

# Data Science-Fondations

## Prétraitement des données

Mustapha El Ossmani

**Filière IADS**  
**ENSAM-Meknès**

Année universitaire 2023/2024





## 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.

- **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.

**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**, ...

## 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.





**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](http://continuum.io/downloads)
- Pour ce cours, nous utiliserons la version Python 3.5. Anaconda comprend **Spyder** (Scientific **PY**thon **D**éveloppement **E**nviRonment) comme environnement de développement intégré.

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/>

## 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.

## 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**

# Réaliser le TP 01



# Prétraitement des 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 les 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



- **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 de data à partir de bases de données SQL
  - Importation de data à partir de SPSS, Stata, and SAS
  - Importation de data de R
  - Importation de données : Satellite, Réseaux, Web, Media, ..

# 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

- 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 :

```
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 :

```
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 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 :

```
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. 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']
```



# Importation à partir de base 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**

```
1 server = "pdcc.c9sqqzd5fulv.us-west-2.rds.amazonaws.com"
2 user = "pdccuser"
3 password = "pdccpass"
4 database = "pdccctest"
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.c9sqzdz5fulv.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.

```

1 newcolorder = ['studentid', 'gradeperiod1', 'gradeperiod2',
2               'gradeperiod3', 'school', 'sex', 'age', 'famsize',
3               'mothereducation', 'fathereducation', 'traveltime',
4               'studytime', 'freetime', 'failures', 'famrel',
5               'goout']
6 studentmath = studentmath[newcolorder]
7 studentmath.studentid.count()
8 studentmath.studentid.nunique()
9 studentmath.set_index('studentid', inplace=True)

```

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

```

1 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

```

1 setvalues={"famrel":{1:"1:very bad",2:"2:bad",3:"3:neutral",
2   4:"4:good",5:"5:excellent"},
3   "freetime":{1:"1:very low",2:"2:low",3:"3:neutral",
4   4:"4:high",5:"5:very high"},
5   "goout":{1:"1:very low",2:"2:low",3:"3:neutral",
6   4:"4:high",5:"5:very high"},
7   "mothereducation":{0:np.nan,1:"1:k-4",2:"2:5-9",
8   3:"3:secondary ed",4:"4:higher ed"},
9   "fathereducation":{0:np.nan,1:"1:k-4",2:"2:5-9",
10  3:"3:secondary ed",4:"4:higher ed"}}
11
12 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 :

```

1 setvalueskeys = [k for k in setvalues]
2 studentmath[setvalueskeys].memory_usage(index=False)
3
4 for col in studentmath[setvalueskeys].columns:
5     studentmath[col] = studentmath[col].astype('category')
6     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 :

```
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 :

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

# 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.
- **Les meilleures bibliothèques Python pour le web scraping :**
  - **Requêtes**
    - La bibliothèque **Requests** vous permet d'exécuter des requêtes HTTP en Python.
    - Requests facilite l'envoi de requêtes HTTP, notamment par rapport à la bibliothèque HTTP standard de Python.
    - Requests joue un rôle clé dans les projets de web scraping sous Python.

En effet, pour récupérer les données contenues dans une page web, vous devez d'abord les récupérer via une requête HTTP GET. Vous pouvez installer des requêtes avec la commande :

```
1 pip install requests
```

- **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
```

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

```
1 page = requests.get('https://quotes.toscrape.com')
```

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 **page.text** au constructeur BeautifulSoup() pour extraire les données de la page web :

```
1 soup = BeautifulSoup(page.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 **page.text** avec l'interpréteur **html.parser** intégré dans Python.

# Sélection d'éléments HTML avec BeautifulSoup

- BeautifulSoup 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

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

- Par Id

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

- Par texte :

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

- Par attribut :

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

- Par classe :

```
1 # find all the centered elements on the page
2 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 :

```
1 # get all "li" elements in the ".navbar" element
2 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')
```

### 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 ».

# 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.

```
1 rows = bs.find('table', {'id':'tblDeaths'}).tbody.find_all('tr')
2 datarows = []
3 labelrows = []
4 for row in rows:
5     rowlabels = row.find('th').get_text()
6     cells = row.find_all('td', {'class':'data'})
7     if (len(rowlabels)>3):
8         labelrows.append(rowlabels)
9     if (len(cells)>0):
10        cellvalues = [j.get_text() for j in cells]
11        datarows.append(cellvalues)
12 pprint.pprint(datarows[0:2])
13 pprint.pprint(labelrows[0:2])
14 for i in range(len(datarows)):
15     datarows[i].insert(0, labelrows[i])
16 pprint.pprint(datarows[0:1])
```

- Chargement des données dans pandas

```
1 totaldeaths = pd.DataFrame(datarows, columns=labelcols)
2 totaldeaths.head()
3 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

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

- 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.

- ➊ Identifier les colonnes qui contiennent une seule valeur
- ➋ Supprimer les colonnes qui contiennent une seule valeur
- ➌ Considérer les colonnes qui ont très peu de valeurs
- ➍ Supprimez les colonnes qui ont une faible variance
- ➎ Identifier les lignes qui contiennent des données en double
- ➏ Supprimer les lignes qui contiennent des données en double

## 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.

## ● Identifier les colonnes qui contiennent une seule valeur

- Les 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.

Ici, une valeur unique signifie que chaque ligne de cette colonne a la même valeur.

- Vous pouvez détecter les lignes qui ont cette propriété en utilisant la fonction `unique()` de NumPy qui indiquera le nombre de valeurs uniques dans chaque colonne.

```
1 from numpy import loadtxt
2 from numpy import unique
3 data = loadtxt('oil-spill.csv' , delimiter= ',' )
4 for i in range(data.shape[1]):
5     print(i, len(unique(data[:, i])))
```

Une approche plus simple consiste à utiliser la fonction `nunique()` de pandas qui fait le travail difficile pour toi. Ci-dessous, le même exemple utilisant la fonction Pandas.

```
1 from pandas import read_csv
2 df = read_csv('oil-spill.csv' , header=None)
3 print(df.nunique())
```

## ● Supprimer les colonnes qui contiennent une seule valeur

- Les colonnes sont relativement faciles à supprimer d'un tableau NumPy ou DataFrame Pandas.
- L'approche consiste à enregistrer toutes les colonnes qui ont une seule valeur unique, puis les supprimer en appelant la fonction drop()

```
1 counts = df.unique()
2 to_del = [i for i,v in enumerate(counts) if v == 1]
3 print(to_del)
4 df.drop(to_del, axis=1, inplace=True)
5 print(df.shape)
```

## ● Considérer les colonnes qui ont très peu de valeurs

- Dans la section précédente, nous avons vu que certaines colonnes de l'exemple de jeu de données avaient très peu de valeurs.
- Par exemple, certaines colonnes n'avaient que 2, 4 et 9 valeurs uniques.
- En tant que tel, n'avoir que 2, 4 ou 9 valeurs numériques uniques dans une colonne peut être surprenant.
- Nous pouvons faire référence à ces colonnes ou prédicteurs comme des prédicteurs de variance proche de zéro, comme leur variance n'est pas nulle, mais un très petit nombre proche de zéro.

Pour aider à mettre en évidence les colonnes de ce type, vous pouvez calculer le nombre de valeurs uniques pour chaque variable en pourcentage du total nombre de lignes dans le jeu de données. Faisons cela manuellement en utilisant NumPy.

L'exemple complet est donné par :

```
1 # summarize the number of unique values in each column
2 for i in range(data.shape[1]):
3     num = len(unique(data[:, i]))
4     percentage = float(num)/data.shape[0]*100
5     print(' %d, %d, %.1f%% ' % (i, num, percentage))
```

Nous pouvons mettre à jour l'exemple pour ne garder que les variables qui ont des valeurs uniques qui représentent moins de 1 % du nombre de lignes.

```
1 # summarize the number of unique values in each column
2 for i in range(data.shape[1]):
3     num = len(unique(data[:, i]))
4     percentage = float(num)/data.shape[0]*100
5     if percentage < 1:
6         print(' %d, %d, %.1f%% ' % (i, num, percentage))
```

Par exemple, si nous voulons supprimer les 11 colonnes dont les valeurs uniques sont inférieures à 1% des lignes, :

```
1 # delete columns where number of unique values is less than 1% of the rows
2 from pandas import read_csv
3 # load the dataset
4 df = read_csv( ' oil-spill.csv ', header=None)
5 print(df.shape)
6 # get number of unique values for each column
7 counts = df.nunique()
8 # record columns to delete
9 to_del = [i for i,v in enumerate(counts) if (float(v)/df.shape[0]*100) < 1]
10 print(to_del)
11 # drop useless columns
12 df.drop(to_del, axis=1, inplace=True)
13 print(df.shape)
```

## ● Supprimez les colonnes qui ont une faible variance

- Une autre approche du problème de la suppression des colonnes ayant peu de valeurs uniques est de considérer la variance de la colonne.
- Rappelons que la variance est une statistique calculée sur une variable en tant que différence moyenne au carré des valeurs de l'échantillon par rapport à la moyenne.
- La variance peut être utilisée comme un filtre pour identifier les colonnes à supprimer de l'ensemble de données. Une colonne qui a une seule valeur a une variance de 0.0, et une colonne qui a très peu de valeurs uniques peut avoir une petite variance.
- La classe **VarianceThreshold** de la bibliothèque **scikit-learn** prend en charge ce type de sélection de caractéristiques.
- Une instance de la classe peut être créée et nous pouvons spécifier l'argument du seuil, qui a la valeur par défaut 0.0 pour éliminer les colonnes avec une seule valeur.
- Elle peut ensuite être ajustée et appliquée à un ensemble de données en appelant la fonction **fit\_transform()** pour créer une version transformée de l'ensemble de données où les colonnes dont la variance est inférieure au seuil ont été supprimées automatiquement.



```

1 # example of applying the variance threshold for feature selection
2 from pandas import read_csv
3 from sklearn.feature_selection import VarianceThreshold
4 # load the dataset
5 df = read_csv('oil-spill.csv' , header=None)
6 # split data into inputs and outputs
7 data = df.values
8 X = data[:, :-1]
9 y = data[:, -1]
10 print(X.shape, y.shape)
11 # define the transform
12 transform = VarianceThreshold()
13 # transform the input data
14 X_sel = transform.fit_transform(X)
15 print(X_sel.shape)

```

Nous pouvons étendre cet exemple et voir ce qui se passe lorsque nous utilisons différents seuils. Nous pouvons définir une séquence de seuils de 0.0 à 0.5 avec un pas de 0.05, par exemple 0.0, 0.05, 0.1, etc.

```

1 thresholds = arange(0.0, 0.55, 0.05)

```

Nous pouvons alors indiquer le nombre de caractéristiques dans l'ensemble de données transformé pour chaque seuil donné.

```
1 # apply transform with each threshold
2 results = list()
3 for t in thresholds:
4     # define the transform
5     transform = VarianceThreshold(threshold=t)
6     # transform the input data
7     X_sel = transform.fit_transform(X)
8     # determine the number of input features
9     n_features = X_sel.shape[1]
10    print(' >Threshold=%.2f, Features=%d' % (t, n_features))
11    # store the result
12    results.append(n_features)
```

Enfin, nous pouvons tracer les résultats de la comparaison entre le seuil de variance et le nombre de caractéristiques sélectionnées

```

1  from numpy import arange
2  from pandas import read_csv
3  from sklearn.feature_selection import VarianceThreshold
4  from matplotlib import pyplot
5  df = read_csv('oil-spill.csv' header=None)
6  data = df.values
7  X = data[:, :-1]
8  y = data[:, -1]
9  print(X.shape, y.shape)
10 thresholds = arange(0.0, 0.55, 0.05)
11 results = list()
12 for t in thresholds:
13     transform = VarianceThreshold(threshold=t)
14     X_sel = transform.fit_transform(X)
15     # determine the number of input features
16     n_features = X_sel.shape[1]
17     print(' >Threshold=%.2f, Features=%d ', (t, n_features))
18     # store the result
19     results.append(n_features)
20     # plot the threshold vs the number of selected features
21     pyplot.plot(thresholds, results)
22     pyplot.show()

```

# Identifier les lignes qui contiennent des données en double

- Les lignes qui contiennent des données identiques peuvent être inutiles au processus de modélisation, voire dangereusement trompeuses lors de l'évaluation du modèle.
- Ici, une ligne dupliquée est une ligne où chaque valeur de chaque colonne de cette ligne apparaît dans le même ordre dans une autre ligne.

La fonction **uplicated()** de Pandas indique si une ligne donnée est dupliquée ou non.

```
1 # locate rows of duplicate data
2 from pandas import read_csv
3 # load the dataset
4 df = read_csv('iris.csv', header=None)
5 # calculate duplicates
6 dups = df.duplicated()
7 # report if there are any duplicates
8 print(dups.any())
9 # list all duplicate rows
10 print(df[dups])
```

# Supprimer les lignes qui contiennent des données en double

- Les lignes de données en double doivent probablement être supprimées de votre ensemble de données avant la modélisation.
- Il existe de nombreuses façons d'y parvenir, mais Pandas fournit la fonction **drop\_duplicates()** qui permet de supprimer la ligne dupliquée.

```
1 # delete rows of duplicate data from the dataset
2 from pandas import read_csv
3 # load the dataset
4 df = read_csv('iris.csv', header=None)
5 print(df.shape)
6 # delete duplicate rows
7 df.drop_duplicates(inplace=True)
8 print(df.shape)
```

- 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 ?

## Définition

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  $x(1), x(2), \dots, x(n)$  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 ?

- 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 observe !**

- Dans ce cas de figure, vous pouvez garder ces valeurs aberrantes !

# 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.

- 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)))
```

## ● 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 :

- **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**.

```

1 ...
2 # calculate summary statistics
3 data_mean, data_std = mean(data), std(data)
4 # define outliers
5 cut_off = data_std * 3
6 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]

```

1 ...
2 # identify outliers
3 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 :

```

1 ...
2 # remove outliers
3 outliers_removed = [x for x in data if x > lower and x < upper]

```

Le code de l'exemple complet est répertorié ci-dessous :

```
1 # identify outliers with standard deviation
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 # generate univariate observations
9 data = 5 * randn(10000) + 50
10 # calculate summary statistics
11 data_mean, data_std = mean(data), std(data)
12 # define outliers
13 cut_off = data_std * 3
14 lower, upper = data_mean - cut_off, data_mean + cut_off
15 # identify outliers
16 outliers = [x for x in data if x < lower or x > upper]
17 print('Identified outliers: %d', len(outliers))
18 # remove outliers
19 outliers_removed = [x for x in data if x >= lower and x <= upper]
20 print('Non-outlier observations: %d', len(outliers_removed))
```

### 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 aberrantes. 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 avons 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 boîte. 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.

```

1 ...
2 # calculate interquartile range
3 q25, q75 = percentile(data, 25), percentile(data, 75)
4 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

```
1 ...  
2 # calculate the outlier cutoff  
3 cut_off = iqr * 1.5  
4 lower, upper = q25 - cut_off, q75 + cut_off
```

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

```
1 ....  
2 # identify outliers  
3 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.

```
1 ...  
2 # remove outliers  
3 outliers_removed = [x for x in data if x > lower and x < upper]
```

Le code de l'exemple complet est répertorié ci-dessous.

```
1 # identify outliers with interquartile range
2 from numpy.random import seed
3 from numpy.random import randn
4 from numpy import percentile
5 # seed the random number generator
6 seed(1)
7 # generate univariate observations
8 data = 5 * randn(10000) + 50
9 # calculate interquartile range
10 q25, q75 = percentile(data, 25), percentile(data, 75)
11 iqr = q75 - q25
12 print('Percentiles: 25th=%.3f, 75th=%.3f, IQR=%.3f', (q25, q75, iqr))
13 # calculate the outlier cutoff
14 cut_off = iqr * 1.5
15 lower, upper = q25 - cut_off, q75 + cut_off
16 # identify outliers
17 outliers = [x for x in data if x < lower or x > upper]
18 print('Identified outliers: %d ', len(outliers))
19 # remove outliers
20 outliers_removed = [x for x in data if x >= lower and x <= upper]
21 print('Non-outlier observations: %d ', len(outliers_removed))
```

# 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**.

## Exemple de dataset de logement de Boston :

- Illustrer la méthode **LocalOutlierFactor** sur le data "housing.csv".
- ❶ Tout d'abord, nous pouvons charger le jeu de données sous la forme d'un tableau NumPy, le séparer en variables d'entrée et de sortie puis divisez-le en ensembles de données d'entraînement et de test.

```
1 from pandas import read_csv
2 from sklearn.model_selection import train_test_split
3 # load the dataset
4 df = read_csv('housing.csv', header=None)
5 # retrieve the array
6 data = df.values
7 # split into input and output elements
8 X, y = data[:, :-1], data[:, -1]
9 # summarize the shape of the dataset
10 print(X.shape, y.shape)
11 # split into train and test sets
12 X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, random_state=1)
13 # summarize the shape of the train and test sets
14 print(X_train.shape, X_test.shape, y_train.shape, y_test.shape)
```

Il s'agit d'un problème de modélisation prédictive de régression (prédire une valeur numérique). Dans ce cas, nous ajusterons un algorithme de régression linéaire et évaluer les performances du modèle en formant le modèle sur l'ensemble de données de test et en faisant une prédiction puis une évaluation sur les données de test et évaluer les prédictions à l'aide de l'erreur absolue moyenne (MAE).

```
1 from pandas import read_csv
2 from sklearn.model_selection import train_test_split
3 from sklearn.linear_model import LinearRegression
4 from sklearn.metrics import mean_absolute_error
5 df = read_csv('housing.csv', header=None) # load the dataset
6 data = df.values # retrieve the array
7 X, y = data[:, :-1], data[:, -1] # split into input and output elements
8 X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, random_state=1) # split
   → into train and test sets
9 model = LinearRegression()
10 model.fit(X_train, y_train) # fit the model
11 # evaluate the model
12 yhat = model.predict(X_test)
13 # evaluate predictions
14 mae = mean_absolute_error(y_test, yhat)
15 print('MAE: %.3f ' % mae)
```

- Ensuite, nous pouvons essayer de supprimer les valeurs aberrantes de l'ensemble de données d'entraînement.
- On s'attend à ce que les valeurs aberrantes amènent le modèle de régression à apprendre une compréhension biaisée du problème, et que la suppression de ces valeurs aberrantes de l'ensemble d'apprentissage permettra un modèle plus efficace à apprendre.
- Nous pouvons y parvenir en définissant le modèle `LocalOutlierFactor` et en l'utilisant pour faire une prédiction sur l'ensemble de données d'entraînement, en marquant chaque ligne de l'ensemble de données d'entraînement comme normale (1) ou une valeur aberrante (-1).

```
1 ...  
2 # identify outliers in the training dataset  
3 lof = LocalOutlierFactor()  
4 yhat = lof.fit_predict(X_train)
```

Nous pouvons ensuite utiliser ces prédictions pour supprimer toutes les valeurs aberrantes de l'ensemble de données d'apprentissage.

```
1 ...  
2 # select all rows that are not outliers  
3 mask = yhat != -1  
4 X_train, y_train = X_train[mask, :], y_train[mask]
```



Nous pouvons ensuite ajuster et évaluer le modèle comme d'habitude.

```
1 # evaluate model on training dataset with outliers removed
2 from pandas import read_csv
3 from sklearn.model_selection import train_test_split
4 from sklearn.linear_model import LinearRegression
5 from sklearn.neighbors import LocalOutlierFactor
6 from sklearn.metrics import mean_absolute_error
7 df = read_csv('housing.csv' , header=None)
8 data = df.values
9 X, y = data[:, :-1], data[:, -1]
10 X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.33,random_state=1)
11 print(X_train.shape, y_train.shape)
12 lof = LocalOutlierFactor() # identify outliers in the training dataset
13 yhat = lof.fit_predict(X_train)
14 mask = yhat != -1 # select all rows that are not outliers
15 X_train, y_train = X_train[mask, :], y_train[mask]# summarize the shape
16 print(X_train.shape, y_train.shape)
17 model = LinearRegression()
18 model.fit(X_train, y_train) # fit the model
19 yhat = model.predict(X_test) # evaluate the model
20 mae = mean_absolute_error(y_test, yhat) # evaluate predictions
21 print('MAE: %.3f ' % mae)
```

# Comment marquer et supprimer les données 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.
- 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.
- Pour bien gérer les valeurs manquantes, on doit savoir :
  - 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.

# 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

```
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 :

- ❶ Plasma glucose concentration
- ❷ Diastolic blood pressure
- ❸ Triceps skinfold thickness
- ❹ 2-Hour serum insulin
- ❺ 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**.

```

1 # example of summarizing the number of missing values for each variable
2 from pandas import read_csv
3 # load the dataset
4 dataset = read_csv('pima-indians-diabetes.csv', header=None)
5 # count the number of missing values for each column
6 num_missing = (dataset[[1,2,3,4,5]] == 0).sum()
7 # report the results
8 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.

```
1 # example of marking missing values with nan values
2 from numpy import nan
3 from pandas import read_csv
4 # load the dataset
5 dataset = read_csv( ' pima-indians-diabetes.csv ' , header=None)
6 # replace ' 0 ' values with ' nan '
7 dataset[[1,2,3,4,5]] = dataset[[1,2,3,4,5]].replace(0, nan)
8 # count the number of nan values in each column
9 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.

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 ?



```

1 # example where missing values cause errors
2 from numpy import nan
3 from pandas import read_csv
4 from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
5 from sklearn.model_selection import KFold
6 from sklearn.model_selection import cross_val_score
7 # load the dataset
8 dataset = read_csv('pima-indians-diabetes.csv' , header=None)
9 # replace '0' values with 'nan'
10 dataset[[1,2,3,4,5]] = dataset[[1,2,3,4,5]].replace(0, nan)
11 # split dataset into inputs and outputs
12 values = dataset.values
13 X = values[:,0:8]
14 y = values[:,8]
15 # define the model
16 model = LinearDiscriminantAnalysis()
17 # define the model evaluation procedure
18 cv = KFold(n_splits=3, shuffle=True, random_state=1)
19 # evaluate the model
20 result = cross_val_score(model, X, y, cv=cv, scoring= 'accuracy' )
21 # report the mean performance
22 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.

```
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.

```
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 Learning.
- 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.
- 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 ("?").

## 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.

```

1  # summarize the horse colic dataset
2  from pandas import read_csv
3  # load dataset
4  dataframe = read_csv('horse-colic.csv', header=None, na_values= '?' )
5  # summarize the first few rows
6  print(dataframe.head())
7  # summarize the number of rows with missing values for each column
8  for i in range(dataframe.shape[1]):
9      # count number of rows with missing values
10     n_miss = dataframe[[i]].isnull().sum()
11     perc = n_miss / dataframe.shape[0] * 100
12     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.

```
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)
```

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.

```
1 ...  
2 # transform the dataset  
3 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.

```
1 # statistical imputation transform for the horse colic dataset
2 from numpy import isnan
3 from pandas import read_csv
4 from sklearn.impute import SimpleImputer
5 # load dataset
6 dataframe = read_csv('horse-colic.csv', header=None, na_values = '?')
7 # split into input and output elements
8 data = dataframe.values
9 ix = [i for i in range(data.shape[1]) if i != 23]
10 X, y = data[:, ix], data[:, 23]
11 print('Missing: %d' % sum(isnan(X).flatten()))# summarize total missing
12 # define imputer
13 imputer = SimpleImputer(strategy = 'mean ' )
14 # fit on the dataset
15 imputer.fit(X)
16 # transform the dataset
17 Xtrans = imputer.transform(X)
18 # summarize total missing
19 print('Missing: %d' % sum(isnan(Xtrans).flatten()))
```

- 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"**

```
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).

```
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**.

```
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')),
8   → ('m', RandomForestClassifier())])
9 # fit the model
10 pipeline.fit(X, y)
11 # define new data
12 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,
13   → nan, 2, 11300, 00000, 00000, 2]
14 # make a prediction
15 yhat = pipeline.predict([row])
16 print( ' Predicted Class: %d ' % yhat[0])
```

### Objectif : Réduire le modèle aux variables explicatives les plus pertinentes

- Nous allons utiliser la base de données diabetes pour prédire la probabilité du diabète en fonction des antécédents cliniques en utilisant le logit et la méthode RFE (Recursive Feature Elimination) de scikit-learn .
- La sélection de variables présente plusieurs avantages : interprétation, déploiement (moins de var. à renseigner)

**Remarque :** Implémenter la méthode RFE de scikit-learn qui élimine au fur et à mesure les poids les plus faibles en valeur absolue et s'arrête quand on arrive à la moitié ou à un nombre spécifié de variables.

```
1 df=pandas.read_excel("diabetes.xlsx",sheetname="data")
2 print(df.head(n=5))
3 #transformation en matrice numpy  seul reconnu par scikit-learn
4 data=numpy.matrix(df) #
5 X = data[:,0:8] # X Matrice des variables explicatives (8 premières)
6 y = data[:,8] # y vecteur de la variable à prédire
```

Utilisation du module `model_selection` de scikit-learn pour subdiviser l'échantillon en un échantillon d'apprentissage et de validation (`éch.test = 300` ; `éch.app = 468`)

```
1 from sklearn.model_selection import train_test_split
2 X_app,X_test,y_app,y_test=train_test_split(X,y,test_size=300,random_state=0)
3 print(X_app.shape,X_test.shape,y_app.shape,y_test.shape)
```

Importer la classe `LogisticRegression`

```
1 from sklearn.linear_model import LogisticRegression
```

Création d'une instance de la classe

```
1 lr = LogisticRegression()
```

Algorithme de sélection de var.

```
1 from sklearn.feature_selection import RFE
2 selecteur = RFE(estimator=lr)
```

Lancer la recherche

```
1 sol=selecteur.fit(X, y)
2 print(sol.n_features_) # 4 = 8/2 nombre de var. sélectionnées
```

Liste des variables sélectionnées

```
1 print(sol.support_) # [ True  True False False False  True  True False]
2 print(sol.ranking_) # [1 1 2 4 5 1 1 3] # ordre de suppression
```

## Réduction de la base d'app. aux var. sélectionnées

```
1 X_new_app = X_app[:,sol.support_]# utiliser le filtre booléen sol.support_  
2 print(X_new_app.shape) # (468, 4) 4 variables restantes
```

## Construction du modèle sur les explicatives sélectionnées

```
1 modele_sel = lr.fit(X_new_app,y_app)
```

## Réduction de la base test aux mêmes variables

```
1 X_new_test = X_test[:,sol.support_]   
2 print(X_new_test.shape) # (300, 4)
```

## Prédiction du modèle réduit sur l'éch. test

```
1 y_pred_sel = modele_sel.predict(X_new_test)
```

## Évaluation

```
1 from sklearn import metrics  
2 print(metrics.accuracy_score(y_test,y_pred_sel)) # 0.787 # Aussi bien  
3 #(presque, 0.793) que le modèle initial, mais avec moitié moins de variables.
```



- 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 standardisation

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**)

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

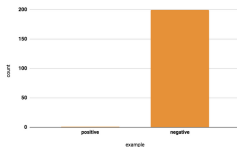
# Échantillonnage et fractionnement des données

- **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

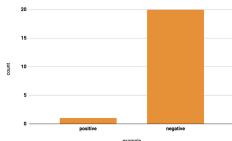
- Par exemple, dans le cas de la détection de fraude, les cas positifs (il y a un fraude) sont rares par rapport au cas négatif (pas de fraude). On va finir par une distribution de données comme dans le schéma suivant (200 négatifs et 1 positif).



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 bons 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.



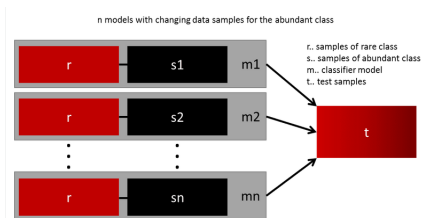
- **Sur-échantillonnage**

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)

- **Ré-échantillonnage 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 entraîner 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 testé sur des données qu'il n'a pas encore rencontrées pour tester s'il a bien généralisé à partir des données qu'il a vues déjà. Donc, on a besoin de diviser notre ensemble de données en 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.



Parfois, lorsqu'on teste notre modèle et on rend compte qu'il donne des résultats médiocres, on veut refaire la phase d'entraînement 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 :

- ❶ Entraîner le système sur l'ensemble des données d'entraînement pour avoir un modèle
- ❷ 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
- ❸ Tester le modèle sur l'ensemble de test pour calculer la performance de votre système et comparer avec les autres systèmes existants.