

Analyse de données avec Python

Document confidentiel.
Ne peut être reproduit ni diffusé
sans l'autorisation de VERTIGO.



Entertainment Marketing Research

Plan du cours

0. **Préambule**
1. **Une analyse, c'est quoi ?**
2. **Python et environnement de travail**
3. **Data : manipulation, exploration, nettoyage**
4. **Data : Visualisation**
5. **Data : Présentation**

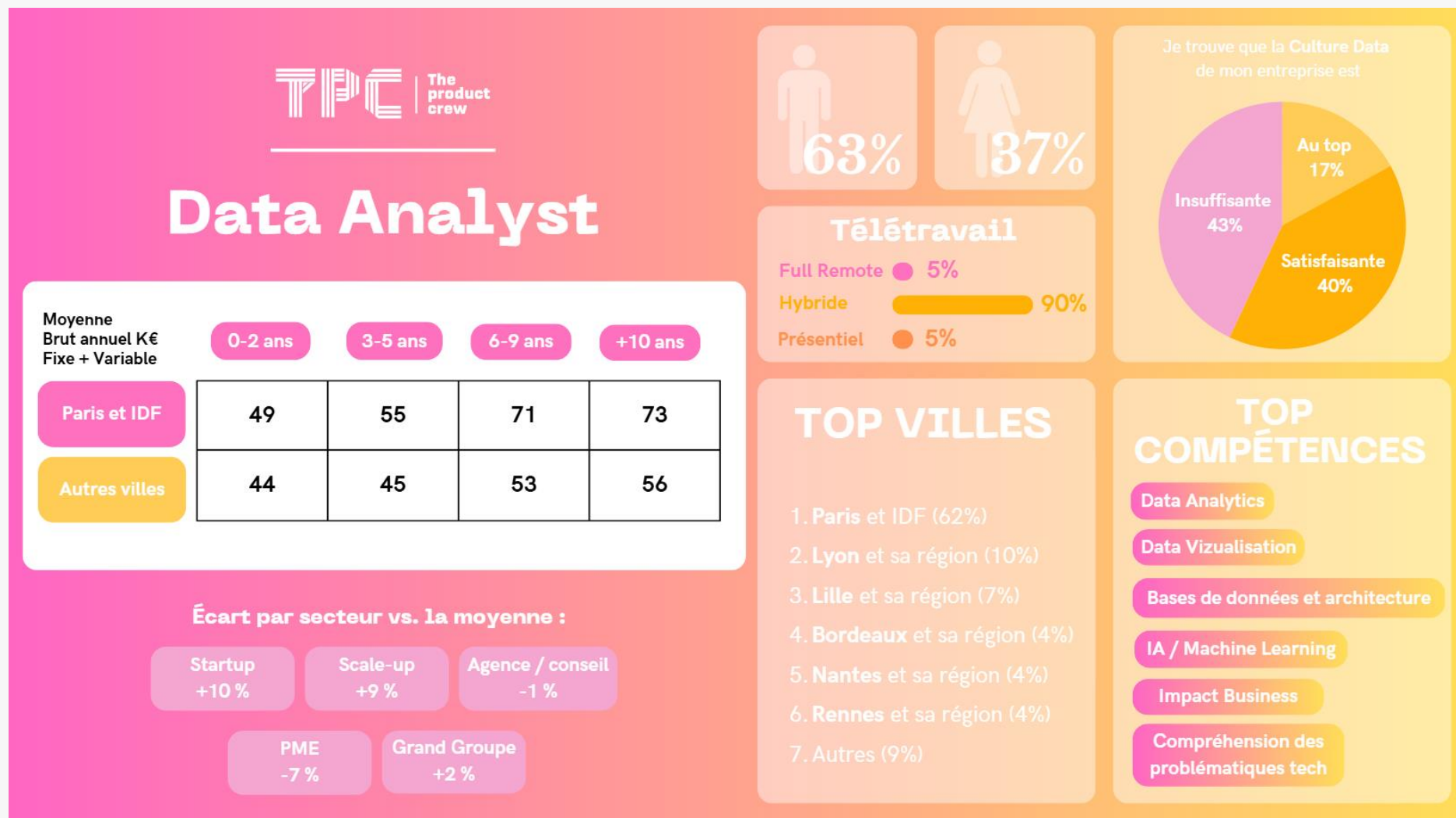
0. Préambule

Document confidentiel.
Ne peut être reproduit ni diffusé
sans l'autorisation de VERTIGO.



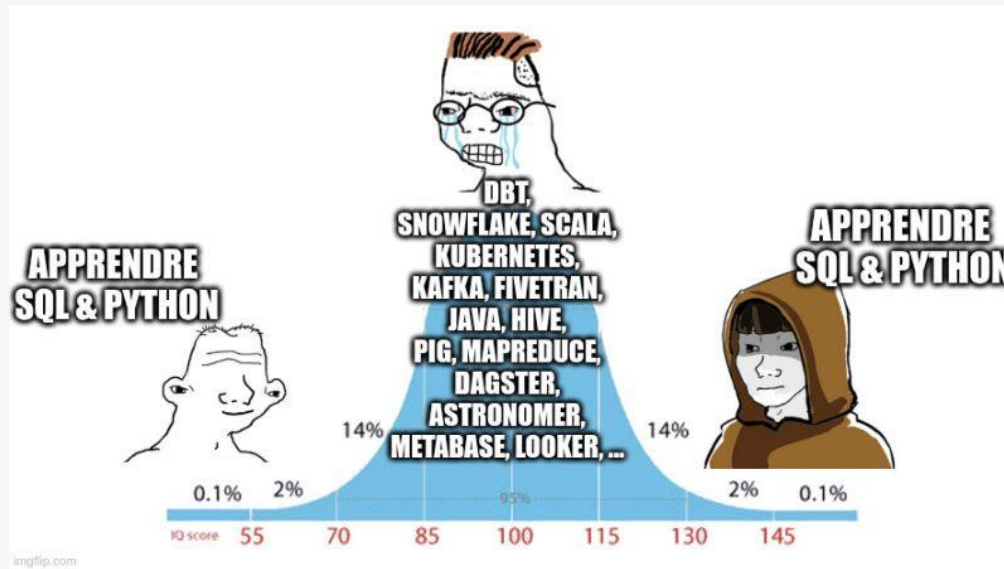
Entertainment Marketing Research

0. Préambule



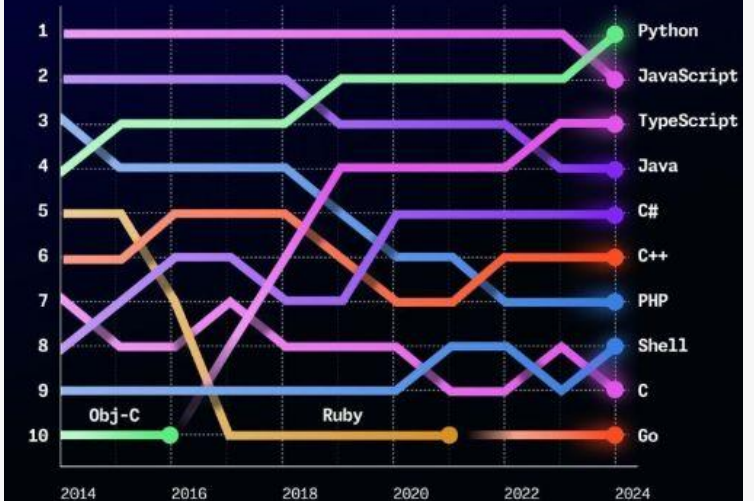
0. Préambule

Analyse : compétences techniques



Top programming languages on GitHub

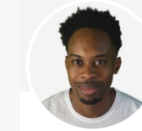
RANKED BY COUNT OF DISTINCT USERS CONTRIBUTING TO PROJECTS OF EACH LANGUAGE.



0. Préambule

Analyse : compétences business

Présent sur
in y t
Abonnes toi :)

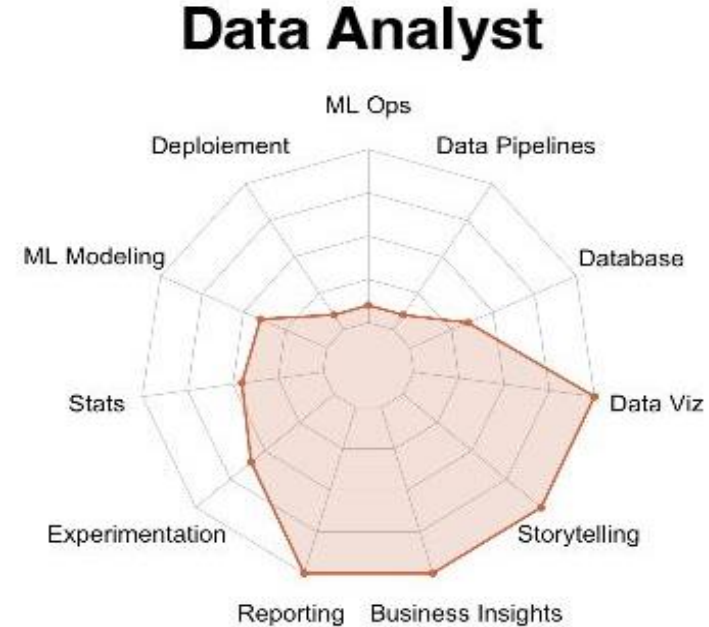


Kevin Rosamont Prombo · 1er
Lead Data | Consultant | Top 5 Contenu Data LinkedIn France
Paris, Île-de-France, France · [Coordonnées](#)
[Ressources Data & Analytics](#) [🔗](#)
22 330 abonnés · [Plus de 500 relations](#)

Deviens Data Analyst
Formation Excel, SQL & Data Storytelling

- + de 400 élèves accompagnés
- + 10 ans xp dans l'analytics
- + 15 missions en entreprise

DW Data Workers
Université de Montréal



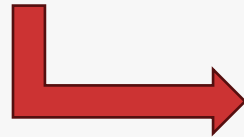
Crédit : Kevin Rosamont Prombo (2023)

0. Préambule

ChatGPT et autres LLM : **OUI MAIS**



Doit rester votre **EXECUTANT**



Rapidité +++

Qualité +++ à --- selon prompt

Architecture --

Vue Globale Projet ---

TOUJOURS lire le code des LLM
COMPRENDRE ce qu'il a fait et **POURQUOI**

0. Préambule

ChatGPT et autres LLM : **OUI MAIS**

Python = Langue = Vocabulaire + Syntaxe

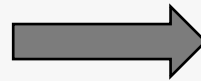


Apprendre Pratiquer



Quand ChatGPT **FAIT**  Vous **NE FAITES PAS**

Utilisations recommandées



- **Commencer un projet**
- **Débugger**
- **Optimiser**

1. Une analyse c'est quoi ?

Document confidentiel.
Ne peut être reproduit ni diffusé
sans l'autorisation de VERTIGO.



Entertainment Marketing Research

1. Une analyse, c'est quoi ?

Réponse à un besoin business

Questions

Définition

1. Une analyse, c'est quoi ?

Cas pratique 1

Vous êtes maire d'une ville.

Vous avez remarqué que vous électeurs et électrices parlent tous et toutes du même sujet (à définir). L'ayant aussi remarqué, vous convoquez un(e) prestataire pour faire une analyse du sujet.

Objectif

Après 4 minutes de dialogue avec le/la prestataire, il/elle doit pouvoir restituer précisément et exactement le besoin à analyser

1. Une analyse, c'est quoi ?

Cas pratique 1

Poser des questions !

Comment ? Pourquoi ?



Définit le besoin

(le client ne le connaît pas
toujours)

Qui ? Quel ? Est-ce que ?



Répond au besoin

1. Une analyse, c'est quoi ?

Cas pratique 2

Editeur de films : Promoteur publicitaire d'un film avant sa sortie en salle

Un éditeur souhaiterait comprendre pourquoi son film, Thunderbolts, n'a pas performé la semaine passée en salle.

Comment faire?

1. Une analyse, c'est quoi ?

Cas pratique 2

Pourquoi la semaine passée précisément ?

Pourquoi pas avant ?

Comment la date de sortie a été choisie ?

Comment la promotion a été menée ?

1. Une analyse, c'est quoi ?

Cas pratique 2

Quel est le film ?

De quoi parle-t 'il ?

Qui est le cœur de cible ?

Quels sont les enjeux?

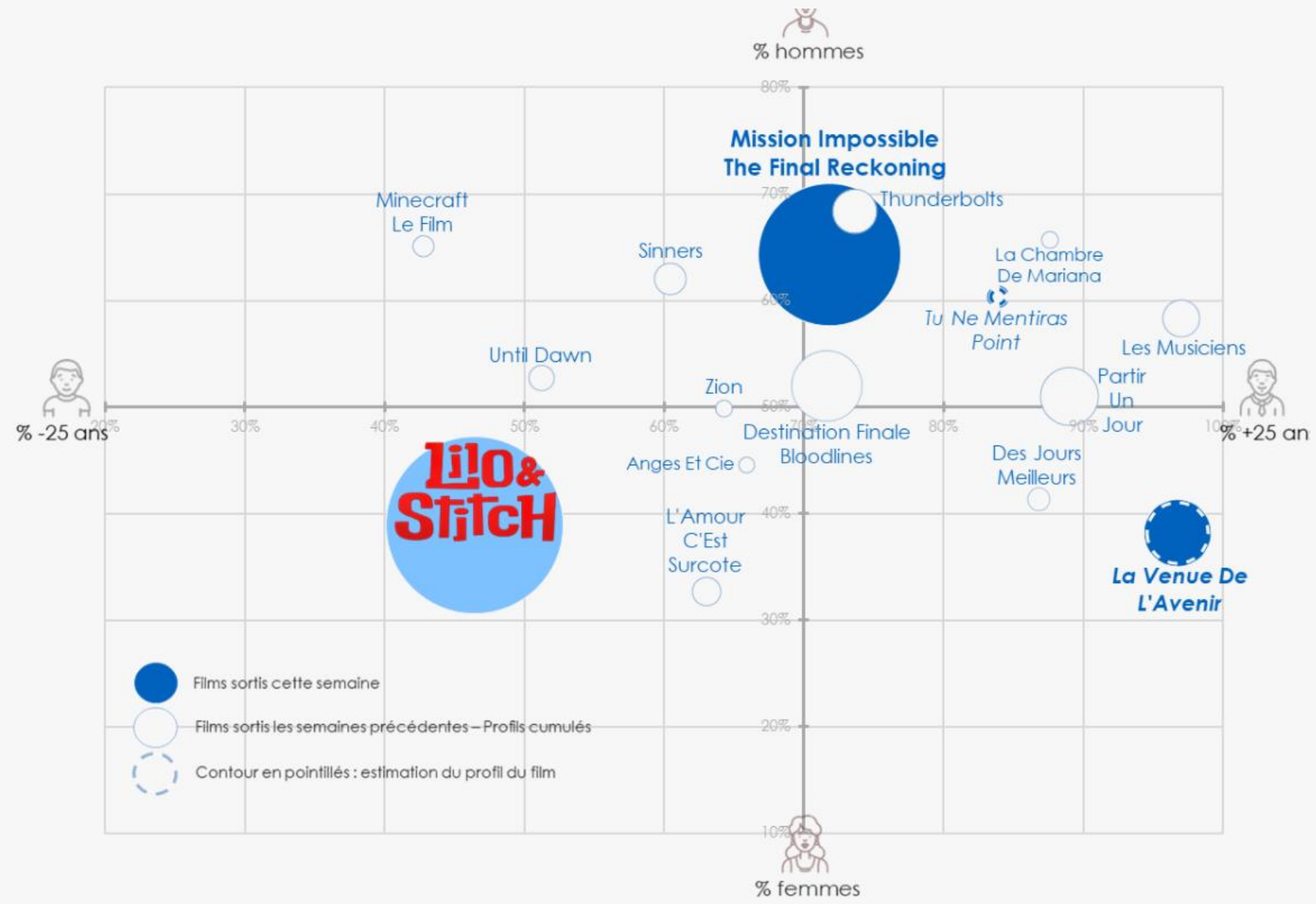
 **Besoin : connaitre le public des films**

1. Une analyse, c'est quoi ?

Cas pratique 2

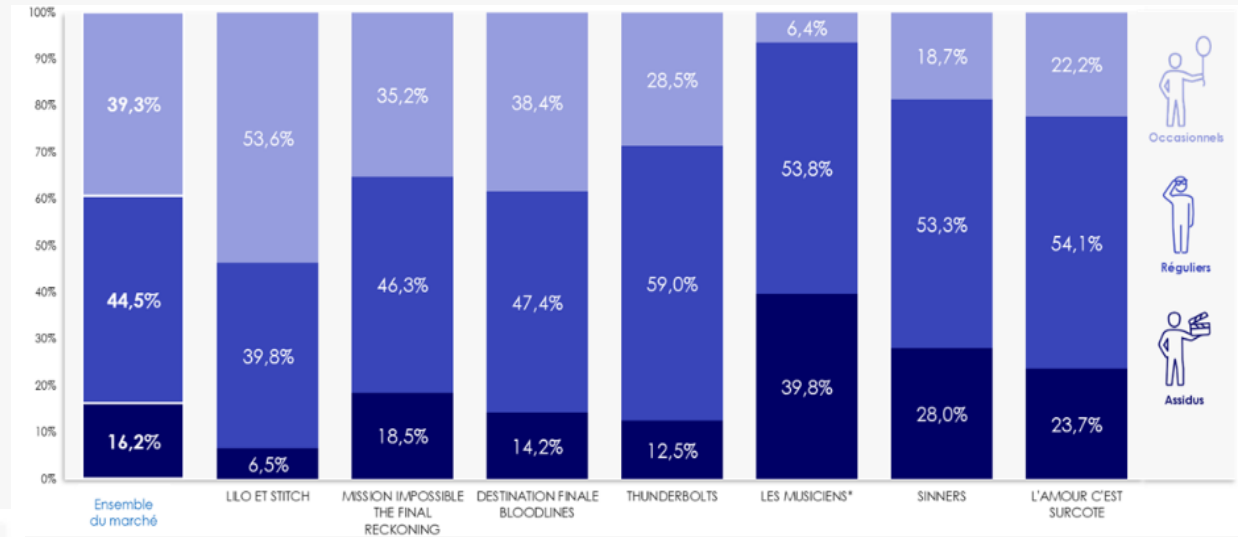
➔ Réponse : Etude **CinExpert** de **Vertigo**

Mesure hebdomadaire du public en salle
= **Qui** voit **quoi**

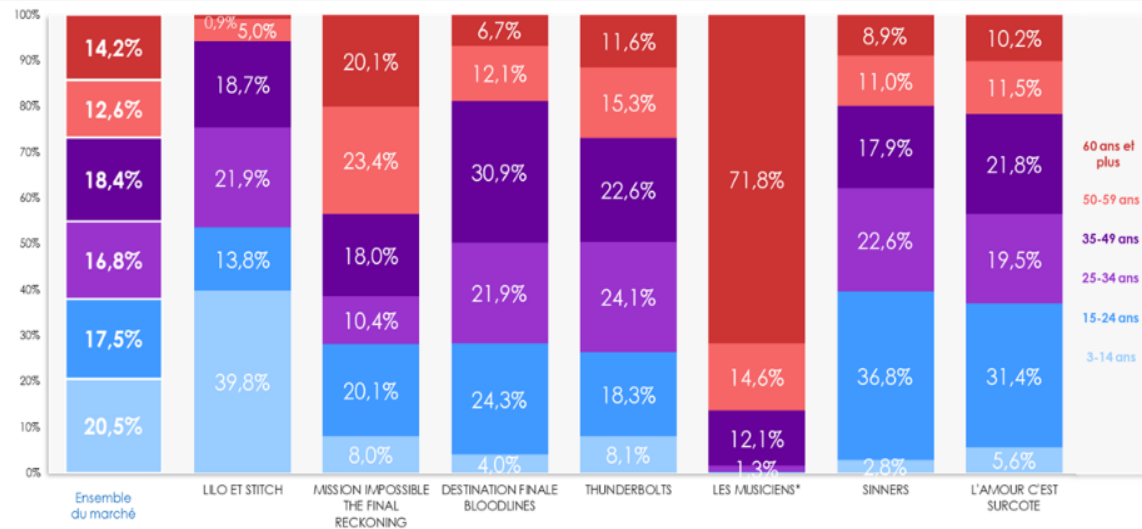


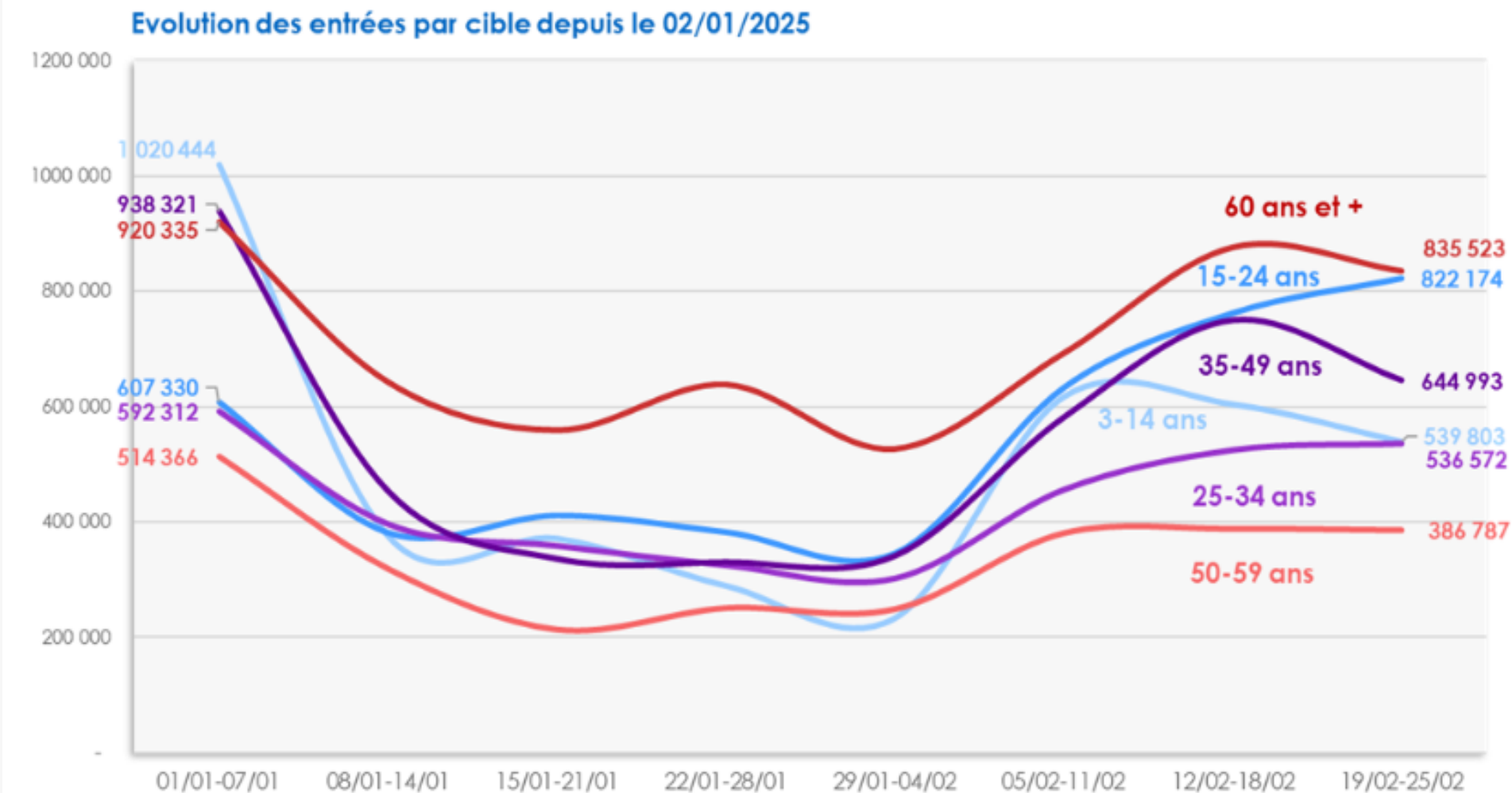
Profil du marché et des principaux films

Par habitude de fréquentation



Par âge





2. Python et environnement de travail

Document confidentiel.
Ne peut être reproduit ni diffusé
sans l'autorisation de VERTIGO.



Entertainment Marketing Research

Environnement de travail ?

- ➔ Maison avec juste les meubles nécessaires
- ➔ Toujours les mêmes conditions
- ➔ Accessible depuis de machines ≠

Multitude d'outils

IDE : VSCode, PyCharm, Spyder, Komodo, Vim...

Notebook : Jupyter, JupyterLab, GoogleColab, Azure, DeepNote, Datalore...

Version Control System : Git, Mercurial, Surversion, Perforce (Jeux Vidéo), ...

Multitude d'outils

Frontend et UX différents

Principes de fonctionnement identiques



Maitrise de l'un => Adaptation rapide à l'autre

Python : Rappels

Avantages

Lecture / Ecriture facile
Courbe d'apprentissage douce
Versatile
Dynamic typing
Open Source
Communauté énorme et active
Nombreuses librairies tierces
Langage interprété
...

Inconvénient

Lenteur +++
Memory inefficient
Weak mobile computation
Dynamic typing
Multithreading limité (GIL)
Support mobile limité
...

2. Python et environnement de travail

Python : Rappels

Objets Natifs

Eléments

```
'cinema' <class 'str'>  
9 <class 'int'>  
1.5 <class 'float'>  
True <class 'bool'>
```

Collections / Itérable

```
[False, 1, 'deux', 3.0] <class 'list'>  
(False, 1, 'deux', 3.0) <class 'tuple'>  
{False, 1, 'deux', 3.0} <class 'set'>  
  
{'booleen': False,  
 'int': 1,  
 'string': 'deux',  
 'float': 3.0} <class 'dict'>
```

Ordonné	Mutable	Appartenance (relative)	Mémoire (relative)
1	1	$O(n) : 1$	1
1	0	$O(n) : 1$	0,1 – 1
0	1	$O(1) : 10^{-4} - 10^{-5}$	10 – 100

Python : Rappels

Objets Natifs

Eléments

```
'cinema'  
9  
1.5  
True
```

Variables

```
cinema = 'cinema'  
neuf = 9  
un_virgule_cinq = 1.5  
vrai = True
```

Collections / Itérable

```
[False, 1, 'deux', 3.0]  
(False, 1, 'deux', 3.0)  
{False, 1, 'deux', 3.0}  
  
{'booleen': False,  
 'int': 1,  
 'string': 'deux',  
 'float': 3.0}
```

```
small_list = [False, 1, 'deux', 3.0]  
small_tuple = (False, 1, 'deux', 3.0)  
small_set = {False, 1, 'deux', 3.0}  
small_dico = {  
    'booleen': False,  
    'int': 1,  
    'string': 'deux',  
    'float': 3.0  
}
```

BONNES PRATIQUES

Minuscule
Mots séparés par « _ » } = snake case

Nom avec du sens
(éviter a, b, c, i, j, k, etc.)

Python : Rappels

Bloc if – elif – else

Exécution conditionnelle

```
price = 10
money_in_pocket = 5

if money_in_pocket >= price :
    money_in_pocket -= price
    print(money_in_pocket)
else:
    print("Not enough money")
```

Not enough money

```
price = 10
money_in_pocket = 22

if money_in_pocket >= price :
    money_in_pocket -= price
    print(money_in_pocket)
else:
    print("Not enough money")
```

12

```
price = 10
money_in_pocket = 10

if money_in_pocket > price :
    money_in_pocket -= price
    print(money_in_pocket)
elif money_in_pocket == price :
    print("No money left")
else:
    print("Not enough money")
```

No money left

Python : Rappels

Boucle for - while

Répéter une opération

```
eaten_candies = 0
gloups = 5
for gloup in range(gloups):
    eaten_candies += 1
    print(f"I ate: {eaten_candies} candies")
```

```
I ate: 1 candies
I ate: 2 candies
I ate: 3 candies
I ate: 4 candies
I ate: 5 candies
```

```
candies_in_jar = 10
eaten_candies = 0
while candies_in_jar :
    eaten_candies += 1
    candies_in_jar -= 1
    print(f"I ate {eaten_candies} candies,",
          f"{candies_in_jar} candies left")
print("I got sick")
```

```
I ate 1 candies, 9 candies left
I ate 2 candies, 8 candies left
I ate 3 candies, 7 candies left
I ate 4 candies, 6 candies left
I ate 5 candies, 5 candies left
I ate 6 candies, 4 candies left
I ate 7 candies, 3 candies left
I ate 8 candies, 2 candies left
I ate 9 candies, 1 candies left
I ate 10 candies, 0 candies left
I got sick
```

Python : Rappels

Fonctions

```
ticket_price = 10 * 0.85
jacket_price = 11.85 * 0.85
car_price = 98521 * 0.85
couple_of_cents = 0.05 * 0.85
print(
    ticket_price,
    jacket_price,
    car_price,
    couple_of_cents,
    sep='\n',
    end='\n')
```

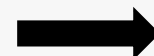
```
8.5
10.0725
83742.84999999999
0.0425
```

Factorisation



```
def dollar_to_euro(dollar, rate) :
    result = round(dollar * rate, 2)
    return result
```

Modulable
Réutilisable
Lisibilité



```
CONVERTER = 0.85
ticket_price = dollar_to_euro(10, CONVERTER)
jacket_price = dollar_to_euro(11.85, CONVERTER)
car_price = dollar_to_euro(98521, CONVERTER)
couple_of_cents = dollar_to_euro(0.05, CONVERTER)
print(
    ticket_price,
    jacket_price,
    car_price,
    couple_of_cents,
    sep='\n',
    end='\n')
```

```
8.5
10.07
83742.85
0.04
```

Python : Rappels

Fonctions

```
students = ["Alice", "Bob", "Charlie"]
grades = [[85, 90, 92], [78, 82, 88], [95, 91, 93]]

for i in range(len(students)):
    total = 0
    for grade in grades[i]:
        total += grade
    average = total / len(grades[i])
    if average >= 90:
        letter = 'A'
    elif average >= 80:
        letter = 'B'
    elif average >= 70:
        letter = 'C'
    else:
        letter = 'F'
    print(f"{students[i]}: {average:.2f} ({letter})")
```

```
Alice: 89.00 (B)
Bob: 82.67 (B)
Charlie: 93.00 (A)
```



```
def calculate_average(grades: List[float]) -> float:
    """Return the average of a list of grades."""
    return sum(grades) / len(grades)

def assign_letter_grade(average: float) -> str:
    """Assign a letter grade based on average score."""
    if average >= 90:
        return 'A'
    elif average >= 80:
        return 'B'
    else:
        return 'F'

def main():
    students = ["Alice", "Bob", "Charlie"]
    grades = [[85, 90, 92], [78, 82, 88], [95, 91, 93]]

    for name, grade_list in zip(students, grades):
        avg = calculate_average(grade_list)
        letter = assign_letter_grade(avg)
        print(f"{name}: {avg:.2f} ({letter})")

main()
```

```
Alice: 89.00 (B)
Bob: 82.67 (B)
Charlie: 93.00 (A)
```

Python : Rappels

Fonctions

BONNES PRATIQUES

Fonctions

```
def dollar_to_euro(dollar, rate) :  
    result = round(dollar * rate, 2)  
    return result
```

Docstring

Snake case

Typehints

```
def dollar_to_euro(dollar: float, rate: float) -> float:  
    """  
    Convert an amount in US dollars to euros using a given exchange rate.  
  
    Args:  
        dollar (float): The amount in US dollars.  
        rate (float): The exchange rate (euros per dollar).  
  
    Returns:  
        float: The equivalent amount in euros, rounded to 2 decimal places.  
    """  
    result = round(dollar * rate, 2)  
    return result
```

3. Data : manipulation, exploration, nettoyage

Document confidentiel.
Ne peut être reproduit ni diffusé
sans l'autorisation de VERTIGO.



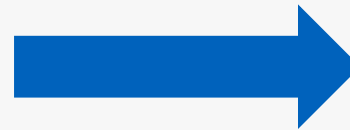
Entertainment Marketing Research

3. Data : manipulation , exploration, nettoyage

	Colonne 1	Colonne 2
Ligne 1	1	2
Ligne 2	3	4

x 2

	Colonne 1	Colonne 2
Ligne 1	2	4
Ligne 2	6	8



Liste de 2 listes

```
[5]: [[1, 2], [3, 4]]  
[5]: [[1, 2], [3, 4]]
```

x 2

Liste de 4 listes

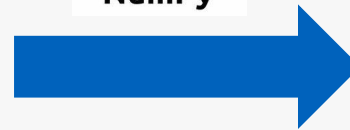
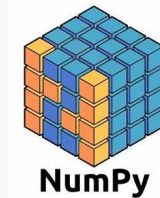
```
[6]: [[1, 2], [3, 4]] * 2  
[6]: [[1, 2], [3, 4], [1, 2], [3, 4]]
```

3. Data : manipulation , exploration, nettoyage

	Colonne 1	Colonne 2
Ligne 1	1	2
Ligne 2	3	4

x 2

	Colonne 1	Colonne 2
Ligne 1	2	4
Ligne 2	6	8



Array de **2** listes = Matrice

```
[7]: import numpy as np  
     np.array([[1, 2], [3, 4]])
```

```
[7]: array([[1, 2],  
           [3, 4]])
```

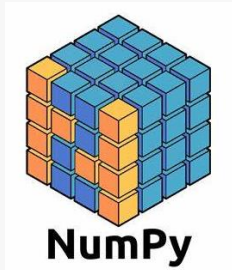
x 2

Array de **2** listes = Matrice

```
[8]: import numpy as np  
     np.array([[1, 2], [3, 4]]) * 2
```

```
[8]: array([[2, 4],  
           [6, 8]])
```

3. Data : manipulation , exploration, nettoyage



Codé en C sous le capot = rapidité +++ Vectorisation des opérations mathématiques

```
n = 1_000_000
```

```
%%timeit  
[i*2 for i in range(n)]
```

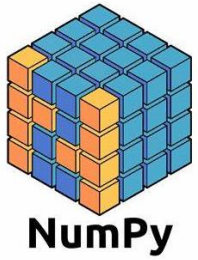
45.6 ms ± 2.47 ms per loop (mean ± std. dev. of 7 runs, 10 loops each)

```
%%timeit  
np.arange(n) * 2
```

1.6 ms ± 50.3 µs per loop (mean ± std. dev. of 7 runs, 1,000 loops each)

45x
+ rapide

3. Data : manipulation , exploration, nettoyage



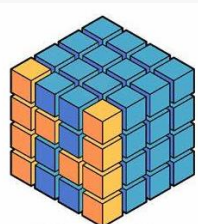
Lisibilité ---

```
array([[ 'ADAMA', 'OCEAN FILMS', 'Animation Mixte', 140.0, 'France',  
        'Simon Rouby'],  
       [ 'A VIF', 'SND', 'Gros Indépendant', 296.0, 'USA', 'Jon Wells'],  
       [ 'NOUS TROIS OU RIEN', 'GAUMONT DISTRIBUTION',  
        'Français Intermédiaire Tout Public', 233.0, 'France', 'Kheiron'],  
       [ 'AVRIL ET LE MONDE TRUQUE', 'STUDIOCANAL', 'Animation Mixte',  
        152.0, 'France', 'Franck Ekinci, Christian Desmares'],  
       [ 'SPECTRE', 'SONY', 'Blockbuster US', 902.0, 'USA', 'Sam Mendes']],  
      dtype=object)
```



	titre	distributeur	typologie_film	nombre_copies	nationalite	realisateur
0	ADAMA	OCEAN FILMS	Animation Mixte	140.0	France	Simon Rouby
1	A VIF	SND	Gros Indépendant	296.0	USA	Jon Wells
2	NOUS TROIS OU RIEN	GAUMONT DISTRIBUTION	Français Intermédiaire Tout Public	233.0	France	Kheiron
3	AVRIL ET LE MONDE TRUQUE	STUDIOCANAL	Animation Mixte	152.0	France	Franck Ekinci, Christian Desmares
4	SPECTRE	SONY	Blockbuster US	902.0	USA	Sam Mendes

3. Data : manipulation , exploration, nettoyage



NumPy



Lisibilité ---

```
array([[ 'ADAMA', 'OCEAN FILMS', 'Animation Mixte', 140.0, 'France',  
        'Simon Rouby'],  
       [ 'A VIF', 'SND', 'Gros Indépendant', 296.0, 'USA', 'Jon Wells'],  
       [ 'NOUS TROIS OU RIEN', 'GAUMONT DISTRIBUTION',  
        'Français Intermédiaire Tout Public', 233.0, 'France', 'Kheiron'],  
       [ 'AVRIL ET LE MONDE TRUQUE', 'STUDIOCANAL', 'Animation Mixte',  
        152.0, 'France', 'Franck Ekinci, Christian Desmares'],  
       [ 'SPECTRE', 'SONY', 'Blockbuster US', 902.0, 'USA', 'Sam Mendes']],  
      dtype=object)
```

TITRE Français	NATIONALITE	REALISATEUR	DISTIBUTEUR FR	Typologie Cinexpert
str	str	str	str	str
"ADAMA"	"France"	"Simon Rouby"	"OCEAN FILMS"	"Animation Mixte"
"A VIF"	"USA"	"Jon Wells"	"SND"	"Gros Indépendant"
"NOUS TROIS OU RIEN"	"France"	"Kheiron"	"GAUMONT DISTRIBUTION"	"Français Intermédiaire Tout Pu..."
"AVRIL ET LE MONDE TRUQUE"	"France"	"Franck Ekinci, Christian Desma..."	"STUDIOCANAL"	"Animation Mixte"
"SPECTRE"	"USA"	"Sam Mendes"	"SONY"	"Blockbuster US"



Polars

3. Data : manipulation , exploration, nettoyage

1. Ne **jamais faire confiance** à un dataset

“In practice, **no dataset is ever truly clean**; all observational data are subject to measurement error, missingness, or structural inconsistencies, **which must be explored and accounted for prior to analysis.**”

Tukey, John W. (1977). Exploratory Data Analysis.

3. Data : manipulation , exploration, nettoyage

D	E	F
code_cinexp	date_de_sortie	titre
738	15/03/2017	01:54
	24/07/2019	303
2359	05/01/2022	355
	04/10/2023	800
1872	15/01/2020	1917
	31/05/2023	2018
1929	26/02/2020	2040
3651	05/02/2025	05-sept
	01/05/2019	#FEMALE PLEASURE
1899	05/02/2020	#JESUISLA
	21/11/2018	#MOSCOU ROYAN
886	02/08/2017	#PIRE SOIREE
317	16/03/2016	10 CLOVERFIELD LANE
2870	12/04/2023	10 JOURS ENCORE SANS MAMAN
1914	19/02/2020	10 JOURS SANS MAMAN
1688	17/07/2019	100 KILOS D'ETOILES
3714	26/03/2025	100 MILLIONS
2063	28/10/2020	100% LOUP

D	E	F
code_cinexp	date_de_sortie	titre
		A USEFUL GHOST
	14/09/2022	A VENDREDI ROBINSON
167	04/11/2015	A VIF
772	12/04/2017	A VOIX HAUTE LA FORCE DE LA PAI
	05/04/2023	A VOL D'OISEAUX
412	01/06/2016	A WAR
	06/12/2023	AALAVANDHAN
1201	09/05/2018	ABDEL ET LA COMTESSE
	16/08/2023	ABDELINHO
3356	29/05/2024	ABIGAIL
1786.5	23/10/2019	ABOMINABLE
1968	15/07/2020	ABOU LEILA
2867	05/04/2023	ABOUT KIM SOHEE
2463	06/04/2022	ABUELA
	14/05/2025	ACCIDENT DOMESTIQUE
3053	20/09/2023	ACIDE
1193	02/05/2018	ACTION OU VERITE

	Film	Semaine	Entrées	Cumul	Cumul réel
33880	FANON	1	33860	33860	33 860
33987	FANON	2	45175	73872	79 035
34105	FANON	3	49661	139117	128 696

3. Data : manipulation , exploration, nettoyage

1. Ne **jamais faire confiance** à un dataset

“In practice, **no dataset is ever truly clean**; all observational data are subject to measurement error, missingness, or structural inconsistencies, **which must be explored and accounted for prior to analysis.**”

Tukey, John W. (1977). Exploratory Data Analysis.

2. Les data ne mentent pas, mais **elles disent ce que nous voulons**

“Beware of the problem of testing too many hypotheses; **the more you torture the data, the more likely they are to confess**, but **confessions obtained under duress may not be admissible in the court of scientific opinion.**”

Professor of Statistics Stephen M. Stigler(1987). Neutral Models in Biology.

3. Data : manipulation , exploration, nettoyage

Pandas CheatSheet

[Pandas Cheat Sheet.pdf](#) (<https://pandas.pydata.org/> originally written by Irv Lustig, Princeton Consultants, inspired by Rstudio Data Wrangling Cheatsheet)

[pandas documentation — pandas 2.3.0 documentation](#)

3. Data : manipulation , exploration, nettoyage

Lire un fichier

```
pd.read_excel(  
    'data/entrees_films.xlsx',  
    usecols=['sem_cine_inv', 'Entrées', 'titre', 'Année'], # or "A, B, C, D, K" or "A:D, K" or [0, 1, 2, 3, 10]  
    engine='calamine' # Possible values: "openpyxl" (default), "xlrd", "odf", "pyxlsb", "calamine" (ultra fast)  
)
```

```
pd.read_csv(  
    'data/poids_statistiques.csv',  
    sep="\t", # tabulation  
    encoding='latin-1', # possible values : "utf-8", "ANSI", "latin-1", "ISO-8859-1", "cp1252"...  
    usecols=['pid', 'age', 'mois_cine',], # or "A, B, C, D, K" or "A:D, K" or [0, 1, 2, 3, 10]  
)
```

```
pd.read_pickle('data/cleaned_data.pkl')
```

3. Data : manipulation , exploration, nettoyage

Explorer un DataFrame

```
a_global.shape
✓ 0.0s
(10255, 7)

a_global.index
✓ 0.0s
RangeIndex(start=0, stop=10255, step=1)
```

```
a_global.columns
✓ 0.0s
Index(['vague_cine_inv', 'pid', 'age', 'gender', 's3ad', 'q2ad',
      'sem_cine_inv'],
      dtype='object')
```

```
a_global.dtypes
✓ 0.0s
vague_cine_inv    int64
pid               int64
age              int32
gender           float64
s3ad             float64
q2ad             int64
sem_cine_inv      int64
dtype: object
```

```
a_global.info()
✓ 0.0s
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10255 entries, 0 to 10254
Data columns (total 7 columns):
#   Column          Non-Null Count  Dtype
---  -
0   vague_cine_inv  10255 non-null  int64
1   pid             10255 non-null  int64
2   age            10255 non-null  int32
3   gender         10255 non-null  float64
4   s3ad           10255 non-null  float64
5   q2ad           10255 non-null  int64
6   sem_cine_inv   10255 non-null  int64
dtypes: float64(2), int32(1), int64(4)
memory usage: 520.9 KB
```

```
a_global.isna().sum()
✓ 0.0s
vague_cine_inv    0
pid               0
age              0
gender            0
s3ad             0
q2ad             0
sem_cine_inv      0
dtype: int64
```

```
a_global.head(n=5)
✓ 0.0s
```

	vague_cine_inv	pid	age
0	2326	3162713	57
1	2326	5203821	50
2	2326	5361385	40
3	2326	9664953	65
4	2326	23992569	68

```
a_global.tail(n=5)
✓ 0.0s
```

	vague_cine_inv	pid	age
10250	2331	1976744286	8
10251	2331	1988359483	36
10252	2331	1988359483	36
10253	2331	2002828674	28
10254	2331	2007777127	47

```
a_global.describe()
✓ 0.0s
```

	vague_cine_inv	pid	age
count	10255.000000	1.025500e+04	10255.000000
mean	2328.427011	6.291306e+08	36.929888
std	1.541099	5.274258e+08	20.340951
min	2326.000000	4.075890e+05	3.000000
25%	2327.000000	3.156061e+08	17.000000
50%	2328.000000	3.949792e+08	38.000000
75%	2330.000000	1.084144e+09	53.000000
max	2331.000000	2.011343e+09	86.000000

3. Data : manipulation , exploration, nettoyage

Slicer un DataFrame : `.loc` == PAR NOM

```
a_global.loc[0] # Renvoie les lignes par NOM
✓ 0.0s
```

vague_cine_inv	2326.0
pid	3162713.0
age	57.0
gender	1.0
s3ad	2.0
q2ad	2921.0
sem_cine_inv	2326.0

Name: 0, dtype: float64

```
a_global.loc[0] # Renvoie les lignes par NOM
✓ 0.0s
```

	vague_cine_inv	pid	age	gender	s3ad	q2ad
0	2326	3162713	57	1.0	2.0	2921
0	2326	5203821	50	1.0	2.0	2964
0	2326	5361385	40	1.0	4.0	2966
0	2326	9664953	65	1.0	4.0	2946
0	2326	23992569	68	1.0	2.0	2964

```
a_global.loc[[0]] # Renvoie les lignes par NOM
```

	vague_cine_inv	pid	age	gender	s3ad	q2ad
0	2326	3162713	57	1.0	2.0	2921

```
a_global.loc[[0, 1, 2]]
a_global.loc[0:2] #Les deux sont équivalents
✓ 0.0s
```

	vague_cine_inv	pid	age	gender	s3ad	q2ad	sem_cine_inv
0	2326	3162713	57	1.0	2.0	2921	2326
0	2326	5203821	50	1.0	2.0	2964	2326
0	2326	5361385	40	1.0	4.0	2966	2326
0	2326	9664953	65	1.0	4.0	2946	2326
0	2326	23992569	68	1.0	2.0	2964	2326
1	2326	44828833	28	2.0	4.0	2860	2326
1	2326	44828833	11	1.0	4.0	2860	2326
1	2326	49778428	6	2.0	4.0	2940	2326
1	2326	72292447	65	1.0	4.0	2924	2326
1	2326	77548686	37	1.0	4.0	2897	2326
2	2326	80445588	20	1.0	5.0	2959	2326
2	2326	82373191	15	1.0	3.0	2921	2326
2	2326	88009078	59	2.0	3.0	2964	2326
2	2326	88025551	25	1.0	3.0	2930	2326
2	2326	88025551	25	1.0	3.0	2937	2326

```
# Renvoie les lignes par NOM et la colonne
a_global.loc[0, 'age']
✓ 0.0s
```

0	57
0	50
0	40
0	65
0	68

Name: age, dtype: int32

```
# Renvoie les lignes par NOM et les colonnes
a_global.loc[0, ['age', 's3ad']]
✓ 0.0s
```

	age	s3ad
0	57	2.0
0	50	2.0
0	40	4.0
0	65	4.0
0	68	2.0

3. Data : manipulation , exploration, nettoyage

Slicer un DataFrame : `.iloc` == PAR INDEX

```
# Renvoie les lignes par index  
a_global.iloc[0]  
✓ 0.0s
```

```
vague_cine_inv    2326.0  
pid              3162713.0  
age              57.0  
gender           1.0  
s3ad             2.0  
q2ad            2921.0  
sem_cine_inv     2326.0  
Name: 0, dtype: float64
```

```
# Renvoie les lignes et les colonnes par index  
a_global.iloc[[0, 1, 2], 0]  
✓ 0.0s
```

```
0    2326  
0    2326  
0    2326  
Name: vague_cine_inv, dtype: int64
```

```
# Renvoie les lignes par index  
a_global.iloc[[0, 1, 2]]  
✓ 0.0s
```

	vague_cine_inv	pid	age	gender	s3ad	q2ad	sem_cine_inv
0	2326	3162713	57	1.0	2.0	2921	2326
0	2326	5203821	50	1.0	2.0	2964	2326
0	2326	9664953	65	1.0	4.0	2946	2326

```
# Renvoie les lignes et les colonnes par index  
a_global.iloc[[0, 1, 2], [0, 1, 2]]  
✓ 0.0s
```

	vague_cine_inv	pid	age
0	2326	3162713	57
0	2326	5203821	50
0	2326	5361385	40

3. Data : manipulation , exploration, nettoyage

Explorer une Series

```
a_global['age']
# Ecriture déconseillée
# Confusion avec les méthodes
# et attributs de la classe DataFrame
a_global.age
✓ 0.0s
```

0	57
0	50
0	40
0	65
0	68
	..
2050	8
2050	36
2050	36
2050	28
2050	47

Name: age, Length: 10255, dtype: int32

```
a_global['age'].sort_values(ascending=True)
✓ 0.0s
```

190	3
190	3
522	3
190	3
1238	3
	..
642	85
1895	85
591	86
561	86
1962	86

Name: age, Length: 10255, dtype: int32

```
a_global['age'].value_counts()
# a_global['age'].value_counts().sort_index()
# pour trier le résultats par age croissant
✓ 0.0s
```

age	
17	323
15	312
16	275
12	266
14	220
	...
86	3
82	3
85	2
83	1
84	1

```
a_global['age'].unique()
# a_global['age'].sort_values().unique()
✓ 0.0s
```

```
array([57, 50, 40, 65, 68, 28, 11,  6, 37, 20, 15, 59, 25, 47, 12, 51, 10,
       24, 43,  9, 73, 53, 62, 46, 60, 13, 61, 17, 64, 36, 38,  8, 67,  5,
        3, 45, 16, 55, 66, 52, 35, 72, 26, 39, 23, 56, 41, 21, 77, 54, 79,
       31, 49, 29, 48, 34, 42, 14, 63,  7, 44, 32, 33, 74, 30, 69, 58, 75,
       27, 22, 19,  4, 78, 80, 71, 18, 76, 70, 83, 86, 82, 85, 84, 81],
      dtype=int32)
```

```
a_global['age'].nunique()
✓ 0.0s
```

84

3. Data : manipulation , exploration, nettoyage

Explorer une Series

```
a_global['age']
# Ecriture déconseillée
# Confusion avec les méthodes
# et attributs de la classe DataFrame
a_global.age
✓ 0.0s
```

0	57
0	50
0	40
0	65
0	68
	..
2050	8
2050	36
2050	36
2050	28
2050	47

Name: age, Length: 10255, dtype: int32

```
a_global['age'].sort_values(ascending=True)
✓ 0.0s
```

190	3
190	3
522	3
190	3
1238	3
	..
642	85
1895	85
591	86
561	86
1962	86

Name: age, Length: 10255, dtype: int32

```
a_global['age'].value_counts()
# a_global['age'].value_counts().sort_index()
# pour trier le résultats par age croissant
✓ 0.0s
```

age	
17	323
15	312
16	275
12	266
14	220
	...
86	3
82	3
85	2
83	1
84	1

```
a_global['age'].unique()
# a_global['age'].sort_values().unique()
✓ 0.0s
```

```
array([57, 50, 40, 65, 68, 28, 11,  6, 37, 20, 15, 59, 25, 47, 12, 51, 10,
       24, 43,  9, 73, 53, 62, 46, 60, 13, 61, 17, 64, 36, 38,  8, 67,  5,
        3, 45, 16, 55, 66, 52, 35, 72, 26, 39, 23, 56, 41, 21, 77, 54, 79,
       31, 49, 29, 48, 34, 42, 14, 63,  7, 44, 32, 33, 74, 30, 69, 58, 75,
       27, 22, 19,  4, 78, 80, 71, 18, 76, 70, 83, 86, 82, 85, 84, 81],
      dtype=int32)
```

```
a_global['age'].nunique()
✓ 0.0s
```

84

3. Data : manipulation , exploration, nettoyage

Filtrer un DataFrame

```
a_global.query('age >= 18') # only rows where age is at least 18 (7563 rows)
a_global.query('age >= 18 & gender == 2') # only rows where age is at least 18 AND gender is women (4115 rows)
a_global.query('age >= 18 | gender == 2') # only rows where age is at least 18 OR gender is women (8862 rows)
✓ 0.0s
```

```
a_global.loc[a_global['age'] >= 18] # only rows where age is at least 18 (7563 rows)
a_global.loc[(a_global['age'] >= 18) & (a_global['gender'] == 2)] # only rows where age is at least 18 AND gender is women (4115 rows)
a_global.loc[(a_global['age'] >= 18) | (a_global['gender'] == 2)] # only rows where age is at least 18 OR gender is women (8862 rows)
✓ 0.0s
```

```
a_global.loc[a_global['age'] >= 18, 'gender'] # Returns only one column
a_global.loc[(a_global['age'] >= 18) & (a_global['gender'] == 2), ['pid, gender']] # Returns two columns
✓ 0.0s
```

```
# Storing condition for better readability and reusability
at_least_18 = a_global['age'] >= 18
women = a_global['gender'] == 2
a_global.loc[(at_least_18) & (women), ['pid, gender']]
✓ 0.0s
```


3. Data : manipulation , exploration, nettoyage

Ajouter une colonne

```
a_global['new_column'] = 1
a_global.head()
```

✓ 0.0s

	vague_cine_inv	pid	age	gender	s3ad	q2ad	sem_cine_inv	new_column
0	2326	3162713	57	1.0	2.0	2921	2326	1
1	2326	5203821	50	1.0	2.0	2964	2326	1
2	2326	5361385	40	1.0	4.0	2966	2326	1
3	2326	9664953	65	1.0	4.0	2946	2326	1
4	2326	23992569	68	1.0	2.0	2964	2326	1

```
a_global.insert(loc=1, column='new_column', value=1,)
a_global.head()
```

✓ 0.0s

	vague_cine_inv	new_column	pid	age	gender	s3ad	q2ad	sem_cine_inv
0	2326	1	3162713	57	1.0	2.0	2921	2326
1	2326	1	5203821	50	1.0	2.0	2964	2326
2	2326	1	5361385	40	1.0	4.0	2966	2326
3	2326	1	9664953	65	1.0	4.0	2946	2326
4	2326	1	23992569	68	1.0	2.0	2964	2326

```
%%timeit
a_global['gender_letter'] = a_global['gender'].apply(lambda x : 'H' if x == 1 else 'F')
```

✓ 7.2s

895 µs ± 7.94 µs per loop (mean ± std. dev. of 7 runs, 1,000 loops each)

```
%%timeit
a_global['gender_letter'] = pd.cut(
    x=a_global['gender'],
    bins=[0, 1, 2],
    right=True, # each bin includes its right edge. Default is True.
    labels=['H', 'F'],
)
```

✓ 2.6s

326 µs ± 3.43 µs per loop (mean ± std. dev. of 7 runs, 1,000 loops each)

```
%%timeit
a_global['gender_letter'] = np.where(a_global['gender']== 1, 'H', 'F')
```

✓ 13.3s

164 µs ± 637 ns per loop (mean ± std. dev. of 7 runs, 10,000 loops each)

3. Data : manipulation , exploration, nettoyage

Supprimer une ligne/colonne

```
# Supprimer les lignes si elles existent
a_global = a_global.drop([0, 3], axis=0,)
a_global.head()
```

✓ 0.0s

	vague_cine_inv	pid	age	gender	s3ad	q2ad	sem_cine_inv
1	2326	5203821	50	1.0	2.0	2964	2326
2	2326	5361385	40	1.0	4.0	2966	2326
4	2326	23992569	68	1.0	2.0	2964	2326

```
# Supprimer la colonne si elle existe
a_global = a_global.drop('gender_letter', axis=1,)
a_global.head()
```

✓ 0.0s

	vague_cine_inv	pid	age	gender	s3ad	q2ad	sem_cine_inv
0	2326	3162713	57	1.0	2.0	2921	2326
1	2326	5203821	50	1.0	2.0	2964	2326
2	2326	5361385	40	1.0	4.0	2966	2326

```
# Supprimer les lignes où TOUTES les valeurs sont NaN
a_global = a_global.dropna(how='all')
a_global.shape
```

✓ 0.0s

(10253, 7)

```
# Supprimer les lignes où au moins 1 valeur est NaN
a_global = a_global.dropna(how='any')
a_global.shape
```

✓ 0.0s

(10253, 7)

3. Data : manipulation , exploration, nettoyage

Exporter un DataFrame

```
# Sauvegarde au format pickle
a_global.to_pickle('data/cleaned_data.pkl')

# Sauvegarde au format CSV (UTF-16 or Latin-1)
a_global.to_csv('data/poids_statistiques.csv', sep="\t", encoding='latin-1', index=False)

# Sauvegarde au format Excel
a_global.to_excel('data/entrees_films.xlsx', sheet_name='Sheet', index=False, engine='openpyxl')
```

```
# Sauvegarde au format CSV (UTF-16 or Latin-1)
with open(
    'data/poids_statistiques.csv',
    mode='w', # Précise qu'il faut écrire : w = write
    encoding='latin-1',
    newline=''
) as f:
    a_global.to_csv(f, sep="\t", index=False)
```

```
# Sauvegarde d'un fichier Excel
with pd.ExcelWriter(
    'data/entrees_films.xlsx',
    engine='openpyxl',
    mode='w'
) as writer:
    a_global.to_excel(writer, sheet_name='Sheet', index=False)
```

```
# Sauvegarde au format pickle
with open(
    'data/cleaned_data.pkl',
    mode='wb' # Ecrire en binaire : wb = write binary
) as f:
    a_global.to_pickle(f)
```

3. Data : manipulation , exploration, nettoyage

Concaténer deux DataFrames

```
# DataFrames l'une à côté de l'autre. Concaténer sur les index communs. Si index absents => Remplis avec NaN
pd.concat([a_global, weights], axis=1)
```

✓ 0.0s

	vague_cine_inv	pid	age	gender	s3ad	q2ad	sem_cine_inv	vague_cine_inv	pid	age	q2ad	Entrées Extrapolées	mois_cine
0	2326	3162713	57	1.0	2.0	2921	2326	2326.0	1.488241e+09	72.0	2964.0	4883.573695	2307.0
1	2326	5203821	50	1.0	2.0	2964	2326	2326.0	2.900808e+08	49.0	2939.0	808.221050	2307.0
2	2326	5361385	40	1.0	4.0	2966	2326	2326.0	1.561725e+09	46.0	2964.0	1508.842008	2307.0
3	2326	9664953	65	1.0	4.0	2946	2326	2326.0	1.488240e+09	32.0	2964.0	1352.103424	2307.0
4	2326	23992569	68	1.0	2.0	2964	2326	2326.0	2.661995e+08	60.0	2964.0	961.496080	2307.0
...
10250	2331	1976744286	8	2.0	4.0	2988	2330	NaN	NaN	NaN	NaN	NaN	NaN
10251	2331	1988359483	36	2.0	2.0	2975	2330	NaN	NaN	NaN	NaN	NaN	NaN
10252	2331	1988359483	36	2.0	2.0	2937	2330	NaN	NaN	NaN	NaN	NaN	NaN
10253	2331	2002828674	28	2.0	5.0	2954	2330	NaN	NaN	NaN	NaN	NaN	NaN
10254	2331	2007777127	47	2.0	4.0	2995	2330	NaN	NaN	NaN	NaN	NaN	NaN

10255 rows × 13 columns

```
# DataFrames l'une au-dessus de l'autre
# Concaténer sur les colonnes communes
# Si colonnes absentes => Remplies avec NaN
pd.concat([a_global, weights])
```

✓ 0.0s

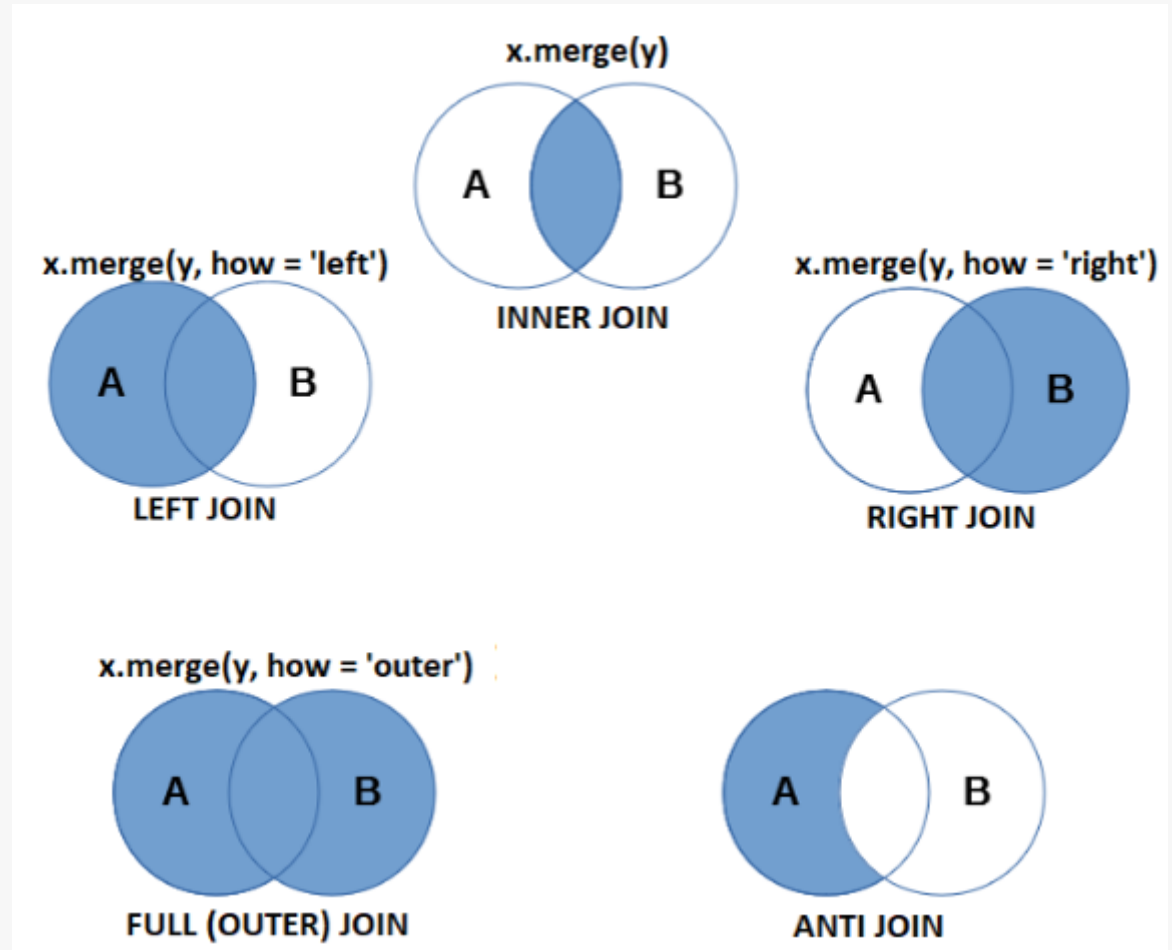
	vague_cine_inv	pid	age	gender	s3ad	q2ad	sem_cine_inv	Entrées Extrapolées	mois_cine
0	2326	3162713	57	1.0	2.0	2921	2326.0	NaN	NaN
1	2326	5203821	50	1.0	2.0	2964	2326.0	NaN	NaN
2	2326	5361385	40	1.0	4.0	2966	2326.0	NaN	NaN
3	2326	9664953	65	1.0	4.0	2946	2326.0	NaN	NaN
4	2326	23992569	68	1.0	2.0	2964	2326.0	NaN	NaN
...
9937	2331	354597332	16	NaN	NaN	2973	NaN	1042.730244	2307.0
9938	2331	93486851	15	NaN	NaN	2994	NaN	2382.030533	2307.0
9939	2331	263130660	15	NaN	NaN	2986	NaN	378.624161	2307.0
9940	2331	418923223	15	NaN	NaN	2988	NaN	1845.376712	2307.0
9941	2331	418923223	15	NaN	NaN	2929	NaN	1199.833042	2307.0

20197 rows × 9 columns

Merge (= Jointure SQL sur valeurs de colonne) deux DataFrames

Associe deux dataframes
par **une ou plusieurs colonnes**
pour **toutes les lignes**
ayant des **valeurs identiques**
sur les **colonnes choisies**

Le paramètre « **how** »
détermine les **lignes renvoyées**



3. Data : manipulation , exploration, nettoyage

Merge (= Jointure SQL sur valeurs de colonne) deux DataFrames

Les opérations **merge** sont **à l'origine** de beaucoup **d'erreurs** en data



Savoir à l'avance si après le merge on attend
Autant / Moins / Plus de lignes qu'avant le merge

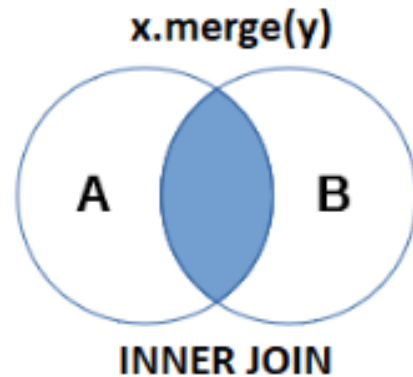
3. Data : manipulation , exploration, nettoyage

Merge (= Jointure SQL sur valeurs de colonne) deux DataFrames

df_users

✓ 0.0s

	id	name
0	1	Alice
1	2	Bob
2	3	Charlie



```
df_users.merge(df_orders, on="id", how="inner")
```

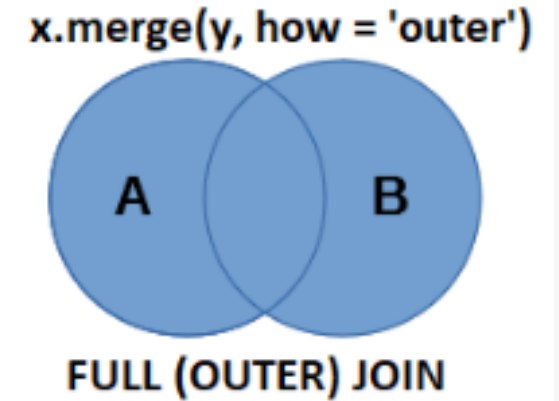
✓ 0.0s

	id	name	order
0	1	Alice	Book
1	2	Bob	Pen
2	2	Bob	Notebook

```
df_users.merge(df_orders, on="id", how="outer")
```

✓ 0.0s

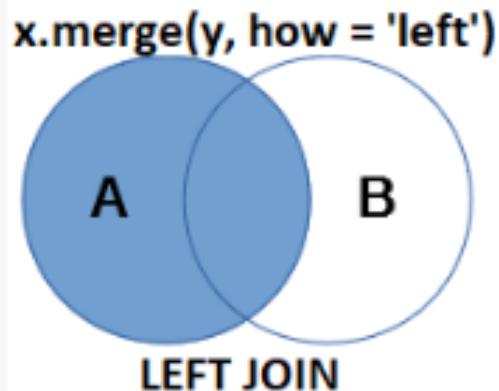
	id	name	order
0	1	Alice	Book
1	2	Bob	Pen
2	2	Bob	Notebook
3	3	Charlie	NaN
4	4	NaN	Pencil



df_orders

✓ 0.0s

	id	order
0	1	Book
1	2	Pen
2	2	Notebook
3	4	Pencil



```
df_users.merge(df_orders, on="id", how="left")
```

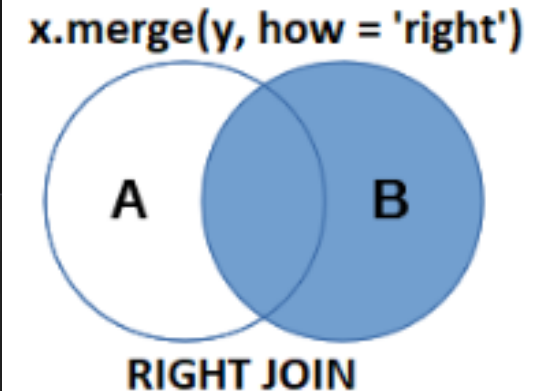
✓ 0.0s

	id	name	order
0	1	Alice	Book
1	2	Bob	Pen
2	2	Bob	Notebook
3	3	Charlie	NaN

```
df_users.merge(df_orders, on="id", how="right")
```

✓ 0.0s

	id	name	order
0	1	Alice	Book
1	2	Bob	Pen
2	2	Bob	Notebook
3	4	NaN	Pencil




3. Data : manipulation , exploration, nettoyage


Merge (= Jointure SQL sur valeurs de colonne) deux DataFrames

```
print(a_global.columns.to_list())
a_global.shape
✓ 0.0s
['vague_cine_inv', 'pid', 'age', 'gender', 's3ad', 'q2ad', 'sem_cine_inv']
(10255, 7)
```


```
print(weights.columns.to_list())
weights.shape
✓ 0.0s
['vague_cine_inv', 'pid', 'age', 'q2ad', 'Entrées Extrapolées', 'mois_cine']
(9942, 6)
```



```
a_global.merge(weights, how='inner', on=['vague_cine_inv']).shape
✓ 2.0s
(18085754, 12)
```



```
a_global.merge(weights, how='inner', on=['vague_cine_inv', 'q2ad']).shape
✓ 0.0s
(1406841, 11)
```



```
a_global.merge(weights, how='inner', on=['vague_cine_inv', 'age', 'q2ad']).shape
✓ 0.0s
(37158, 10)
```

```
a_global.merge(weights, how='inner', on=['vague_cine_inv', 'pid', 'age', 'q2ad']).shape
✓ 0.0s
(9802, 9)
```


3. Data : manipulation , exploration, nettoyage

Merge (= Jointure SQL sur valeurs de colonne) deux DataFrames

sort=True #Default

```
a_global.merge(weights, how='inner', on=['vague_cine_inv']).shape
```

✓ 2.0s

(18085754, 12)

sort=False

```
a_global.merge(weights, how='inner', on=['vague_cine_inv'], sort=False).shape
```

✓ 1.1s

(18085754, 12)

45 % de gain

3. Data : manipulation , exploration, nettoyage

Join (=Jointure sur index ou nom de colonne) deux DataFrames

Même principe que merge sur les **index/noms de colonne**

```
a_global = a_global.set_index('vague_cine_inv')
weights = weights.set_index('vague_cine_inv')
# rsuffix needed if the same column names are present in both DataFrames
# lsuffix is also available if needed
a_global.join(weights, how='inner', rsuffix='_weight', sort=False).shape
✓ 4.1s
(18085754, 11)
```



- **Petit dataset**
- **Jointures sur index**
- **Multiples jointures consécutives**

3. Data : manipulation , exploration, nettoyage

Agréger un DataFrame / Une Series

```
a_global.sum()
✓ 0.0s
```

pid	6.451735e+12
age	3.787160e+05
gender	1.566900e+04
s3ad	3.658800e+04
q2ad	2.859071e+07
sem_cine_inv	2.387283e+07
dtype:	float64

```
a_global.agg({'age': 'mean', 'sem_cine_inv': 'sum'}).round(2)
✓ 0.0s
```

age	36.93
sem_cine_inv	23872831.00
dtype:	float64

```
a_global['age'].agg(
    ('median', 'max', 'min', 'mean', 'std', 'var', 'count',)
)
✓ 0.0s
```

median	38.000000
max	86.000000
min	3.000000
mean	36.929888
std	20.340951
var	413.754300
count	10255.000000
Name:	age, dtype: float64

```
weights.groupby('vague_cine_inv')\
    [['Entrées Extrapolées']].sum()
✓ 0.0s
```

vague_cine_inv	Entrées Extrapolées
2326	1.653242e+06
2327	3.392680e+06
2328	3.298210e+06
2329	4.197077e+06
2330	4.852108e+06
2331	2.331370e+06

```
weights.groupby('vague_cine_inv', as_index=False).agg(
    {'Entrées Extrapolées': 'sum',
     'age': 'mean',})
✓ 0.0s
```

	vague_cine_inv	Entrées Extrapolées	age
0	2326	1.653242e+06	35.000000
1	2327	3.392680e+06	39.258310
2	2328	3.298210e+06	37.632086
3	2329	4.197077e+06	35.396070
4	2330	4.852108e+06	36.503308
5	2331	2.331370e+06	40.779034

```
weights.pivot_table(
    index='vague_cine_inv',
    columns='mois_cine',
    values='Entrées Extrapolées',
    aggfunc='sum', # default "mean"
)
✓ 0.0s
```

	mois_cine	2307
vague_cine_inv		
	2326	1.653242e+06
	2327	3.392680e+06
	2328	3.298210e+06
	2329	4.197077e+06
	2330	4.852108e+06
	2331	2.331370e+06

4. Data : Visualisation

Document confidentiel.
Ne peut être reproduit ni diffusé
sans l'autorisation de VERTIGO.



Entertainment Marketing Research

**1 Graphique
=
1 message clair**

Message clair = compris en -3 sec

Bon choix de graphiques

Contraste de couleurs, de formes

Un titre explique et contextualisé

Pas de répétition

Cohérence dans le sens de lecture

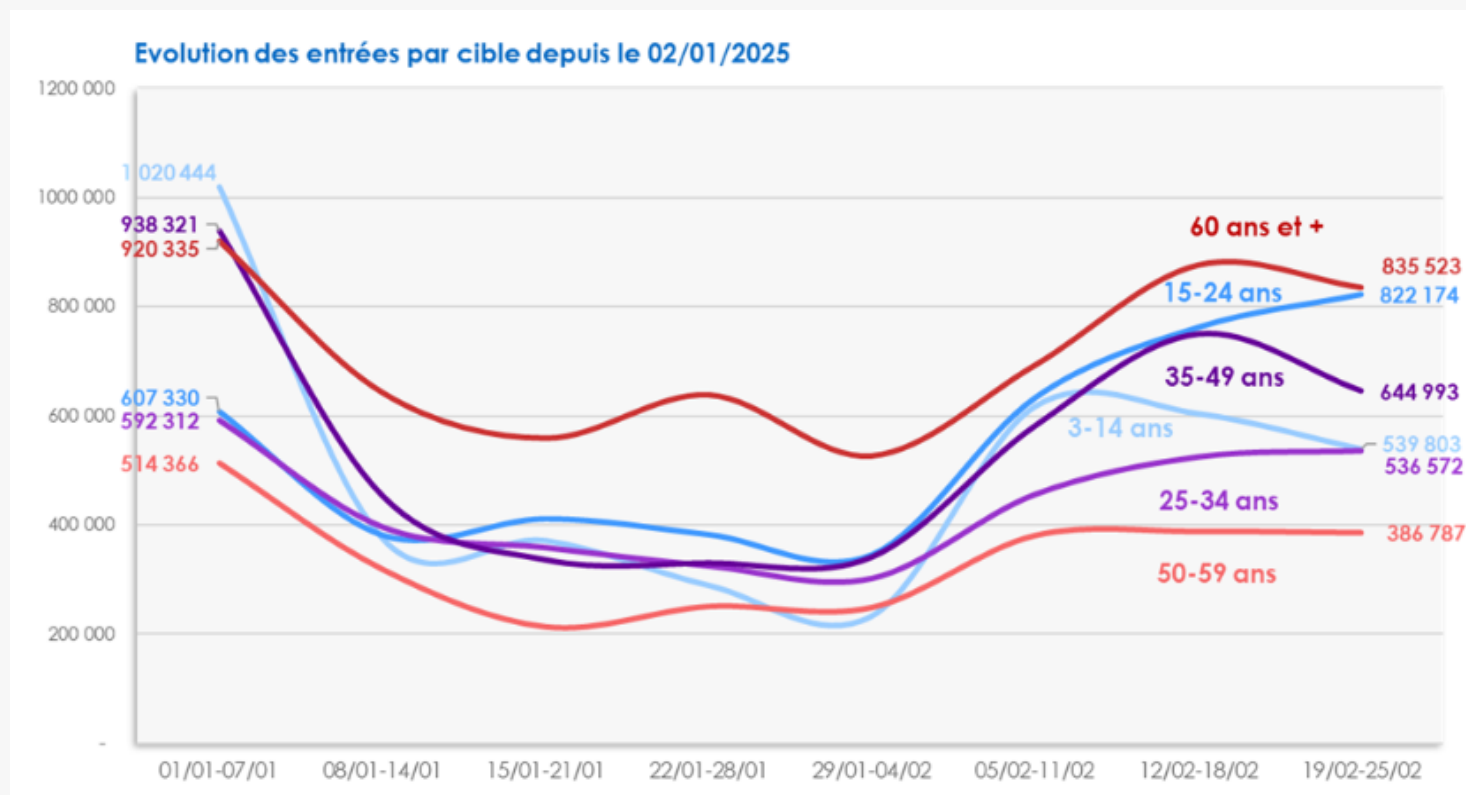
...

4. Data : Visualisation

Types : Graphique en Ligne

Objectifs : Evolution temporelle, Analyses de tendance

Avantages : Lecture, Comparaison, Contraste (forme, couleur...)

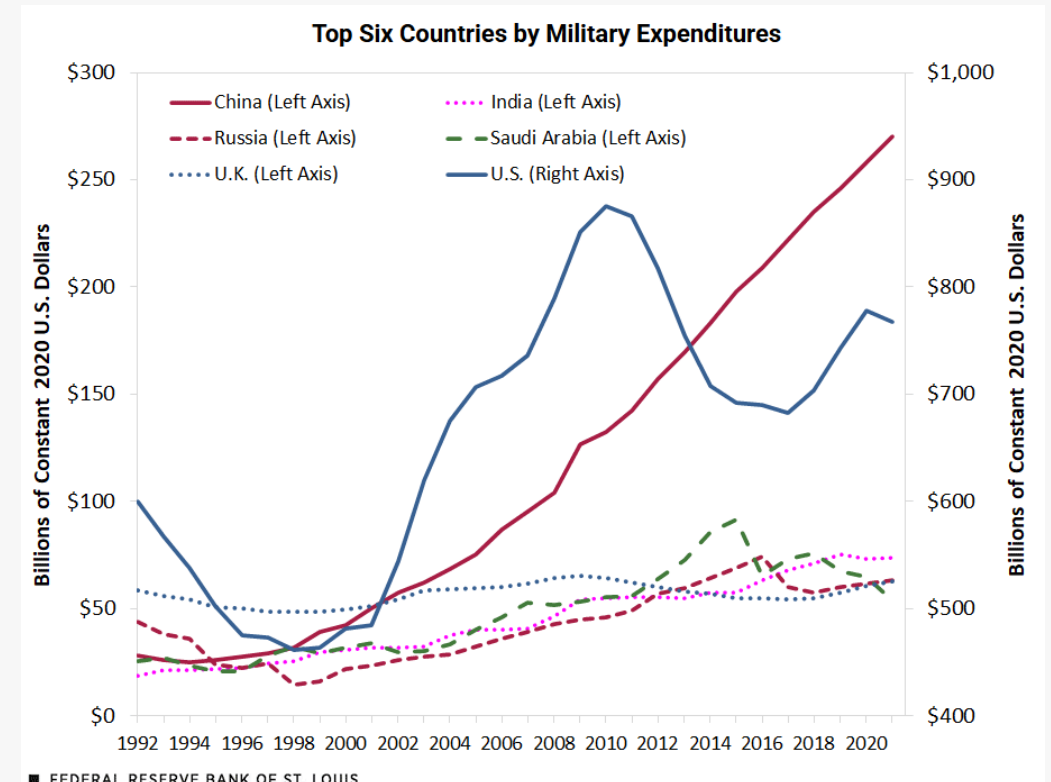
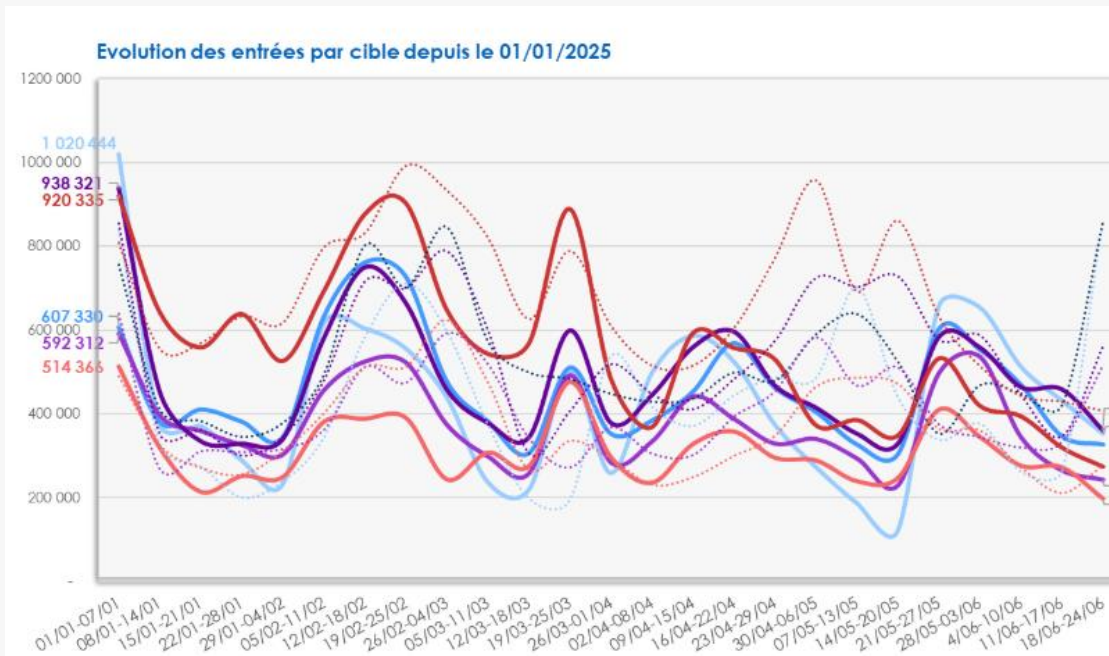


Source : CinExpert Vertigo

4. Data : Visualisation

Types : Graphique en Ligne

ERREURS A EVITER : Trop de lignes , 2 axes + identification ---, ...

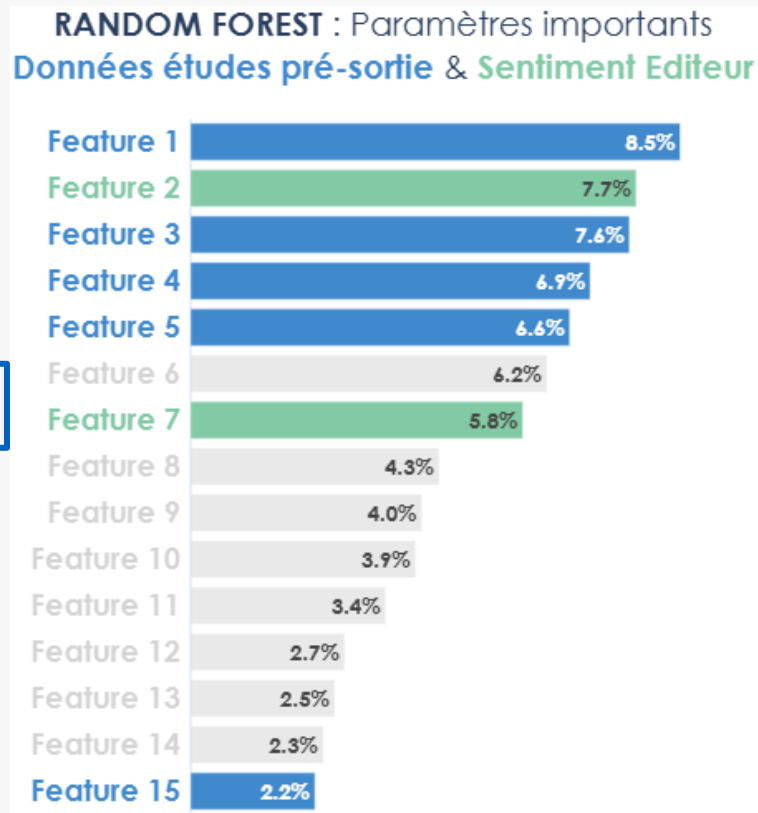


4. Data : Visualisation

Types : Diagramme en barres

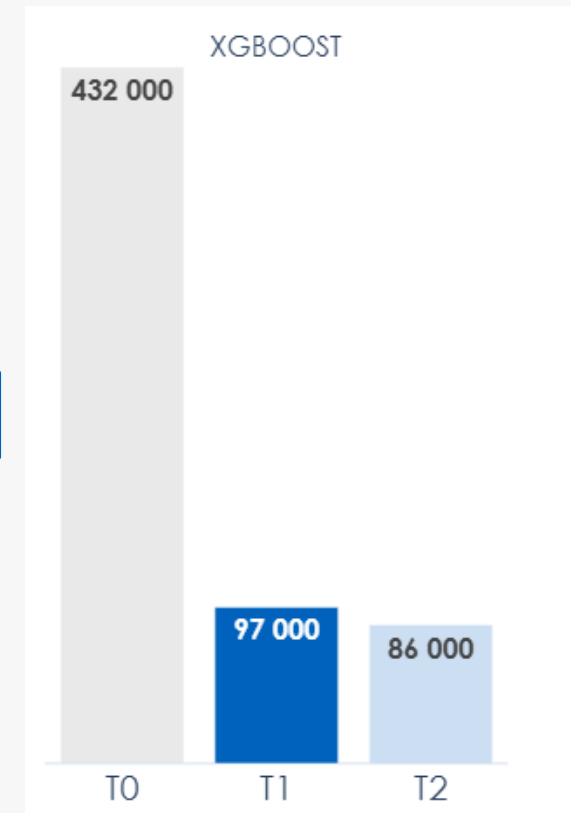
Objectifs : Classement, Comparer des quantités par catégorie

Avantages : Lecture, Comparaison, Nombre élevé de catégories



Horizontal

Copyright : Vertigo

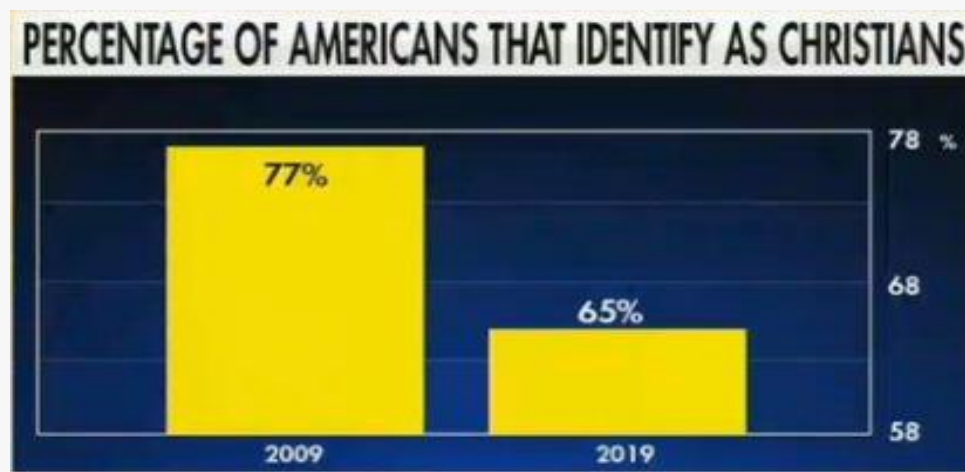


Vertical

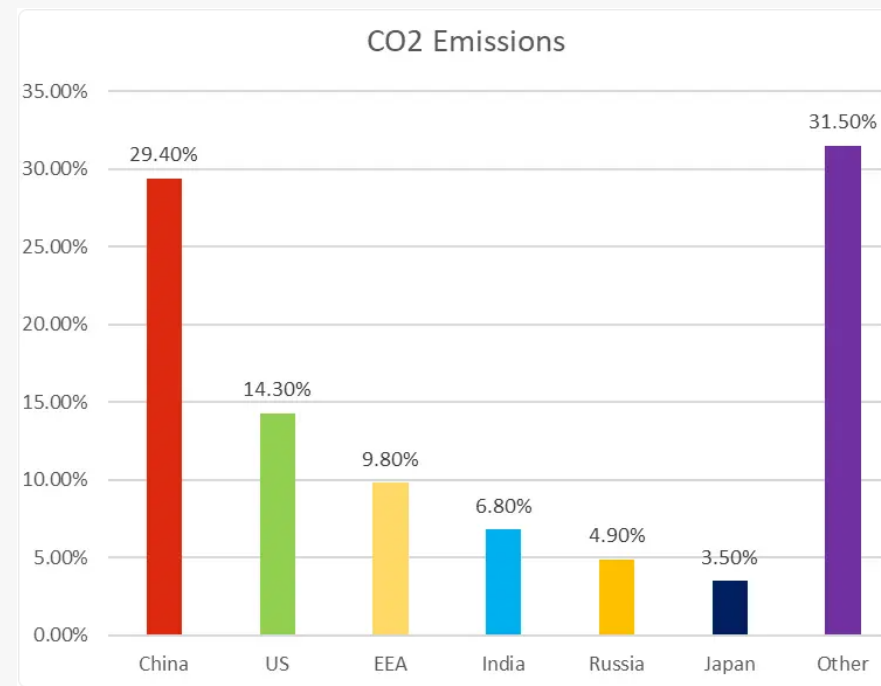
Copyright : Vertigo

Types : Diagramme en barres

ERREURS A EVITER : Ne pas commencer à 0, Trop de couleurs, ...



Source : [12 Bad Data Visualization Examples Explained - Code Conquest](#)
Diffusé sur Fox News



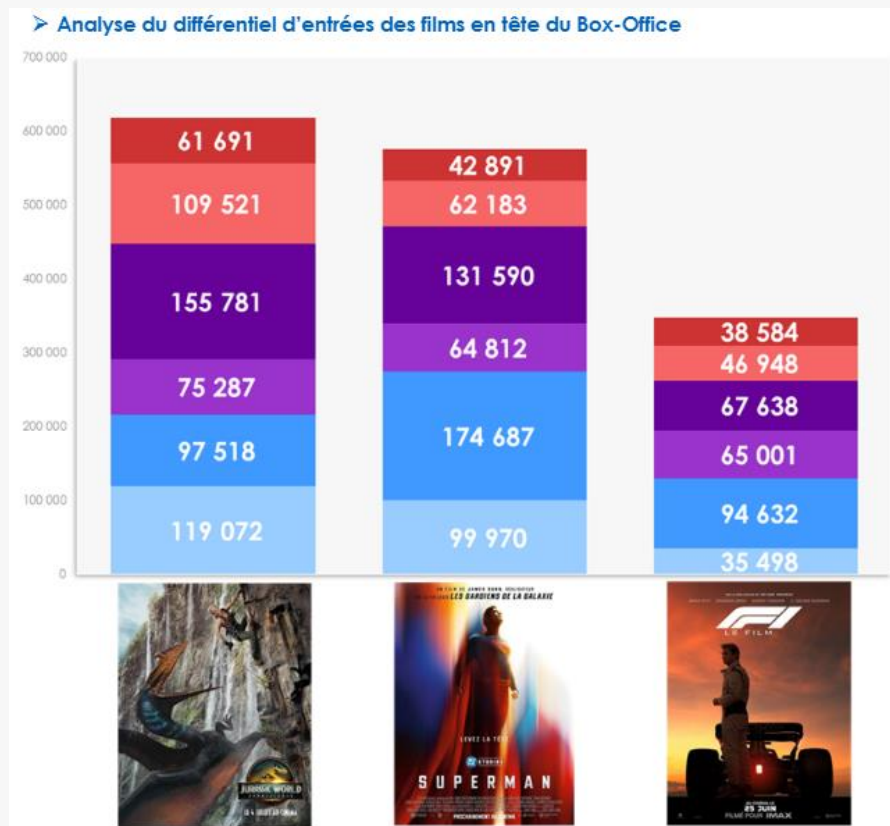
Source : [10 Good and Bad Examples of Data Visualization - Polymer](#)

4. Data : Visualisation

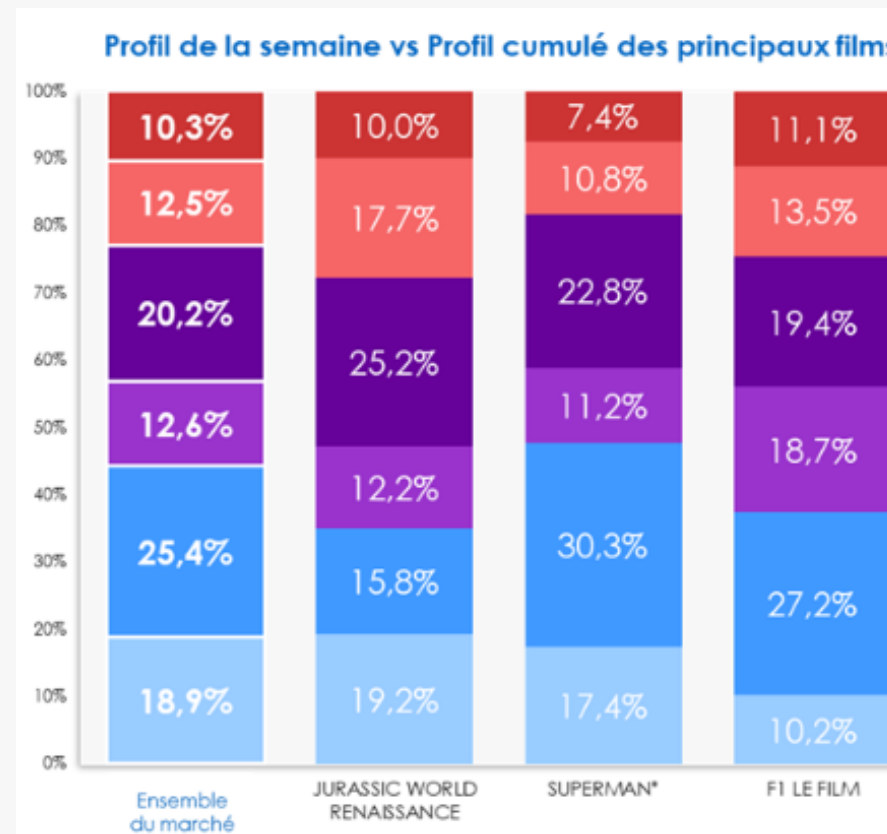
Diagramme en barres empilées

Objectifs : + 1 dimension aux diagrammes en barres, Comparer les parties d'un tout

Avantages : Lecture, Comparaison, Analyse poussée



Source : CinExpert Vertigo

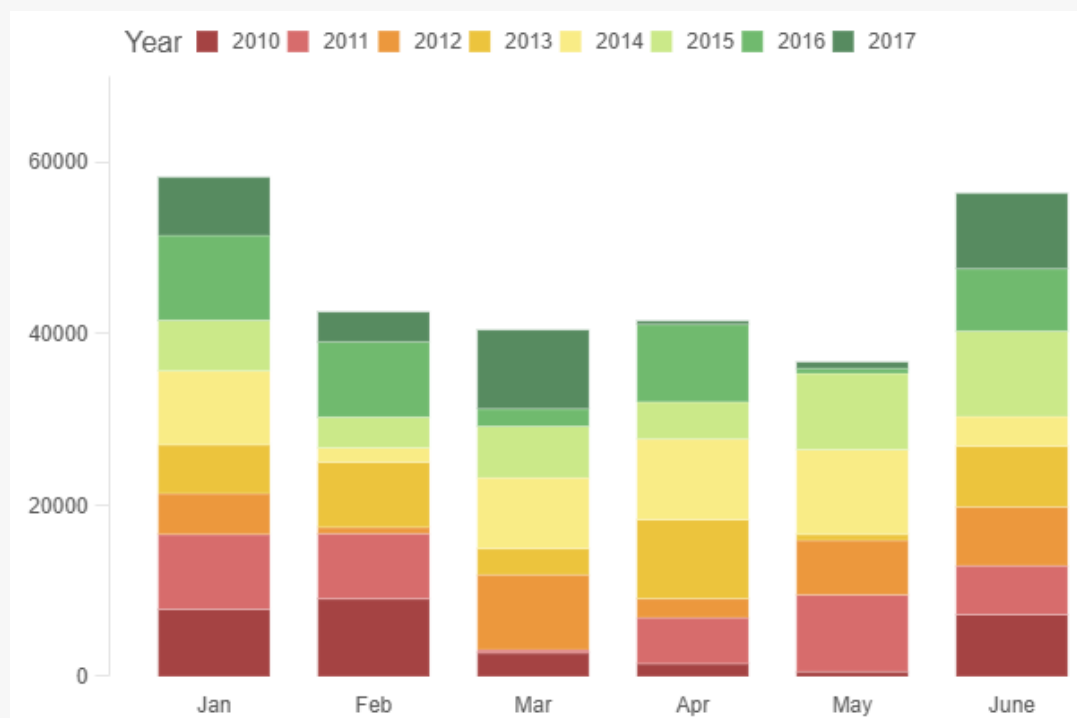


Source : CinExpert Vertigo

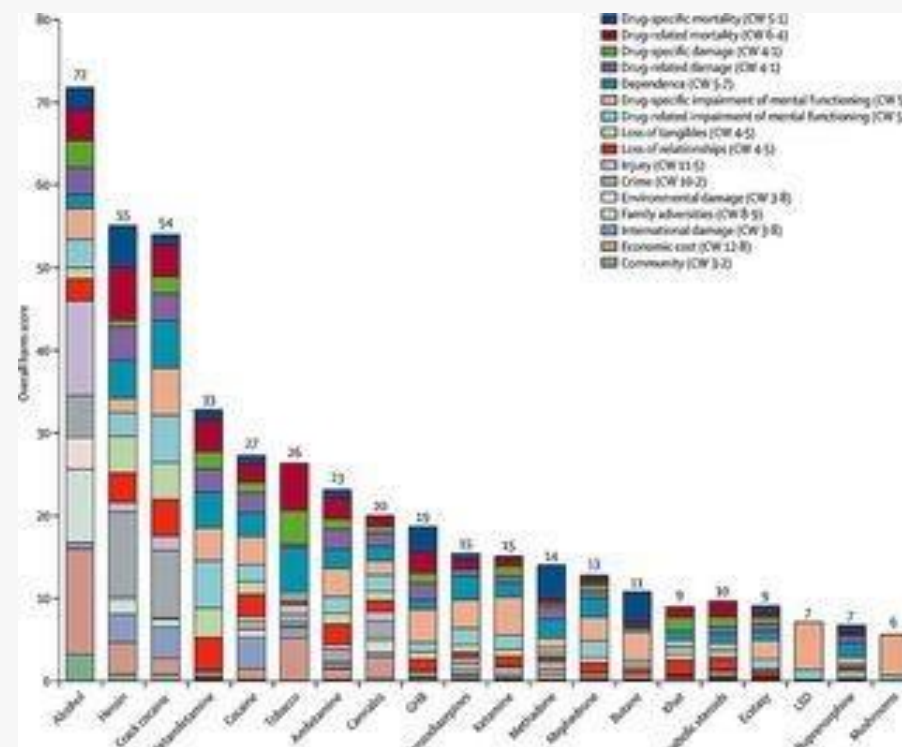
4. Data : Visualisation

Diagramme en barres empilées

ERREURS A EVITER : Ne pas mettre les valeurs, trop de catégories, ...



Source : <https://observablehq.com/@miralemd/picasso-js-stacked-bar-chart>



Source : <https://365datascience.com/trending/chart-types-and-how-to-select-the-right-one/>

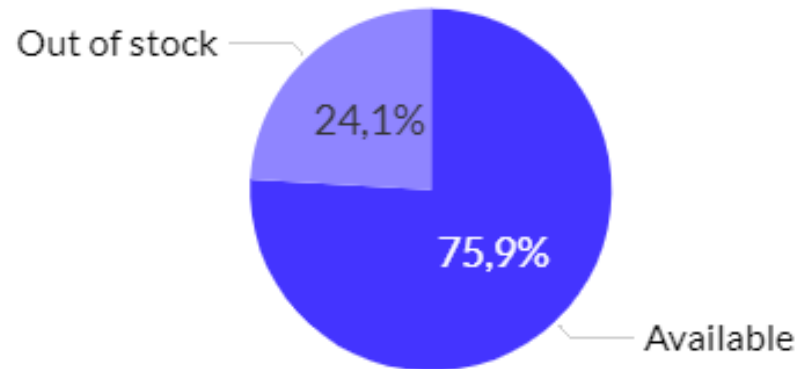
4. Data : Visualisation

Types : Camembert

Objectifs : Comparer les parties d'un tout

Avantages : Connu et rassurant, Change le rythme de présentation

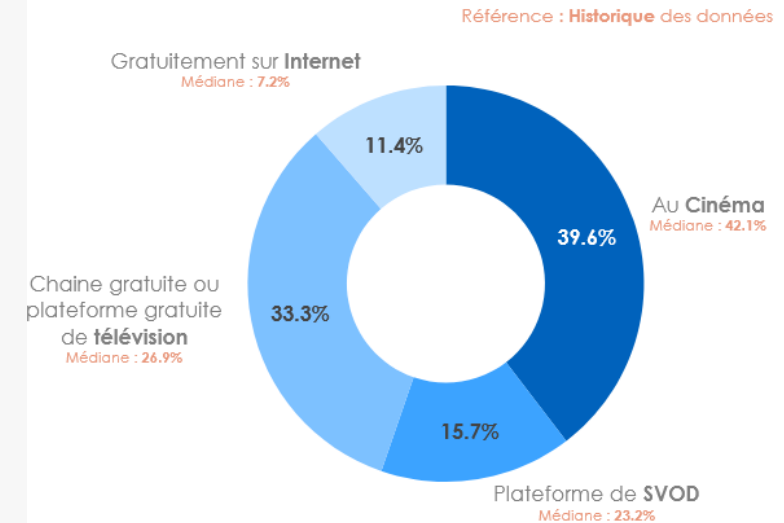
Product inventory



Source: https://www.luzmo.com/blog/chart-types?utm_source=chatgpt.com

Version Donut

Support pour voir - Toutes Cibles

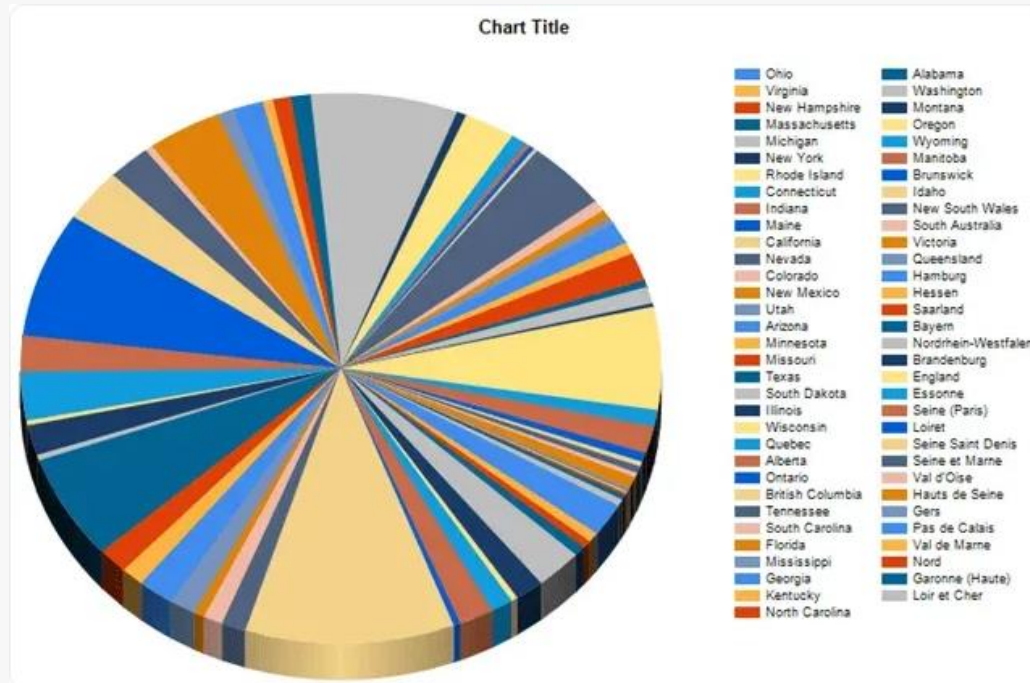


Source: Flash Film Target®/Gravity® Vertigo

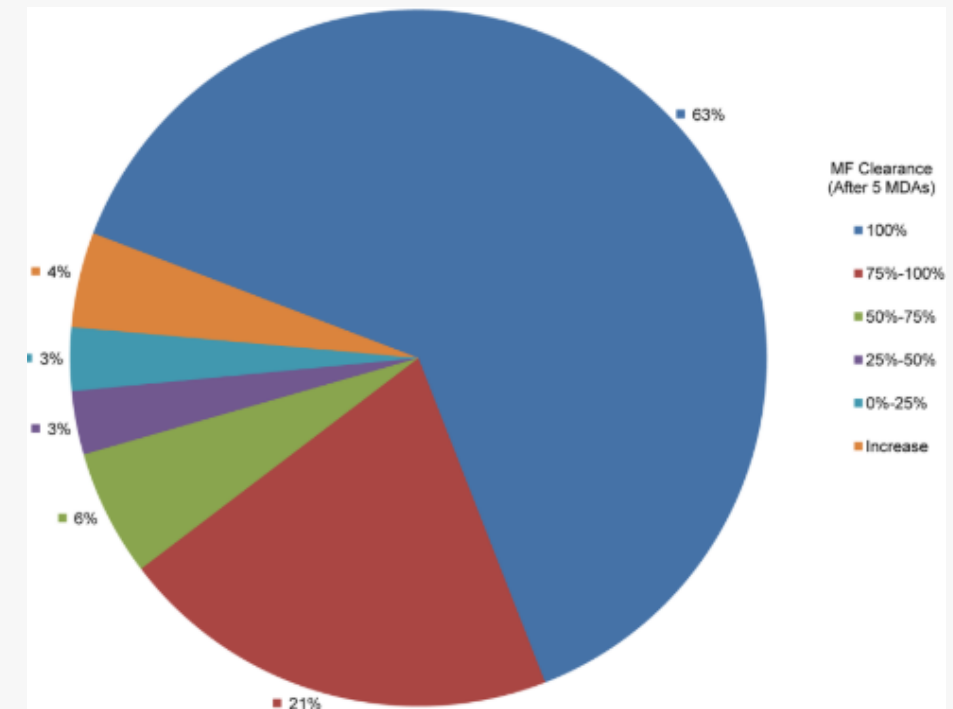
4. Data : Visualisation

Types : Camembert

ERREURS A EVITER : Trop de catégories, Ne pas trier, Pas de verticale, total \neq 100 %, ...



Source: <https://365datascience.com/trending/chart-types-and-how-to-select-the-right-one/>

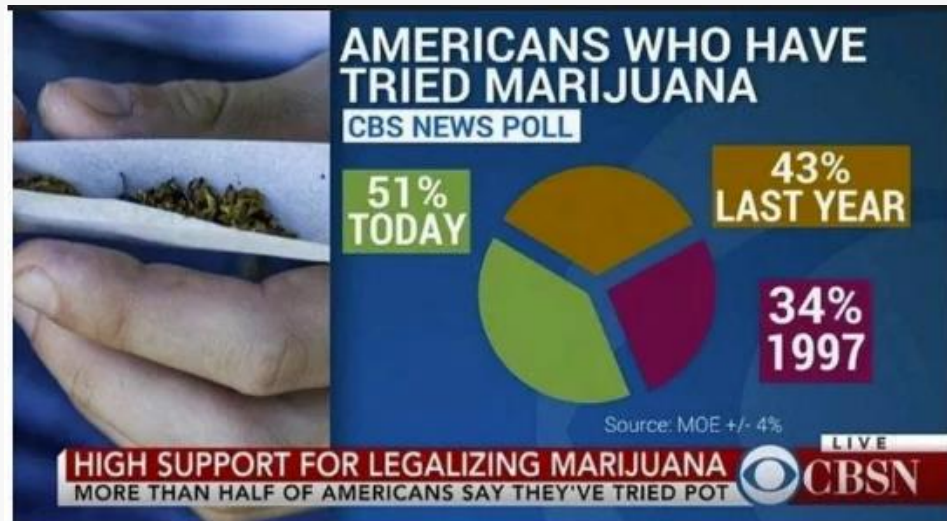


Source: <https://animalia-life.club/qa/pictures/elephantiasis-worm-life-cycle>

4. Data : Visualisation

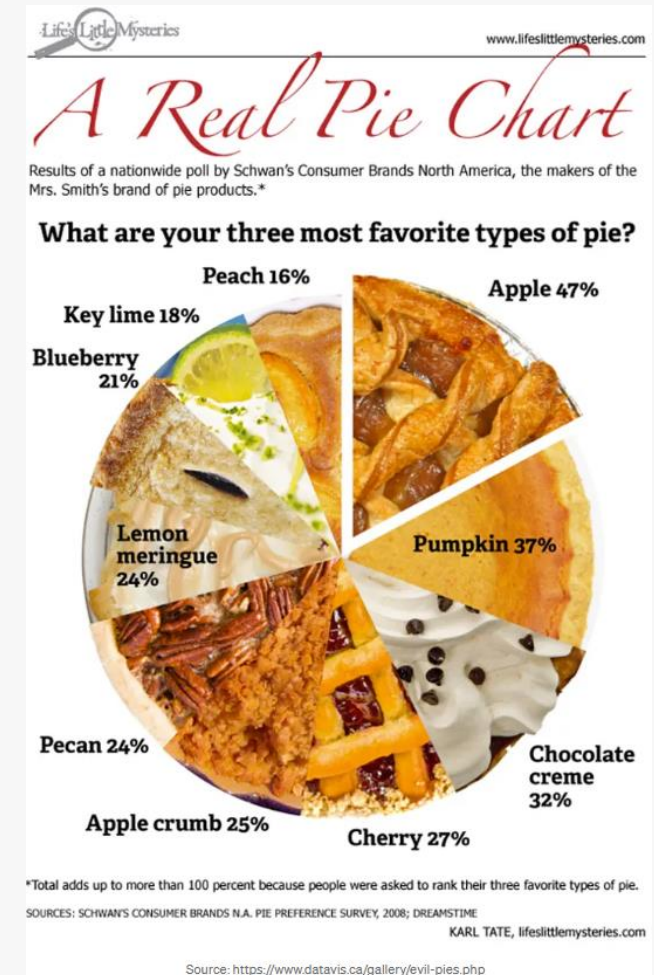
Types : Camembert

ERREURS A EVITER : Trop de catégories, Ne pas trier, Pas de verticale, total \neq 100 %



Source: <https://www.painting-with-numbers.com/blog/getting-high-on-bad-data-visualization/>

Source : <https://medium.com/learning-data/hall-of-shame-pie-charts-that-should-have-never-been-baked-d4030778c9ee>

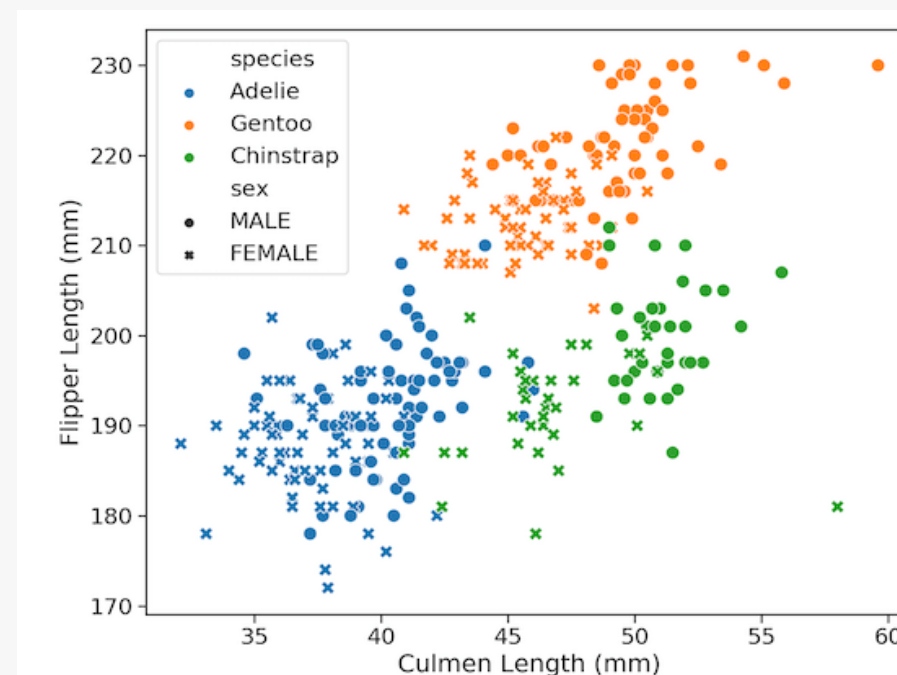
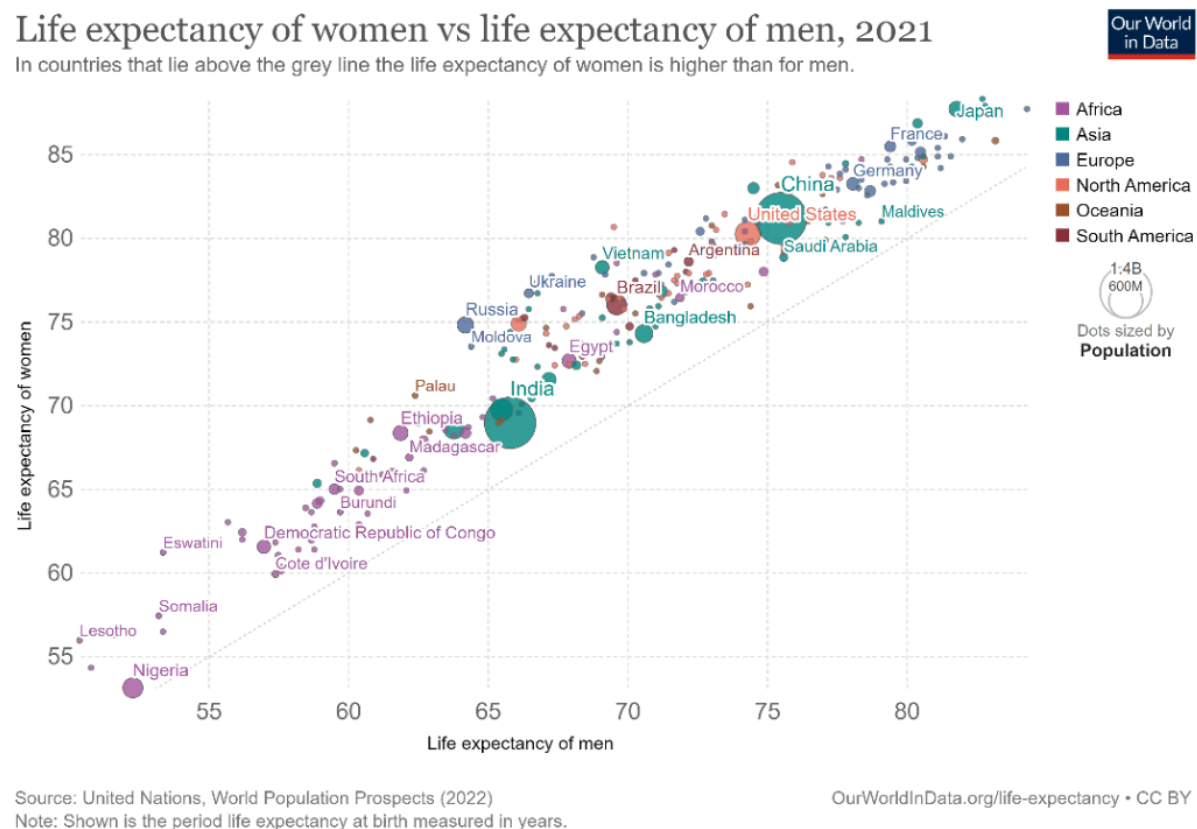


4. Data : Visualisation

Types : Nuage de points

Objectifs : Relation/corrélation entre variables, Détection de clusters/outliers

Avantages : Tailles, formes et couleurs variables, Business Insights rapides

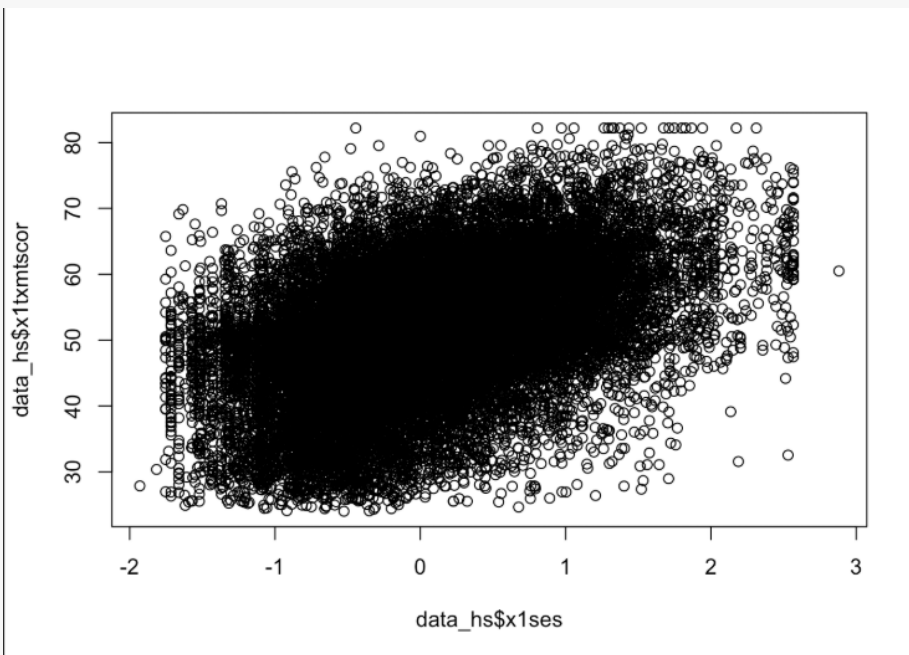


Source : <https://jokersmooth.weebly.com/blog/seaborn-scatter-plot-color>

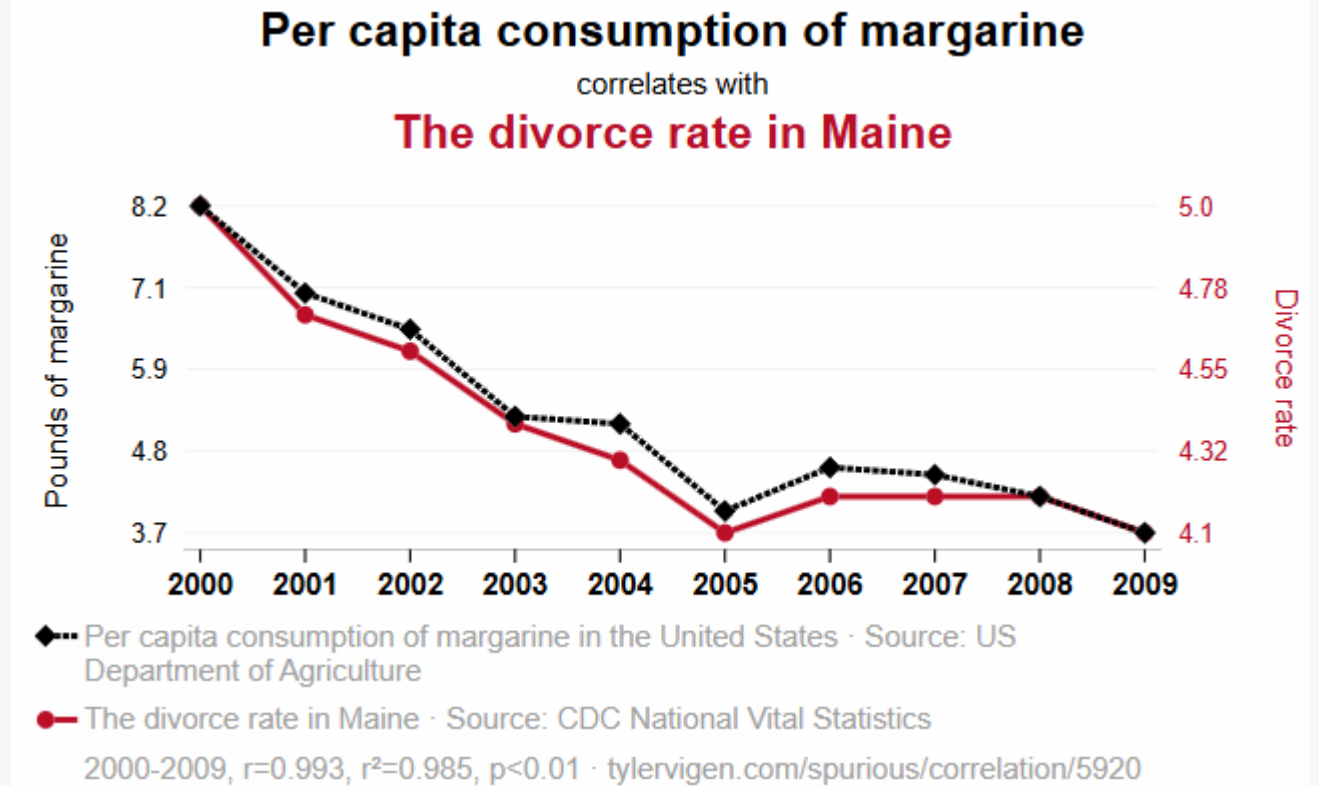
4. Data : Visualisation

Types : Nuage de points

ERREURS A EVITER : Trop de points, confondre corrélation et causalité, ...



Source : <https://capaldi.info/7916/05-viz-i.html>



Source : <https://www.tylervigen.com/spurious-correlations>

4. Data : Visualisation

Heatmap:

Patterns temporels,
Analyse de corrélation

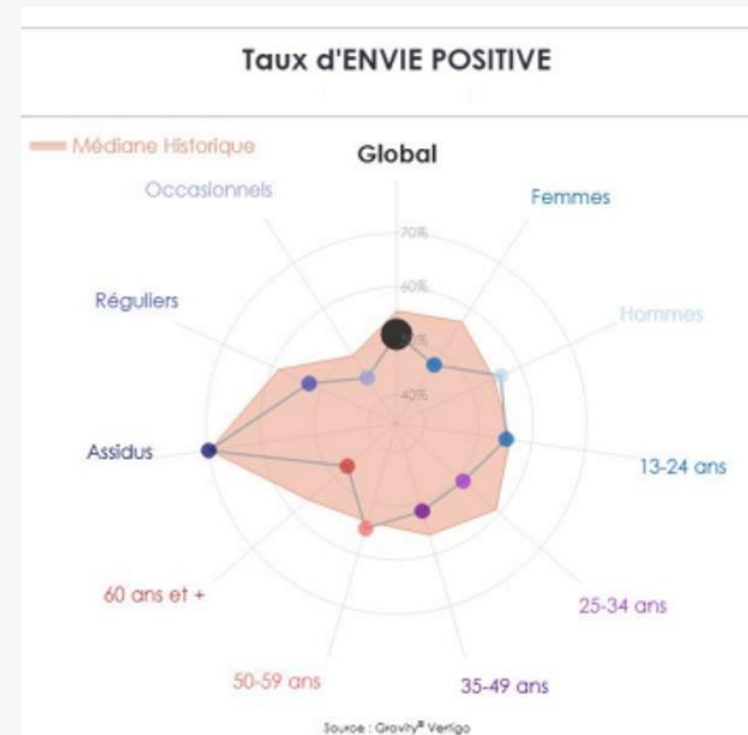


Source : <https://bidataintel.com/2021/04/using-colour-in-data-visualisation/>

Types : Autres

Radar:

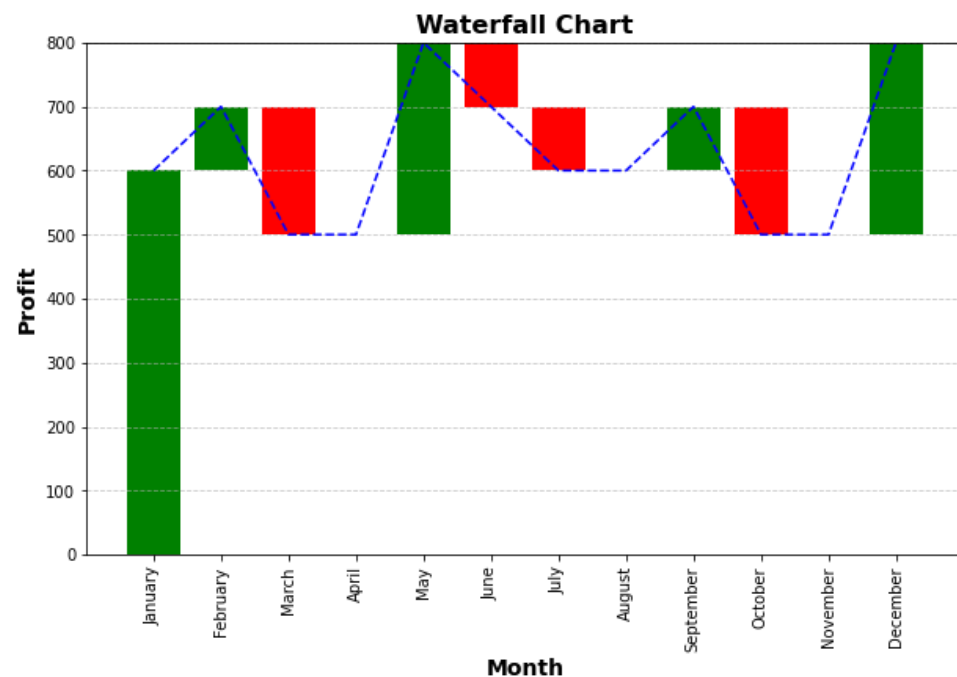
Comparaison plusieurs
catégories



Source : Vertigo Gravity®

4. Data : Visualisation

Waterfall : Gain/Perte

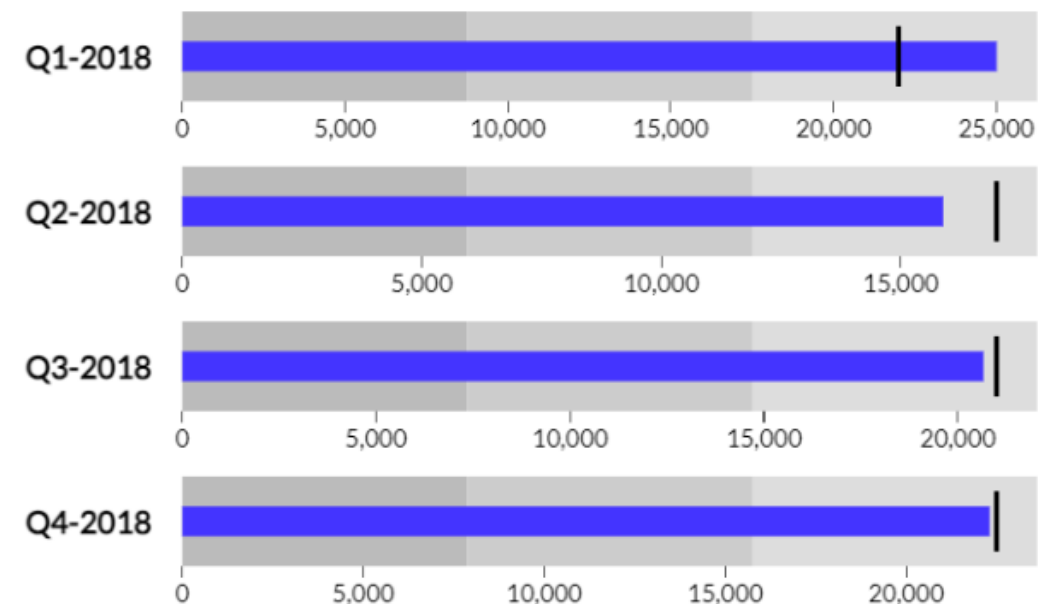


Source : https://www.luzmo.com/blog/chart-types?utm_source=chatgpt.com

Types : Autres

Bullet : Atteinte objectifs

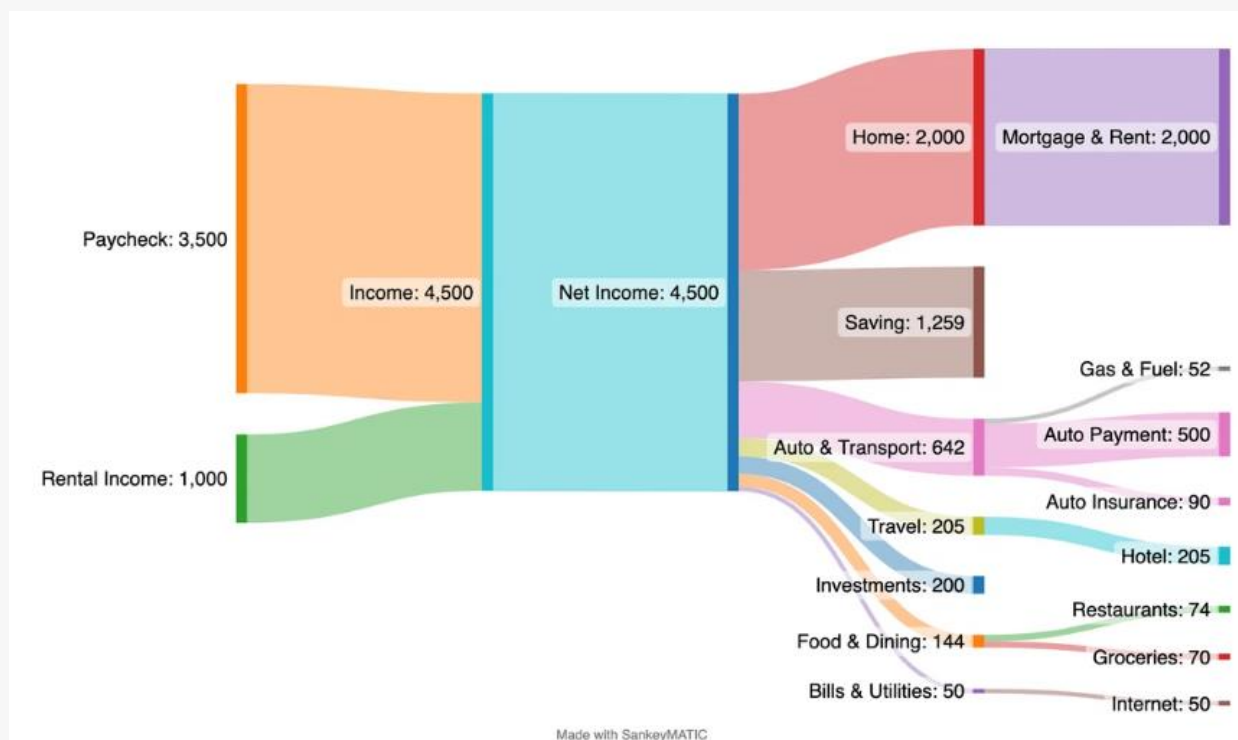
Sales target attainment per quarter



Source : https://www.luzmo.com/blog/chart-types?utm_source=chatgpt.com

4. Data : Visualisation

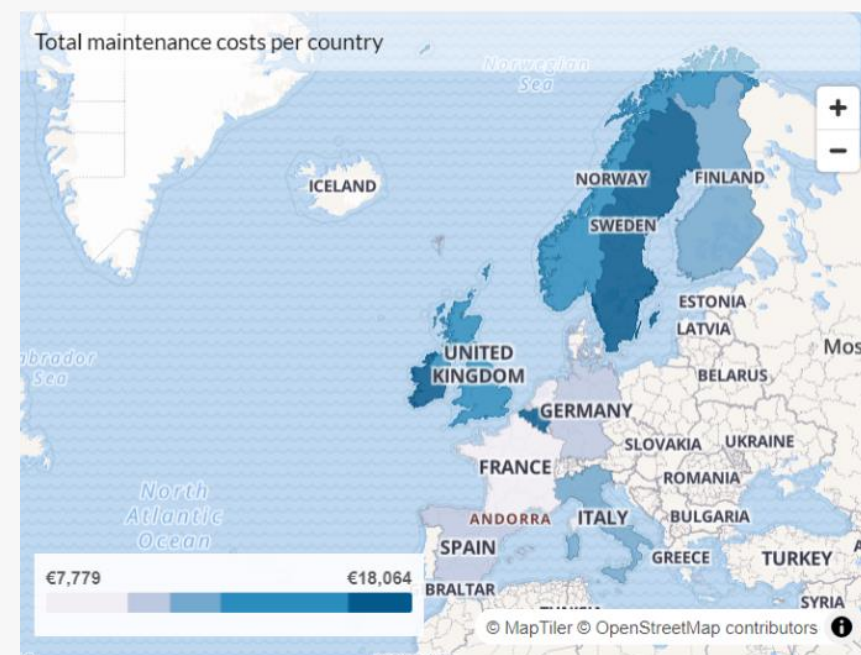
Sankey : Modéliser les flux
(personnes, argent, énergie)



Source : <https://nyan.im/p/sankey-diagram-mint>

Types : Autres

Choropleth : Carte géographique

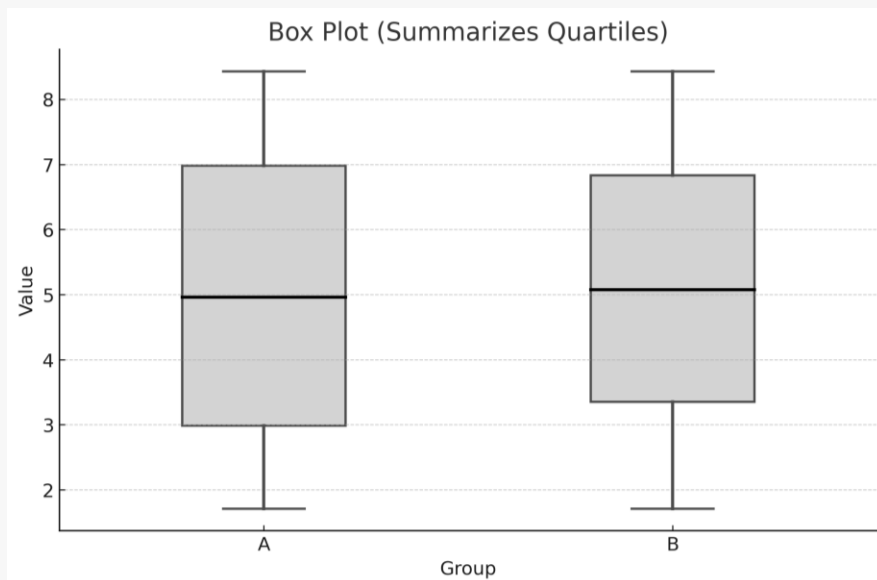


Source : https://www.luzmo.com/blog/chart-types?utm_source=chatgpt.com

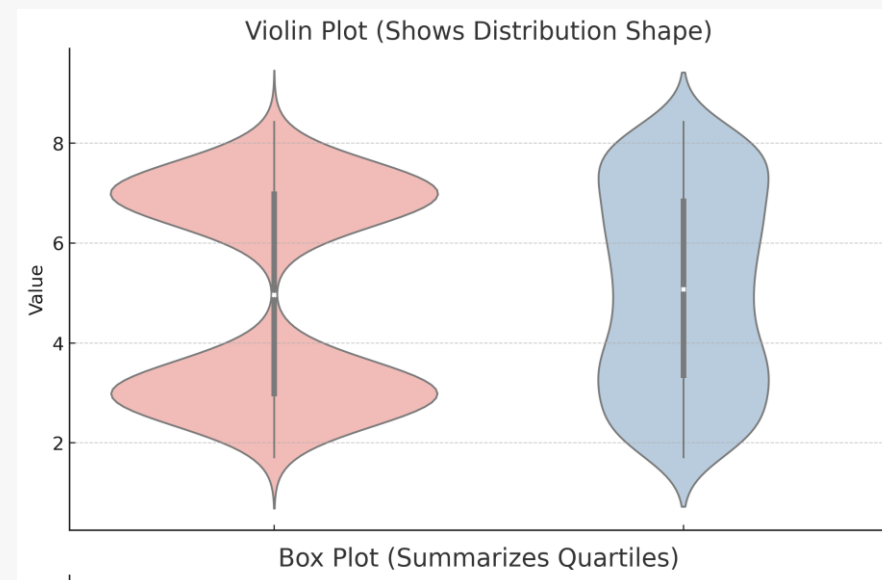
4. Data : Visualisation

Types : Autres

Boxplot : statistiques de groupes



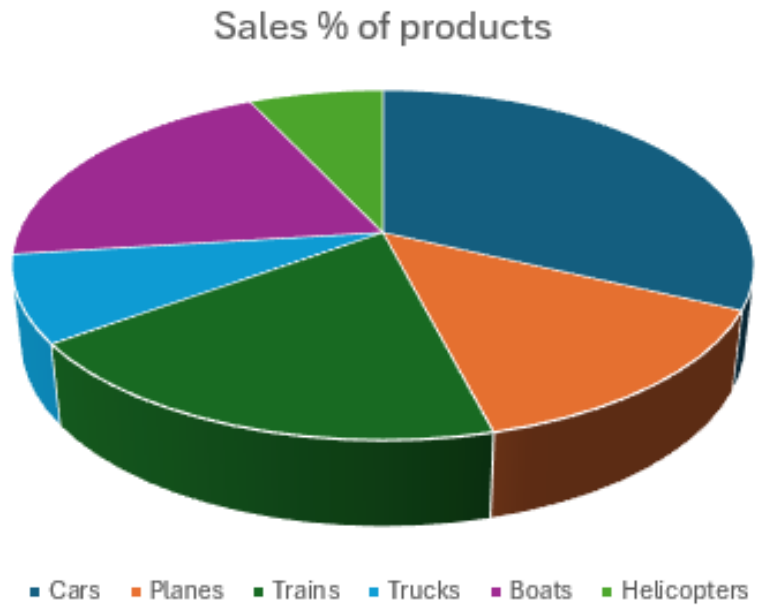
Violin plot : stats + répartition



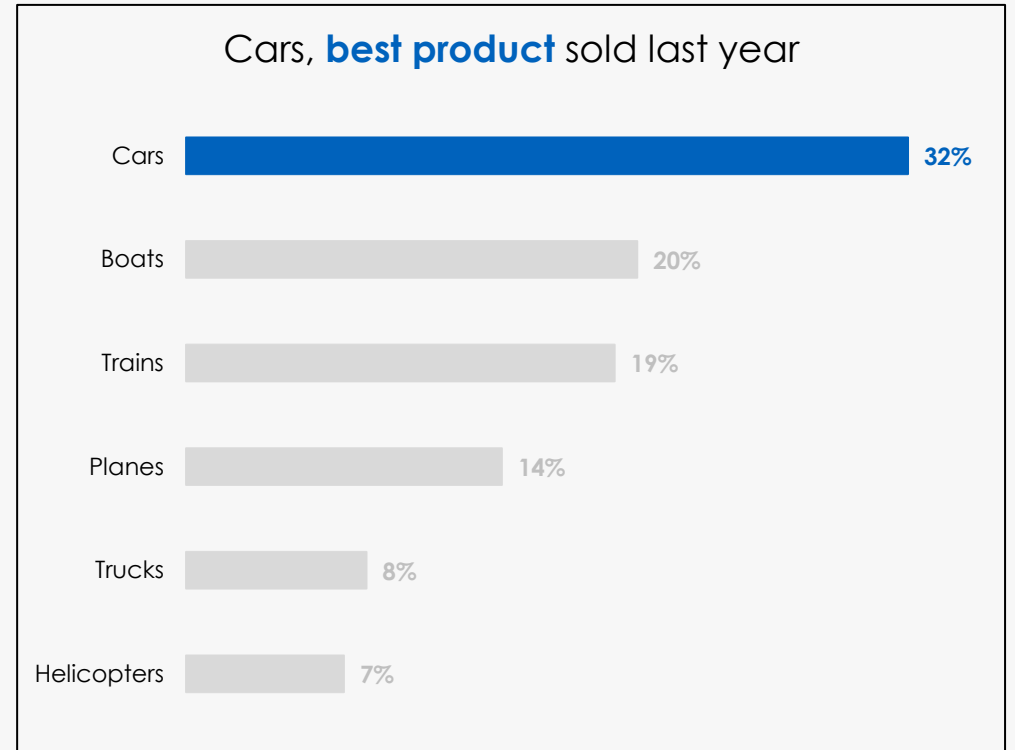
GENERAL GUIDELINES

- Ecrire un titre qui met l'emphasis sur le message à faire passer
- Utiliser des sous-titres pour plus de contexte si besoin
- Enlever les traits et les labels non nécessaires (grid, axes, ...)
- Utiliser la bonne échelle (avec 0 si barre, pas nécessaire si nuage de points)
- Utiliser le gris pour mettre en valeur le message
- Utiliser un dégradé d'une couleur plutôt que plusieurs couleurs
- N'oubliez pas vos sources !

LESS IS MORE



Fuir les effets 3D



Comparaison + facile avec longueur

Librairies

Plotly.express (px)

+

Interactivité
Visuel par défaut
Simple et rapide
Syntaxe concise

-

Contrôle limité
Pas de subplot
Debug difficile

➡ **Exploration Rapide**

Plotly.graph_objects (go)

+

Interactivité
Visuel
Full control
Subplot multiples

-

Verbeux
Learning curve
Nommer df

➡ **Livable Client**

Matplotlib (plt)

Classique
Syntaxe --
Communauté +++

Seaborn (sns)

Construit sur matplotlib
Syntaxe ++
Visuel ++

Autres

Bokeh
Cutechart (main levée)
Pygwalker (Tableau) ...

4. Data : Visualisation

Plotly.express (px)

```
import plotly.express as px

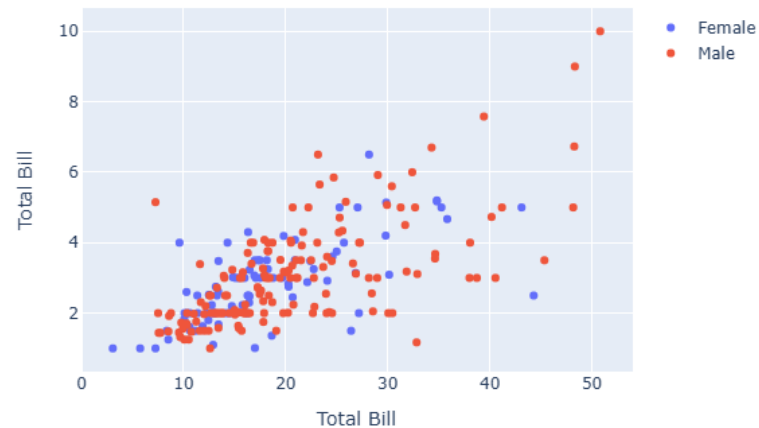
df = px.data.tips()

fig = px.scatter(
    df, # Naming df once
    x='total_bill',
    y='tip',
    color='sex', # auto separation by gender
    title='Tips vs Total Bill',
    width=600
)

fig.show()
```



Tips vs Total Bill



Plotly.graph_objects (go)

```
import plotly.graph_objects as go

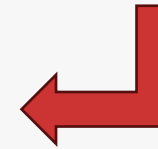
df = px.data.tips()

fig = go.Figure() # Instantiate the chart

# Each gender trace must be added separately to the figure
for sex in df['sex'].unique():
    sub_df = df[df['sex'] == sex] # Filtering gender
    fig.add_trace( # Adding the gender trace
        go.Scatter(
            x=sub_df['total_bill'], # Necessity to name the filtered dataframe
            y=sub_df['tip'],
            mode='markers', # == nuage de points etc.
            name=sex # Legend name
        )
    )

fig.update_layout(
    title=dict(text='Tips vs Total Bill'),
    xaxis=dict(title='Total Bill'),
    yaxis=dict(title='Tip'),
    width=600
)

fig.show()
```



4. Data : Visualisation

Plotly.express (px) ❌

✅ Plotly.graph_objects (go)

Données :

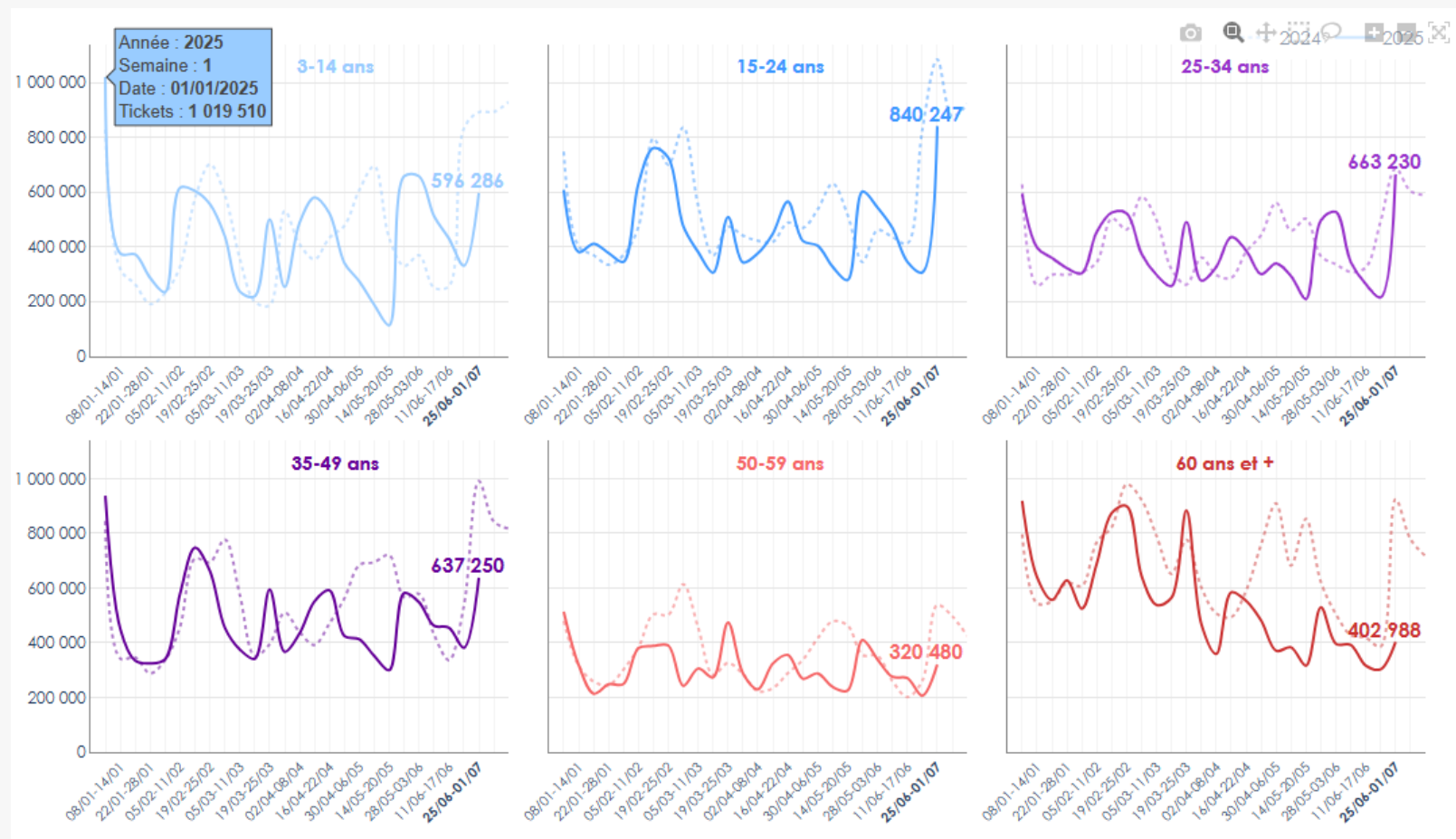
Taille (2024 > 2025)
Look (dashed vs solid)

Layout :

Espaces entre subplots
Abscisses, que dates 2025
Pas de label smart formatting

Customisation par subplot :

Textbox
Titre et chiffres
Légende



Source : Vertigo CinExpert ®

5. Data : Présentation

Document confidentiel.
Ne peut être reproduit ni diffusé
sans l'autorisation de VERTIGO.



Entertainment Marketing Research

GENERAL GUIDELINES

- Connaitre son public
- Raconter une histoire
 - ⇒ Introduction = **Contexte**
 - ⇒ Résultat + Explication = **Insight**
 - ⇒ Conclusion = **Recommandation**
- Focus sur le **résultat**, pas sur le moyen
 - ✗ « J'ai utilisé Python pour ... »
 - ✓ « On a gagné 3h/sem sur cette tâche ... »
- Dire ce qu'on va dire, le dire, dire ce qu'on a dit
 - ⇒ Répéter = Retenir

CONSTRUCTION

1. Résultats = Corps de l'analyse

2. Conclusion = « A retenir » des résultats

3. Introduction = Histoire qui amène aux résultats