

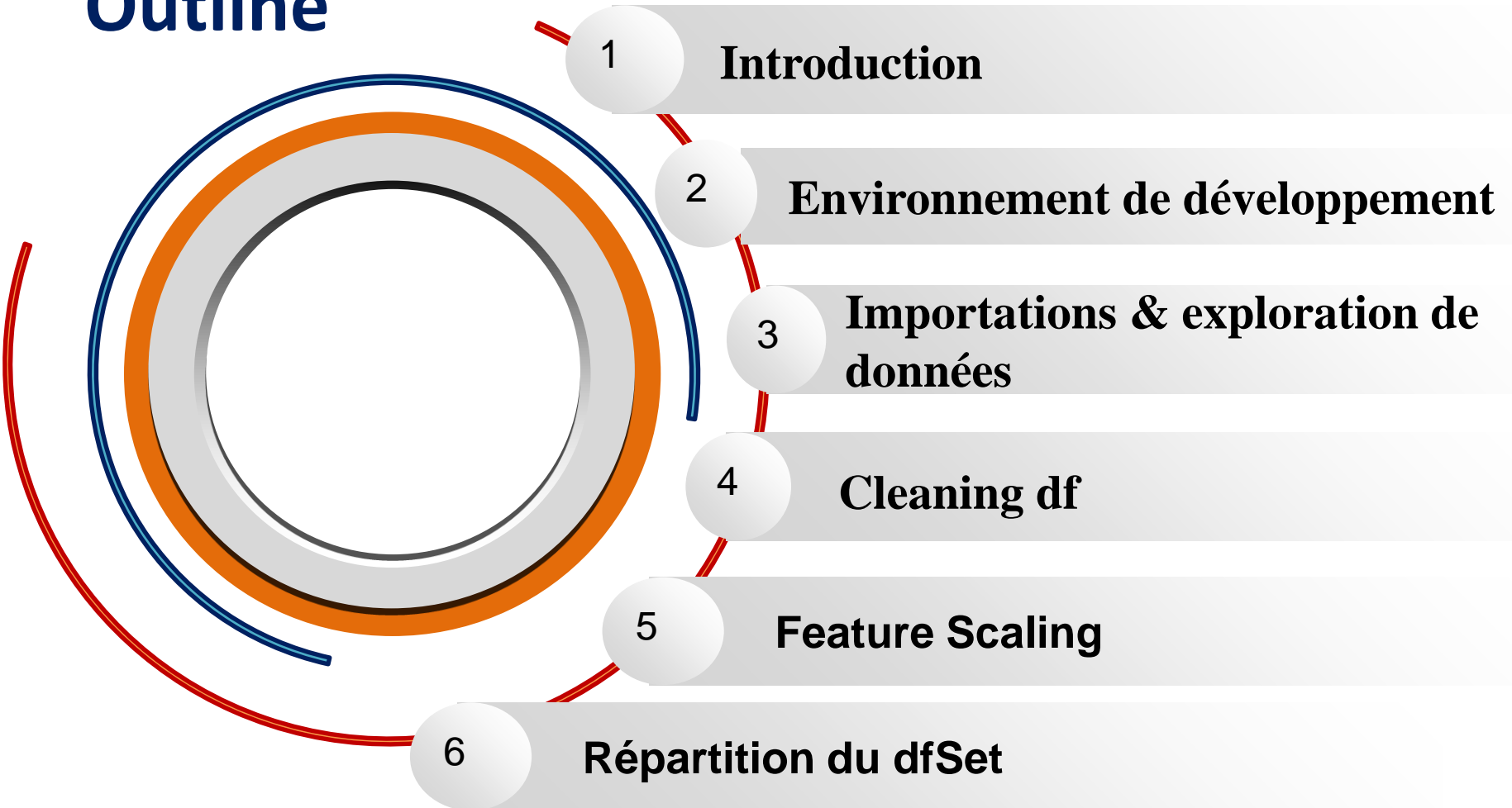
## Chapitre II : Pré-Traitement de données

*Enseignantes :*

**Naïma Halouani**  
**Hounaïda Moalla**  
**ISET Sfax**



# Outline



La préparation des données **peut nécessiter** les étapes suivantes :

1. Importations et exploration des données

- Importation de Librairies
- Importation de la dataset
- Exploration des données
- Visualisation

2. Cleaning df

- Ajouter les données manquantes
- Traiter les valeurs aberrantes
- Catégorisation des données

3. Mise à l'échelle des valeurs : Normalisation / Standardisation

4. Répartition de la base en Training Set and Test Set

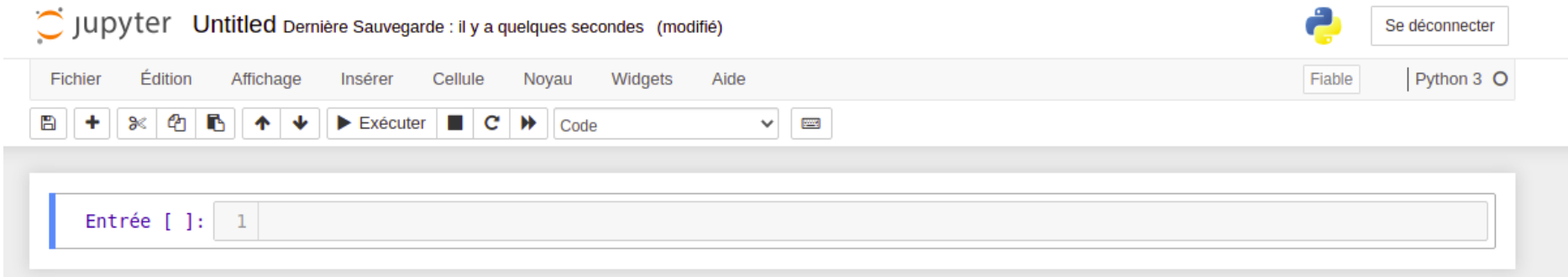
### 1

### Développement en local

Pour lancer le notebook Python, dans le terminal et dans votre dossier de travail, tapez la commande `jupyter notebook`.

```
jupyter notebook
```

Une fenêtre va se lancer dans votre navigateur pour ouvrir l'application Jupyter.



Créer un nouveau notebook Python et commencer à taper votre code dans les cellules.

### 2

### Développement sur cloud

Le Cloud Google Colab :



- Un service cloud gratuit et supporte les processeurs CPU, GPU et TPU.
- Des libraries pré-installées : Keras, TensorFlow, PyTorch, OpenCV, ...



Limite de 12 heures d'utilisation de la machine virtuelle.

## 2 Environnements de développement

Utiliser Google Colab

1 URL : *<https://colab.research.google.com>*

**Bienvenue dans Colaboratory**

Fichier Modifier Affichage Insérer Exécution Outils Aide

+ Code + Texte Copier sur Drive

Connecter Modification

**Table des matières** Extraits de code FichiersX

- Présentation de Colaboratory
- Premiers pas
- Autres ressources
- Exemples de machine learning : projet Seedbank
- + Section

**Bienvenue dans Colaboratory !**

Colaboratory est un environnement de notebook Jupyter qui ne nécessite aucune configuration et qui s'exécute entièrement dans le cloud.

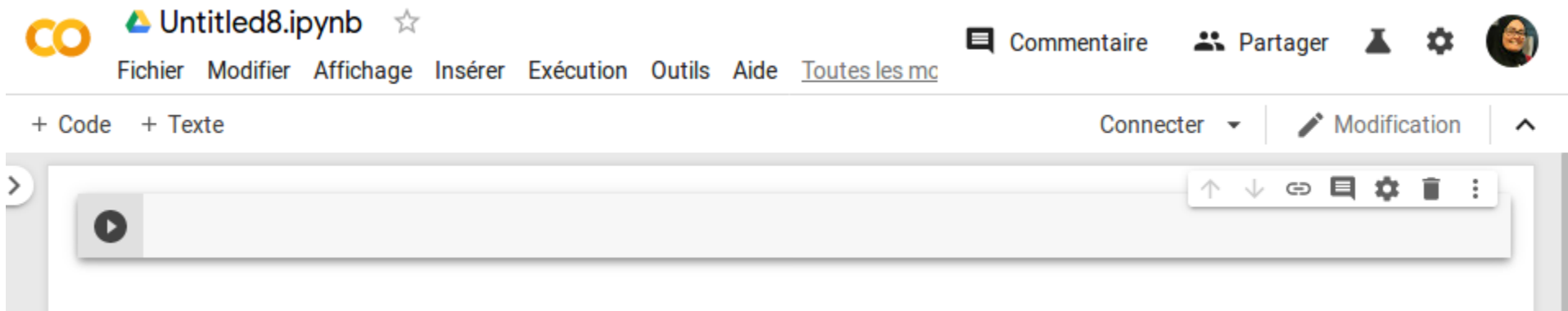
Il vous permet d'écrire et d'exécuter du code, de sauvegarder et partager vos analyses, et d'accéder à de puissantes ressources informatiques. Tout cela gratuitement, depuis votre navigateur.

**Présentation de Colaboratory**

Cette vidéo de trois minutes offre un aperçu des fonctionnalités principales de Colaboratory :

## 2 Environnements de développement

2 Click ***File/New Python 3 notebook***



Un notebook Jupyter est ouvert.

3 Renommer votre fichier et commencer à coder votre programme.

**NB :** Votre code sera sauvegardé sur votre cloud.

## 1

## Importation de Librairies

```
import numpy as np  
import pandas as pd  
import matplotlib as plt  
Import seaborn as sns
```

## 2

## Importation de la dataset

```
df=pd.read_csv('fichier.csv')
```

ou

```
df=pd.read_excel('fichier.xls')
```

```
df=pd.read_table('fichier.txt')
```

Selon le type du fichier

dfFrame



Syntaxe de création d'un **DataFrame** avec **pandas.read**:

**Df=pd.read\_***type***(‘fichier.ext’,sep=‘c’,header=0)**

- **Type** : correspond au type du fichier à ouvrir
- **sep=‘c’** : c est le caractère séparateur de champs (de colonnes)
- **header=0** : les noms de colonnes sont en première ligne

Type(df)

```
pandas.core.frame.DataFrame
```

## Exploration des données

Consulter le contenu du dataFrame obtenu :

```
df.head(10)
```

ou

```
df.head() # 5 lignes par défaut
```

Pays	Age	Salaire	Achat
France	44	72000	No
Spain	27	48000	Yes
Germany	30	54000	No
Spain	38	61000	No
Germany	40	-----	Yes
France	35	58000	Yes
Spain	----	52000	No
France	48	79000	Yes
Germany	50	83000	No
France	37	67000	Yes

## Variables prédictives :

3 variables indépendantes :

**Pays, Age, Salaire**

## Variables à prédire :

1 variable “décision” qui dépend des autres :

**Achat**



Prédire si le client achète ou non le produit ?



## Description

Consulter une description du dataframe :

```
df.describe( )
```



**count, mean, std, min, max, ...**  
par variable / feature

```
df.describe(include=['object'])
```



description des variables  
catégoriques

Distribution des différentes classes :

```
df.groupby('Age').size()
```



Pour chaque valeur de Age, compter le nombre de lignes



## Description

Consulter une description du dataframe :

- **count** : Le nombre total de valeurs non nulles dans chaque colonne.
- **mean** : La moyenne des valeurs de la colonne.
- **std** : L'écart-type, qui mesure la dispersion des données autour de la moyenne.
- **min** : La valeur minimale de la colonne.
- **25%** : Le premier quartile (Q1)
- **50%** : Le deuxième quartile (Q2 ou médiane),
- **75%** : Le troisième quartile (Q3),
- **max** : La valeur maximale dans la colonne.

	age	hypertension	heart_disease
<b>count</b>	5110.000000	5110.000000	5110.000000
<b>mean</b>	43.215264	0.097456	0.054012
<b>std</b>	22.633866	0.296607	0.226063
<b>min</b>	0.000000	0.000000	0.000000
<b>25%</b>	25.000000	0.000000	0.000000
<b>50%</b>	45.000000	0.000000	0.000000
<b>75%</b>	61.000000	0.000000	0.000000
<b>max</b>	82.000000	1.000000	1.000000

## Description statistique

<code>df.shape</code>	dimension du <code>dfFrame</code>
<code>df.isnull()</code>	les <code>False</code> indiquent qu'il n'existe aucune valeur manquante
<code>df.isnull().count()</code>	nombre de valeurs manquantes par colonne
<code>df.isnull().any()</code>	si une colonne de la dataframe contient une donnée manquante ( <code>NAN</code> ou <code>NULL</code> )
<code>df.isnull().sum()</code>	nombre de lignes contenant des valeurs manquantes
<code>df.columns</code> <code>[df.isnull().any()]</code>	déduire une liste de colonnes contenant des données manquantes
<code>df['Col'].unique()</code>	nombre de valeurs distinctes dans la colonne <code>nomCol</code>
<code>df.loc[df['Col']=="val",:]</code>	consulter seulement les lignes dont <code>nomCol=val</code>
<code>df.iloc[lgi:lgj,coln:colm]</code>	Accès aux données avec les indices

## Manipulation des données

```
df.drop(['col1','col2',..],  
axis=1, inplace=True)
```

Supprimez les étiquettes (colonnes) spécifiées pour toutes les lignes (axis=1), le résultat est placé dans le même dataframe (inplace=True).

```
df=df.iloc[3:,:]
```

Supprimer les trois premières ligne du dataframe

```
df=df.loc[df['age']<80,:]
```

Supprimer les lignes (entrées) correspondant aux personnes dont l'âge est supérieur à 20.

```
df['col'].astype(NouvType)
```

Cast un objet pandas vers un type spécifié

```
df['col'].fillna(df['col'].mean(),  
inplace=True)
```

Remplacer les valeurs NaN dans la variable 'col' par la moyenne (ou median) de la colonne.

```
df.value_col.values_count()
```

Une des valeurs de la variable 'col' est value\_col. On affiche le nombre de valeurs cherchées.



## Détection de valeurs aberrantes (outliers)



**Qu'est-ce que une valeur aberrante ?**



C'est une valeur d'une variable qui diffère considérablement de toutes les autres valeurs.



C'est aussi une observation incorrecte ou anormale dans les données par rapport aux autres observations.



## Détection de valeurs aberrantes



**Quelles sont les causes et conséquences des valeurs aberrantes ?**

**Cause**



Causées par une incertitude de mesure ou par une erreur expérimentale, de saisie,...etc

**Conséquence**



Les VA peuvent déparer et tromper la phase d'entraînement des modèles d'apprentissage automatique et aboutir à de mauvaises performances de prédiction

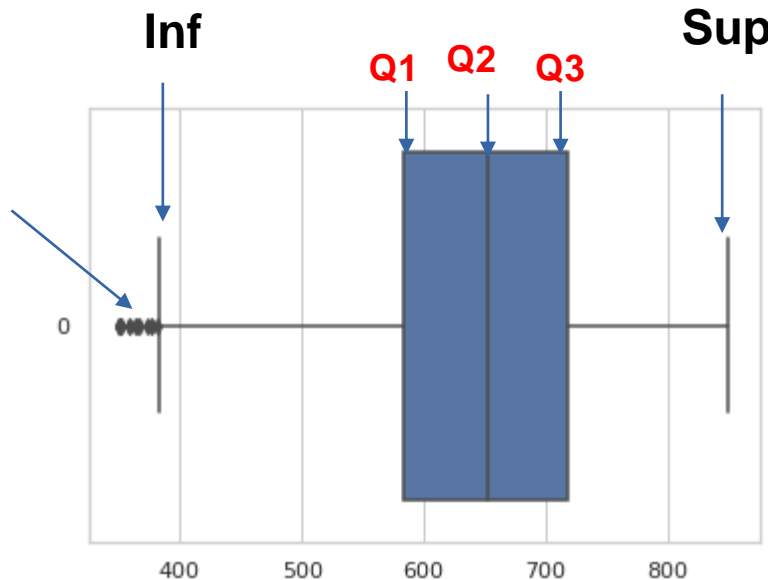




## Visualisations

```
import seaborn as sns
sns.set(style='whitegrid')
ax=sns.boxplot (df=df['Salaire'],orient='h')
```

Valeurs  
aberrantes  
(outliers)



**Q1**=df['Salaire'].quantile(0.25)

**Q3**=df['Salaire'].quantile(0.75)

**Q2** = (Q1+Q3) / 2

**[Q1,Q3]** : intervalle de confiance

**IQR=Q3-Q1**



**inf=Q1-1.5\*IQR**

**sup=Q3+1.5\*IQR**



## Visualisations

Le quartile est calculé, sur les valeurs d'une variable triées dans l'ordre croissant, en tant que 4-quartiles:

- Q1 : est la plus petite valeur de la variable tel que 25% au moins des autres valeurs lui sont inférieures ou égales.
- Q2 : est la valeur de la variable qui sépare les 50% des données inférieurs.
- Q3 : est la valeur de la variable qui sépare les 75% au moins des autres valeurs lui sont inférieures ou égales.
- IQR : l'écart **interquartile** ( **interquartile range**) est une mesure de dispersion qui s'obtient en faisant la différence entre le troisième et le premier quartile :  $IQR = Q_3 - Q_1$ . L'IQR est un estimateur statistique robuste.



## Visualisation → Calcul

Print(data)

10	11	12	13	14	15	16	17	18
0	1	2	3	4	5	6	7	8

Calculer  $Q1=?$

9 : nombre de valeur de data

$$i=0.25*(9-1)$$

$i=2$  (indice de  $Q1$ )

$$Q1=12$$

Calculer  $Q3=?$

9 : nombre de valeur de data

$$i=0.75*(9-1)$$

$$i=6$$

$$Q3=16$$

$$Q2=(Q1+Q3)/2$$

$$Q2=14$$



Si l'indice  $i$  n'est pas un entier (2,2 par exemple), on fait une interpolation

$$Q1=12+0,25*(13-12) \text{ \# différence entre l'élément d'indice 2 et 3}$$



## Visualisations



**Comment afficher les valeurs aberrante pour une variable v ?**

```
Outliers=[ x for x in df['v'] if x <Inf or x > Sup]
```

```
df[ df['v']<Inf]
```

```
df[ df['v']>Sup]
```



**Comment connaître le nombre de valeurs aberrante ?**

```
df[ df['v']<Inf].count()
```

```
df[ df['v']>Sup].count()
```

1

## Solutions pour les valeurs aberrantes et/ou manquantes (VAM)

Première démarche : Supprimer les lignes contenant les VAM.



Sans drop :

Outliers

```
df_sans_outliers = [ x for x in df if df['colonne'] > Inf and df['colonne'] < Sup ]
```

ou

```
df_sans_outliers = df [ df['colonne'] < Sup & df['colonne'] > Inf ]
```

NaN

```
df_sans_nan = df [ pd.notnull ( df ['colonne'] ) ]
```

ou

```
df_sans_nan = df [ df ['colonne'].notna() ]
```

 Avec drop :

Outliers

```
df.drop (df [ df['colonne'] >= Sup ].index, inplace=True )
```

ou

```
df.drop ( df [ df['colonne'] <= Inf ].index, inplace=True )
```

NaN

```
df = df.dropna ( )
```



Si la taille du dataframe est réduite, l'élimination des observations n'est pas une bonne idée.

## Deuxième démarche : Imputation des valeurs manquantes

Cette méthode consiste à deviner et changer une valeur pour une donnée aberrante ou manquante par une valeur artificielle.

 Imputation avec **fillna ( )** :

```
df.fillna ( value=None, method=None, axis=None, inplace=False,  
            limit=None, downcast=None )
```



- Cette fonction est disponible dans le package pandas.
- Il renvoie un objet en sortie dans lequel les valeurs nulles/manquantes sont remplies.

Avec **.fillna()**, nous pourrons remplacer avec des **valeurs calculées** (mean, median) ou des **valeurs personnalisées**.

### 1- remplir les valeurs manquantes avec la moyenne :

```
df.colonne.fillna ( value = df['colonne'].mean(), inplace=True)
```

ou tout simplement :

```
df = df.fillna ( df['colonne'].mean() )
```

### 2- remplir les valeurs manquantes avec la médiane :

```
df.colonne.fillna ( value = df['colonne'].median(), inplace=True)
```

### 3- remplir les valeurs manquantes avec une valeurs choisie :

```
df.colonne.fillna ( value = 300, inplace=True)
```





## Deuxième démarche : Imputation des valeurs **abérantes**

Cette méthode consiste à deviner et changer une valeur pour une donnée aberrante ou manquante par une valeur artificielle.

 Imputation avec **where ( )** :

```
df['colonne'] = np.where(df['colonne'] > sup, df['colonne'].median(),  
                        df['colonne'])
```



- Cette fonction est disponible dans le package numpy.
- La valeur de remplacement « *df['colonne'].median()* » peut être aussi 'sup' ou *df['colonne'].mean()* ou toute autre valeur conventionnelle.

## Deuxième démarche : Imputation des valeurs **abérantes**

Cette méthode consiste à deviner et changer une valeur pour une donnée aberrante ou manquante par une valeur artificielle.



Imputation avec **du code Python**:

```
for i in df['colonne']:  
    if i >=Sup or i <=Inf:  
        df['colonne']=df['colonne'].replace(i,np.median(df['colonne']))
```



## Corrélation entre variables

### 1 - Définition

- Dès lors que l'on analyse des données, il est important en Machine Learning) de détecter si vos variables (features) sont liées (corrélées).
- Le coefficient **de** Pearson permet **de** mesurer le niveau **de corrélation entre les deux variables**. Il renvoie une valeur **entre** -1 et 1. Si il est proche **de** 1 cela signifie que **les variables** sont corrélées, proche **de** 0 que **les variables** sont décorrélées et proche **de** -1 qu'elles sont corrélées négativement.



## Corrélation entre variables

### 2 - Types de corrélation

#### 1. Corrélation positive

Lorsque la valeur d'une variable augmente, la valeur de l'autre variable a également tendance à augmenter.

**Exemple** : La taille et le poids des personnes. En général, plus une personne est grande, plus elle tend à être lourde.

**Graphiquement**, cela se manifeste par une ligne ascendante dans un nuage de points.

**Valeur du coefficient de corrélation : (entre 0 et 1).**



## Corrélation entre variables

### 2 - Types de corrélation

#### 2. Corrélation négative

Lorsque la valeur d'une variable augmente, la valeur de l'autre variable tend à diminuer.

**Exemple :** Le prix d'un produit et la quantité vendue. Si le prix augmente, la demande peut diminuer.

**Graphiquement,** cela se manifeste par une ligne descendante dans un nuage de points.

**Valeur du coefficient de corrélation : (entre -1 et 0).**



## Corrélation entre variables

### 2 - Types de corrélation

#### 3. Corrélation nulle

Aucune relation linéaire apparente entre les variables.  
Lorsque l'une varie, l'autre ne montre pas de tendance spécifique à augmenter ou diminuer.

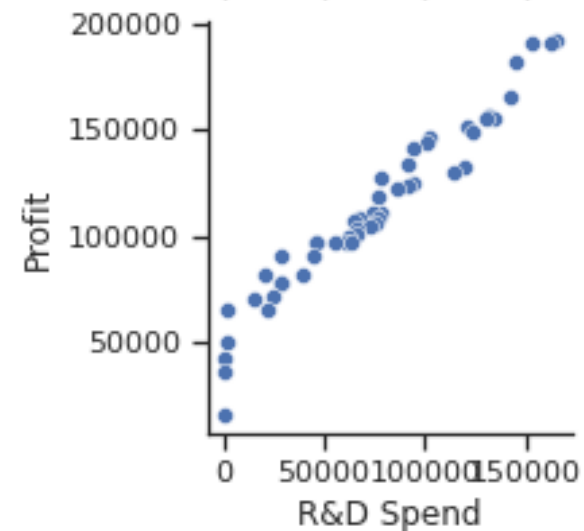
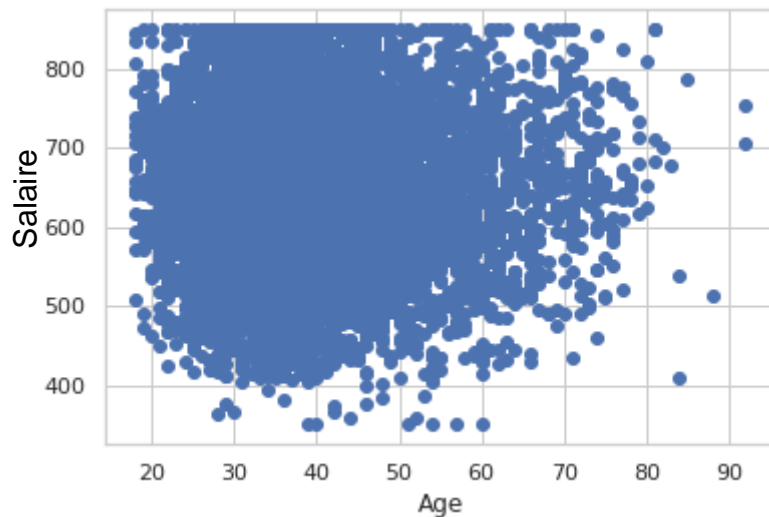
**Exemple** : Le poids des personnes et leur numéro de téléphone. Il n'y a aucune raison que ces deux variables soient liées.

**Valeur du coefficient de corrélation est 0.**

## 2 - Visualisation

Visualisation des dispersions en nuages de points : **corrélation** entre deux variables :

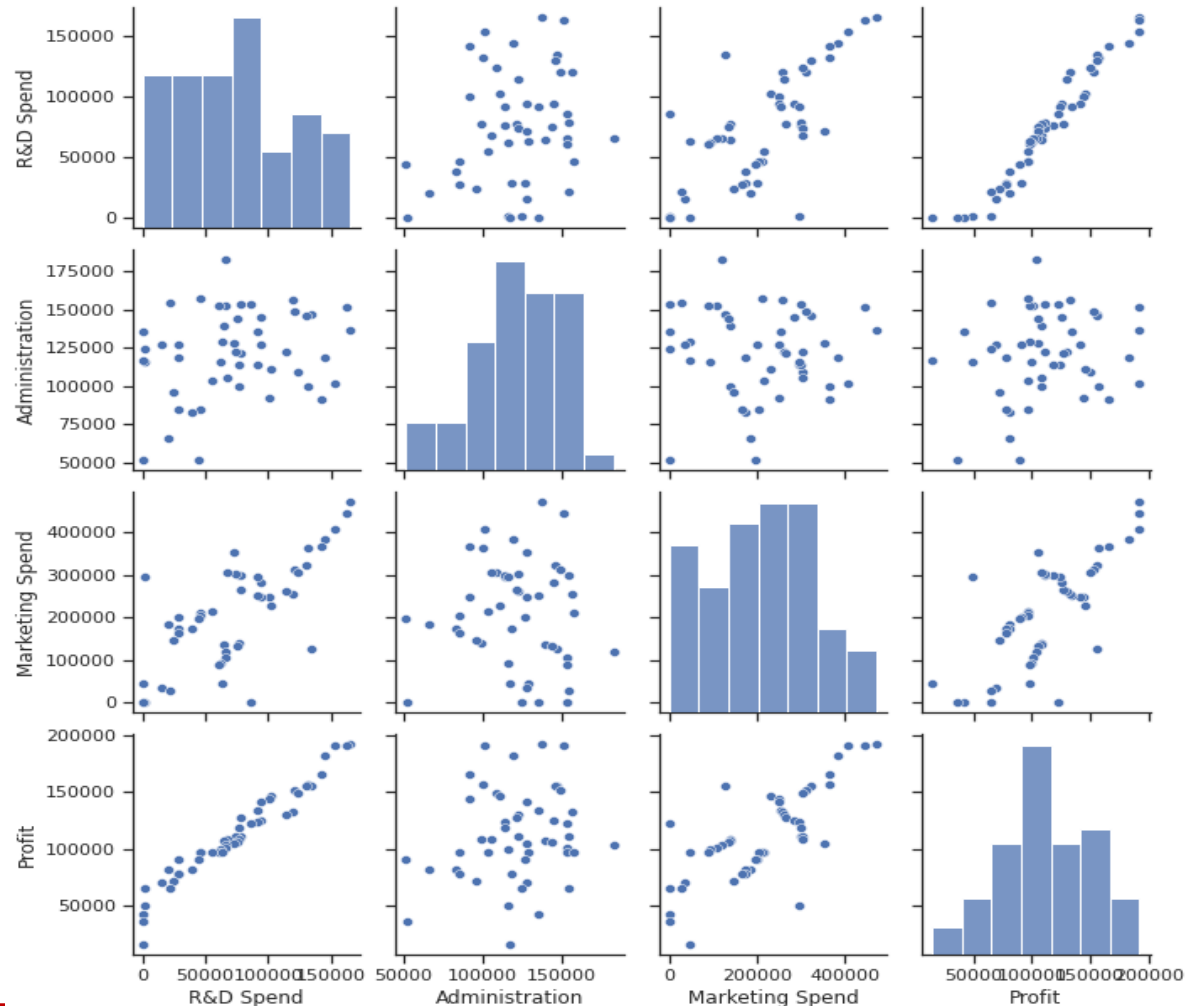
```
import matplotlib as plt  
plt.scatter(x=df['Age'],y=df['Salaire'])  
plt.xlabel('Age')  
plt.ylabel('Salaire')
```



## 2 - Visualisation

Corrélation par paires de variables :

```
sns.set(style='ticks')  
sns.pairplot(df)
```





### 3 - Calcul de coefficient de corrélation

Formule de calcul de la covariance et le coefficient de corrélation de Pearson

$$cov(x, y) = \sum_{i=1}^n (x_i - mean_i) (y_j - mean_j), \quad pearson := \rho_{x,y} = \frac{cov(x, y)}{std_x std_y}$$

$$Co(X, Y) = \sum_{i=0}^N \frac{(X_i - \bar{X})(Y_i - \bar{Y})}{N}$$

$$Std_x = \sqrt{\frac{1}{n} \sum_{i=1}^n (x_i - \bar{x})^2}$$

Syntaxe d'affichage des corrélations:

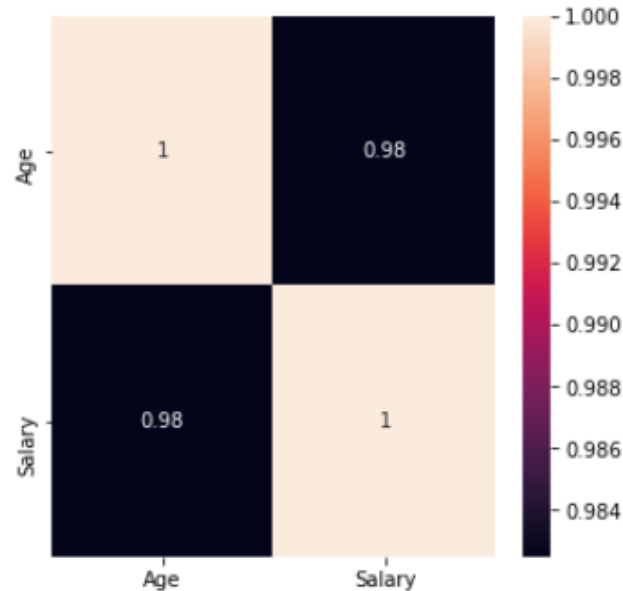
**`correlations = dataset.corr(method='pearson')`**

### 3 - Calcul de coefficient de corrélation

Visualisation graphique des coefficients de corrélation

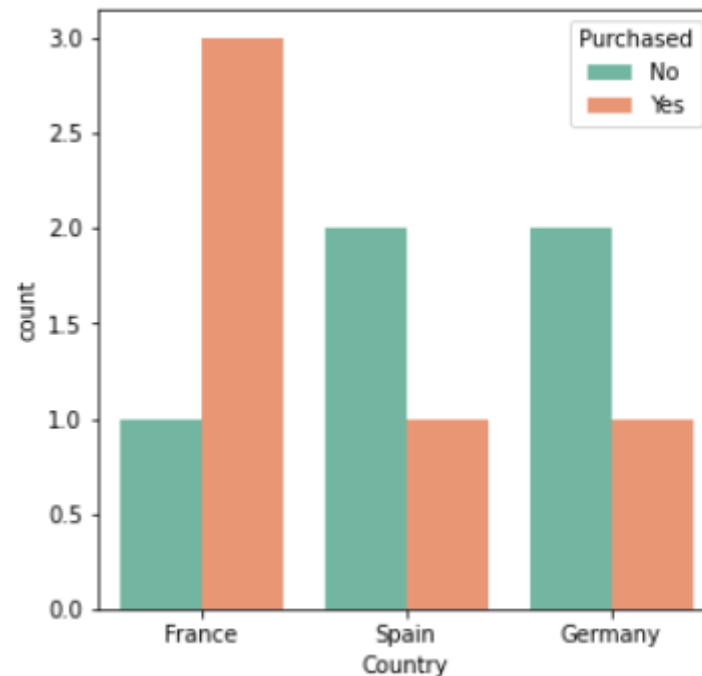
```
if, ax = plt.subplots(figsize = (5, 5))  
sns.heatmap(correlations, annot = True)
```

Out[10]: <AxesSubplot:>



## 4 – Autres visualisations de données

```
plt.figure(figsize=(5,5))  
sns.countplot( x='Country', hue=df['Purchased'], palette='Set2',  
data=df);
```



## 4 – Traitement des données corrélées

- Supprimer une des deux variables corrélées : permet de réduire la redondance et simplifie le modèle.
- Combiner les deux variables : en faisant la moyenne ou toute autre combinaison linéaire.
- Utiliser des algorithmes robustes à la multicollinéarité : arbre de décision ou Random forest



## Traitement des doublons

### 1- Recherche des doublons

Pour identifier les lignes en double dans un DataFrame, on peut utiliser la méthode `uplicated()`. Elle retourne un tableau de booléens, où chaque valeur indique si une ligne est dupliquée ou non.

```
import pandas as pd

# Exemple de dataframe
df = pd.DataFrame({
    'col1': [1, 2, 2, 3],
    'col2': ['A', 'B', 'B', 'C']
})

# Identifier les doublons
duplicated_rows = df.duplicated()

print(duplicated_rows)
```



## Traitement des doublons

### 1- Traitement des doublons

- Pour supprimer les doublons, tu peux utiliser **drop\_duplicates()**.  
défaut, cette méthode garde la première occurrence de chaque doublon.

```
# Supprimer les doublons  
df_cleaned = df.drop_duplicates()
```

- Si tu veux garder la dernière occurrence au lieu de la première, tu peux utiliser l'argument `keep='last'`

```
df_cleaned = df.drop_duplicates(keep='last')
```

- Supprimer les doublons basés sur certaines colonnes

```
df_cleaned = df.drop_duplicates(subset=['col1'])
```

- Réinitialiser les index après suppression

```
df_cleaned.reset_index(drop=True, inplace=True)
```

## 4

## Input data - Target

Créer une matrice pour les variables indépendantes : elle contient les valeurs des trois premières variables avec toutes les lignes.



Utilisez la technique **iloc** de Pandas :

## Données

```
X=df.iloc[:, :-1]
```

ou

```
X=df.iloc[:, 0:3]
```

## Cible (Target)

```
Y=df.iloc[:, -1]
```

ou

```
Y=df.iloc[:, 3]
```

## Transformer le Dataframe en matrice numpy

```
Y=Y.values
```

et

```
X=X.values
```

## 2

## Catégorisation des données

Les variables textuelles ne peuvent pas participer à la formulation mathématique du problème.



Il faut les convertir en variables numériques.

Variables non numériques

	Pays	Age	Salaire	Achat
0	France	44	72000	No
1	Spain	27	48000	Yes
2	Germany	30	54000	No
3	Spain	38	61000	No
4	Germany	40	63777.777778	Yes
5	France	35	58000	Yes
6	Spain	38.777778	52000	No
7	France	48	79000	Yes
8	Germany	50	83000	No
9	France	37	67000	Yes



## Première conversion

bibliothèque

module

classe

```
from sklearn.preprocessing import LabelEncoder  
  
# créer un objet de la classe LabelEncoder  
labEnc_X = LabelEncoder ( )  
  
#adapter la colonne 0 à transformer  
X[:,0] = labEnc_X.fit_transform ( X [ : , 0 ] )
```

 La classe **LabelEncoder** transforme les textes France, Allemagne et Espagne en valeurs numériques 0, 1 et 2.

## Deuxième conversion

```
from sklearn.preprocessing import OneHotEncoder  
from sklearn.compose import ColumnTransformer  
  
#préparation de transformation : préciser le critère et le type de transformation  
#une colonne en 3 colonnes binaires et indiquer la position de la colonne  
ct=ColumnTransformer([('Pays',OneHotEncoder(),[1])],  
                       remainder='passthrough')  
  
#exécution de transformation  
x=ct.fit_transform(x)
```



La colonne transformée sera ajoutée au début du dataframe.

**Résultat obtenu :**

Pays			Age	Salaire
1.00	0.00	0.00	44.00	72000.00
0.00	0.00	1.00	27.00	48000.00
0.00	1.00	0.00	30.00	54000.00
0.00	0.00	1.00	38.00	61000.00
0.00	1.00	0.00	40.00	63777.77
1.00	0.00	0.00	35.00	58000.00
0.00	0.00	1.00	38.77	52000.00
1.00	0.00	0.00	48.00	79000.00
0.00	1.00	0.00	50.00	83000.00
1.00	0.00	0.00	37.00	67000.00




Maintenant, la matrice X est prête à être intégrée dans des équations pour définir certains modèles de ML.

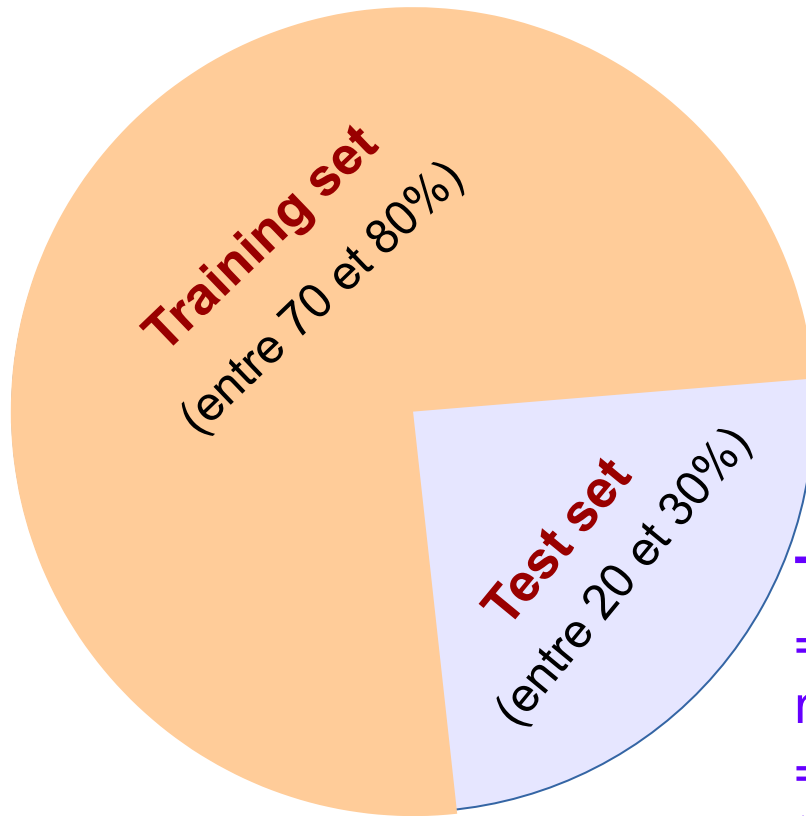
## Catégorisation de la dernière colonne (“Achat”) :

De la même manière :

```
labEnc_y = LabelEncoder ( )  
y=labEnc_y . fit_transform ( y )
```

Achat		Achat
No		0
Yes		1
No		0
No		0
Yes		1
Yes		1
No		0
Yes		1
No		0
Yes		1

## Répartition de la base en **Training Set** and **Test Set**



### **Training Set :**

==> utilisée pour l'apprentissage ;  
==> fournit un modèle qui définit une corrélation entre les variables indépendantes et dépendantes.

### **Test Set :**

==> utilisée pour vérifier si le modèle maintient toujours ses corrélations ;  
==> le modèle est testé sur une base de test, jamais vue en apprentissage.



**train\_test\_split** est une fonction ==> on n'a pas besoin de créer d'objets.

```
from sklearn.model_selection import train_test_split  
X_train , X_test , Y_train , Y_test = train_test_split( X , Y, test_size = 0.2, random_state=0)
```

unifier le choix de l'ensemble de  
train et de l'ensemble de test

Résultat de répartition :

X\_train

Pays			Age	Salaire	Achat
0	1	0	40	63777.778	1
1	0	0	37	67000	1
0	0	1	27	48000	1
0	0	1	38.7778	52000	0
1	0	0	48	79000	1
0	0	1	38	61000	0
1	0	0	44	72000	0
1	0	0	35	58000	1

Y\_train

X\_test

Y\_test

Pays			Age	Salaire	Achat
0	1	0	30	54000	0
0	1	0	50	83000	0

## Mise à l'échelle de valeurs

Il est à remarquer que la variable Age n'a pas la même échelle que la variable salaire :

$$27 < \text{Age} < 50$$

$$48000 < \text{Salaire} < 83000$$

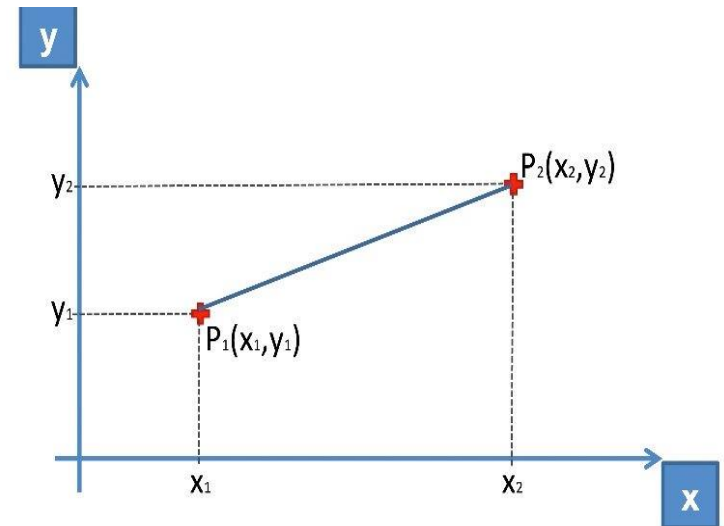
Le salaire peut dominer et même écraser la variable Age .

Cette dernière ne sera pas prise en compte dans le modèle alors qu'elle peut avoir un impact sur la variable Achat



Si nous prenons deux échantillons de la base et nous calculons la distance Euclidienne :

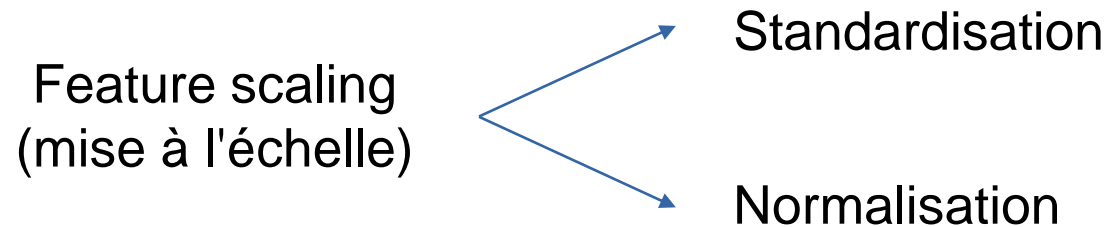
Pays	Age	Salaire	Achat
France	44	72000	No
Spain	27	48000	Yes
Germany	30	54000	No
Spain	38	61000	No
Germany	40	63777.7778	Yes
France	35	58000	Yes
Spain	38.7778	52000	No
France	48	79000	Yes
Germany	50	83000	No
France	37	67000	Yes



$$\text{Euclidean Distance between } P_1 \text{ and } P_2 = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

$$(79000 - 48000)^2 = 961000000$$

$$(48 - 27)^2 = 441 \implies \text{très négligeable}$$



## 1

**Standardisation**

La standardisation remet à l'échelle les données dans la plage [-1,1]. Cependant, nous conservons les valeurs aberrantes.

Nous calculons **la moyenne** et **l'écart-type** de chaque variable indépendante de **X\_train**.

$$x_{stand} = \frac{x - \text{mean}(x)}{\text{StandardDeviation}(x)}$$

```
from sklearn.preprocessing import StandardScaler  
sc = StandardScaler ( )  
X_train = sc . fit_transform ( X_train )  
X_test=sc . transform ( X_test )
```



nous n'avons pas appliqué de mise à l'échelle des entités à la variable Y car elle prend les valeurs 0 et 1 qui sont déjà sous la même échelle que celles de la variable X.



Si la variable dépendante Y prend de très grandes valeurs, nous devons appliquer le feature scaling des entités pour transformer Y sous la même échelle que les variables X.

## 2

**Normalisation**

La normalisation (la plus utilisée) remet à l'échelle les valeurs dans la plage [0,1].

Cela peut être utile dans certains cas où toutes les valeurs ont besoin de la même échelle positive.

Cependant, les valeurs aberrantes des données sont perdues.

$$x_{norm} = \frac{x - \min(x)}{\max(x) - \min(x)}$$

```
from sklearn import preprocessing  
mm_scaler = preprocessing.MinMaxScaler()  
X_train_minmax = mm_scaler.fit_transform ( X_train )  
X_test_minmax = mm_scaler.transform ( X_test )
```

1. Entraînement (train) : Lorsqu'on standardise/normalise les données d'entraînement, on utilise la méthode **fit\_transform** car elle effectue deux opérations :

- **Fit (ajustement)** : Elle calcule les statistiques (par exemple, la moyenne et l'écart type) à partir des données d'entraînement.
- **Transform (transformation)** : Ensuite, elle applique cette standardisation (centrage des données autour de la moyenne et mise à l'échelle avec l'écart type) sur l'ensemble d'entraînement.

2. Test : Lorsqu'on travaille avec l'ensemble de test, on utilise **transform** car les données de test doivent être normalisées/standardisées en utilisant les mêmes statistiques (moyenne et écart type) que celles calculées sur l'ensemble d'entraînement.