Prétraitement des données

Mustapha El Ossmani

Filière TIADS
ENSAM-Meknès

Plan

Outline

Introduction

Qu'est-ce que la **Science des Données ?**

La science des données est un domaine **interdisciplinaire** qui combine des compétences en **statistiques**, en informatique et en domaines spécifiques pour extraire des connaissances et des insights à partir de **données structurées et non structurées**.

- La science des données implique **l'acquisition, le nettoyage, l'analyse et l'interprétation** des données.
- Elle repose sur des méthodes statistiques, des **algorithmes d'apprentissage automatique** et des techniques de visualisation de données.
- La data science est désormais **l'outil essentiel d'aide à la décision** dans des domaines d'activités extrêmement variés : **la santé, la finance, le marketing, etc.**

Compétences Clés en Science des Données

- **Statistiques** : Compréhension des concepts statistiques fondamentaux pour analyser les données et évaluer les modèles.
- **Informatique** : Maîtrise de langages de programmation comme Python ou R pour manipuler les données et implémenter des algorithmes.
- **Domaine d'Application** : Connaissance approfondie du domaine spécifique pour interpréter correctement les résultats et formuler des questions pertinentes.
- **Communication** : Capacité à présenter les résultats de manière claire et à communiquer efficacement avec les parties prenantes.

La plupart des **analystes de données** utilisent des programmes de **feuilles de calcul** comme **Microsoft Excel** et **Google Sheets** ou **des logiciels statistiques propriétaires** comme **SAS, Stata, ou SPSS**. Toutefois, **ces différents outils présentent des limites**.

- **Excel** ne peut prendre en charge des ensembles de données au-delà d'une certaine limite.
- **Excel** ne permet pas de reproduire des analyses sur de nouveaux ensembles de données.
- La principale faiblesse des programmes comme **SAS** est qu'ils ont été développés pour un usage très spécifique, et ne bénéficient pas d'une vaste communauté de contributeurs capables d'ajouter de nouveaux outils.

Python et R pour les sciences des données

R et Python permettent de **dépasser les limites des programmes comme SAS et Excel**

R et Python sont deux principaux langages de programmation **utilisés par les analystes de données et les data scientists**.

- Les deux langages **R** ou **Python** **sont gratuits et open source**, et furent développés au début des années 90.
- **R est dédié aux analyses statistiques**, et **Python est un langage de programmation plus généraliste**.

Ces deux langages sont idéaux pour travailler sur de **larges ensembles de données** ou créer des **data visualisations complexes**, ...

Python et R pour les sciences des données

Quel est le meilleur de ces langages de programmation à apprendre pour l'analyse de données ?

- Contrairement à **Python**, **R** ne permet pas de créer des **site web et d'automatiser les processus**. En revanche, **R** est plus adapté pour les projets lourds en **statistiques et les explorations ponctuelles d'ensembles de données**.
- Un autre avantage de **Python** est qu'il s'agit d'un langage **plus généraliste**, qui pourra être utilisé également pour la **création de site web ou autre programme informatique**.

De fait, pour une personne qui souhaite devenir programmeur, **Python** est mieux adapté.

Outline

Environnement et outils de travail

Python est l'un des langages les plus populaires utilisés par les scientifiques et les développeurs de logiciels pour les tâches liées à la science des données (Data Science), couvrant des domaines tels que le traitement des données, la modélisation et la visualisation.

- Durant tout le cours, nous utiliserons **Python** comme langage de programmation.
- Les versions 2 et 3 du langage **Python** présentent des différences. Nous donnerons par défaut des programmes adaptés pour les versions 3 et ultérieures.
- Pour exécuter les programmes, il vous faudra disposer de **Python** et des bibliothèques adéquates.
- Vous pouvez installer le logiciel gratuit **Anaconda** développé par la société Continuum car il contient l'ensemble de l'environnement de développement : http ://continuum.io/downloads
- Pour ce cours, nous utiliserons la version Python 3.5. Anaconda comprend **Spyder** (**S**cientific **PY**thon **D**eveloppement **E**nviRonment) comme environnement de développement intégré.

Librairies Python pour DS

Voici une liste non exhaustive des bibliothèques Python pour la science des données que vous devez connaître.

1. **Extraction de données**
   - **Scrapy** : permet de récupérer des données structurées sur le web -par exemple, des URL ou des informations de contact. C'est un excellent outil pour scraper les données utilisées, par exemple, dans les modèles de Machine Learning en Python. https ://github.com/scrapy/scrapy
   - **BeautifulSoup** : très populaire pour le crawling sur le web et le scraping de données. Si vous souhaitez collecter des données disponibles sur un site web mais non via un fichier CSV BeautifulSoup peut vous aider à les scraper et à les organiser dans le format dont vous avez besoin. https ://www.crummy.com/software/BeautifulSoup/bs4/doc/

Librairies Python ....

2. **Calcul scientifique** : Traitement et modélisation des données
   - **NumPy** : (Numerical Python) est un outil parfait pour le calcul scientifique et la réalisation d'opérations de base et avancées avec des tableaux.
   - **SciPy** : comprend des modules pour l'algèbre linéaire, l'intégration, l'optimisation et les statistiques. Sa fonctionnalité principale a été construite sur NumPy, donc ses tableaux utilisent cette bibliothèque.
   - **Pandas** : Elle est basée sur deux structures de données principales : « Série » (unidimensionnelle, comme une liste Python) et « Dataframe » (bidimensionnelle, comme un tableau à plusieurs colonnes). Pandas permet de gérer les données manquantes et d'ajouter/supprimer des colonnes de DataFrame, d'imputer les fichiers manquants et de tracer les données avec un histogramme ou une boîte à moustache. C'est un outil indispensable pour la manipulation et la visualisation des données.

Librairies Python ....

3. **Visualisation de données**
   - **Matplotlib** : pour générer des visualisations de données telles que des diagrammes et des graphiques bidimensionnels (histogrammes, diagrammes de dispersion, ..).
   - **Seaborn** : basé sur Matplotlib et très utile pour la visualisation de modèles statistiques–cartes thermiques et autres types de visualisations. En utilisant cette bibliothèque, vous bénéficiez d'une vaste galerie de visualisations (y compris des visualisations complexes comme les séries temporelles et les tracés conjoints).
4. **IA : Deep Learning & Machine learning**
   - **Keras** : est une excellente bibliothèque pour la construction de réseaux de neurones et la modélisation.
   - **Scikit-Learn** : utilise les opérations mathématiques de SciPy pour traiter les tâches standard de Machine Learning.
   - **PyTorch**
   - **TensorFlow**

Prise en main du Python : un peu de statistique

Réaliser le LAB 0

Outline

Prétraitement de ces données

Dû à la grande taille des bases de données actuelles, les données brutes sont généralement de faible qualité.

- On ne peut pas faire de la science des données d'une manière fiable sur des données brutes qui peuvent avoir plusieurs anomalies : **codes erronés, données manquantes, données aberrantes, variables inutiles, variables redondantes...etc.**
- Le prétraitement de ces données est une étape cruciale dans le processus de découverte de connaissances à partir de grandes bases de données.
- Les étapes d'un **prétraitement des données** sont :
  - Importation des données
  - Nettoyage
    - Corrections des doublons, des erreurs de saisie.
    - Détection des valeurs aberrantes.
    - Détection des informations manquantes.
  - Enrichissement des données
  - Détermination des valeurs catégorielles
  - Transformation et mise à l'échelle

Importation des données

- **Sources des données** : Les données nous parviennent sous une grande variété de formats : fichiers CSV ou Excel, tableaux de bases de données SQL, logiciels d'analyse statistique tels que SPSS, Stata, SAS ou R, de sources non tabulaires telles que JSON(JavaScript Objet Notation) et de pages Web.

Importation des données

- **Sources des données** : Les données nous parviennent sous une grande variété de formats : fichiers CSV ou Excel, tableaux de bases de données SQL, logiciels d'analyse statistique tels que SPSS, Stata, SAS ou R, de sources non tabulaires telles que JSON(JavaScript Objet Notation) et de pages Web.
  - Importation de fichiers CSV
  - Importation de fichiers Excel files
  - Importation à partir de basse de données SQL
  - Importation de données à partir des pages WEB
    - Requests
    - Beautiflsoop
    - scrap

Importation de fichiers CSV

Nous allons importer un fichier CSV dans pandas, en tirant parti de certaines options très utiles de la fonction **read_csv** très utiles :

1. **Importer la bibliothèque pandas et configurer l'environnement pour faciliter l'affichage des résultats.**

Importation de fichiers CSV

Nous allons importer un fichier CSV dans pandas, en tirant parti de certaines options très utiles de la fonction **read_csv** très utiles :

1. **Importer la bibliothèque pandas et configurer l'environnement pour faciliter l'affichage des résultats.**

```
1 import pandas as pd
2 pd.options.display.float_format = '{:,.2f}'.format
3 pd.set_option('display.width', 75)
4 pd.set_option('display.max_columns', 5)
```

Importation de fichiers CSV

Nous allons importer un fichier CSV dans pandas, en tirant parti de certaines options très utiles de la fonction **read_csv** très utiles :

1. **Importer la bibliothèque pandas et configurer l'environnement pour faciliter l'affichage des résultats.**

```
1 import pandas as pd
2 pd.options.display.float_format = '{:,.2f}'.format
3 pd.set_option('display.width', 75)
4 pd.set_option('display.max_columns', 5)
```

2. **Lire le fichier de données, définir de nouveaux noms pour les rubriques et analyser la colonne de date.**

Importation de fichiers CSV

Nous allons importer un fichier CSV dans pandas, en tirant parti de certaines options très utiles de la fonction **read_csv** très utiles :

1. **Importer la bibliothèque pandas et configurer l'environnement pour faciliter l'affichage des résultats.**

```
1 import pandas as pd
2 pd.options.display.float_format = '{:,.2f}'.format
3 pd.set_option('display.width', 75)
4 pd.set_option('display.max_columns', 5)
```

2. **Lire le fichier de données, définir de nouveaux noms pour les rubriques et analyser la colonne de date.**
   Passer 1 à **skiprows** pour sauter la première ligne. Utiliser **parse_dates** pour créer une colonne datetime, et mettre **low_memory** à False pour réduire l'utilisation de la mémoire pendant l'importation :

Importation de fichiers CSV

Nous allons importer un fichier CSV dans pandas, en tirant parti de certaines options très utiles de la fonction **read_csv** très utiles :

1. **Importer la bibliothèque pandas et configurer l'environnement pour faciliter l'affichage des résultats.**

```
1 import pandas as pd
2 pd.options.display.float_format = '{:,.2f}'.format
3 pd.set_option('display.width', 75)
4 pd.set_option('display.max_columns', 5)
```

2. **Lire le fichier de données, définir de nouveaux noms pour les rubriques et analyser la colonne de date.**
   Passer 1 à **skiprows** pour sauter la première ligne. Utiliser **parse_dates** pour créer une colonne datetime, et mettre **low_memory** à False pour réduire l'utilisation de la mémoire pendant l'importation :

```
1 landtemps = pd.read_csv('data/landtempssample.csv',
2     names=['stationid','year','month','avgtemp','latitude',
3       'longitude','elevation','station','countryid','country'],
4     skiprows=1,
5     parse_dates=[['month','year']],
6     low_memory=False)
7 type(landtemps)
```

3. **Obtenir un aperçu rapide sur les données.**
   Afficher les premières lignes. Affichez le type de données de toutes les colonnes, ainsi que le nombre de lignes et de colonnes.

```
1 landtemps.head(7)
2 landtemps.dtypes
3 landtemps.shape
```

4. **Donner un meilleur nom à la colonne de date et afficher les statistiques récapitulatives pour la température mensuelle moyenne :**

```
1 landtemps.rename(columns={'month_year':'measuredate'}, inplace=True)
2 landtemps.dtypes
3 landtemps.avgtemp.describe()
```

5. **Rechercher les valeurs manquantes pour chaque colonne.**
   Utiliser **isnull**, qui renvoie Vrai pour chaque valeur manquante pour chaque colonne, et False lorsqu'elle n'est pas manquante. Enchaîner ceci avec **sum** pour compter les valeurs manquantes pour chaque colonne. Lorsque vous travaillez avec des valeurs booléennes, **sum** traite True comme 1 et False comme 0 .

```
1 landtemps.isnull().sum()
```

6. Supprimer les lignes avec des données manquantes pour avgtemp

6. **Supprimer les lignes avec des données manquantes pour avgtemp**
   - Utiliser le paramètre **subset** pour indiquer à **dropna** de supprimer les lignes où **avgtemp** est manquant.
   - Définir **inplace** à True.
   - Utiliser l'attribut **shape** pour obtenir le nombre de lignes et de colonnes

6 Supprimer les lignes avec des données manquantes pour avgtemp

- Utiliser le paramètre **subset** pour indiquer à **dropna** de supprimer les lignes où **avgtemp** est manquant.
- Définir **inplace** à True.
- Utiliser l'attribut **shape** pour obtenir le nombre de lignes et de colonnes

```
1 # remove rows with missing values
2 landtemps.dropna(subset=['avgtemp'], inplace=True)
3 landtemps.shape
```

Importation de fichiers EXCEL

- Sélectionner la feuille contenant les données dont nous avons besoin, mais ignorer les colonnes et les lignes que nous ne voulons pas.
- Utiliser le paramètre **sheet_name** pour spécifier la feuille.
- Définir **skiprows** à 4 et **skipfooter** à 1 pour ignorer les quatre premières lignes et la dernière ligne.
- Fournissons des valeurs pour **usecols** afin d'obtenir les données de la colonne A et des colonnes C à T (la colonne B est vide).
- Utilisez **head** pour afficher les premières lignes :

Importation de fichiers EXCEL

- Sélectionner la feuille contenant les données dont nous avons besoin, mais ignorer les colonnes et les lignes que nous ne voulons pas.
- Utiliser le paramètre **sheet_name** pour spécifier la feuille.
- Définir **skiprows** à 4 et **skipfooter** à 1 pour ignorer les quatre premières lignes et la dernière ligne.
- Fournissons des valeurs pour **usecols** afin d'obtenir les données de la colonne A et des colonnes C à T (la colonne B est vide).
- Utilisez **head** pour afficher les premières lignes :

```
1 # import the land temperature data
2 import pandas as pd
3 percapitaGDP = pd.read_excel("data/GDPpercapita.xlsx",
4    sheet_name="OECD.Stat export",
5    skiprows=4,
6    skipfooter=1,
7    usecols="A,C:T")
8    percapitaGDP.head()
```

- Utilisez la méthode **info** pour visualiser les types de données et le nombre de données non nulles :

- Utilisez la méthode **info** pour visualiser les types de données et le nombre de données non nulles :

```
1 percapitaGDP.info()
```

- Utilisez la méthode **info** pour visualiser les types de données et le nombre de données non nulles :

```
1 percapitaGDP.info()
```

- Donner un nom approprié à la colonne des régions métropolitaines (Year to metro).
- Il y a des espaces supplémentaires avant les valeurs métropolitaines dans certains cas, et des espaces supplémentaires après les valeurs métropolitaines dans d'autres cas.
- Nous pouvons vérifier la présence d'espaces avant avec **startswith(' ')**, puis utiliser **any** pour établir s'il y a une ou plusieurs occasions où le premier caractère est blanc.
- Nous pouvons utiliser **endswith(' ')** pour examiner les espaces à la fin.
- Nous utilisons **strip** pour pour supprimer les espaces initiaux et finaux.

- Utilisez la méthode **info** pour visualiser les types de données et le nombre de données non nulles :

```
1 percapitaGDP.info()
```

- Donner un nom approprié à la colonne des régions métropolitaines (Year to metro).
- Il y a des espaces supplémentaires avant les valeurs métropolitaines dans certains cas, et des espaces supplémentaires après les valeurs métropolitaines dans d'autres cas.
- Nous pouvons vérifier la présence d'espaces avant avec **startswith(' ')**, puis utiliser **any** pour établir s'il y a une ou plusieurs occasions où le premier caractère est blanc.
- Nous pouvons utiliser **endswith(' ')** pour examiner les espaces à la fin.
- Nous utilisons **strip** pour pour supprimer les espaces initiaux et finaux.

```
1 percapitaGDP.rename(columns={'Year':'metro'}, inplace=True)
2 percapitaGDP.metro.str.startswith(' ').any()
3 percapitaGDP.metro.str.endswith(' ').any()
4 percapitaGDP.metro = percapitaGDP.metro.str.strip()
```

- Itérer sur toutes les colonnes d'année (2001-2018) et convertir le type de données d'objet à flottant.
- **Coerce** la conversion même lorsqu'il y a des données de type caractère ... dans cet exemple.
- Nous voulons que les valeurs de caractères dans ces colonnes deviennent manquantes, ce qui est le cas.
- Renommer les colonnes d'année pour mieux refléter les données dans ces colonnes :

- Itérer sur toutes les colonnes d'année (2001-2018) et convertir le type de données d'objet à flottant.
- **Coerce** la conversion même lorsqu'il y a des données de type caractère ... dans cet exemple.
- Nous voulons que les valeurs de caractères dans ces colonnes deviennent manquantes, ce qui est le cas.
- Renommer les colonnes d'année pour mieux refléter les données dans ces colonnes :

```
1 for col in percapitaGDP.columns[1:]:
2   percapitaGDP[col] = pd.to_numeric(percapitaGDP[col], errors='coerce')
3   percapitaGDP.rename(columns={col:'pcGDP'+col}, inplace=True)
```

- Itérer sur toutes les colonnes d'année (2001-2018) et convertir le type de données d'objet à flottant.
- **Coerce** la conversion même lorsqu'il y a des données de type caractère ... dans cet exemple.
- Nous voulons que les valeurs de caractères dans ces colonnes deviennent manquantes, ce qui est le cas.
- Renommer les colonnes d'année pour mieux refléter les données dans ces colonnes :

```
1 for col in percapitaGDP.columns[1:]:
2   percapitaGDP[col] = pd.to_numeric(percapitaGDP[col], errors='coerce')
3   percapitaGDP.rename(columns={col:'pcGDP'+col}, inplace=True)
```

- Utiliser la méthode **describe** pour générer des statistiques récapitulatives pour toutes les données numériques dans le Data Frame :

- Itérer sur toutes les colonnes d'année (2001-2018) et convertir le type de données d'objet à flottant.
- **Coerce** la conversion même lorsqu'il y a des données de type caractère ... dans cet exemple.
- Nous voulons que les valeurs de caractères dans ces colonnes deviennent manquantes, ce qui est le cas.
- Renommer les colonnes d'année pour mieux refléter les données dans ces colonnes :

```
1 for col in percapitaGDP.columns[1:]:
2   percapitaGDP[col] = pd.to_numeric(percapitaGDP[col], errors='coerce')
3   percapitaGDP.rename(columns={col:'pcGDP'+col}, inplace=True)
```

- Utiliser la méthode **describe** pour générer des statistiques récapitulatives pour toutes les données numériques dans le Data Frame :

```
1 percapitaGDP.head()
2 percapitaGDP.dtypes
3 percapitaGDP.describe()
```

- Utiliser le paramètre **subset** de **dropna** pour inspecter toutes les colonnes, en commençant par la deuxième colonne jusqu'à la dernière colonne.
- Utiliser **how** pour spécifier que nous voulons supprimer des lignes uniquement si toutes les colonnes spécifiées dans **subset** sont manquantes. item Utiliser **shape** pour afficher le nombre de lignes et de colonnes dans le Data Frame résultant :

- Utiliser le paramètre **subset** de **dropna** pour inspecter toutes les colonnes, en commençant par la deuxième colonne jusqu'à la dernière colonne.
- Utiliser **how** pour spécifier que nous voulons supprimer des lignes uniquement si toutes les colonnes spécifiées dans **subset** sont manquantes. item Utiliser **shape** pour afficher le nombre de lignes et de colonnes dans le Data Frame résultant :

```
1 percapitaGDP.dropna(subset=percapitaGDP.columns[1:], how="all", inplace=True)
2 percapitaGDP.describe()
3 percapitaGDP.head()
4 percapitaGDP.shape
```

- Définir l'index pour le Data Frame en utilisant la colonne de la région métropolitaine.
- Confirmer qu'il y a 480 valeurs valides pour metro et qu'il y a 480 valeurs uniques, avant de définir l'index.

```
1 percapitaGDP.metro.count()
2 percapitaGDP.metro.nunique()
3 percapitaGDP.set_index('metro', inplace=True)
4 percapitaGDP.head()
5 percapitaGDP.loc['AUS02: Greater Melbourne']
```

Réaliser le LAB 1

Importation à partir de basse de données SQL

- **Importer pandas, numpy, pymssql, et mysql**.
  Cette étape suppose que vous avez installé les API **pymssql** et **mysql** (pip install **pymssql** or pip install **mysql-connector-python**)

Importation à partir de basse de données SQL

- **Importer pandas, numpy, pymssql, et mysql**.
  Cette étape suppose que vous avez installé les API **pymssql** et **mysql** (pip install **pymssql** or pip install **mysql-connector-python**)

```
1 import pandas as pd
2 import numpy as np
3 import pymssql
4 import mysql.connector
```

Importation à partir de basse de données SQL

- **Importer pandas, numpy, pymssql, et mysql.**
  Cette étape suppose que vous avez installé les API **pymssql** et **mysql** (pip install **pymssql** or pip install **mysql-connector-python**)

```
1 import pandas as pd
2 import numpy as np
3 import pymssql
4 import mysql.connector
```

- **Sélectionner les colonnes que nous voulons à partir des données du serveur SQL et utiliser des alias SQL pour améliorer les noms de colonnes (par exemple, fedu AS fathereducation).**

Importation à partir de basse de données SQL

- **Importer pandas, numpy, pymssql, et mysql.**
  Cette étape suppose que vous avez installé les API **pymssql** et **mysql** (pip install **pymssql** or pip install **mysql-connector-python**)

```
1 import pandas as pd
2 import numpy as np
3 import pymssql
4 import mysql.connector
```

- **Sélectionner les colonnes que nous voulons à partir des données du serveur SQL et utiliser des alias SQL pour améliorer les noms de colonnes (par exemple, fedu AS fathereducation).**

```
1 query = "SELECT studentid, school, sex, age, famsize,\
2   medu AS mothereducation, fedu AS fathereducation,\
3   traveltime, studytime, failures, famrel, freetime,\
4   goout, g1 AS gradeperiod1, g2 AS gradeperiod2,\
5   g3 AS gradeperiod3 From studentmath"
```

- Créer une connexion aux données du serveur SQL en passant les informations d'identification de la base de données à la fonction pymssql

- **Créer une connexion aux données du serveur SQL en passant les informations d'identification de la base de données à la fonction pymssql**

```
1 server = "pdcc.c9sqqzd5fulv.us-west-2.rds.amazonaws.com"
2 user = "pdccuser"
3 password = "pdccpass"
4 database = "pdcctest"
5 conn = pymssql.connect(server=server,
6   user=user, password=password, database=database)
```

- **Créer une connexion aux données du serveur SQL en passant les informations d'identification de la base de données à la fonction pymssql**

```
1 server = "pdcc.c9sqqzd5fulv.us-west-2.rds.amazonaws.com"
2 user = "pdccuser"
3 password = "pdccpass"
4 database = "pdcctest"
5 conn = pymssql.connect(server=server,
6   user=user, password=password, database=database)
```

- Créez une **data frame pandas** en passant l'instruction **SELECT** et l'objet de connexion à **read_sql**.
- fermer la connexion.

- **Créer une connexion aux données du serveur SQL en passant les informations d'identification de la base de données à la fonction pymssql**

```
1 server = "pdcc.c9sqqzd5fulv.us-west-2.rds.amazonaws.com"
2 user = "pdccuser"
3 password = "pdccpass"
4 database = "pdcctest"
5 conn = pymssql.connect(server=server,
6   user=user, password=password, database=database)
```

- Créez une **data frame pandas** en passant l'instruction **SELECT** et l'objet de connexion à **read_sql**.
- fermer la connexion.

```
1 studentmath = pd.read_sql(query,conn)
2 conn.close()
3 studentmath.dtypes
4 studentmath.head()
```

- **(Alternative) Utilisez le connecteur mysql et readsql pour obtenir des données de MySQL.**

```
1 host = "pdccmysql.c9sqqzd5fulv.us-west-2.rds.amazonaws.com"
2 user = "pdccuser"
3 password = "pdccpass"
4 database = "pdccschema"
5 connmysql = mysql.connector.connect(host=host,
6 database=database,user=user,password=password)
7 studentmath = pd.read_sql(sqlselect,connmysql)
8 connmysql.close()
```

- **(Alternative) Utilisez le connecteur mysql et readsql pour obtenir des données de MySQL.**

```
1 host = "pdccmysql.c9sqqzd5fulv.us-west-2.rds.amazonaws.com"
2 user = "pdccuser"
3 password = "pdccpass"
4 database = "pdccschema"
5 connmysql = mysql.connector.connect(host=host,
6 database=database,user=user,password=password)
7 studentmath = pd.read_sql(sqlselect,connmysql)
8 connmysql.close()
```

- **Réorganiser les colonnes, définir un index et vérifier les valeurs manquantes :**

- **(Alternative) Utilisez le connecteur mysql et readsql pour obtenir des données de MySQL.**

```
1 host = "pdccmysql.c9sqqzd5fulv.us-west-2.rds.amazonaws.com"
2 user = "pdccuser"
3 password = "pdccpass"
4 database = "pdccschema"
5 connmysql = mysql.connector.connect(host=host,
6 database=database,user=user,password=password)
7 studentmath = pd.read_sql(sqlselect,connmysql)
8 connmysql.close()
```

- **Réorganiser les colonnes, définir un index et vérifier les valeurs manquantes :**
  - Déplacer les données relatives aux notes à gauche du data frame juste après studentid.
  - Déplacer également la colonne freetime vers la droite, après traveltime et studytime.
  - Confirmer que chaque ligne a un ID et que les ID sont uniques.
  - Définir studentid comme index.

```
newcolorder = ['studentid', 'gradeperiod1', 'gradeperiod2',
  'gradeperiod3', 'school', 'sex', 'age', 'famsize',
  'mothereducation', 'fathereducation', 'traveltime',
  'studytime', 'freetime', 'failures', 'famrel',
  'goout']
studentmath = studentmath[newcolorder]
studentmath.studentid.count()
studentmath.studentid.nunique()
studentmath.set_index('studentid', inplace=True)
```

```
newcolorder = ['studentid', 'gradeperiod1', 'gradeperiod2',
  'gradeperiod3', 'school', 'sex', 'age', 'famsize',
  'mothereducation', 'fathereducation', 'traveltime',
  'studytime', 'freetime', 'failures', 'famrel',
  'goout']
studentmath = studentmath[newcolorder]
studentmath.studentid.count()
studentmath.studentid.nunique()
studentmath.set_index('studentid', inplace=True)
```

- **Utiliser la fonction count du data frame pour vérifier les valeurs manquantes :**

```
newcolorder = ['studentid', 'gradeperiod1', 'gradeperiod2',
  'gradeperiod3', 'school', 'sex', 'age', 'famsize',
  'mothereducation', 'fathereducation', 'traveltime',
  'studytime', 'freetime', 'failures', 'famrel',
  'goout']
studentmath = studentmath[newcolorder]
studentmath.studentid.count()
studentmath.studentid.nunique()
studentmath.set_index('studentid', inplace=True)
```

- **Utiliser la fonction count du data frame pour vérifier les valeurs manquantes :**

```
studentmath.count()
```

```
newcolorder = ['studentid', 'gradeperiod1', 'gradeperiod2',
  'gradeperiod3', 'school', 'sex', 'age', 'famsize',
  'mothereducation', 'fathereducation', 'traveltime',
  'studytime', 'freetime', 'failures', 'famrel',
  'goout']
studentmath = studentmath[newcolorder]
studentmath.studentid.count()
studentmath.studentid.nunique()
studentmath.set_index('studentid', inplace=True)
```

- **Utiliser la fonction count du data frame pour vérifier les valeurs manquantes :**

```
studentmath.count()
```

- **Remplacer les valeurs de données codées par des valeurs plus informatives.**
  Créer un dictionnaire pour remplacer les valeurs des colonnes par des valeurs plus significatives, puis utiliser la commande replace pour insérer ces valeurs

```
setvalues={"famrel":{1:"1:very bad",2:"2:bad",3:"3:neutral",
    4:"4:good",5:"5:excellent"},
  "freetime":{1:"1:very low",2:"2:low",3:"3:neutral",
    4:"4:high",5:"5:very high"},
  "goout":{1:"1:very low",2:"2:low",3:"3:neutral",
    4:"4:high",5:"5:very high"},
  "mothereducation":{0:np.nan,1:"1:k-4",2:"2:5-9",
    3:"3:secondary ed",4:"4:higher ed"},
  "fathereducation":{0:np.nan,1:"1:k-4",2:"2:5-9",
    3:"3:secondary ed",4:"4:higher ed"}}

studentmath.replace(setvalues, inplace=True)
```

```
setvalues={"famrel":{1:"1:very bad",2:"2:bad",3:"3:neutral",
    4:"4:good",5:"5:excellent"},
  "freetime":{1:"1:very low",2:"2:low",3:"3:neutral",
    4:"4:high",5:"5:very high"},
  "goout":{1:"1:very low",2:"2:low",3:"3:neutral",
    4:"4:high",5:"5:very high"},
  "mothereducation":{0:np.nan,1:"1:k-4",2:"2:5-9",
    3:"3:secondary ed",4:"4:higher ed"},
  "fathereducation":{0:np.nan,1:"1:k-4",2:"2:5-9",
    3:"3:secondary ed",4:"4:higher ed"}}

studentmath.replace(setvalues, inplace=True)
```

- Changer le type des colonnes dont les données ont été modifiées en catégorie.
- Vérifier si l'utilisation de la mémoire a changé :

```
setvalues={"famrel":{1:"1:very bad",2:"2:bad",3:"3:neutral",
    4:"4:good",5:"5:excellent"},
  "freetime":{1:"1:very low",2:"2:low",3:"3:neutral",
    4:"4:high",5:"5:very high"},
  "goout":{1:"1:very low",2:"2:low",3:"3:neutral",
    4:"4:high",5:"5:very high"},
  "mothereducation":{0:np.nan,1:"1:k-4",2:"2:5-9",
    3:"3:secondary ed",4:"4:higher ed"},
  "fathereducation":{0:np.nan,1:"1:k-4",2:"2:5-9",
    3:"3:secondary ed",4:"4:higher ed"}}

studentmath.replace(setvalues, inplace=True)
```

- Changer le type des colonnes dont les données ont été modifiées en catégorie.
- Vérifier si l'utilisation de la mémoire a changé :

```
    setvalueskeys = [k for k in setvalues]
    studentmath[setvalueskeys].memory_usage(index=False)

    for col in studentmath[setvalueskeys].columns:
    studentmath[col] = studentmath[col].astype('category')
    studentmath[setvalueskeys].memory_usage(index=False)
```

- **Calculer des pourcentages pour les valeurs de la colonne famrel.**
  Exécutez value_counts et définissez normalize sur True pour générer des pourcentages :

- **Calculer des pourcentages pour les valeurs de la colonne famrel.**
  Exécutez value_counts et définissez normalize sur True pour générer des pourcentages :

```
1 studentmath['famrel'].value_counts(sort=False, normalize=True)
2 studentmath[['freetime','goout']].\
3   apply(pd.Series.value_counts, sort=False, normalize=True)
4 studentmath[['mothereducation','fathereducation']].\
5   apply(pd.Series.value_counts, sort=False, normalize=True)
```

- **Calculer des pourcentages pour les valeurs de la colonne famrel.**
  Exécutez value_counts et définissez normalize sur True pour générer des pourcentages :

```
1 studentmath['famrel'].value_counts(sort=False, normalize=True)
2 studentmath[['freetime','goout']].\
3   apply(pd.Series.value_counts, sort=False, normalize=True)
4 studentmath[['mothereducation','fathereducation']].\
5   apply(pd.Series.value_counts, sort=False, normalize=True)
```

- **Utiliser l'option apply pour calculer des pourcentages pour plusieurs colonnes :**

- **Calculer des pourcentages pour les valeurs de la colonne famrel.**
  Exécutez value_counts et définissez normalize sur True pour générer des pourcentages :

```
studentmath['famrel'].value_counts(sort=False, normalize=True)
studentmath[['freetime','goout']].\
  apply(pd.Series.value_counts, sort=False, normalize=True)
studentmath[['mothereducation','fathereducation']].\
  apply(pd.Series.value_counts, sort=False, normalize=True)
```

- **Utiliser l'option apply pour calculer des pourcentages pour plusieurs colonnes :**

```
studentmath[['freetime','goout']].\
  apply(pd.Series.value_counts, sort=False, normalize=True)
studentmath[['mothereducation','fathereducation']].\
  apply(pd.Series.value_counts, sort=False, normalize=True)
```

Réaliser le LAB 2

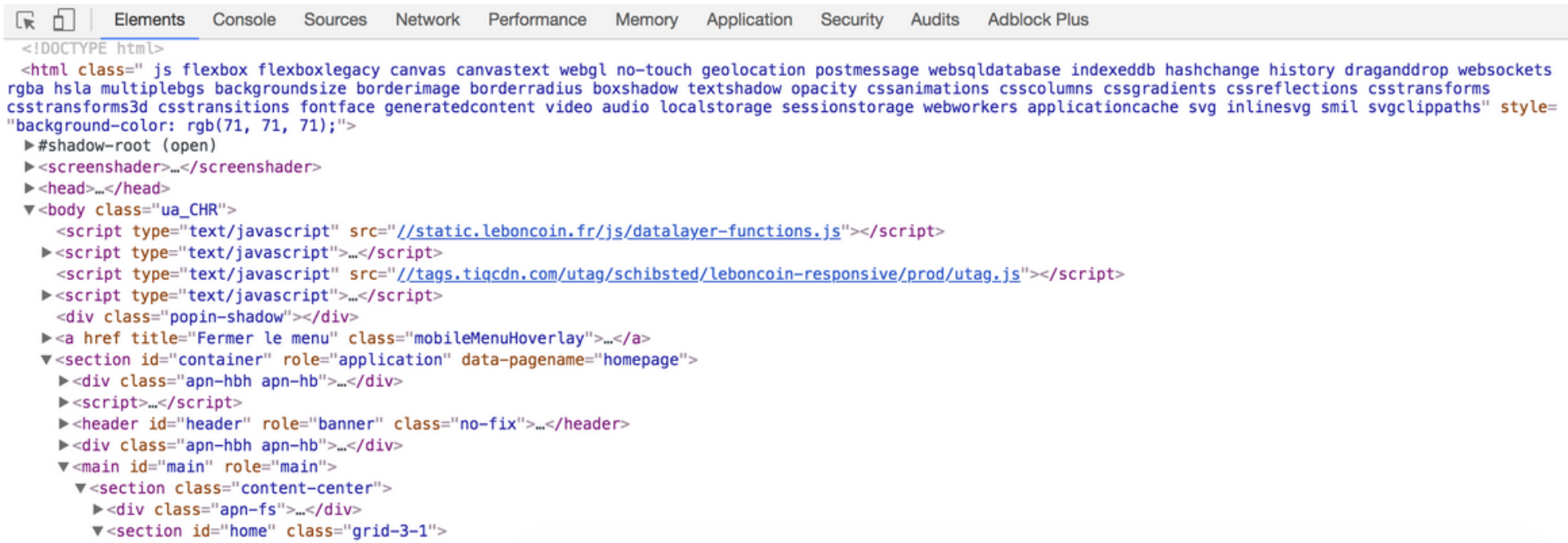
Importation de données à partir de pages Web

- **Le scraping Web** est très utile lorsqu'un site Web contient des données mises à jour régulièrement.
- Nous pouvons ré-exécuter notre code pour générer de nouvelles données chaque fois que la page est mise à jour.
- Il existe deux grandes pratiques pour scraper un site efficacement. Nous allons aborder les deux :
  - **Récupération et parsing du code HTML**. Cette solution nécessite une compréhension du code et des notions basiques de DOM et architecture HTML.
  - **Récupération des appels API** aux serveurs permettant de récupérer les informations directement à la source, la plupart du temps au format JSON. Cette deuxième solution est la plus efficace et facile mais les appels d'API sont souvent cachés ou bloqués.

Dans les deux cas, nous utiliserons des requêtes HTTP et le package **requests**.

Celui-ci permet de faire des requêtes très rapidement et facilement via un interpréteur Python. De nombreux paramètres sont modifiables.

Pour effectuer ces opérations, il est recommandé d'utiliser les outils de développement intégrés à Chrome ou Firefox. Je recommande particulièrement celui de Google, qui est plus intuitif. Bien qu'il existe plusieurs raccourcis clavier, la méthode la plus simple consiste à faire un clic droit et à sélectionner "**Inspecter**".



Deux onglets sont importants dans notre cas :

- **Element** : la partie correspondant au code HTML : elle permet de visualiser la structure et repérer les pointeurs des balises qui encapsulent nos données.
- **Network** : cette partie permet d'analyser tous les appels réseaux réalisés depuis le front. C'est ici que les appels de récupération de données sont effectués.

- **Récupération des données**

  Nombreuses librairies en python ont été développée dans ce sens :

  - **Requests** (+ requests_cache qui fait gagner un temps précieux lors des développements)
  - **BeautifulSoup** (permet de créer un objet python à partir de code HTML pour faciliter l'extraction)
  - **Readability** (permet de récupérer le texte pertinent d'une page web)
  - **Scrapy** (Framework de Spider permettant de gérer le Crawling et le Scraping de site web très efficacement)

- **Stockage**

  Un grand nombre de bases de données sont disponibles sur le marché. Dans le contexte du web, les bases **noSQL** sont très recherchées et demandées, à cause de leur structure très flexible et optimisée pour le stockage de données hétérogènes. Quelques exemples de bases de données :

  - **MySQL** (Base de données relationnelles, stockage sous forme de tableau, optimale pour le stockage de chiffres)
  - **Cassandra** (Base de données noSQL developpée pour garantir une scalabilité et intégrité d'un grand nombre de données)
  - **MongoDB** (Base de données noSQL, stockage sous la forme de document JSON)
  - **Redis** (Base de données noSQL, stockage sous la forme de données clé :valeurs avec très grandes performances).

- **Les meilleures bibliothèques Python pour le web scraping :**
  - **requests** : vous permet d'exécuter des requêtes HTTP en Python.
    Un requête HTTP est une requête basé sur le protocole TCP. Elle permet d'accéder aux données mise à disposition sur une adresse IP (ou url résolue par un DNS) et un port.

    Il existe de nombreux types de requêtes selon la convention REST :
    - GET
    - POST
    - PUT
    - DELETE
    - UPDATE.

    Dans notre cas nous allons utiliser la plupart du temps des **GET** et potentiellement des **POST**.
    - Le **GET** permet comme sont nom l'indique de récupérer des informations en fonction de certain paramètres.
    - Le **POST** nécéssite un envoie de données pour récupérer des données.

    Ces requêtes encapsulent un certain nombre de paramètres qui permettent soient d'identifier une provenance et un utilisateur ou de réaliser différentes actions. Vous pouvez installer des requêtes avec la commande :

```
1 pip install requests
```

Exploitation du HTML

- **Beautiful Soup**
  - La bibliothèque Python **Beautiful Soup** facilite l'extraction d'informations sur des pages web.
  - **Beautiful Soup** fonctionne avec n'importe quel analyseur HTML ou XML et vous fournit tout ce dont vous avez besoin pour vos instructions d'itération, de recherche et de modification sur l'arborescence d'analyse.

Notez que vous pouvez utiliser **Beautiful Soup** avec html.parser, interpréteur fourni avec la bibliothèque standard de Python, qui vous permet d'analyser des fichiers texte HTML. En particulier, **Beautiful Soup** peut vous aider à parcourir le DOM et à en extraire les données dont vous avez besoin.
Vous pouvez installer **Beautiful Soup** avec le pip de la manière suivante :

```
1 pip install beautifulsoup4
```

Connexion à l'URL cible

- La première chose que vous devez faire dans un web scraper est de vous connecter à votre site cible.
- Tout d'abord, récupérez l'URL complète de la page cible sur votre navigateur.
- Assurez-vous de copier également la partie http :// ou https :// du protocole HTTP.
- Dans notre cas, il s'agit de l'URL complète du site cible :

```
1 https://quotes.toscrape.com
```

Connexion à l'URL cible

- La première chose que vous devez faire dans un web scraper est de vous connecter à votre site cible.
- Tout d'abord, récupérez l'URL complète de la page cible sur votre navigateur.
- Assurez-vous de copier également la partie http :// ou https :// du protocole HTTP.
- Dans notre cas, il s'agit de l'URL complète du site cible :

```
1 https://quotes.toscrape.com
```

Maintenant, vous pouvez utiliser Requests pour télécharger une page web avec la ligne de code suivante :

```
1 url='https://quotes.toscrape.com'
2 response = requests.get(url)
3 statut=reponse.status_code
```

Cette ligne affecte simplement le résultat de la méthode **request.get()** à la page des variables. En arrière-plan, request.get() exécute une requête GET en utilisant l'URL transmise comme paramètre. Il renvoie ensuite un objet **response** contenant la réponse du serveur à la requête HTTP.

Analyse du contenu HTML

Pour analyser le document HTML renvoyé par le serveur suite à la requête GET, passez **response.text** au constructeur BeautifulSoup() pour extraire les données de la page web :

```
1 import requests
2 from bs4 import BeautifulSoup
3 soup = BeautifulSoup(response.text, 'html.parser')
```

Analyse du contenu HTML

Pour analyser le document HTML renvoyé par le serveur suite à la requête GET, passez **response.text** au constructeur BeautifulSoup() pour extraire les données de la page web :

```
1 import requests
2 from bs4 import BeautifulSoup
3 soup = BeautifulSoup(response.text, 'html.parser')
```

- Le second paramètre spécifie l'interpréteur que **Beautiful Soup** utilisera.
- La variable soup contient maintenant un objet **BeautifulSoup**. Il s'agit d'une structure arborescente générée à partir de l'analyse syntaxique du document HTML contenu dans **response.text** avec l'interpréteur **html.parser** intégré dans Python.

Il se peut qu'un message d'erreur arrive à ce point là si vous n'avez pas la librairie lxml installée, pour se faire vous avez juste à lancer la commande suivante :

```
1 pip install lxml
```

Sélection d'éléments HTML avec Beautiful Soup

- Beautiful Soup propose différentes approches pour sélectionner des éléments :
  - find() : renvoie le premier élément HTML correspondant à la stratégie de sélecteur d'entrée, le cas échéant.
  - find_all() : renvoie une liste des éléments HTML correspondant à la condition de sélecteur passée par paramètre.
- En fonction des paramètres fournis à ces deux méthodes, celles-ci rechercheront des éléments sur la page de différentes manières. Plus précisément, vous pouvez sélectionner des éléments HTML :
  - Par Balise

```
# get all <h1> elements on the page
h1_elements = soup.find_all('h1')
```

  - Par Id

```
# get the element with id="main-title"
main_title_element = soup.find(id='main-title')
```

- Par texte :

```
# find the footer element based on the text it contains
footer_element = soup.find(text={'Powered by WordPress'})
```

- Par attribut :

```
# find the email input element through its "name" attribute
email_element = soup.find(attrs={'name': 'email'})
```

- Par classe :

```
# find all the centered elements on the page
centered_element = soup.find_all(class_='text-center')
```

- Par texte :

```
# find the footer element based on the text it contains
footer_element = soup.find(text={'Powered by WordPress'})
```

- Par attribut :

```
# find the email input element through its "name" attribute
email_element = soup.find(attrs={'name': 'email'})
```

- Par classe :

```
# find all the centered elements on the page
centered_element = soup.find_all(class_='text-center')
```

En concaténant ces méthodes, vous pouvez extraire n'importe quel élément HTML de la page. Observez l'exemple ci-dessous :

```
# get all "li" elements in the ".navbar" element
soup.find(class_='navbar').find_all('li')
```

Pour faciliter les choses, **Beautiful Soup** utilise également la méthode **select()**. Cela vous permet d'appliquer directement un sélecteur **CSS** :

```
1 # get all "li" elements
2 # in the ".navbar" element
3 soup.select('.navbar > li')
```

Pour faciliter les choses, **Beautiful Soup** utilise également la méthode **select()**. Cela vous permet d'appliquer directement un sélecteur **CSS** :

```
1 # get all "li" elements
2 # in the ".navbar" element
3 soup.select('.navbar > li')
```

Important à apprendre :

- Identifier les éléments HTML qui vous intéressent.
- En particulier, vous devez définir une stratégie de sélection pour les éléments qui contiennent les données que vous souhaitez extraire.
- Vous pouvez y parvenir en utilisant les outils de développement proposés par votre navigateur. Dans Chrome, cliquez avec le bouton droit de la souris sur l'élément HTML souhaité et sélectionnez l'option « Inspect ».

• **Exploitation des appels d'API**

Lorsque le front du site récupère des données sur une API géré par le back, un appel d'API est réalisé. Cet appel est recensé dans les appels réseaux. Il est alors possible de re-jouer cet appel pour récupérer à nouveau les données. Il est très facile de récupérer ces appels dans l'onglet Network de la console développeur de Chrome ou FireFox. La console vous permet de copier le code CURL pour effectuée et vous pouvez ensuite la transformer en code Python depuis le site https://curl.trillworks.com/.

Exemple d'importation de données à partir de pages Web

- Nous allons extraire des données d'une page web.
- Trouver le tableau suivant dans cette page, et le charger dans un DataFrame pandas

Exemple d'importation de données à partir de pages Web

- Nous allons extraire des données d'une page web.
- Trouver le tableau suivant dans cette page, et le charger dans un DataFrame pandas

Country	Cases	Deaths	Cases per Million	Deaths per Million	population	population_density	median_age	gdp_per_capita	hospital_beds_per_100k
Algeria	9,394	653	214	15	43,851,043	17	29	13,914	1.9
Austria	16,642	668	1848	74	9,006,400	107	44	45,437	7.4
Bangladesh	47,153	650	286	4	164,689,383	1265	28	3,524	0.8
Belgium	58,381	9467	5037	817	11,589,616	376	42	42,659	5.6
Brazil	514,849	29314	2422	138	212,559,409	25	34	14,103	2.2
Canada	90,936	7295	2409	193	37,742,157	4	41	44,018	2.5

TABLE – Données COVID-19 de 6 pays

- Importation des bibliothèques **pprint, requests, et BeautifulSoup**

```
1 import pandas as pd
2 import numpy as np
3 import json
4 import pprint
5 import requests
6 from bs4 import BeautifulSoup
```

- Analyse de la page Web et l'obtention de la ligne d'en-tête du tableau
  - Utiliser la méthode **find** de **Beautiful Soup** pour obtenir la table souhaitée,
  - puis utiliser la méthode **find_all** pour récupérer les éléments imbriqués dans les éléments pour ce tableau.
  - Créez une liste d'étiquettes de colonne basée sur le texte des lignes :

```
1 webpage = requests.get("http://www.alrb.org/datacleaning/covidcaseoutliers.html")
2 bs = BeautifulSoup(webpage.text, 'html.parser')
3 theadrows = bs.find('table', {'id':'tblDeaths'}).
4 thead.find_all('th')
```

- Obtenez les données des cellules du tableau.

```
rows = bs.find('table', {'id':'tblDeaths'}).tbody.find_all('tr')
datarows = []
labelrows = []
for row in rows:
    rowlabels = row.find('th').get_text()
    cells = row.find_all('td', {'class':'data'})
    if (len(rowlabels)>3):
       labelrows.append(rowlabels)
    if (len(cells)>0):
        cellvalues = [j.get_text() for j in cells]
        datarows.append(cellvalues
pprint.pprint(datarows[0:2])
pprint.pprint(labelrows[0:2])
for i in range(len(datarows)):
        datarows[i].insert(0, labelrows[i])
pprint.pprint(datarows[0:1])
```

- Chargement des données dans pandas

```
totaldeaths = pd.DataFrame(datarows, columns=labelcols)
totaldeaths.head()
totaldeaths.dtypes
```

- Correction des noms des colonnes et conversion des données en valeurs numériques
  - Supprimer les espaces des noms de colonnes.
  - Supprimer toutes les données non numériques des premières colonnes contenant des données, y compris les virgules
  - Convertir en valeurs numériques, sauf pour la colonne rowheadings

```
        totaldeaths.columns = totaldeaths.columns.str.replace(" ", "_").str.lower()
for col in totaldeaths.columns[1:-1]:
        totaldeaths[col] = totaldeaths[col].str.replace("[^0-9]","").astype('int64')
totaldeaths['hospital_beds_per_100k'] =
↪  totaldeaths['hospital_beds_per_100k'].astype('float')
totaldeaths.head()
totaldeaths.dtypes
```

Réaliser le LAB 3

Réaliser le LAB 4

Web Scraping with WebDriver

Nettoyage des données–Data Cleaning

- Le nettoyage des données est une étape cruciale dans tout projet de Machine Learning.
- Il existe des opérations de nettoyage de données très basiques que vous devriez probablement effectuer sur chaque projet d'apprentissage automatique.

Nettoyage des données–Data Cleaning

- Le nettoyage des données est une étape cruciale dans tout projet de Machine Learning.
- Il existe des opérations de nettoyage de données très basiques que vous devriez probablement effectuer sur chaque projet d'apprentissage automatique.

1. Identifier les colonnes qui contiennent une seule valeur
2. Supprimer les colonnes qui contiennent une seule valeur
3. Considérer les colonnes qui ont très peu de valeurs
4. Supprimez les colonnes qui ont une faible variance
5. Identifier les lignes qui contiennent des données en double
6. Supprimer les lignes qui contiennent des données en double

Nettoyage des Données : Valeurs Uniques et Faible Variance

- Lors du nettoyage des données, il est essentiel de comprendre la distribution des variables.
- **Valeurs uniques** : Colonnes avec très peu de valeurs distinctes.
  Ici, une valeur unique signifie que chaque ligne de cette colonne a la même valeur.
- **Faible variance** : Colonnes où la majorité des valeurs sont identiques ou proches.

Pourquoi C'est Important ?

- **Valeurs uniques** :
  - Colonnes qui ont une seule observation ou valeur sont probablement inutiles pour la modélisation.
  - Lorsqu'un prédicteur contient une seule valeur, il est appelé prédicteur à variance nulle car il n'y a aucune variation affichée par le prédicteur.
  - Risquent d'introduire du bruit dans les modèles d'apprentissage automatique.
- **Peu de valeurs uniques** :
  - Un pourcentage très faible signifie que la plupart des valeurs se répètent.
  - Si une colonne contient un très faible pourcentage de valeurs uniques, elle pourrait ne pas contribuer à l'analyse.
- **Faible variance** :
  - Colonnes avec une faible variation peuvent ne pas contribuer à la différenciation des données.
  - Elles risquent de ne pas apporter de valeur dans les modèles

Identification des Valeurs Uniques avec Pandas

Exemple de code en Python :

```
1  import pandas as pd
2
3  # Chargement des données
4  df = pd.read_csv("data.csv")
5
6  # Identification des colonnes ayant une seule valeur unique
7  colonnes_uniques = df.columns[df.nunique() == 1]
8  print(colonnes_uniques)
```

- **Explication** : `nunique()` compte le nombre de valeurs uniques dans chaque colonne.
- Si une colonne contient une seule valeur unique, elle peut être ignorée pour l'analyse.

Exemple de Détection des Valeurs Uniques

Dataset :

ID	Nom	Age	Ville
1	Alice	30	Paris
2	Bob	25	Paris
3	Carol	28	Paris
4	Dave	22	Paris

Sortie du code : La colonne `Ville` a une seule valeur "Paris", elle pourrait donc être supprimée.

Identifier les Colonnes avec peu de Valeurs Uniques

Exemple de code en Python :

```
1  # Calculer le pourcentage de valeurs uniques
2  pourcentage_uniques = df.nunique() / df.shape[0] * 100
3
4  # Seuil pour faible pourcentage de valeurs uniques (ex: moins de 10%)
5  seuil_bas = 10
6
7  # Colonnes avec un faible pourcentage de valeurs uniques
8  colonnes_bas_pourcentage = pourcentage_uniques[pourcentage_uniques < seuil_bas].index
9
10 print(colonnes_bas_pourcentage)
```

- **Explication** : Les colonnes avec un faible pourcentage de valeurs uniques peuvent être ignorées si elles n'apportent pas suffisamment de diversité à l'analyse.

Exemple de Détection - Peu de Valeurs Uniques

Dataset :

ID	Couleur	Ville	Catégorie
1	Rouge	Paris	A
2	Rouge	Lyon	A
3	Rouge	Paris	B
4	Bleu	Paris	A
5	Rouge	Lyon	A
6	Rouge	Paris	A

Sortie du code :

- La colonne `Couleur` a un faible pourcentage de valeurs uniques : 2 couleurs ("Rouge" et "Bleu") pour 6 lignes. Cela représente un pourcentage unique de 33%.
- La colonne `Catégorie` a un pourcentage de valeurs uniques de 2 valeurs distinctes ("A" et "B") sur 6 lignes, soit 33%.
- Ces colonnes peuvent être ignorées si elles ne différencient pas suffisamment les données.

Pourquoi Supprimer ces Colonnes ?

- Les colonnes avec peu de diversité dans les valeurs uniques peuvent introduire du bruit sans vraiment enrichir les informations utiles dans un modèle.
- Dans l'exemple ci-dessus, la colonne `Couleur` contient principalement la valeur "Rouge", donc elle n'apporte pas de variabilité significative.
- La colonne `Catégorie` pourrait être ignorée si "A" domine les autres catégories.

Identifier les Colonnes à Faible Variance

Exemple de code en Python :

```
1  # Calcul de la variance des colonnes numériques
2  variance = df.var()
3
4  # Seuil pour faible variance (ex: moins de 0.01)
5  seuil_variance = 0.01
6  colonnes_faible_variance = variance[variance < seuil_variance].index
7  print(colonnes_faible_variance)
```

- **Explication** : Une variance proche de 0 indique une faible variation dans la colonne.

Exemple de Détection de Faible Variance

Dataset :

ID	Score_Test	Age	Hauteur
1	0.99	30	1.70
2	0.98	25	1.69
3	0.99	28	1.70
4	0.99	22	1.69

Sortie du code : La colonne `Score_Test` a une variance très faible (0.0001), donc elle pourrait être supprimée.

Utilisation de VarianceThreshold avec scikit-learn

```python
from sklearn.feature_selection import VarianceThreshold

# Chargement des données (exemple)
import pandas as pd
df = pd.DataFrame({
    'A': [1, 1, 1, 1, 1],
    'B': [0, 0, 1, 1, 1],
    'C': [0, 1, 2, 3, 4],
    'D': [0.1, 0.1, 0.1, 0.1, 0.1]
})

# Appliquer le filtre de VarianceThreshold
seuil = 0.1  # Seuil de variance
selector = VarianceThreshold(threshold=seuil)

# Ajuster le sélecteur sur les données
df_reduit = selector.fit_transform(df)

# Colonnes conservées
print(selector.get_support(indices=True))
```

- **Explication** : `VarianceThreshold` supprime les colonnes dont la variance est inférieure au seuil défini.

Exemple de Dataset

#	A	B	C	D
1	1	0	0	0.1
2	1	0	1	0.1
3	1	1	2	0.1
4	1	1	3	0.1
5	1	1	4	0.1

Exemple de Dataset

#	A	B	C	D
1	1	0	0	0.1
2	1	0	1	0.1
3	1	1	2	0.1
4	1	1	3	0.1
5	1	1	4	0.1

- **Interprétation des résultats**

Exemple de Dataset

#	A	B	C	D
1	1	0	0	0.1
2	1	0	1	0.1
3	1	1	2	0.1
4	1	1	3	0.1
5	1	1	4	0.1

- **Interprétation des résultats**
  - La colonne `A` a une variance nulle (toutes les valeurs sont identiques) et sera supprimée.
  - La colonne `D` a également une variance très faible (0.1) et sera supprimée si elle est en dessous du seuil de 0.1.
  - Les colonnes `B` et `C` ont une variance plus élevée et seront conservées.

Exemple de Dataset

#	A	B	C	D
1	1	0	0	0.1
2	1	0	1	0.1
3	1	1	2	0.1
4	1	1	3	0.1
5	1	1	4	0.1

- **Interprétation des résultats**
  - La colonne `A` a une variance nulle (toutes les valeurs sont identiques) et sera supprimée.
  - La colonne `D` a également une variance très faible (0.1) et sera supprimée si elle est en dessous du seuil de 0.1.
  - Les colonnes `B` et `C` ont une variance plus élevée et seront conservées.
- **Pourquoi Utiliser `VarianceThreshold` ?**

Exemple de Dataset

#	A	B	C	D
1	1	0	0	0.1
2	1	0	1	0.1
3	1	1	2	0.1
4	1	1	3	0.1
5	1	1	4	0.1

- **Interprétation des résultats**
  - La colonne `A` a une variance nulle (toutes les valeurs sont identiques) et sera supprimée.
  - La colonne `D` a également une variance très faible (0.1) et sera supprimée si elle est en dessous du seuil de 0.1.
  - Les colonnes `B` et `C` ont une variance plus élevée et seront conservées.
- **Pourquoi Utiliser `VarianceThreshold` ?**
  - Automatisation : L'utilisation de `VarianceThreshold` permet de filtrer automatiquement les colonnes sans les évaluer manuellement.
  - Flexibilité : Vous pouvez ajuster le seuil de variance selon les besoins de votre projet.
  - Réduction de la dimensionnalité : En supprimant les colonnes inutiles, vous réduisez la taille de votre jeu de données, ce qui améliore l'efficacité des modèles.

Que Faire Après l'Identification ?

- **Valeurs uniques** :
  - Supprimer les colonnes ou les ignorer lors de l'analyse.
- **Pourcentage de valeurs uniques** :
  - Supprimer les colonnes avec un pourcentage trop bas de valeurs uniques, si elles n'apportent pas de valeur informative.
- **Faible variance** :
  - Envisager de supprimer les colonnes à faible variance, car elles risquent d'introduire du bruit.

Réaliser le LAB 5

Exemple : **Oil Spill Dataset :**

Nettoyage des données–Data Cleaning

Exemple : **Oil Spill Dataset :**

- C'est un ensemble de données sur la détection des déversements d'hydrocarbures contient des images jpg extraites de données satellitaires de radar à ouverture synthétique (SAR)
- La tâche consiste à prédire si le patch contient ou non un déversement d'hydrocarbures, par ex. du déversement illégal ou accidentel de pétrole dans l'océan.
- Il y a 937 cas. Chaque cas est composé de 48 caractéristiques dérivées de la vision numérique par ordinateur, d'un numéro de patch et d'une étiquette de classe.
- Le cas normal est l'absence de déversement d'hydrocarbures avec l'étiquette de classe 0, alors qu'un déversement d'hydrocarbures est indiqué par une étiquette de classe 1.
- Il y a 896 cas d'absence de déversement d'hydrocarbures et 41 cas de déversement d'hydrocarbures.

Ce dataset contient des colonnes avec très peu de valeurs uniques qui fournissent une bonne base pour le nettoyage des données.

Identification et Suppression des Doublons dans un Dataset

- Doublons : lignes avec les mêmes valeurs dans toutes les colonnes ou dans un sous-ensemble de colonnes.
- Importance :
  - Les lignes qui contiennent des données identiques peuvent être inutiles au processus de modélisation, voire dangereusement trompeuses lors de l'évaluation du modèle.
  - Biais dans les analyses statistiques
  - Perturbation des modèles de Machine Learning.

Chargement des données dans Pandas

Exemple de dataset :

```
1  import pandas as pd
2
3  data = {'Nom': ['Alice', 'Bob', 'Alice', 'David', 'Alice'],
4          'Âge': [25, 30, 25, 40, 25],
5          'Ville': ['Paris', 'Lyon', 'Paris', 'Marseille', 'Paris']}
6  df = pd.DataFrame(data)
7
8  print(df)
```

Résultat :

```
1        Nom   Âge       Ville
2  0   Alice    25       Paris
3  1     Bob    30        Lyon
4  2   Alice    25       Paris
5  3   David    40  Marseille
6  4   Alice    25       Paris
```

Identification des doublons

Méthode Pandas : `duplicated()`

Identifie si une ligne est un doublon.

```
# Identifier les doublons
doublons = df.duplicated()
print(doublons)
```

Résultat :

```
0    False
1    False
2     True
3    False
4     True
```

Affichage des doublons

Pour afficher uniquement les lignes dupliquées :

```
# Afficher les lignes dupliquées
df_doublons = df[df.duplicated()]
print(df_doublons)
```

Résultat :

```
      Nom   Âge     Ville
2   Alice    25     Paris
4   Alice    25     Paris
```

Suppression des doublons

Méthode Pandas : `drop_duplicates()`

Supprime les lignes en double, en gardant la première occurrence.

```
1 # Supprimer les doublons
2 df_sans_doublons = df.drop_duplicates()
3 print(df_sans_doublons)
```

Résultat :

```
1       Nom   Âge       Ville
2 0   Alice    25       Paris
3 1     Bob    30        Lyon
4 3   David    40  Marseille
```

Suppression basée sur des colonnes spécifiques

Vous pouvez spécifier les colonnes à considérer pour la suppression des doublons :

```
# Supprimer les doublons basés sur 'Nom'
df_sans_doublons_col = df.drop_duplicates(subset=['Nom'])
print(df_sans_doublons_col)
```

Résultat :

```
      Nom   Âge       Ville
0   Alice    25       Paris
1     Bob    30        Lyon
3   David    40  Marseille
```

Gestion avancée des doublons

Options supplémentaires :

- Conserver la dernière occurrence :

```
1 df_last = df.drop_duplicates(keep='last')
```

- Supprimer tous les doublons (sans garder la première ou la dernière) :

```
1 df_none = df.drop_duplicates(keep=False)
```

Conclusion

- Un dataset sans doublons est plus propre et produit des résultats d'analyse plus fiables.
- **Bonnes pratiques** : Toujours vérifier les doublons avant d'analyser ou d'entraîner un modèle.

Réaliser le LAB 6

Exemple : **Iris Dataset :**

Identification et suppression des valeurs aberrantes : Outlier Identification

- Lors de la modélisation, il est important de nettoyer l'échantillon de données pour s'assurer que les observations représentent au mieux le problème.
- Parfois, un ensemble de données peut contenir des valeurs extrêmes qui se situent en dehors de l'éventail de ce qui est attendu et qui sont différentes des autres données.
- **Ces valeurs sont appelées valeurs aberrantes.**

Identification et suppression des valeurs aberrantes : Outlier Identification

- Lors de la modélisation, il est important de nettoyer l'échantillon de données pour s'assurer que les observations représentent au mieux le problème.
- Parfois, un ensemble de données peut contenir des valeurs extrêmes qui se situent en dehors de l'éventail de ce qui est attendu et qui sont différentes des autres données.
- **Ces valeurs sont appelées valeurs aberrantes.**

Dans ce tutoriel, vous découvrirez les valeurs aberrantes et comment les identifier et les supprimer de vos données d'apprentissage automatique.

Identification et suppression des valeurs aberrantes : Outlier Identification

- Lors de la modélisation, il est important de nettoyer l'échantillon de données pour s'assurer que les observations représentent au mieux le problème.
- Parfois, un ensemble de données peut contenir des valeurs extrêmes qui se situent en dehors de l'éventail de ce qui est attendu et qui sont différentes des autres données.
- **Ces valeurs sont appelées valeurs aberrantes.**

Dans ce tutoriel, vous découvrirez les valeurs aberrantes et comment les identifier et les supprimer de vos données d'apprentissage automatique.

- Comment utiliser des statistiques univariées simples telles que l'écart type et l'écart interquartile pour identifier et supprimer les observations aberrantes d'un échantillon de données.
- Comment utiliser un modèle de détection des valeurs aberrantes pour identifier et supprimer les lignes d'un ensemble de données d'entraînement afin d'améliorer la modélisation prédictive.

Que sont les valeurs aberrantes ?

Défiition

Une valeur aberrante est une valeur qui diffère de façon significative de la tendance globale des autres observations quand on observe un ensemble de données ayant des caractéristiques communes.

Que sont les valeurs aberrantes ?

Défiition

Une valeur aberrante est une valeur qui diffère de façon significative de la tendance globale des autres observations quand on observe un ensemble de données ayant des caractéristiques communes.

Les valeurs aberrantes peuvent avoir de nombreuses causes, telles que :

Que sont les valeurs aberrantes ?

Défiition

Une valeur aberrante est une valeur qui diffère de façon significative de la tendance globale des autres observations quand on observe un ensemble de données ayant des caractéristiques communes.

Les valeurs aberrantes peuvent avoir de nombreuses causes, telles que :

- Une erreur de mesure ou de saisie.
- Corruption des données.
- Une véritable observation aberrante.

Que sont les valeurs aberrantes ?

Défiition

Une valeur aberrante est une valeur qui diffère de façon significative de la tendance globale des autres observations quand on observe un ensemble de données ayant des caractéristiques communes.

Les valeurs aberrantes peuvent avoir de nombreuses causes, telles que :

- Une erreur de mesure ou de saisie.
- Corruption des données.
- Une véritable observation aberrante.

Il n'existe pas de méthode précise pour définir et identifier les valeurs aberrantes en général, en raison des spécificités de chaque ensemble de données.
Au lieu de cela, vous devez interpréter les observations brutes et décider si une valeur est aberrante ou non.
L'identification des valeurs aberrantes et des mauvaises données dans votre ensemble de données est probablement l'une des parties les plus difficiles du nettoyage des données, il s'agit toujours d'un sujet à explorer avec prudence.

Valeurs aberrantes ou non ?

- Les valeurs identifiées ne signifie pas toujours que sont des valeurs aberrantes et doivent être supprimées.
- Soit $x(1)$, $x(2)$,..., $x(n)$ les données ordonnées dans l'ordre croissant. Les valeurs $x(1)$ et $x(n)$ sont respectivement l'observation extrême inférieure et supérieure :

Valeurs aberrantes ou non ?

- Les valeurs identifiées ne signifie pas toujours que sont des valeurs aberrantes et doivent être supprimées.
- Soit $x(1)$, $x(2)$,..., $x(n)$ les données ordonnées dans l'ordre croissant. Les valeurs $x(1)$ et $x(n)$ sont respectivement l'observation extrême inférieure et supérieure :
  - Les valeurs extrêmes peuvent être ou ne pas être des valeurs aberrantes.

Valeurs aberrantes ou non ?

- Les valeurs identifiées ne signifie pas toujours que sont des valeurs aberrantes et doivent être supprimées.
- Soit $x(1)$, $x(2)$,..., $x(n)$ les données ordonnées dans l'ordre croissant. Les valeurs $x(1)$ et $x(n)$ sont respectivement l'observation extrême inférieure et supérieure :
  - Les valeurs extrêmes peuvent être ou ne pas être des valeurs aberrantes.
  - Une valeur aberrante est toujours une valeur extrême de l'échantillon.

Valeurs aberrantes ou non ?

- Les valeurs identifiées ne signifie pas toujours que sont des valeurs aberrantes et doivent être supprimées.
- Soit $x(1)$, $x(2)$,..., $x(n)$ les données ordonnées dans l'ordre croissant. Les valeurs $x(1)$ et $x(n)$ sont respectivement l'observation extrême inférieure et supérieure :
  - Les valeurs extrêmes peuvent être ou ne pas être des valeurs aberrantes.
  - Une valeur aberrante est toujours une valeur extrême de l'échantillon.
- Des outils qu'on va décrire un peu plus loin peuvent être utiles pour mettre en lumière des événements rares qui peuvent nécessiter un second regard.
- Un bon conseil est d'envisager de tracer les valeurs aberrantes identifiées pour voir comment elles sont perçues dans le contexte des valeurs non aberrantes.
- Si c'est le cas, peut-être qu'elles ne sont pas aberrantes et peuvent être expliquées.

Garder ou non un Outlier ?

Garder ou non un Outlier ?

- C'est la question à 1 million d'euros !

Garder ou non un Outlier ?

- C'est la question à 1 million d'euros ! Pourtant la réponse relève du bon sens bien qu'elle ne soit pas systématique.

Garder ou non un Outlier ?

- C'est la question à 1 million d'euros ! Pourtant la réponse relève du bon sens bien qu'elle ne soit pas systématique.
- Pour vous simplifier la décision, **retenez ceci :**
- Si votre modèle prédictif est **sensible aux outliers**, et que vous souhaitez découvrir une pattern ou une fonction de prédiction, garder les données aberrantes biaisera votre modèle.

Garder ou non un Outlier ?

- C'est la question à 1 million d'euros ! Pourtant la réponse relève du bon sens bien qu'elle ne soit pas systématique.
- Pour vous simplifier la décision, **retenez ceci :**
- Si votre modèle prédictif est **sensible aux outliers**, et que vous souhaitez découvrir une pattern ou une fonction de prédiction, garder les données aberrantes biaisera votre modèle.

  La conséquence c'est que vous n'aurez pas un modèle optimal.
- Dans ce cas, sans hésiter, supprimez ces valeurs !

Garder ou non un Outlier ?

- C'est la question à 1 million d'euros ! Pourtant la réponse relève du bon sens bien qu'elle ne soit pas systématique.
- Pour vous simplifier la décision, **retenez ceci :**
- Si votre modèle prédictif est **sensible aux outliers**, et que vous souhaitez découvrir une pattern ou une fonction de prédiction, garder les données aberrantes biaisera votre modèle.

  La conséquence c'est que vous n'aurez pas un modèle optimal.

- Dans ce cas, sans hésiter, supprimez ces valeurs !
- Par ailleurs, **si vous devez détecter des comportements anormaux comme des mouvements bancaires frauduleux ou douteux**, ces outliers vous seront utiles. En effet :

Garder ou non un Outlier ?

- C'est la question à 1 million d'euros ! Pourtant la réponse relève du bon sens bien qu'elle ne soit pas systématique.
- Pour vous simplifier la décision, **retenez ceci :**
- Si votre modèle prédictif est **sensible aux outliers**, et que vous souhaitez découvrir une pattern ou une fonction de prédiction, garder les données aberrantes biaisera votre modèle.

La conséquence c'est que vous n'aurez pas un modèle optimal.

- Dans ce cas, sans hésiter, supprimez ces valeurs !
- Par ailleurs, **si vous devez détecter des comportements anormaux comme des mouvements bancaires frauduleux ou douteux**, ces outliers vous seront utiles. En effet :

pour qu'un algorithme apprenne la notion de comportement anormal, il faudra qu'il les observent !

Garder ou non un Outlier ?

- C'est la question à 1 million d'euros ! Pourtant la réponse relève du bon sens bien qu'elle ne soit pas systématique.
- Pour vous simplifier la décision, **retenez ceci :**
- Si votre modèle prédictif est **sensible aux outliers**, et que vous souhaitez découvrir une pattern ou une fonction de prédiction, garder les données aberrantes biaisera votre modèle.

La conséquence c'est que vous n'aurez pas un modèle optimal.

- Dans ce cas, sans hésiter, supprimez ces valeurs !
- Par ailleurs, **si vous devez détecter des comportements anormaux comme des mouvements bancaires frauduleux ou douteux**, ces outliers vous seront utiles. En effet :

pour qu'un algorithme apprenne la notion de comportement anormal, il faudra qu'il les observent !

Conclusion

- Approcher les données aberrantes n'est pas toujours évident.
- **Il n'y a pas d'approche systématique pour les gérer.**
- Il faut comprendre le contexte métier dans lequel elles se présentent.
- **Cette compréhension du contexte métier est toujours souhaitable pour tout Data Scientist qui se respecte.**

Test Dataset

- Avant d'examiner les **méthodes d'identification des valeurs aberrantes**, définissons un ensemble de données que nous pouvons utiliser pour tester ces méthodes.
- Nous allons générer une population de 10 000 nombres aléatoires tirés d'une distribution gaussienne avec une moyenne de 50 et un écart type de 5.
- Les nombres tirés d'une distribution gaussienne auront des valeurs aberrantes. C'est-à-dire qu'en vertu de la distribution elle-même, il y aura quelques valeurs qui seront très éloignées de la moyenne.
- Nous utiliserons la fonction **randn()** pour générer des valeurs gaussiennes aléatoires avec une moyenne de 0 et un écart type de 1.
- Puis nous multiplierons les résultats par notre propre écart-type et nous ajouterons la moyenne pour décaler les valeurs dans la plage des valeurs extrêmes.
- Le générateur de nombres pseudo-aléatoires **est ensemencé** pour s'assurer que nous obtenons le même échantillon de nombres à chaque fois que le code est exécuté.

Méthodes pour identifier les valeurs aberrantes

```
1  # generate gaussian data
2  from numpy.random import seed
3  from numpy.random import randn
4  from numpy import mean
5  from numpy import std
6  # seed the random number generator
7  seed(1)
8  data = 5 * randn(10000) + 50
9  print(' mean=%.3f stdv=%.3f' % (mean(data), std(data)))
```

Méthodes pour identifier les valeurs aberrantes

```
1  # generate gaussian data
2  from numpy.random import seed
3  from numpy.random import randn
4  from numpy import mean
5  from numpy import std
6  # seed the random number generator
7  seed(1)
8  data = 5 * randn(10000) + 50
9  print(' mean=%.3f stdv=%.3f' % (mean(data), std(data)))
```

- Méthode d'écart type "Standard Deviation Method"
  - Si la distribution des valeurs de l'échantillon est gaussienne, on peut utiliser l'écart-type comme seuil pour identifier les valeurs aberrantes.
  - Par exemple, plus au moins d'écart-type de la moyenne couvrira 68 % des données.
  - Ainsi, si la moyenne est de 50 et l'écart-type est 5, comme c'est le cas dans les données de test.
  - Toutes les données de l'échantillon comprises entre 45 et 55 représenteront environ 68 % de l'échantillon de données.

Méthode d'écart type

Nous pouvons couvrir une plus grande partie de l'échantillon de données si nous étendons là comme suit :

Méthode d'écart type

Nous pouvons couvrir une plus grande partie de l'échantillon de données si nous étendons là comme suit :

- **1 écarts-type** par rapport à la moyenne : **68 %**
- **2 écarts-types** par rapport à la moyenne : **95 %**
- **3 écarts-types** par rapport à la moyenne : **99.7 %**

Méthode d'écart type

Nous pouvons couvrir une plus grande partie de l'échantillon de données si nous étendons là comme suit :

- **1 écarts-type** par rapport à la moyenne : **68 %**
- **2 écarts-types** par rapport à la moyenne : **95 %**
- **3 écarts-types** par rapport à la moyenne : **99.7 %**

Remarque :

Méthode d'écart type

Nous pouvons couvrir une plus grande partie de l'échantillon de données si nous étendons là comme suit :

- **1 écarts-type** par rapport à la moyenne : **68 %**
- **2 écarts-types** par rapport à la moyenne : **95 %**
- **3 écarts-types** par rapport à la moyenne : **99.7 %**

Remarque :

- Une valeur qui tombe en dehors de **3 écarts-types** fait partie de la distribution, mais c'est un événement peu probable ou rare à environ 1 échantillon sur 370.
- Trois écarts-types par rapport à la la moyenne est un seuil commun dans la pratique pour identifier les valeurs aberrantes dans une distribution Gaussienne.
- Pour des échantillons de taille petite, une valeur de **2 écarts-types (95 %)** peut être utilisé, et pour des échantillons plus grands, une valeur de **4 écarts-types (99,9%)** peut être utilisé.

Méthode d'écart type

Nous pouvons couvrir une plus grande partie de l'échantillon de données si nous étendons là comme suit :

- **1 écarts-type** par rapport à la moyenne : **68 %**
- **2 écarts-types** par rapport à la moyenne : **95 %**
- **3 écarts-types** par rapport à la moyenne : **99.7 %**

Remarque :

- Une valeur qui tombe en dehors de **3 écarts-types** fait partie de la distribution, mais c'est un événement peu probable ou rare à environ 1 échantillon sur 370.
- Trois écarts-types par rapport à la la moyenne est un seuil commun dans la pratique pour identifier les valeurs aberrantes dans une distribution Gaussienne.
- Pour des échantillons de taille petite, une valeur de **2 écarts-types (95 %)** peut être utilisé, et pour des échantillons plus grands, une valeur de **4 écarts-types (99,9%)** peut être utilisé.

Exemple :

Calculer la moyenne et l'écart type d'un échantillon donné, puis calculer le seuil pour identifier les valeurs aberrantes à plus de **3 écarts-types de la moyenne.**

```
...
# calculate summary statistics
data_mean, data_std = mean(data), std(data)
# define outliers
cut_off = data_std * 3
lower, upper = data_mean - cut_off, data_mean + cut_off
```

```
...
# calculate summary statistics
data_mean, data_std = mean(data), std(data)
# define outliers
cut_off = data_std * 3
lower, upper = data_mean - cut_off, data_mean + cut_off
```

Nous pouvons alors identifier les valeurs aberrantes comme celles qui ne font pas partie de l'intervalle [lower, upper]

```
...
# calculate summary statistics
data_mean, data_std = mean(data), std(data)
# define outliers
cut_off = data_std * 3
lower, upper = data_mean - cut_off, data_mean + cut_off
```

Nous pouvons alors identifier les valeurs aberrantes comme celles qui ne font pas partie de l'intervalle [lower, upper]

```
...
# identify outliers
outliers = [x for x in data if x < lower or x > upper]
```

```
...
# calculate summary statistics
data_mean, data_std = mean(data), std(data)
# define outliers
cut_off = data_std * 3
lower, upper = data_mean - cut_off, data_mean + cut_off
```

Nous pouvons alors identifier les valeurs aberrantes comme celles qui ne font pas partie de l'intervalle [lower, upper]

```
...
# identify outliers
outliers = [x for x in data if x < lower or x > upper]
```

Alternativement, nous pouvons filtrer les valeurs de l'échantillon qui ne sont pas dans les plages limites définies :

```
...
# calculate summary statistics
data_mean, data_std = mean(data), std(data)
# define outliers
cut_off = data_std * 3
lower, upper = data_mean - cut_off, data_mean + cut_off
```

Nous pouvons alors identifier les valeurs aberrantes comme celles qui ne font pas partie de l'intervalle [lower, upper]

```
...
# identify outliers
outliers = [x for x in data if x < lower or x > upper]
```

Alternativement, nous pouvons filtrer les valeurs de l'échantillon qui ne sont pas dans les plages limites définies :

```
...
# remove outliers
outliers_removed = [x for x in data if x > lower and x < upper]
```

Réaliser le LAB 7

Exemple : outliers identification with standard deviation

Remarque :

Jusqu'à présent, nous n'avons parlé que de données univariées avec une distribution gaussienne, par ex. une seule variable. Vous pouvez utiliser la même approche si vous avez des données multivariées, par ex. données avec plusieurs variables, chacune avec une distribution gaussienne différente. Vous pouvez imaginer des limites en deux dimensions cela définirait une ellipse si vous avez deux variables. Les observations qui sortent du cadre de l'ellipse seraient considérées comme aberrante. En trois dimensions, ce serait un ellipsoïde, et ainsi de suite dans des dimensions supérieures.

Méthode de l'intervalle interquartile : InterQuartile Range Method (IQR)

- Toutes les données ne sont pas normales ou suffisamment normales pour être traitées comme étant tirées d'une distribution gaussienne.
- Une bonne statistique pour résumer un échantillon de données de distribution non gaussienne est l'intervalle interquartile
- L'IQR est calculé comme la différence entre le 75e et le 25e centiles des données et définit dans un tracé en boîte à moustaches.
- Rappelez-vous que les centiles peuvent être calculés en triant les observations et en sélectionnant des valeurs à des indices spécifiques.
- Le 50e centile est la valeur médiane, ou la moyenne des deux valeurs médianes pour un nombre pair. Si nous avions 10 000 échantillons, le 50e centile serait la moyenne des 5000e et 5001e valeurs.
- On appelle les centiles des quartiles (quart signifiant 4) car les données sont divisées en quatre groupes via les 25e, 50e et 75e valeurs. L'IQR définit le milieu de 50 % des données

- L'IQR peut être utilisé pour identifier les valeurs aberrantes en définissant des limites sur les valeurs d'échantillon qui ont un facteur k de L'IQR inférieur au 25e centile ou supérieur au 75e centile.
- La valeur commune pour le facteur k est la valeur 1,5. Un facteur k de 3 ou plus peut être utilisé pour identifier les valeurs qui sont les valeurs aberrantes extrêmes ou les sorties éloignées lorsqu'elles sont décrites dans le diagramme en boîte de moustaches.
- Sur une boîte de moustaches, ces limites sont dessinées comme des clôtures sur les moustaches (ou les lignes) qui sont tracées de la boite. Les valeurs qui se situent en dehors de ces valeurs sont dessinées sous forme de points.
- Nous pouvons calculer la centile à l'aide de la fonction **percentile()** de **NumPy** qui prend l'ensemble de données et spécification du centile souhaité.
- L'IQR peut alors être calculé comme la différence entre les 75e et 25e centiles.

- L'IQR peut être utilisé pour identifier les valeurs aberrantes en définissant des limites sur les valeurs d'échantillon qui ont un facteur k de L'IQR inférieur au 25e centile ou supérieur au 75e centile.
- La valeur commune pour le facteur k est la valeur 1,5. Un facteur k de 3 ou plus peut être utilisé pour identifier les valeurs qui sont les valeurs aberrantes extrêmes ou les sorties éloignées lorsqu'elles sont décrites dans le diagramme en boîte de moustaches.
- Sur une boîte de moustaches, ces limites sont dessinées comme des clôtures sur les moustaches (ou les lignes) qui sont tracées de la boite. Les valeurs qui se situent en dehors de ces valeurs sont dessinées sous forme de points.
- Nous pouvons calculer la centile à l'aide de la fonction **percentile()** de **NumPy** qui prend l'ensemble de données et spécification du centile souhaité.
- L'IQR peut alors être calculé comme la différence entre les 75e et 25e centiles.

```
...
# calculate interquartile range
q25, q75 = percentile(data, 25), percentile(data, 75)
iqr = q75 - q25
```

- L'IQR peut être utilisé pour identifier les valeurs aberrantes en définissant des limites sur les valeurs d'échantillon qui ont un facteur k de L'IQR inférieur au 25e centile ou supérieur au 75e centile.
- La valeur commune pour le facteur k est la valeur 1,5. Un facteur k de 3 ou plus peut être utilisé pour identifier les valeurs qui sont les valeurs aberrantes extrêmes ou les sorties éloignées lorsqu'elles sont décrites dans le diagramme en boîte de moustaches.
- Sur une boîte de moustaches, ces limites sont dessinées comme des clôtures sur les moustaches (ou les lignes) qui sont tracées de la boite. Les valeurs qui se situent en dehors de ces valeurs sont dessinées sous forme de points.
- Nous pouvons calculer la centile à l'aide de la fonction **percentile()** de **NumPy** qui prend l'ensemble de données et spécification du centile souhaité.
- L'IQR peut alors être calculé comme la différence entre les 75e et 25e centiles.

```
...
# calculate interquartile range
q25, q75 = percentile(data, 25), percentile(data, 75)
iqr = q75 - q25
```

Nous pouvons ensuite calculer le seuil pour les valeurs aberrantes comme 1,5 fois l'IQR et soustraire ce seuil du 25e centile et ajoutez-le au 75e centile pour donner les limites actuelles des données.

Identification et suppression des outliers par IQR

```
...
# calculate the outlier cutoff
cut_off = iqr * 1.5
lower, upper = q25 - cut_off, q75 + cut_off
```

Identification et suppression des outliers par IQR

```
...
# calculate the outlier cutoff
cut_off = iqr * 1.5
lower, upper = q25 - cut_off, q75 + cut_off
```

Nous pouvons ensuite utiliser ces limites pour identifier les valeurs aberrantes.

Identification et suppression des outliers par IQR

```
...
# calculate the outlier cutoff
cut_off = iqr * 1.5
lower, upper = q25 - cut_off, q75 + cut_off
```

Nous pouvons ensuite utiliser ces limites pour identifier les valeurs aberrantes.

```
....
# identify outliers
outliers = [x for x in data if x < lower or x > upper]
```

Identification et suppression des outliers par IQR

```
...
# calculate the outlier cutoff
cut_off = iqr * 1.5
lower, upper = q25 - cut_off, q75 + cut_off
```

Nous pouvons ensuite utiliser ces limites pour identifier les valeurs aberrantes.

```
....
# identify outliers
outliers = [x for x in data if x < lower or x > upper]
```

Nous pouvons également utiliser les limites pour filtrer les valeurs aberrantes de l'ensemble de données.

Identification et suppression des outliers par IQR

```
...
# calculate the outlier cutoff
cut_off = iqr * 1.5
lower, upper = q25 - cut_off, q75 + cut_off
```

Nous pouvons ensuite utiliser ces limites pour identifier les valeurs aberrantes.

```
....
# identify outliers
outliers = [x for x in data if x < lower or x > upper]
```

Nous pouvons également utiliser les limites pour filtrer les valeurs aberrantes de l'ensemble de données.

```
...
# remove outliers
outliers_removed = [x for x in data if x > lower and x < upper]
```

Réaliser le LAB 7

Exemple : **outliers identification with IQR**

Détection automatique des valeurs aberrantes : Automatic Outlier Detection

- En Machine Learning, une approche pour résoudre le problème de la détection des valeurs aberrantes est la classification à une classe
- Un classificateur à une classe vise à capturer les caractéristiques des instances d'entraînement, afin d'être en mesure de les distinguer des valeurs aberrantes potentielles à apparaître.
- Une approche simple pour identifier les valeurs aberrantes consiste à localiser les exemples de données qui sont éloignés d'autres exemples dans l'espace des caractéristiques multidimensionnelles.
- Le facteur de valeur aberrante locale est une technique qui tente d'exploiter l'idée des voisins les plus proches pour détecter les valeurs aberrantes.
- Chaque exemple se voit attribuer une note indiquant à quel point il est isolé ou susceptible d'être une valeur aberrante en fonction de la taille de son voisinage local.
- Les exemples avec le score le plus élevé sont plus susceptibles d'être des valeurs aberrantes.
- La bibliothèque **scikit-learn** fournit une implémentation de cette approche dans la Classe **LocalOutlierFactor**.

Introduction à LOF (Local Outlier Factor)

- Le **Local Outlier Factor (LOF)** est un algorithme de détection d'anomalies basé sur la densité locale des données.
- Il compare la densité locale d'un point à celle de ses voisins proches pour déterminer si un point est un outlier.
- LOF est particulièrement utile pour détecter des outliers locaux dans des données multidimensionnelles.

Fonctionnement de LOF

- **Étape 1** : Calcul des distances entre chaque point et ses voisins les plus proches (*k-nearest neighbors*).
- **Étape 2** : Calcul de la *distance atteignable* entre un point et ses voisins.
- **Étape 3** : Calcul de la densité locale basée sur la distance atteignable.
- **Étape 4** : Le LOF d'un point est le ratio de la densité moyenne de ses voisins par rapport à sa propre densité.
- **Résultat** : Un LOF élevé (>1) indique un outlier.

Paramètres importants de LOF

- **n_neighbors** : Nombre de voisins à prendre en compte pour la comparaison de densité locale.
- **contamination** : Proportion estimée d'outliers dans le dataset.
- **leaf_size** : Optimisation pour la recherche des voisins dans l'arbre.

Exemple en Python - LOF

Exemple d'application du LOF en Python :

```python
from sklearn.neighbors import LocalOutlierFactor
import numpy as np
import matplotlib.pyplot as plt

# Génération de données
X_inliers = 0.3 * np.random.randn(100, 2)
X_outliers = np.random.uniform(low=-4, high=4, size=(20, 2))
X = np.r_[X_inliers + 2, X_inliers - 2, X_outliers]

# Création du modèle LOF
lof = LocalOutlierFactor(n_neighbors=20, contamination=0.1)

# Ajustement du modèle et détection des outliers
y_pred = lof.fit_predict(X)

# Points prédit comme outliers
outliers = X[y_pred == -1]

# Visualisation
plt.scatter(X[:, 0], X[:, 1], label='Inliers')
plt.scatter(outliers[:, 0], outliers[:, 1], color='r', label='Outliers')
plt.legend()
plt.show()
```

Analyse des résultats

- Les points normaux sont étiquetés par ‘1‘ et les outliers par ‘-1‘.
- Les outliers sont les points dont la densité locale est beaucoup plus faible que celle de leurs voisins.
- La visualisation montre clairement les points anormaux en rouge.

Comparaison avec d'autres méthodes

- **LOF** : Détecte les outliers locaux, particulièrement efficace dans des datasets complexes.
- **Isolation Forest** : Méthode basée sur l'isolation des points aberrants.
- **Méthode de l'écart-type** : Simple mais moins efficace pour des anomalies locales ou multivariées.
- **Boîte à moustaches (Boxplot)** : Adaptée pour des données unidimensionnelles.

Conclusion

- Le **Local Outlier Factor** est un outil puissant pour la détection des anomalies locales dans des données multidimensionnelles.
- Son efficacité dépend du réglage de ‘n_neighbors‘ et de la bonne estimation de la contamination.
- Utiliser le LOF pour des cas où les outliers sont dispersés localement plutôt que globalement.

Réaliser le LAB 7

Exemple : outliers identification with LocalOutlierFactor

Exemple de dataset de logement de Boston :

Introduction aux valeurs manquantes

- Les données du monde réel ont souvent des valeurs manquantes. Les données peuvent avoir des valeurs manquantes pour un certain nombre de raisons telles que
  - les observations non enregistrées et la corruption des données.
  - Erreurs humaines lors de la saisie.
  - Problèmes techniques (dysfonctionnement de capteurs, absence de réponse dans des enquêtes, etc.).
- La gestion des données manquantes est important car de nombreux algorithmes d'apprentissage automatique ne prennent pas en charge les données avec des valeurs manquantes.
- Comment marquer les valeurs non valides ou corrompues comme manquantes dans votre ensemble de données.
- Comment confirmer que la présence de valeurs manquantes marquées cause des problèmes d'apprentissage.
- Comment supprimer des lignes avec des données manquantes de votre jeu de données et évaluer un algorithme d'apprentissage sur le jeu de données transformé.

Types de valeurs manquantes

Les valeurs manquantes peuvent être classées en trois catégories :

- **MCAR (Missing Completely At Random)** : la probabilité de manquer une donnée ne dépend d'aucune variable, y compris la variable elle-même.
- **MAR (Missing At Random)** : la probabilité de manquer une donnée dépend d'autres variables observées.
- **MNAR (Missing Not At Random)** : la probabilité de manquer une donnée dépend de la valeur manquante elle-même.

- MCAR (Missing Completely At Random)
  - Les données sont manquantes de façon complètement aléatoire.
  - Exemple : certaines personnes oublient par hasard de répondre à une question dans une enquête.
  - Conséquence : les données restantes sont représentatives et la suppression des lignes peut être envisagée sans biais.
- MAR (Missing At Random)
  - Les données sont manquantes de manière aléatoire mais corrélée à une ou plusieurs autres variables observées.
  - Exemple : les personnes âgées omettent plus souvent de déclarer leur revenu dans une enquête.
  - Conséquence : l'imputation basée sur d'autres variables connues peut être utilisée.

MNAR (Missing Not At Random)

Dans ce cas, les données manquantes ne sont pas **aléatoires** et leur absence est liée à des facteurs qui **ne sont pas observés** dans les autres variables de l'ensemble de données.

- En d'autres termes, la probabilité qu'une donnée soit manquante dépend de la valeur elle-même ou d'autres facteurs non mesurés dans le modèle.
- Exemple : les personnes avec des revenus très élevés peuvent choisir de ne pas répondre par souci de confidentialité.
- Conséquence : plus difficile à traiter, nécessitant des modèles spécifiques ou des hypothèses sur le mécanisme des données manquantes.

Nature de l'absence de certaines données

- Il n'est pas possible de savoir, uniquement à partir des données, si des données manquent de façon aléatoire ou non (MAR ou MNAR)
  - Chercher d'autres travaux sur le même problème (caractérisé par les données manquantes) qui peuvent informer sur ce mécanisme
  - Inclure dans l'étude un maximum de variables qui pourraient expliquer les données manquantes, afin de rendre MAR bien plus probable (dans les variables observées, on en trouve qui expliquent les variables à données manquantes) que MNAR
- Lorsque les données manquent de façon non aléatoire (cas MNAR), un modèle pour ce manque est indisponible
  - Possible de compléter la collecte (observer de nouvelles variables, explicatives pour le manque de données) afin de passer de MNAR à MAR ?

Solution 1 : suppression des observations à données manquantes

- Solution souvent employée par défaut
- Quel est son impact suivant la nature de l'absence de données ?
  - MCAR :
    - Ignorer les observations à données manquantes = conserver un échantillon aléatoire des observations
    - Ne biaise pas les résultats, mais peut réduire la qualité du modèle résultant (surtout en présence de classes déséquilibrées) s'il y a beaucoup d'observations à données manquantes
  - MAR :
    - Ignorer les observations à données manquantes biaise les résultats
    - Préférable de compléter les observations à données manquantes (imputer les données manquantes)
  - MNAR :
    - Ignorer les observations à données manquantes biaise les résultats
    - L'imputation des données manquantes est plus difficile à justifier, surtout à partir des autres variables observées, vu que l'absence dépend de variable(s) non observée(s)

Solution 2 : imputation des données manquantes

- Compléter les observations à données manquantes peut améliorer la précision du modèle résultant ou réduire son biais
- Nécessaire de comparer le modèle obtenu par imputation avec celui obtenu (si cela est possible) par suppression des observations à données manquantes
- Diverses méthodes d'imputation :
  1. Par une valeur unique (moyenne, médiane, etc.)
  2. Par le centre du groupe
  3. A partir des k plus proches voisins
  4. Par une moyenne partielle
  5. Autres méthode

Identification des valeurs manquantes en Python

Utilisation des bibliothèques pandas pour identifier les valeurs manquantes :

- `df.isnull()` : Renvoie un DataFrame de booléens indiquant les cellules manquantes.
- `df.isnull().sum()` : Compte le nombre de valeurs manquantes par colonne.

Exemple

```
1 import pandas as pd
2 data = {'Nom': ['Alice', 'Bob', 'Charlie', None],
3         'Âge': [25, None, 30, 22],
4         'Ville': ['Paris', 'Lyon', None, 'Marseille']}
5 df = pd.DataFrame(data)
6 print(df.isnull())
7 print(df.isnull().sum())
```

Visualisation des valeurs manquantes

Utilisation de la bibliothèque `missingno` pour visualiser les valeurs manquantes :

- `msno.matrix(df)` : Crée une matrice visuelle des valeurs manquantes.
- `msno.heatmap(df)` : Visualise les corrélations entre les valeurs manquantes.
- `msno.bar(df)` : Montre la proportion de valeurs manquantes dans chaque colonne.
- `msno.dendrogram(df)` : Analyse la corrélation entre les valeurs manquantes

```
1 import missingno as msno
2 msno.matrix(df)
3 msno.bar(df)
4 msno.heatmap(df)
5 msno.dendrogram(df)
```

Conclusion

- Comprendre le type de valeurs manquantes (MCAR, MAR, MNAR) est essentiel pour choisir la bonne méthode de traitement.
- Les outils disponibles en Python comme `pandas`, `missingno`, et `scikit-learn` offrent une gamme de solutions pour identifier et traiter ces valeurs.
- La stratégie de traitement dépend du contexte des données et des objectifs de l'analyse.

Diabetes Dataset

- Comme base de ce tutoriel, nous utiliserons l'ensemble de données sur le diabète qui a été largement étudié comme ensemble de données d'apprentissage automatique depuis les années 1990.
- L'ensemble de données classe les données des patients comme soit un début de diabète dans les cinq ans ou non.Il y a 768 exemples et huit entrées variables.
- C'est un problème de classification binaire.

Cet ensemble de données est connu pour avoir des valeurs manquantes. Plus précisément, il manque des observations pour certaines colonnes qui sont marquées comme une valeur nulle. Nous pouvons corroborer cela par la définition de ces colonnes et le domaine savent qu'une valeur nulle n'est pas valide pour ces mesures, par ex. un zéro pour l'indice de masse corporelle ou la tension artérielle n'est pas valide.

Marquer les valeurs manquantes

- La plupart des données ont des valeurs manquantes, et la probabilité d'avoir des valeurs manquantes augmente avec la taille du jeu de données.
- Nous pouvons identifier et marquer les valeurs comme manquantes en utilisant par exemple des graphiques et statistiques récapitulatives

Marquer les valeurs manquantes

- La plupart des données ont des valeurs manquantes, et la probabilité d'avoir des valeurs manquantes augmente avec la taille du jeu de données.
- Nous pouvons identifier et marquer les valeurs comme manquantes en utilisant par exemple des graphiques et statistiques récapitulatives

```
1 # load and summarize the dataset
2 from pandas import read_csv
3 # load the dataset
4 dataset = read_csv('pima-indians-diabetes.csv', header=None)
5 # summarize the dataset
6 print(dataset.describe())
```

Marquer les valeurs manquantes

- La plupart des données ont des valeurs manquantes, et la probabilité d'avoir des valeurs manquantes augmente avec la taille du jeu de données.
- Nous pouvons identifier et marquer les valeurs comme manquantes en utilisant par exemple des graphiques et statistiques récapitulatives

```
1 # load and summarize the dataset
2 from pandas import read_csv
3 # load the dataset
4 dataset = read_csv('pima-indians-diabetes.csv', header=None)
5 # summarize the dataset
6 print(dataset.describe())
```

Remarques :

- Nous pouvons voir qu'il y a des colonnes qui ont une valeur minimale de zéro (0).
- Sur certaines colonnes, une valeur de zéro n'a pas de sens et indique une valeur invalide ou manquante.
- Les valeurs manquantes sont fréquemment indiquées par des entrées hors plage.
- Ça peut-être un **nombre négatif** (par exemple, -1) dans un champ numérique qui est normalement positif,
- ou un **0** dans un champ numérique champ qui ne peut "normalement" jamais être égal à 0.

Plus précisément, les colonnes suivantes ont une valeur minimale de zéro non valide :

1. Plasma glucose concentration
2. Diastolic blood pressure
3. Triceps skinfold thickness
4. 2-Hour serum insulin
5. Body mass index

Confirmons cela en regardant les données brutes, l'exemple imprime les 20 premières lignes de données.

```
1 # load the dataset and review rows
2 from pandas import read_csv
3 # load the dataset
4 dataset = read_csv('pima-indians-diabetes.csv', header=None)
5 # summarize the first 20 rows of data
6 print(dataset.head(20))
```

Plus précisément, les colonnes suivantes ont une valeur minimale de zéro non valide :

1. Plasma glucose concentration
2. Diastolic blood pressure
3. Triceps skinfold thickness
4. 2-Hour serum insulin
5. Body mass index

Confirmons cela en regardant les données brutes, l'exemple imprime les 20 premières lignes de données.

```
1 # load the dataset and review rows
2 from pandas import read_csv
3 # load the dataset
4 dataset = read_csv('pima-indians-diabetes.csv', header=None)
5 # summarize the first 20 rows of data
6 print(dataset.head(20))
```

Question :

Calculer le nombre de valeurs manquantes sur chacune colonne ?

Vous pouvez faire ceci en marquant toutes les valeurs "qui vous intéressent" et qui ont des valeurs nulles comme **True**.

```
# example of summarizing the number of missing values for each variable
from pandas import read_csv
# load the dataset
dataset = read_csv('pima-indians-diabetes.csv', header=None)
# count the number of missing values for each column
num_missing = (dataset[[1,2,3,4,5]] == 0).sum()
# report the results
print(num_missing)
```

```python
# example of summarizing the number of missing values for each variable
from pandas import read_csv
# load the dataset
dataset = read_csv('pima-indians-diabetes.csv', header=None)
# count the number of missing values for each column
num_missing = (dataset[[1,2,3,4,5]] == 0).sum()
# report the results
print(num_missing)
```

Remarques :

- Nous pouvons voir que les colonnes 1, 2 et 5 n'ont que quelques valeurs nulles, alors que les colonnes 3 et 4 représentent beaucoup plus, près de la moitié des lignes.
- En Python, en particulier Pandas, NumPy et Scikit-Learn, nous marquons les valeurs manquantes comme NaN.
- Les valeurs avec une valeur **NaN** sont ignorées des opérations comme la somme, le comptage, etc.
- Nous pouvons marquer les valeurs comme **NaN** facilement avec **Pandas** en utilisant la fonction **replace ()** sur les colonnes qui nous intéressent.
- Après avoir marqué les valeurs manquantes, nous pouvons utiliser la fonction **isnull()** pour marquer toutes les valeurs **NaN** de l'ensemble de données comme vraies et obtenir un décompte des valeurs manquantes pour chaque colonne.

```
# example of marking missing values with nan values
from numpy import nan
from pandas import read_csv
# load the dataset
dataset = read_csv( ' pima-indians-diabetes.csv ' , header=None)
# replace ' 0 ' values with ' nan '
dataset[[1,2,3,4,5]] = dataset[[1,2,3,4,5]].replace(0, nan)
# count the number of nan values in each column
print(dataset.isnull().sum())
```

```python
# example of marking missing values with nan values
from numpy import nan
from pandas import read_csv
# load the dataset
dataset = read_csv( ' pima-indians-diabetes.csv ' , header=None)
# replace ' 0 ' values with ' nan '
dataset[[1,2,3,4,5]] = dataset[[1,2,3,4,5]].replace(0, nan)
# count the number of nan values in each column
print(dataset.isnull().sum())
```

Avant d'examiner la gestion des valeurs manquantes, démontrons d'abord que le fait d'avoir des valeurs manquantes dans un ensemble de données peut causer des problèmes!!!

Les valeurs manquantes causent des problèmes

- Avoir des valeurs manquantes dans un ensemble de données peut entraîner des erreurs pour certains algorithmes d'apprentissage automatique.
- Les valeurs manquantes sont des occurrences courantes dans les données. Malheureusement, de nombreuses techniques de modélisation prédictive ne peuvent pas gérer les valeurs manquantes. Par conséquent, ce problème doit être abordé avant la modélisation.

Les valeurs manquantes causent des problèmes

- Avoir des valeurs manquantes dans un ensemble de données peut entraîner des erreurs pour certains algorithmes d'apprentissage automatique.
- Les valeurs manquantes sont des occurrences courantes dans les données. Malheureusement, de nombreuses techniques de modélisation prédictive ne peuvent pas gérer les valeurs manquantes. Par conséquent, ce problème doit être abordé avant la modélisation.

Dans cette section, nous allons essayer d'évaluer l'algorithme d'analyse discriminante linéaire (LDA) sur le jeu de données avec des valeurs manquantes. C'est un algorithme qui ne fonctionne pas lorsqu'il y a des valeurs manquantes.

Les valeurs manquantes causent des problèmes

- Avoir des valeurs manquantes dans un ensemble de données peut entraîner des erreurs pour certains algorithmes d'apprentissage automatique.
- Les valeurs manquantes sont des occurrences courantes dans les données. Malheureusement, de nombreuses techniques de modélisation prédictive ne peuvent pas gérer les valeurs manquantes. Par conséquent, ce problème doit être abordé avant la modélisation.

Dans cette section, nous allons essayer d'évaluer l'algorithme d'analyse discriminante linéaire (LDA) sur le jeu de données avec des valeurs manquantes. C'est un algorithme qui ne fonctionne pas lorsqu'il y a des valeurs manquantes.

Question :
Tenter d'évaluer l'algorithme LDA en utilisant une validation croisée triple et imprimer la précision moyenne ?

```python
# example where missing values cause errors
from numpy import nan
from pandas import read_csv
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
# load the dataset
dataset = read_csv('pima-indians-diabetes.csv' , header=None)
# replace '0' values with 'nan'
dataset[[1,2,3,4,5]] = dataset[[1,2,3,4,5]].replace(0, nan)
# split dataset into inputs and outputs
values = dataset.values
X = values[:,0:8]
y = values[:,8]
# define the model
model = LinearDiscriminantAnalysis()
# define the model evaluation procedure
cv = KFold(n_splits=3, shuffle=True, random_state=1)
# evaluate the model
result = cross_val_score(model, X, y, cv=cv, scoring= 'accuracy' )
# report the mean performance
print('Accuracy: %.3f' % result.mean())
```

Supprimer les lignes avec des valeurs manquantes

- La stratégie la plus simple pour gérer les données manquantes consiste à supprimer les enregistrements qui contiennent un élément manquant.
- Nous pouvons le faire en créant un nouveau **Pandas DataFrame** en supprimant les lignes contenant des valeurs manquantes.
- **Pandas** fournit la fonction **dropna()** qui peut être utilisée pour supprimer des colonnes ou lignes avec des données manquantes.

Supprimer les lignes avec des valeurs manquantes

- La stratégie la plus simple pour gérer les données manquantes consiste à supprimer les enregistrements qui contiennent un élément manquant.
- Nous pouvons le faire en créant un nouveau **Pandas DataFrame** en supprimant les lignes contenant des valeurs manquantes.
- **Pandas** fournit la fonction **dropna()** qui peut être utilisée pour supprimer des colonnes ou lignes avec des données manquantes.

```
1 from numpy import nan
2 from pandas import read_csv
3 dataset = read_csv('pima-indians-diabetes.csv', header=None)#
4 print(dataset.shape)# summarize the shape of the raw data
5 # replace '0' values with 'nan'
6 dataset[[1,2,3,4,5]] = dataset[[1,2,3,4,5]].replace(0, nan)
7 dataset.dropna(inplace=True) # drop rows with missing values
8 # summarize the shape of the data with missing rows removed
9 print(dataset.shape)
```

Supprimer les lignes avec des valeurs manquantes

- La stratégie la plus simple pour gérer les données manquantes consiste à supprimer les enregistrements qui contiennent un élément manquant.
- Nous pouvons le faire en créant un nouveau **Pandas DataFrame** en supprimant les lignes contenant des valeurs manquantes.
- **Pandas** fournit la fonction **dropna()** qui peut être utilisée pour supprimer des colonnes ou lignes avec des données manquantes.

```
1 from numpy import nan
2 from pandas import read_csv
3 dataset = read_csv('pima-indians-diabetes.csv', header=None)#
4 print(dataset.shape)# summarize the shape of the raw data
5 # replace '0' values with 'nan'
6 dataset[[1,2,3,4,5]] = dataset[[1,2,3,4,5]].replace(0, nan)
7 dataset.dropna(inplace=True) # drop rows with missing values
8 # summarize the shape of the data with missing rows removed
9 print(dataset.shape)
```

En exécutant cet exemple, nous pouvons voir que le nombre de lignes a été fortement réduit de **768** dans l'ensemble de données d'origine à **392** avec toutes les lignes contenant un **NaN** supprimées.

Nous avons maintenant un jeu de données que nous pourrions utiliser pour évaluer un algorithme sensible aux valeurs manquantes comme LDA.

Nous avons maintenant un jeu de données que nous pourrions utiliser pour évaluer un algorithme sensible aux valeurs manquantes comme LDA.

```
1  from numpy import nan
2  from pandas import read_csv
3  from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
4  from sklearn.model_selection import KFold
5  from sklearn.model_selection import cross_val_score
6  dataset = read_csv('pima-indians-diabetes.csv', header=None)
7  dataset[[1,2,3,4,5]] = dataset[[1,2,3,4,5]].replace(0, nan)
8  dataset.dropna(inplace=True) # drop rows with missing values
9  values = dataset.values
10 X = values[:,0:8]
11 y = values[:,8]
12 model = LinearDiscriminantAnalysis()
13 cv = KFold(n_splits=3,shuffle=True, random_state=1)
14 result = cross_val_score(model, X, y, cv=cv, scoring= 'accuracy' )#
15 print('Accuracy: %.3f' % result.mean()) # report the mean performance
```

Nous avons maintenant un jeu de données que nous pourrions utiliser pour évaluer un algorithme sensible aux valeurs manquantes comme LDA.

```
1  from numpy import nan
2  from pandas import read_csv
3  from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
4  from sklearn.model_selection import KFold
5  from sklearn.model_selection import cross_val_score
6  dataset = read_csv('pima-indians-diabetes.csv', header=None)
7  dataset[[1,2,3,4,5]] = dataset[[1,2,3,4,5]].replace(0, nan)
8  dataset.dropna(inplace=True) # drop rows with missing values
9  values = dataset.values
10 X = values[:,0:8]
11 y = values[:,8]
12 model = LinearDiscriminantAnalysis()
13 cv = KFold(n_splits=3,shuffle=True, random_state=1)
14 result = cross_val_score(model, X, y, cv=cv, scoring= 'accuracy' )#
15 print('Accuracy: %.3f' % result.mean()) # report the mean performance
```

Supprimer des lignes avec des valeurs manquantes peut être trop limitant sur certains problèmes de modélisation prédictive, une alternative consiste à imputer les valeurs manquantes.

Comment utiliser l'imputation statistique

- Les ensembles de données peuvent avoir des valeurs manquantes, ce qui peut causer des problèmes pour de nombreux algorithmes de Machine Learing.
- Il est recommandé d'identifier et de remplacer les valeurs manquantes dans vos données d'entrée avant de modéliser votre tâche de prédiction.
- C'est ce qu'on appelle l'imputation des données manquantes.
- Une approche populaire pour l'imputation des données consiste à calculer une valeur pour chaque colonne (telle qu'une moyenne) et remplacer toutes les valeurs manquantes par la statistique.

Comment utiliser l'imputation statistique

- Les ensembles de données peuvent avoir des valeurs manquantes, ce qui peut causer des problèmes pour de nombreux algorithmes de Machine Learing.
- Il est recommandé d'identifier et de remplacer les valeurs manquantes dans vos données d'entrée avant de modéliser votre tâche de prédiction.
- C'est ce qu'on appelle l'imputation des données manquantes.
- Une approche populaire pour l'imputation des données consiste à calculer une valeur pour chaque colonne (telle qu'une moyenne) et remplacer toutes les valeurs manquantes par la statistique.

Dans ce tutoriel, vous découvrirez comment utiliser des stratégies d'imputation statistique pour les données manquantes dans l'apprentissage automatique.

Comment utiliser l'imputation statistique

- Les ensembles de données peuvent avoir des valeurs manquantes, ce qui peut causer des problèmes pour de nombreux algorithmes de Machine Learing.
- Il est recommandé d'identifier et de remplacer les valeurs manquantes dans vos données d'entrée avant de modéliser votre tâche de prédiction.
- C'est ce qu'on appelle l'imputation des données manquantes.
- Une approche populaire pour l'imputation des données consiste à calculer une valeur pour chaque colonne (telle qu'une moyenne) et remplacer toutes les valeurs manquantes par la statistique.

Dans ce tutoriel, vous découvrirez comment utiliser des stratégies d'imputation statistique pour les données manquantes dans l'apprentissage automatique.

- Les valeurs manquantes doivent être marquées avec des valeurs **NaN** et peuvent être remplacées par des valeurs statistiques.
- Comment charger un fichier **CSV** avec des valeurs manquantes et les marquer avec des **NaN** et indiquer le nombre et le pourcentage de valeurs manquantes pour chaque colonne.
- Comment imputer les valeurs manquantes avec des statistiques comme méthode de préparation des données lors de l'évaluation d'un modèle pour faire des prédictions sur de nouvelles données.

- Un jeu de données peut avoir des valeurs manquantes. Les valeurs peuvent manquer complètement ou elles peuvent être marquées avec un caractère spécial ou une valeur, comme un point d'interrogation **(?)**, une chaîne vide **[...]**, la chaîne explicite **NULL** ou undefined ou **N/A** ou **NaN**, etc.
- Ils peuvent se produire pour un certain nombre de raisons, telles qu'un dysfonctionnement de l'équipement de mesure, les changements dans la conception expérimentale pendant la collecte des données et la collation des plusieurs ensembles de données similaires mais non identiques.
- La plupart des algorithmes de Machine Learning nécessitent la présence des valeurs d'entrée numériques pour chaque ligne et colonne.
- Les valeurs manquantes peuvent causer des problèmes pour les algorithmes d'apprentissage. Pour cette raison, il faut les identifier et les remplacer par une valeur numérique (**imputation des données manquantes.**)
- Utiliser des méthodes statistiques pour estimer une valeur pour une colonne à partir des valeurs présentes, puis remplacer toutes les valeurs manquantes dans la colonne avec la statistique calculée.
- C'est simple car les statistiques sont rapides à calculer et il est populaire car il s'avère souvent très efficace.

- Un jeu de données peut avoir des valeurs manquantes. Les valeurs peuvent manquer complètement ou elles peuvent être marquées avec un caractère spécial ou une valeur, comme un point d'interrogation **(?)**, une chaîne vide **[...]**, la chaîne explicite **NULL** ou undefined ou **N/A** ou **NaN**, etc.
- Ils peuvent se produire pour un certain nombre de raisons, telles qu'un dysfonctionnement de l'équipement de mesure, les changements dans la conception expérimentale pendant la collecte des données et la collation des plusieurs ensembles de données similaires mais non identiques.
- La plupart des algorithmes de Machine Learning nécessitent la présence des valeurs d'entrée numériques pour chaque ligne et colonne.
- Les valeurs manquantes peuvent causer des problèmes pour les algorithmes d'apprentissage. Pour cette raison, il faut les identifier et les remplacer par une valeur numérique (**imputation des données manquantes.**)
- Utiliser des méthodes statistiques pour estimer une valeur pour une colonne à partir des valeurs présentes, puis remplacer toutes les valeurs manquantes dans la colonne avec la statistique calculée.
- C'est simple car les statistiques sont rapides à calculer et il est populaire car il s'avère souvent très efficace.
- Les statistiques courantes calculées incluent :
  - La valeur moyenne de la colonne.
  - La valeur médiane de la colonne.
  - La valeur mode de la colonne.
  - Une valeur constante.

Dataset sur les coliques du cheval "Horse Colic Dataset"

- Ce dataset décrit les caractéristiques médicales des chevaux souffrant de coliques et s'ils ont vécu ou sont morts. Il y a 300 lignes et 26 variables d'entrée avec une variable de sortie. C'est une prédiction binaire de classification qui consiste à prédire **1** si le cheval a vécu et **2** si le cheval est mort.
- Il existe de nombreux champs que nous pourrions sélectionner pour prédire dans cet ensemble de données. Dans ce cas, nous allons prédire si le problème était chirurgical ou non **(indice de colonne 23)**, ce qui en fait une classification binaire problème.
- L'ensemble de données comporte de nombreuses valeurs manquantes pour de nombreuses colonnes où chaque valeur manquante la valeur est marquée d'un point d'interrogation (" ?").

Dataset sur les coliques du cheval "Horse Colic Dataset"

- Ce dataset décrit les caractéristiques médicales des chevaux souffrant de coliques et s'ils ont vécu ou sont morts. Il y a 300 lignes et 26 variables d'entrée avec une variable de sortie. C'est une prédiction binaire de classification qui consiste à prédire **1** si le cheval a vécu et **2** si le cheval est mort.
- Il existe de nombreux champs que nous pourrions sélectionner pour prédire dans cet ensemble de données. Dans ce cas, nous allons prédire si le problème était chirurgical ou non **(indice de colonne 23)**, ce qui en fait une classification binaire problème.
- L'ensemble de données comporte de nombreuses valeurs manquantes pour de nombreuses colonnes où chaque valeur manquante la valeur est marquée d'un point d'interrogation (" ?").

Travail à faire :

- Marquer des valeurs manquantes avec une valeur NaN. (**read csv()** et spécifier **na_valeurs** pour charger les valeurs de " ?" comme manquant, marqué d'une valeur NaN.)
- Afficher les données chargées pour confirmer que " ?" les valeurs sont marquées comme NaN.
- Énumérer chaque colonne et signaler le nombre de lignes avec des valeurs manquantes pour la colonne.

Dataset sur les coliques du cheval "Horse Colic Dataset"

- Ce dataset décrit les caractéristiques médicales des chevaux souffrant de coliques et s'ils ont vécu ou sont morts. Il y a 300 lignes et 26 variables d'entrée avec une variable de sortie. C'est une prédiction binaire de classification qui consiste à prédire **1** si le cheval a vécu et **2** si le cheval est mort.
- Il existe de nombreux champs que nous pourrions sélectionner pour prédire dans cet ensemble de données. Dans ce cas, nous allons prédire si le problème était chirurgical ou non **(indice de colonne 23)**, ce qui en fait une classification binaire problème.
- L'ensemble de données comporte de nombreuses valeurs manquantes pour de nombreuses colonnes où chaque valeur manquante la valeur est marquée d'un point d'interrogation (" ?").

Travail à faire :

- Marquer des valeurs manquantes avec une valeur NaN. (**read csv()** et spécifier **na_valeurs** pour charger les valeurs de " ?" comme manquant, marqué d'une valeur NaN.)
- Afficher les données chargées pour confirmer que " ?" les valeurs sont marquées comme NaN.
- Énumérer chaque colonne et signaler le nombre de lignes avec des valeurs manquantes pour la colonne.

```
 # summarize the horse colic dataset
from pandas import read_csv
# load dataset
dataframe = read_csv('horse-colic.csv', header=None, na_values= '?' )
# summarize the first few rows
print(dataframe.head())
# summarize the number of rows with missing values for each column
for i in range(dataframe.shape[1]):
# count number of rows with missing values
n_miss = dataframe[[i]].isnull().sum()
perc = n_miss / dataframe.shape[0] * 100
print('> %d, Missing: %d (%.1f%%)' % (i, n_miss, perc))
```

```
 # summarize the horse colic dataset
from pandas import read_csv
# load dataset
dataframe = read_csv('horse-colic.csv', header=None, na_values= '?' )
# summarize the first few rows
print(dataframe.head())
# summarize the number of rows with missing values for each column
for i in range(dataframe.shape[1]):
# count number of rows with missing values
n_miss = dataframe[[i]].isnull().sum()
perc = n_miss / dataframe.shape[0] * 100
print('> %d, Missing: %d (%.1f%%)' % (i, n_miss, perc))
```

Maintenant que nous connaissons l'ensemble de données sur les coliques du cheval qui a des valeurs manquantes, examinons comment nous pouvons utiliser l'imputation statistique.

```
 # summarize the horse colic dataset
from pandas import read_csv
# load dataset
dataframe = read_csv('horse-colic.csv', header=None, na_values= '?' )
# summarize the first few rows
print(dataframe.head())
# summarize the number of rows with missing values for each column
for i in range(dataframe.shape[1]):
# count number of rows with missing values
n_miss = dataframe[[i]].isnull().sum()
perc = n_miss / dataframe.shape[0] * 100
print('> %d, Missing: %d (%.1f%%)' % (i, n_miss, perc))
```

Maintenant que nous connaissons l'ensemble de données sur les coliques du cheval qui a des valeurs manquantes, examinons comment nous pouvons utiliser l'imputation statistique.

SimpleImputer

La librairie **scikit-learn** fournit la classe *SimpleImputer* qui prend en charge l'imputation statistique.

Imputation Statistique avec *SimpleImputer*

Le *SimpleImputer* est une transformation de données qui est d'abord configurée en fonction du type de statistique à calculer pour chaque colonne, par ex. moyenne.

Imputation Statistique avec *SimpleImputer*

Le *SimpleImputer* est une transformation de données qui est d'abord configurée en fonction du type de statistique à calculer pour chaque colonne, par ex. moyenne.

```
1 ...
2 # define imputer
3 imputer = SimpleImputer(strategy= 'mean')
```

Imputation Statistique avec *SimpleImputer*

Le *SimpleImputer* est une transformation de données qui est d'abord configurée en fonction du type de statistique à calculer pour chaque colonne, par ex. moyenne.

```
...
# define imputer
imputer = SimpleImputer(strategy= 'mean')
```

Ensuite, l'imputeur est ajusté sur un ensemble de données pour calculer la statistique pour chaque colonne.

Imputation Statistique avec *SimpleImputer*

Le *SimpleImputer* est une transformation de données qui est d'abord configurée en fonction du type de statistique à calculer pour chaque colonne, par ex. moyenne.

```
1 ...
2 # define imputer
3 imputer = SimpleImputer(strategy= 'mean')
```

Ensuite, l'imputeur est ajusté sur un ensemble de données pour calculer la statistique pour chaque colonne.

```
1 ...
2 # fit on the dataset
3 imputer.fit(X)
```

Imputation Statistique avec *SimpleImputer*

Le *SimpleImputer* est une transformation de données qui est d'abord configurée en fonction du type de statistique à calculer pour chaque colonne, par ex. moyenne.

```
...
# define imputer
imputer = SimpleImputer(strategy= 'mean')
```

Ensuite, l'imputeur est ajusté sur un ensemble de données pour calculer la statistique pour chaque colonne.

```
...
# fit on the dataset
imputer.fit(X)
```

L'imputation d'ajustement est ensuite appliquée à un ensemble de données pour créer une copie de l'ensemble de données avec tous les éléments manquants. valeurs pour chaque colonne remplacées par une valeur statistique.

Imputation Statistique avec *SimpleImputer*

Le *SimpleImputer* est une transformation de données qui est d'abord configurée en fonction du type de statistique à calculer pour chaque colonne, par ex. moyenne.

```
...
# define imputer
imputer = SimpleImputer(strategy= 'mean')
```

Ensuite, l'imputeur est ajusté sur un ensemble de données pour calculer la statistique pour chaque colonne.

```
...
# fit on the dataset
imputer.fit(X)
```

L'imputation d'ajustement est ensuite appliquée à un ensemble de données pour créer une copie de l'ensemble de données avec tous les éléments manquants. valeurs pour chaque colonne remplacées par une valeur statistique.

```
...
# transform the dataset
Xtrans = imputer.transform(X)
```

Imputation Statistique avec *SimpleImputer*

Le *SimpleImputer* est une transformation de données qui est d'abord configurée en fonction du type de statistique à calculer pour chaque colonne, par ex. moyenne.

```
...
# define imputer
imputer = SimpleImputer(strategy= 'mean')
```

Ensuite, l'imputeur est ajusté sur un ensemble de données pour calculer la statistique pour chaque colonne.

```
...
# fit on the dataset
imputer.fit(X)
```

L'imputation d'ajustement est ensuite appliquée à un ensemble de données pour créer une copie de l'ensemble de données avec tous les éléments manquants. valeurs pour chaque colonne remplacées par une valeur statistique.

```
...
# transform the dataset
Xtrans = imputer.transform(X)
```

Nous pouvons démontrer son utilisation sur l'ensemble de données sur les coliques du cheval et confirmer qu'il fonctionne en résumant le nombre total de valeurs manquantes dans l'ensemble de données avant et après la transformation.

Nous pouvons démontrer son utilisation sur l'ensemble de données sur les coliques du cheval et confirmer qu'il fonctionne en résumant le nombre total de valeurs manquantes dans l'ensemble de données avant et après la transformation.

```
# statistical imputation transform for the horse colic dataset
from numpy import isnan
from pandas import read_csv
from sklearn.impute import SimpleImputer
# load dataset
dataframe = read_csv('horse-colic.csv', header=None, na_values= '?')
# split into input and output elements
data = dataframe.values
ix = [i for i in range(data.shape[1]) if i != 23]
X, y = data[:, ix], data[:, 23]
print('Missing: %d' % sum(isnan(X).flatten()))# summarize total missing
# define imputer
imputer = SimpleImputer(strategy= 'mean ' )
# fit on the dataset
imputer.fit(X)
# transform the dataset
Xtrans = imputer.transform(X)
# summarize total missing
print('Missing: %d' % sum(isnan(Xtrans).flatten()))
```

SimpleImputer et évaluation de modèle

- C'est une bonne pratique d'évaluer les modèles d'apprentissage automatique sur un ensemble de données en utilisant k-fold cross-validation.
- Pour appliquer correctement l'imputation statistique des données manquantes et éviter les fuites de données, il est requis que les statistiques calculées pour chaque colonne soient calculées sur l'ensemble de données d'apprentissage uniquement, puis appliqué aux ensembles d'entraînement et de test pour chaque pli dans l'ensemble de données.

SimpleImputer et évaluation de modèle

- C'est une bonne pratique d'évaluer les modèles d'apprentissage automatique sur un ensemble de données en utilisant k-fold cross-validation.
- Pour appliquer correctement l'imputation statistique des données manquantes et éviter les fuites de données, il est requis que les statistiques calculées pour chaque colonne soient calculées sur l'ensemble de données d'apprentissage uniquement, puis appliqué aux ensembles d'entraînement et de test pour chaque pli dans l'ensemble de données.
- Ceci peut être réalisé en créant un pipeline de modélisation où la première étape est l'imputation statistique, la deuxième étape est le modèle.
- Ceci peut être réalisé en utilisant la classe Pipeline. Par exemple, le Pipeline ci-dessous utilise un SimpleImputer avec une stratégie "mean", suivi d'un modèle de "random forest".

SimpleImputer et évaluation de modèle

- C'est une bonne pratique d'évaluer les modèles d'apprentissage automatique sur un ensemble de données en utilisant k-fold cross-validation.
- Pour appliquer correctement l'imputation statistique des données manquantes et éviter les fuites de données, il est requis que les statistiques calculées pour chaque colonne soient calculées sur l'ensemble de données d'apprentissage uniquement, puis appliqué aux ensembles d'entraînement et de test pour chaque pli dans l'ensemble de données.
- Ceci peut être réalisé en créant un pipeline de modélisation où la première étape est l'imputation statistique, la deuxième étape est le modèle.
- Ceci peut être réalisé en utilisant la classe Pipeline. Par exemple, le Pipeline ci-dessous utilise un SimpleImputer avec une stratégie "mean", suivi d'un modèle de "random forest".

```
1 ...
2 # define modeling pipeline
3 model = RandomForestClassifier()
4 imputer = SimpleImputer(strategy= 'mean' )
5 pipeline = Pipeline(steps=[('i', imputer), ('m' , model)])
```

Nous pouvons évaluer l'ensemble de données à moyenne imputée et le pipeline de modélisation de **"random forest"** pour le jeu de données sur les coliques du cheval avec validation croisée répétée **"10-fold cross-validation"**

Nous pouvons évaluer l'ensemble de données à moyenne imputée et le pipeline de modélisation de **"random forest"** pour le jeu de données sur les coliques du cheval avec validation croisée répétée **"10-fold cross-validation"**

```
1  from numpy import mean
2  from numpy import std
3  from pandas import read_csv
4  from sklearn.ensemble import RandomForestClassifier
5  from sklearn.impute import SimpleImputer
6  from sklearn.model_selection import cross_val_score
7  from sklearn.model_selection import RepeatedStratifiedKFold
8  from sklearn.pipeline import Pipeline
9  dataframe = read_csv('horse-colic.csv', header=None, na_values= '?')
10 data = dataframe.values
11 ix = [i for i in range(data.shape[1]) if i != 23]
12 X, y = data[:, ix], data[:, 23] # split into input and output elements
13 model = RandomForestClassifier() # define modeling pipeline
14 imputer = SimpleImputer(strategy= 'mean' )
15 pipeline = Pipeline(steps=[('i', imputer), ('m', model)])
16 # define model evaluation
17 cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=3, random_state=1)
18 scores=cross_val_score(pipeline, X, y, scoring='accuracy',cv=cv,n_jobs=-1)
19 print('Mean Accuracy: %.3f (%.3f)' % (mean(scores), std(scores)))
```

Comparaison de différentes statistiques imputées

- Nous pouvons tester chaque stratégie statistique et découvrir ce qui fonctionne le mieux pour ce dataset, en comparant la moyenne, la médiane, modale (la plus fréquente) et constante (0).

Comparaison de différentes statistiques imputées

- Nous pouvons tester chaque stratégie statistique et découvrir ce qui fonctionne le mieux pour ce dataset, en comparant la moyenne, la médiane, modale (la plus fréquente) et constante (0).

```
1  dataframe = read_csv('horse-colic.csv', header=None, na_values= '?' )
2  data = dataframe.values
3  ix = [i for i in range(data.shape[1]) if i != 23]
4  X, y = data[:, ix], data[:, 23]
5  results = list() # evaluate each strategy on the dataset
6  strategies = ['mean', 'median', 'most_frequent', 'constant']
7  for s in strategies:
8  pipeline = Pipeline(steps=[('i', SimpleImputer(strategy=s)), ('m',RandomForestClassifier())])
9  cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=3, random_state=1)
10 scores = cross_val_score(pipeline, X, y, scoring= 'accuracy', cv=cv, n_jobs=-1)
11 results.append(scores)# store results
12 print('>%s %.3f (%.3f)' % (s, mean(scores), std(scores)))
13 # plot model performance for comparison
14 pyplot.boxplot(results, labels=strategies, showmeans=True)
15 pyplot.show()
```

SimpleImputer lors d'une prédiction

- Créer un pipeline de modélisation final avec la stratégie d'imputation constante et l'algorithme de "random forest".
- Faire une prédiction pour de nouvelles données.
- Il est important de noter que la ligne de nouvelles données doit marquer toutes les valeurs manquantes en utilisant la valeur **NaN**.

SimpleImputer lors d'une prédiction

- Créer un pipeline de modélisation final avec la stratégie d'imputation constante et l'algorithme de "random forest".
- Faire une prédiction pour de nouvelles données.
- Il est important de noter que la ligne de nouvelles données doit marquer toutes les valeurs manquantes en utilisant la valeur **NaN**.

```
1  ...
2  dataframe = read_csv('horse-colic.csv', header=None, na_values= '?' )
3  data = dataframe.values
4  ix = [i for i in range(data.shape[1]) if i != 23]
5  X, y = data[:, ix], data[:, 23]
6  results = list() # evaluate each strategy on the dataset
7  pipeline = Pipeline(steps=[('i', SimpleImputer(strategy='constant')),
   ↪  ('m',RandomForestClassifier())])
8  # fit the model
9  pipeline.fit(X, y)
10 # define new data
11 row = [2, 1, 530101, 38.50, 66, 28, 3, 3, nan, 2, 5, 4, 4, nan, nan, nan, 3, 5, 45.00, 8.40, nan,
   ↪  nan, 2, 11300, 00000, 00000, 2]
12 # make a prediction
13 yhat = pipeline.predict([row])
14 print( ' Predicted Class: %d ' % yhat[0])
```

Sélection automatique des variables explicatives à l'aide de la méthode RFE

Objectif : Réduire le modèle aux variables explicatives les plus pertinentes

- La méthode **RFE (Recursive Feature Elimination)** est une méthode d'emballage (wrapper) qui fonctionne en sélectionnant progressivement les variables les plus pertinentes en éliminant de façon itérative celles qui contribuent le moins au modèle.
- À chaque étape, elle entraîne le modèle sur toutes les variables disponibles, puis évalue leur importance en se basant sur des métriques comme les poids des coefficients (par exemple, en régression).
- Elle est basée sur la performance du modèle pour identifier les variables qui contribuent le moins à la prédiction.
- **Objectif** : Réduire la complexité du modèle tout en maximisant sa performance.

Fonctionnement de la Méthode RFE

1. Entraînement du modèle avec toutes les variables.
2. Évaluation de l'importance de chaque variable.
3. Suppression de la variable la moins importante.
4. Répétition du processus jusqu'à atteindre le nombre de variables souhaité.

Exemples d'algorithmes utilisés avec RFE

- Régression linéaire
- Régression logistique
- Arbres de décision
- Support Vector Machines

Exemple d'Implémentation de RFE en Python

- Nous allons utiliser un jeu de données généré artificiellement pour illustrer la sélection de variables avec **RFE**.
- Le modèle de base utilisé est une **régression linéaire**.

```
1  from sklearn.feature_selection import RFE
2  from sklearn.linear_model import LinearRegression
3  from sklearn.datasets import make_friedman1
4
5  # Chargement d'un jeu de données fictif
6  X, y = make_friedman1(n_samples=50, n_features=10, random_state=0)
7
8  # Modèle de base
9  model = LinearRegression()
10
11 # Initialisation de RFE en fixant le nombre de variables finales souhaitées
12 selector = RFE(model, n_features_to_select=5, step=1)
13 selector = selector.fit(X, y)
14 print(selector.n_features_) # nombre de var. sélectionnées
15
16 # Affichage des résultats
17 print("Variables sélectionnées :")
18 print(selector.support_)
19 print("Classement des variables (1 = moins importante) :")
20 print(selector.ranking_) # ordre de suppression
```

Analyse des Résultats

- La sortie de RFE indique quelles variables ont été sélectionnées pour rester dans le modèle.
- Le `ranking_` donne le classement des variables, où les variables de rang 1 sont les plus importantes.

Interprétation

- **Variables sélectionnées** : Celles qui sont jugées les plus pertinentes.
- **Variables éliminées** : Celles ayant un impact moindre et pouvant être omises.

Avantages et Inconvénients de RFE

Avantages

- Permet une sélection rigoureuse basée sur la performance du modèle.
- Réduction de la dimensionnalité et amélioration de l'interprétabilité.

Inconvénients

- Consommation élevée de ressources pour les grands ensembles de données.
- Nécessite de bien configurer les paramètres pour éviter le surajustement.

Réaliser le LAB 10

Dataset : **diabetes.xlsx**

Utiliser la base de données diabetes pour prédire la probabalité du dibabète en fonction des antécédents cliniques en utilisant la régression logistique et la méthode RFE (Recursive Feature Elimination) de scikit-learn

Enrichissement

- Quand on joint deux schémas de données, on doit vérifier :
  - Problème de nommage : il se peut qu'on ait des données identiques avec des nominations différentes. Par exemple, si on veut joindre deux tables de données **b1** et **b2** qui ont deux attributs avec le même sens mais différents noms **b1.numclient** et **b2.clientid**, on doit unifier les noms des attributs.
  - Conflits de valeurs : les valeurs des attributs provenant de sources différentes sont représentées différemment. Par exemple, une source de données qui représente la taille en cm et une autre qui représente la taille en pouces.
  - Redondance : les attributs qu'on puisse déduire des autres, les enregistrements identiques.
  - Types différents des attributs
  - Introduction de nouvelles valeurs manquantes et/ou aberrantes.

Transformation : normalisation et standarisation

Avant de passer à l'entraînement d'un modèle, il est préférable pour la majorité des algorithmes d'utiliser les connaissances acquises lors de l'exploration de données pour transformer les données dans un format plus propice à l'apprentissage. Cette étape de transformation est souvent ce qu'on appelle le **feature engineering**.

Normalisation :

La normalisation des données permet de ramener les données autour d'une distribution plus "standard". Pour certains types de modèles, en particulier ceux qui sont basés sur des calculs de distances comme les k-PPV ou le **clustering**, il est primordial de normaliser les données avant d'en faire l'apprentissage.

- Min-Max normalization

$$\tilde{x} = \frac{x - \min x}{\max x - \min x} \in [0, 1]$$

- Standard normalization (**Z-score**)

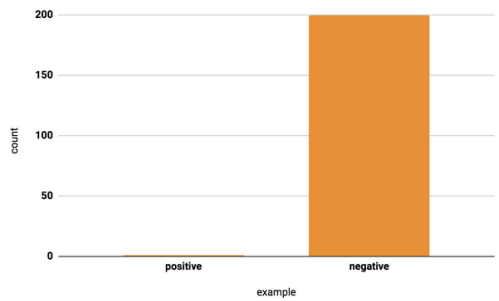
$$\tilde{x} = \frac{x - \bar{x}}{\sigma}$$

Échantillonnage et fractionnement des données

- **Données déséquilibrées**
  Dans la classification, les données d'entraînement peuvent avoir des classes avec des proportions asymétriques. Les classes qui constituent une grande (petite) proportion de données sont appelées **classes majoritaires (minoritaires)** respectivement. Le degré de déséquilibre peut aller de léger à extrême, comme le montre le tableau suivant :

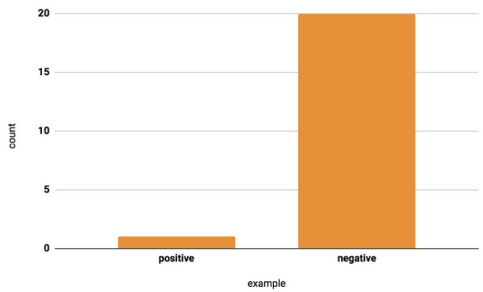
degré de déséquilibre	Proportion de classe minoritaire
léger	20-40% de données
modéré	1-20% de données
extrême	<1% de données

- Par exemple, dans le cas de la détection de fraude, les cas positifs (il y a un fraude) sont rares par rapport au cas négatif (pas de fraude). On va finir par une distribution de données comme dans le schéma suivant (200 négatifs et 1 positif).



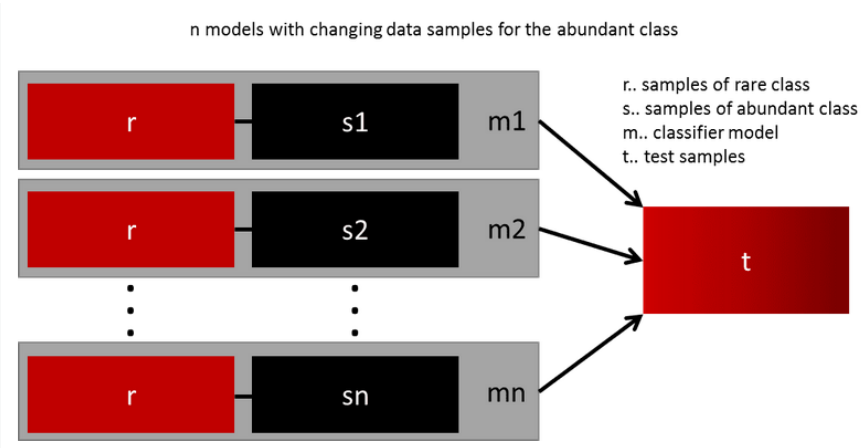
Lors de la phase d'entraînement, le système va prendre plus de temps à apprendre le cas négatif (pas de fraude) que le cas positif. Même si on a ce problème, on essaye d'entraîner notre système. Si le modèle ne donne pas de bonnes résultats lors du test, on essaye de régler ça.

- **Sous-échantillonnage**
  Le sous-échantillonnage équilibre le jeu de données en réduisant la taille de la classe majoritaire. Cette méthode est utilisée lorsque la quantité de données est suffisante, donc on peut supprimer des échantillons de la classe majoritaire au hasard. Cela peut aider le système à converger rapidement et, aussi, préserver l'espace de stockage du modèle généré. Dans l'exemple précédent, on peut diminuer la taille des négatifs 10 fois pour avoir 20 échantillons.



- **Sur-échantillonnage**
  Le sur-échantillonnage équilibre le jeu de données en augmentant la taille de la classe minoritaire. Cette méthode est utilisée lorsque la quantité de données est insuffisante. On peut augmenter la taille de la classe minoritaire en utilisant plusieurs techniques :
  - Répétition : réplication aléatoire des échantillons de la classe minoritaire
  - Techniques de bootstrap
  - SMOTE (Synthetic Minority Over-Sampling Technique)

- **Sur-échantillonnage**
  Le sur-échantillonnage équilibre le jeu de données en augmentant la taille de la classe minoritaire. Cette méthode est utilisée lorsque la quantité de données est insuffisante. On peut augmenter la taille de la classe minoritaire en utilisant plusieurs techniques :
  - Répétition : réplication aléatoire des échantillons de la classe minoritaire
  - Techniques de bootstrap
  - SMOTE (Synthetic Minority Over-Sampling Technique)
- **Ré-échantillonage en ensembles de données équilibrées**
  Dans ce cas, on peut créer plusieurs ensembles de données en divisant la classe majoritaire sur plusieurs ensemble et fusionnant la classe minoritaire avec chaque ensemble. Ensuite, on peut entrainer plusieurs modèles sur ces ensembles.



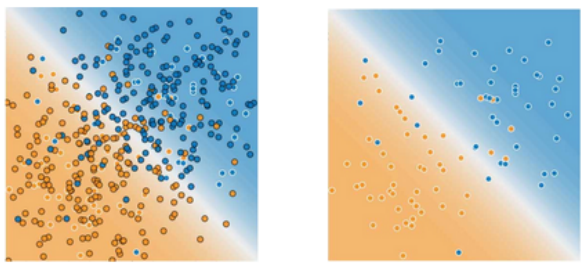
Fractionnement des données

Dans le cas d'apprentissage supervisé, il ne faut pas entraîner et tester votre modèle sur les mêmes données. Le système doit être tester sur des données qu'il n'a pas encore rencontré pour tester s'il a bien généralisé à partir des données qu'il a vu déjà. Donc, on a besoin de diviser notre ensemble de données sur deux sous-ensembles :

- Données d'entraînement avec une majorité des échantillons **(70-80%)**
- Données de test avec une minorité des échantillons **(30-20%)**

Lors du fractionnement, il faut prendre en considération ces deux conditions :

1. Les données de test sont suffisantes pour avoir des résultats significatifs.
2. Les données de test sont **représentatives**. Il ne faut pas prendre un ensemble avec des caractéristiques différentes de celles des données d'entraînement.



Fractionnement des données

Parfois, lorsqu'on teste notre modèle et on rend compte qu'il donne des résultats médiocres, on veut refaire la phase d'entrainement en changeant les paramètres de notre système. En faisant ça plusieurs fois, notre modèle sera ajusté aux données de test. Pour faire face à ce problème, on peut créer un **troisième ensemble pour la validation**.
Les processus d'apprentissage sera comme suit :

1. Entraîner le système sur l'ensemble des données d'entraînement pour avoir un modèle
2. Tester le modèle sur l'ensemble des **données de validation**
   - Si la performance est bonne, aller vers l'étape suivante
   - Sinon, changer les paramètres de votre système et refaire l'étape précédente
3. Tester le modèle sur l'ensemble de test pour calculer la performance de votre système et comparer avec les autres systèmes existants.

Outline