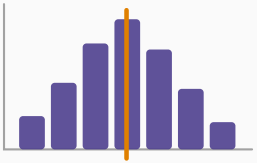
Statistique descriptive

Chapitre 16 : Mise en œuvre sur Python



Jaouad Madkour

Faculté d'Économie et de Gestion de Tanger · 27 août 2026

Plan du chapitre

La même chaîne, un autre outil

Ce qui change, ce qui ne change pas

Les trois bibliothèques

Bibliothèque	Rôle
`pandas`	tableaux de données, tableaux croisés
`numpy`	calcul numérique
`matplotlib`	graphiques

`scipy.stats` complète pour les coefficients de forme.

Ce qui change, ce qui ne change pas (suite)

Le seul vrai piège du chapitre

Là où R divise par $n - 1$ **partout**, pandas et numpy ne font **pas le même choix** :

- `numpy` divise par n par défaut ;
- `pandas` divise par $n - 1$ par défaut.

Deux bibliothèques du même écosystème, deux conventions opposées. C'est la source d'erreur numéro un, et elle explique la plupart des écarts inexpliqués entre un calcul manuel et un résultat Python.

```
import numpy as np, pandas as pd
import matplotlib.pyplot as plt
from scipy import stats
```

Charger et explorer

```python
sal = pd.read_csv("salaires.csv")

sal.info()          # types et valeurs manquantes
sal.head()
sal.describe()      # quantitatives seulement
sal.describe(include="object")   # qualitatives
```

La sortie de describe()

```
             salaire  anciennete
count     500.000000  500.000000
mean     7158.260000   10.142000
std      2600.701688    5.970354
min      2810.000000    0.000000
25%      5317.500000    6.000000
50%      6770.000000    9.000000
75%      8330.000000   13.000000
max     19050.000000   35.000000
```

Lire cette sortie avec l'œil du chapitre 8

`std` vaut $2\,600{,}70$ et non $2\,598{,}10$: **pandas divise par $n-1$**.

Pour l'écart-type **descriptif** du cours, il faut écrire `sal.salaire.std(ddof=0)`.

L'argument `ddof` — *delta degrees of freedom* — vaut 1 par défaut dans pandas et 0 dans numpy. **Le préciser explicitement est la seule façon de ne jamais se tromper.**

Le contrôle des types

```
sal.isna().sum()                    # valeurs manquantes par colonne
sal.salaire.nunique()               # 381 valeurs distinctes
sal.diplome.value_counts()
```

Le tableau statistique

Variable qualitative

```
n  = len(sal)
ni = sal.diplome.value_counts().sort_index()
fi = ni / n

tableau = pd.DataFrame({"ni": ni, "fi": fi,
                        "Ni": ni.cumsum(), "Fi": fi.cumsum()})
print(tableau.round(3))
```

Le résultat

	ni	fi	Ni	Fi
Bac	219	0.438	219	0.438
Bac+2	174	0.348	393	0.786
Bac+5	107	0.214	500	1.000

Variable qualitative (suite)

Trier l'index n'est pas décoratif

`value_counts()` trie par effectif **décroissant** par défaut. Sur une variable **ordinale**, cela détruit l'ordre — et les cumuls deviennent absurdes.

`sort_index()` rétablit l'ordre alphabétique, qui coïncide ici avec l'ordre des diplômes. Pour d'autres modalités, il faudrait déclarer un type `Categorical` ordonné.

Variable continue

```
bornes = [2000, 4000, 6000, 8000, 10000, 14000, 20000]
cl = pd.cut(sal.salaire, bins=bornes, right=False)   # [a ; b[
ni = cl.value_counts().sort_index()
ai = np.diff(bornes)

pd.DataFrame({"ni": ni.values, "fi": ni.values/n,
              "ai": ai, "di": ni.values/ai},
             index=ni.index).round(4)
```

right=False, encore

Comme dans R, la convention par défaut est $]a\,;\,b]$. `right=False` donne $[a\,;\,b[$.

Deux logiciels, deux langages, le même piège au même endroit.

Les graphiques

Les quatre représentations

```
fig, ax = plt.subplots(2, 2, figsize=(11, 7))

sal.diplome.value_counts().sort_index().plot.bar(
    ax=ax[0,0], color="steelblue", rot=0, title="Diplôme")

sal.region.value_counts().plot.pie(
    ax=ax[0,1], autopct="%1.0f%%", ylabel="", title="Région")

ax[1,0].hist(sal.salaire, bins=bornes, density=True, color="steelblue")
ax[1,0].set(title="Salaire", xlabel="dirhams", ylabel="densité")

x = np.sort(sal.salaire)
ax[1,1].step(x, np.arange(1, n+1)/n, color="orange")
ax[1,1].set(title="Cumulative croissante", xlabel="dirhams", ylabel="F(x)")

plt.tight_layout(); plt.show()
```

Les boîtes à moustaches

```
sal.boxplot(column="salaire", by="diplome", grid=False)
plt.suptitle(""); plt.title("Salaire selon le diplôme")
plt.ylabel("dirhams"); plt.show()
```

Les indicateurs univariés

Tendance centrale, dispersion, forme

```
x = sal.salaire

resume = {
    "moyenne":      x.mean(),
    "mediane":      x.median(),
    "ecart_type":   x.std(ddof=0),
    "CV":           x.std(ddof=0)/x.mean(),
    "D1":           x.quantile(0.10),
    "Q1":           x.quantile(0.25),
    "Q3":           x.quantile(0.75),
    "D9":           x.quantile(0.90),
    "IQ":           x.quantile(0.75) - x.quantile(0.25),
    "asymetrie":    stats.skew(x),
    "aplatissement": stats.kurtosis(x, fisher=False),
}
pd.Series(resume).round(3)
```

fisher=False, l'argument à ne jamais oublier

Par défaut, `scipy.stats.kurtosis` renvoie l'**excès** $\gamma_2 - 3$, soit $1{,}837$ ici.

`fisher=False` renvoie γ_2 lui-même, $4{,}837$, la convention du chapitre 9.

Une valeur d'aplatissement lue sans savoir quelle convention a été employée est **ininterprétable** : $1{,}84$ peut vouloir dire « queues plus fines que la normale » ou « queues nettement plus épaisses ».

Concentration

```
def gini(v):
    v = np.sort(np.asarray(v, dtype=float)); m = len(v)
    return 2*np.sum(np.arange(1, m+1)*v)/(m*v.sum()) - (m+1)/m

gini(x)                          # 0.1954

# Courbe de Lorenz
v = np.sort(x); F = np.arange(1, n+1)/n; Q = np.cumsum(v)/v.sum()
plt.plot([0,1], [0,1], color="grey")
plt.plot(np.r_[0,F], np.r_[0,Q], color="steelblue", lw=2)
plt.fill_between(np.r_[0,F], np.r_[0,Q], np.r_[0,F], alpha=.15)
plt.xlabel("part cumulée des salariés")
plt.ylabel("part cumulée de la masse salariale"); plt.show()
```

Le bivarié

Contingence

```
T = pd.crosstab(sal.diplome, sal.region)
pd.crosstab(sal.diplome, sal.region, margins=True)          # avec marges
pd.crosstab(sal.diplome, sal.region, normalize="index")     # profils-lignes
pd.crosstab(sal.diplome, sal.region, normalize="columns")   # profils-colonnes

# effectifs théoriques sous indépendance
E = np.outer(T.sum(1), T.sum(0)) / n
pd.DataFrame(E, index=T.index, columns=T.columns).round(1)
```

normalize, en trois valeurs

`"index"` divise par les totaux de ligne, `"columns"` par ceux de colonne, `"all"` par le total général.

Les trois lectures du chapitre 11, en un seul argument.

Conditionnelles et décomposition

```
sal.groupby("diplome").salaire.agg(
    n="size", moyenne="mean", ecart_type=lambda v: v.std(ddof=0),
    mediane="median").round(1)

# Décomposition de la variance
g  = sal.groupby("diplome").salaire
nj = g.size(); mj = g.mean(); vj = g.apply(lambda v: v.var(ddof=0))
mg = sal.salaire.mean()

intra = (nj*vj).sum()/n                    # 5010426
inter = (nj*(mj-mg)**2).sum()/n            # 1739696
eta2  = inter/(intra+inter)                # 0.2577
```

Le ddof=0 dans le lambda

Sans lui, `vj` serait calculé en $n_j - 1$ et la somme `intra + inter` ne redonnerait pas la variance totale.

Le contrôle est toujours le même : `intra + inter` doit valoir `sal.salaire.var(ddof=0)`, soit $6\,750\,122$.

```
z = sal.anciennete

cov = np.cov(z, x, ddof=0)[0, 1]        # 9859.91
r   = np.corrcoef(z, x)[0, 1]           # 0.6363
r2  = r**2                              # 0.4049

b = cov / z.var(ddof=0)                 # 277.17
a = x.mean() - b*z.mean()               # 4347.2
```

La version statsmodels, pour la sortie complète

```
import statsmodels.formula.api as smf
mod = smf.ols("salaire ~ anciennete", data=sal).fit()
print(mod.summary())
```

```
                coef
Intercept    4347.2
anciennete    277.17
```

La syntaxe formule, à nouveau

`"salaire ~ anciennete"` est **exactement** la syntaxe de R. `statsmodels` l'a reprise volontairement, pour que le passage d'un langage à l'autre soit immédiat.

Le graphique final

```
plt.figure(figsize=(8,5))
plt.scatter(z, x, s=8, alpha=.35, color="steelblue")
xx = np.linspace(0, 35, 2)
plt.plot(xx, a + b*xx, color="orange", lw=2)
plt.scatter(z.mean(), x.mean(), s=60, color="darkviolet", zorder=3)
plt.xlabel("ancienneté (années)"); plt.ylabel("salaire (dirhams)")
plt.show()
```

Comparaison des deux langages

Le tableau de correspondance

R et Python, côte à côte

Opération	R	Python
Importer	`read.csv()`	`pd.read_csv()`
Structure	`str()`	`.info()`
Résumé	`summary()`	`.describe()`
Effectifs	`table()`	`.value_counts()`
Classes	`cut(right=FALSE)`	`pd.cut(right=False)`
Histogramme	`hist(freq=FALSE)`	`.hist(density=True)`
Écart-type descriptif	`sd()*sqrt((n-1)/n)`	`.std(ddof=0)`
Asymétrie	`moments::skewness()`	`stats.skew()`
Aplatissement	`moments::kurtosis()`	`stats.kurtosis(fisher=False)`

R et Python, côte à côte

Opération	R	Python
Croisement	`table(a, b)`	`pd.crosstab(a, b)`
Par groupe	`aggregate(y ~ x, ...)`	`.groupby("x").y.agg()`
Corrélation	`cor()`	`np.corrcoef()`
Régression	`lm(y ~ x)`	`smf.ols("y ~ x")`

Le tableau de correspondance (suite)

Lequel choisir

Les deux donnent **les mêmes nombres**, ce qui est le seul point réellement important.

R est plus direct pour la statistique : `lm`, `aggregate`, `boxplot` prennent la syntaxe formule sans rien installer.

Python est plus polyvalent : le même langage sert à récupérer les données, les nettoyer, les analyser et publier le résultat.

En Licence, apprenez celui que vos enseignants emploient. En pratique professionnelle, vous rencontrerez les deux.

Bilan des seize chapitres

Ce que le cours a établi

Le fil complet

Étape	Chapitres	Résultat sur nos données
Définir	1, 2	500 salariés, six variables typées
Résumer	3, 4	six classes de salaire
Représenter	5	quatre graphiques
Situer	6, 7	moyenne $7\,158$, médiane $6\,770$
Disperser	8	$\sigma = 2\,598$, $CV = 0{,}36$
Décrire la forme	9	$\gamma_1 = 1{,}16$, $\gamma_2 = 4{,}84$
Mesurer le partage	10	Gini $= 0{,}195$
Croiser	11, 12	$\eta^2 = 0{,}258$ pour le diplôme
Lier	13, 14	$r = 0{,}636$, $r^2 = 0{,}405$
Automatiser	15, 16	quinze lignes de code

Ce que le cours a établi (suite)

La phrase qui résume le semestre

Un tableau de mille lignes ne dit rien tant qu'on ne l'a pas résumé — et un résumé ne vaut rien tant qu'on ne sait pas ce qu'il a laissé de côté.

Vous savez maintenant faire les deux.

Ce qu’il faut retenir

Le résumé du chapitre

Six points

1. `pandas` divise par $n-1$, `numpy` par n : **toujours préciser `ddof`**.
2. `pd.cut(..., right=False)` pour des classes $[a\,;\,b[$.
3. `density=True` dans `hist` : la densité, jamais l'effectif.
4. `stats.kurtosis(..., fisher=False)` pour obtenir γ_2 et non l'excès.
5. `pd.crosstab(..., normalize=)` : `"index"`, `"columns"` ou `"all"`.
6. `statsmodels` reprend la syntaxe formule de R : `"y ~ x"`.

La suite du parcours

Ce cours s'est arrêté au constat. Il reste trois questions, et trois cours pour y répondre :

- **Probabilités**, pour modéliser le hasard ;
- **Statistique inférentielle**, pour généraliser au-delà de l'échantillon ;
- **Introduction à l'économétrie**, pour passer de la corrélation à l'explication.

Ces trois cours travaillent tous sur les objets que celui-ci a construits.