] = pd.Categorical(2*['S','M','L'])`
    - `dic["Start"] = 0`
    - `dic["Stop"]  = np.random.randint(0,100,(6,), dtype='u1')`
    - `dic["Score"] = np.linspace(0, 3, 6)`
    - `df = pd.DataFrame(dic)`

Création de dataframe (table)

- **2-D matrice = dataframe**
  - Création plus évoluées : modification de l'index
    - On peut remplacer l'index des lignes par les valeurs d'une colonne
    - ATTENTION : doublons interdits
    - **`df.index = df.Date`** ⇒ **`on utilise la colonne 'Date' comme index`**
    - **`df.index.name = None`** ⇒ **`le label de l'index est inutile, donc on le supprime`**

Création de dataframe (table)

- **2-D matrice = dataframe**
  - Lecture sur disque
    - `csv = pd.read_csv('TEST/test-CSV.csv')`
    - `json = pd.read_json('TEST/test-JSON.json')`
    - `authors = pd.read_html('TEST/test-HTML.html')`

Manipulation des séries et des tables

- **Information générale**
  - `df.head(3)` ⇒ 'n' premières lignes (5 par défaut)
  - `df.tail(1)` ⇒ 'n' dernières lignes (5 par défaut)
  - `df.info()` ⇒ informations générales
  - `df.describe()` ⇒ statistiques descriptives sur chaque colonne à valeurs numériques
  - `df.describe(include='all', datetime_is_numeric=True)`
    ⇒ on force l'inclusion des colonnes non-numériques

Manipulation des séries et des tables

- **Extraction de données**
  - Sélection rectangulaire dans le DataFrame par tranche
    - `df['CCC':'EEE'].T['Model':'Test'].T` ⇒ **accès avec double tranche de labels (peu pratique)**
    - `df.loc['CCC':'EEE','Model':'Test']` ⇒ **mieux avec 'loc'**
    - `df.iloc[2:5,1:6]` ⇒ **idem avec 'iloc' avec les positions**

Manipulation des séries et des tables

- **Extraction de données**
  - Sélection rectangulaire dans le DataFrame par énumération
    - `df.T[['FFF','AAA','DDD']].T[['Date','Score','Model']]`
      ⇒ **double énumération de labels (peu pratique)**
    - `df.loc[('FFF','AAA','DDD'),('Date','Score','Model')]`
      ⇒ **mieux avec 'loc'**
    - `df.iloc[[5,0,3],[6,4,1]]`
      ⇒ **idem avec 'iloc' avec les positions)**

Manipulation des séries et des tables

- **Extraction de données**
  - Sélection par prédicat
    - `df.Score > 1.5` ⇒ **création d'un prédicat sur une colonne**
    - `(df.Model == 'S') & (dd.Score > 1.5)` ⇒ **idem sur plusieurs colonnes (parenthèses obligatoires)**
    - `df[df.Model == 'S']` ⇒ **extraction des lignes vérifiant un prédicat**
    - `df[df.Model == 'S'].Score` ⇒ **idem avec sélection d'une seule colonne**
    - `df[df.Model == 'S'][['Date','Test','Score']]` ⇒ **idem avec sélection de plusieurs colonnes**

Manipulation des séries et des tables

- **Insertion de données**
  - Ajout d'une colonne comme pour un dictionnaire Python
  - `df['Rank'] = range(6,0,-1)` ⇒ insertion d'une colonne complète
  - `df['Final'] = dd.Stop * dd.Score` ⇒ insertion d'une colonne complète par combinaison
  - `df['Color'] = pd.Series(dict(AAA='blue', CCC='green', DDD='red'))` ⇒ insertion d'une colonne incomplète
  - `df['Color-Name'] = dd.Color + '-' + dd.Name` ⇒ insertion d'une colonne incomplète par combinaison

Manipulation des séries et des tables

- **Insertion de données**
  - Ajout d'une ligne ⇒ il faut y accéder par loc
  - `df.loc['ZZZ'] = dict(Name='ZZZ', Model='S', Start=1, Test=False, Color='white')`

Manipulation des séries et des tables

- **Suppression de données**
  - La suppression est faite par défaut sur une copie, sans modifier l'original. Il faut le spécifier explicitement (**`inplace=True`**) pour la suppression se fasse.
  - Type python
    - **`del df['Color-Name']` ⇒ suppression d'une colonne (sur place)**
    - **`df.drop(columns=['Date','Delta'])` ⇒ suppression d'un groupe de colonnes (sur copie)**
    - **`df.drop(columns=['Date','Delta'], inplace=True)` # suppression d'un groupe de colonnes (sur l'original)**

Manipulation des séries et des tables

- **Modification de données**
  - Il faut y accéder par loc ou iloc
    - `df.loc['FFF','Color'] = 'yellow'` ⇒ **modification d'un élément individuel (en utilisant les labels)**
    - `df.iloc[2,2] = 3` ⇒ **idem en utilisant les positions**
  - Effacer une données
    - `df.loc['FFF','Color'] = np.nan`
    - `df.iloc[2,2] = None`
  - Remplacement des NaN
    - `df.Color.fillna('black')` ⇒ **remplacement par une valeur constante**
    - `df.Score.fillna(dd.Score.mean())` ⇒ **remplacement par une valeur calculée**