Types de diagrammes Matplotlib

Maîtriser Matplotlib : Un guide complet pour la visualisation de données · https://moncoachdata.com/tutos/types-de-diagrammes-matplotlib/

Après le tracé linéaire de base, Matplotlib propose beaucoup d'autres **types de diagrammes** : diagrammes à barres, histogrammes, diagrammes de dispersion, diagrammes circulaires, diagrammes en boîte, cartes thermiques, affichage d'images, graphiques en pile... Voyons quand les utiliser, avec des cas d'usage et du code. Tous les exemples supposent ces imports :

```python
import numpy as np
import matplotlib.pyplot as plt
```

Python

Diagramme à barres

Les diagrammes à barres représentent des données catégorielles avec des barres rectangulaires, dont la hauteur (ou la longueur) correspond à une valeur. Utilise `plt.bar(x, y)` pour des barres verticales et `plt.barh(x, y)` pour des barres horizontales.

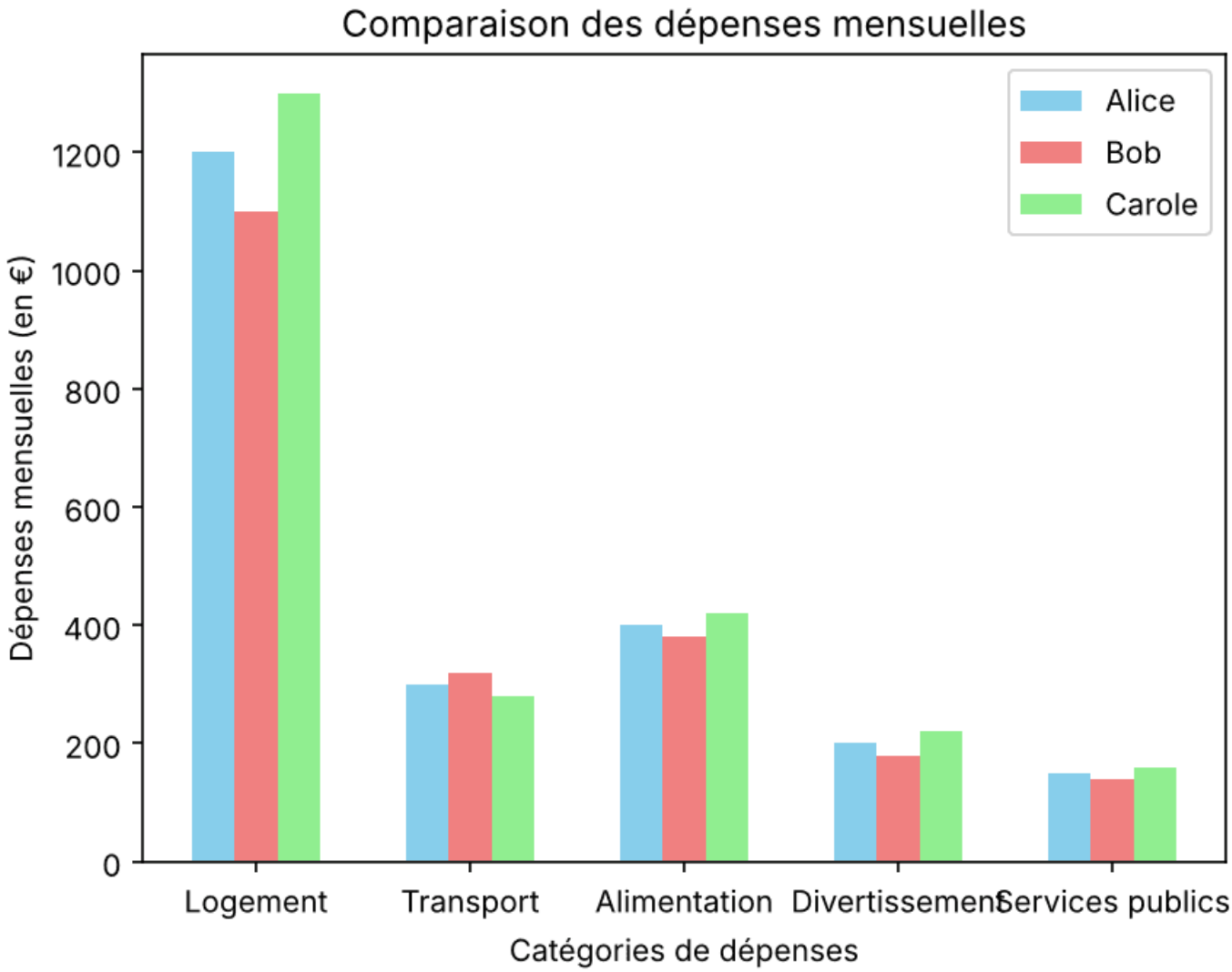
Cas d'usage : comparer les ventes de différents produits, représenter la population par pays.

Voici un diagramme à barres multiples dans un même graphique :



```python
# Catégories de dépenses
categories = ['Logement', 'Transport', 'Alimentation', 'Divertissement', 'Services publics']

# Dépenses mensuelles pour Alice, Bob et Carole
alice_expenses = [1200, 300, 400, 200, 150]
bob_expenses = [1100, 320, 380, 180, 140]
carol_expenses = [1300, 280, 420, 220, 160]

# Positions des catégories sur l'axe x
x = np.arange(len(categories))

# Largeur des barres : nécessaire pour placer plusieurs barres côte à côte
bar_width = 0.2

# Barres d'Alice, décalées vers la gauche
plt.bar(x - bar_width, alice_expenses, width=bar_width, label='Alice', color='skyblue')

# Barres de Bob, centrées sur la position
plt.bar(x, bob_expenses, width=bar_width, label='Bob', color='lightcoral')

# Barres de Carole, décalées vers la droite
plt.bar(x + bar_width, carol_expenses, width=bar_width, label='Carole', color='lightgreen')

# Étiquettes, titre et légende
plt.xlabel('Catégories de dépenses')
plt.ylabel('Dépenses mensuelles (en €)')
plt.title('Comparaison des dépenses mensuelles')

# Afficher les noms des catégories aux positions de l'axe x
plt.xticks(x, categories)
plt.legend()
plt.show()
```

Python



Pour placer les barres côte à côte, on soustrait la largeur de barre aux positions `x` pour la première série (elle passe à gauche), et on l'ajoute pour la dernière (elle passe à droite). Le paramètre `width` est identique pour toutes les barres.

Histogramme

Les histogrammes visualisent la distribution de données continues ou numériques, et aident à repérer des motifs. Les données sont regroupées en intervalles (*bins*), et la hauteur de chaque barre représente le nombre de points dans cet intervalle.

Utilise `plt.hist(x)`. Contrairement au diagramme à barres, tu n'as pas besoin de `y` : il ne représente que la fréquence d'une donnée continue. Le nombre de bins par défaut est de 10, mais tu peux le modifier, ou fournir tes propres intervalles. Tu peux aussi définir la couleur du bord des barres. Pour ajouter une ligne (la moyenne ou la médiane, par exemple), calcule la valeur et passe-la à `plt.axvline()`. Cette méthode marche avec n'importe quel autre graphique.

Cas d'usage : analyser la répartition des âges dans une population, la distribution des notes d'une classe.

```python
# Notes d'examen
exam_scores = [68, 72, 75, 80, 82, 84, 86, 90, 92, 95, 98, 100]

# Intervalles personnalisés
bin_ranges = [60, 70, 80, 90, 100]

# Histogramme avec des intervalles personnalisés
plt.hist(exam_scores, bins=bin_ranges, color='lightblue', edgecolor='black', alpha=0.7)

# Étiquettes et titre
plt.xlabel("Notes d'examen")
plt.ylabel('Fréquence')
plt.title("Histogramme des notes d'examen avec intervalles personnalisés")

# Ligne verticale pour la médiane
median_score = np.median(exam_scores)
plt.axvline(median_score, color='red', linestyle='dashed', linewidth=2,
            label=f'Médiane : {median_score}')

plt.legend()
plt.show()
```

Python

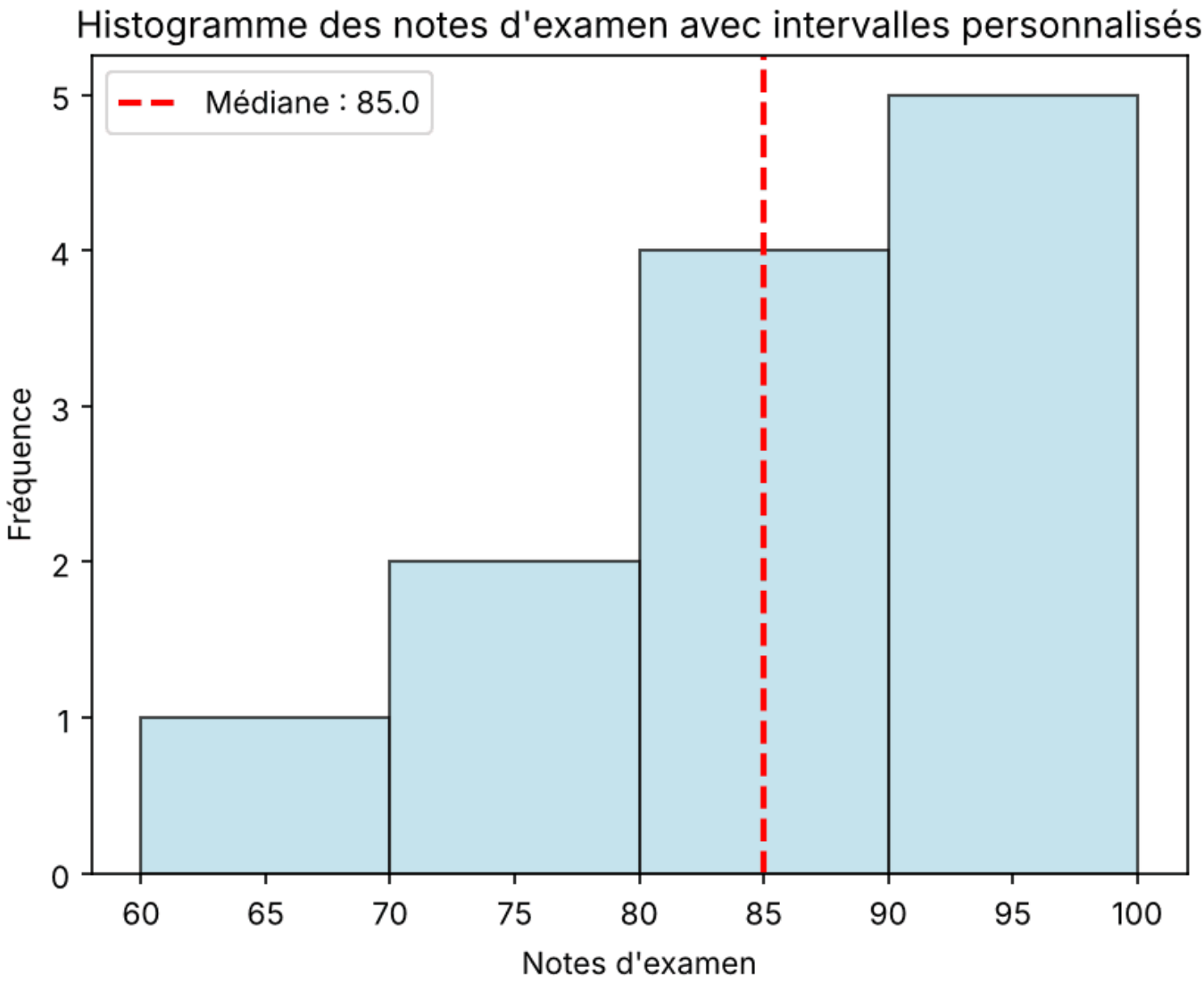


Diagramme de dispersion

Les diagrammes de dispersion (*scatter plots*) représentent des points individuels dans un plan à deux dimensions. Ils servent à explorer les relations ou les corrélations entre deux variables numériques : chaque axe représente une variable.

Utilise `plt.scatter(x, y)`. Le paramètre `s` règle la taille des points (tu peux passer une liste de tailles, une par point), `c` la couleur, `marker` le symbole, et `alpha` la transparence.

Cas d'usage : étudier le lien entre les heures de révision et les résultats d'examen, la corrélation entre la température et les ventes de glaces.

```python
# Données de magasins
stores = ['Magasin A', 'Magasin B', 'Magasin C', 'Magasin D', 'Magasin E']
customers = [120, 90, 150, 80, 200]
revenue = [20000, 18000, 25000, 17000, 30000]
store_size = [10, 5, 15, 8, 20]     # taille du magasin, en 100 m²

# Mise à l'échelle des tailles de magasin pour les tailles de points
point_sizes = [size * 100 for size in store_size]

# Diagramme de dispersion avec une taille de point différente par magasin
# alpha contrôle la transparence
plt.scatter(customers, revenue, s=point_sizes, c='skyblue', alpha=0.7, edgecolors='b')

plt.xlabel('Nombre de clients')
plt.ylabel("Chiffre d'affaires (€)")
plt.title("Relation entre les clients, le chiffre d'affaires et la taille du magasin")
plt.show()
```

Python

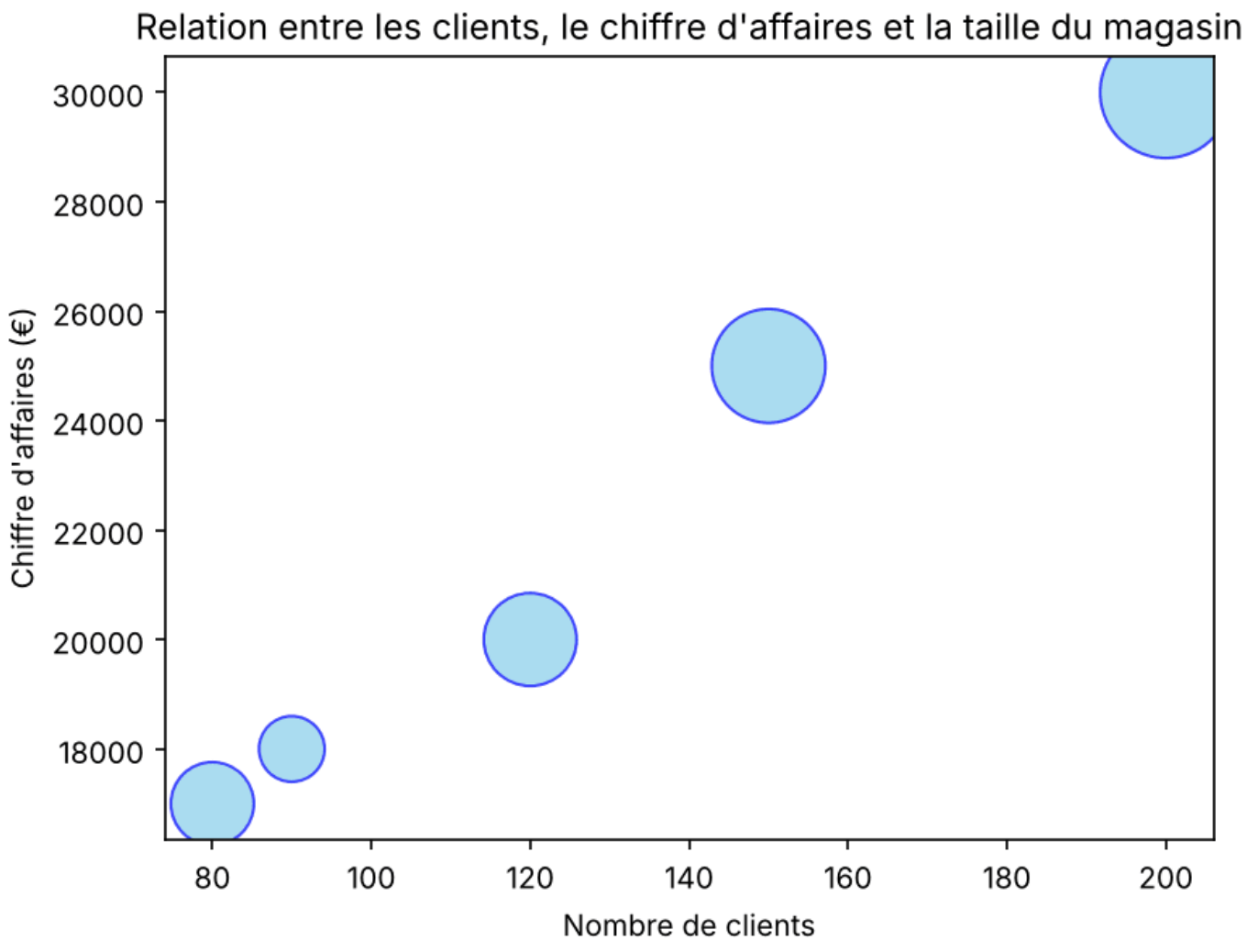


Diagramme circulaire

Les diagrammes circulaires représentent les parties d'un tout sous forme de parts d'un gâteau. Ils conviennent pour montrer la composition d'une seule variable catégorielle en pourcentages. Mais au-delà de six catégories, ils deviennent illisibles : mieux vaut alors utiliser des barres horizontales.

Utilise `plt.pie(x, labels=..., colors=...)`. Tu peux changer la couleur du bord des parts avec `wedgeprops={'edgecolor': ma_couleur}`, et mettre en évidence une part avec `explode` : un tuple dont chaque valeur indique de combien la part correspondante est décalée. Le paramètre `autopct` affiche les pourcentages avec le format de ton choix.

Cas d'usage : répartition d'un budget par catégorie de dépenses, parts de marché de marques de smartphones.

```python
# Ventes par catégorie de produits
categories = ['Électronique', 'Vêtements', 'Décoration intérieure', 'Livres', 'Jouets']
sales = [3500, 2800, 2000, 1500, 1200]

# Détacher la part 'Vêtements' (0.1 = décalage de la deuxième part)
explode = (0, 0.1, 0, 0, 0)

# Diagramme circulaire avec explode, ombre et pourcentages à 1 décimale
plt.pie(sales, labels=categories, explode=explode, shadow=True, autopct='%1.1f%%')
plt.title('Ventes par catégorie de produits')
plt.show()
```

Python

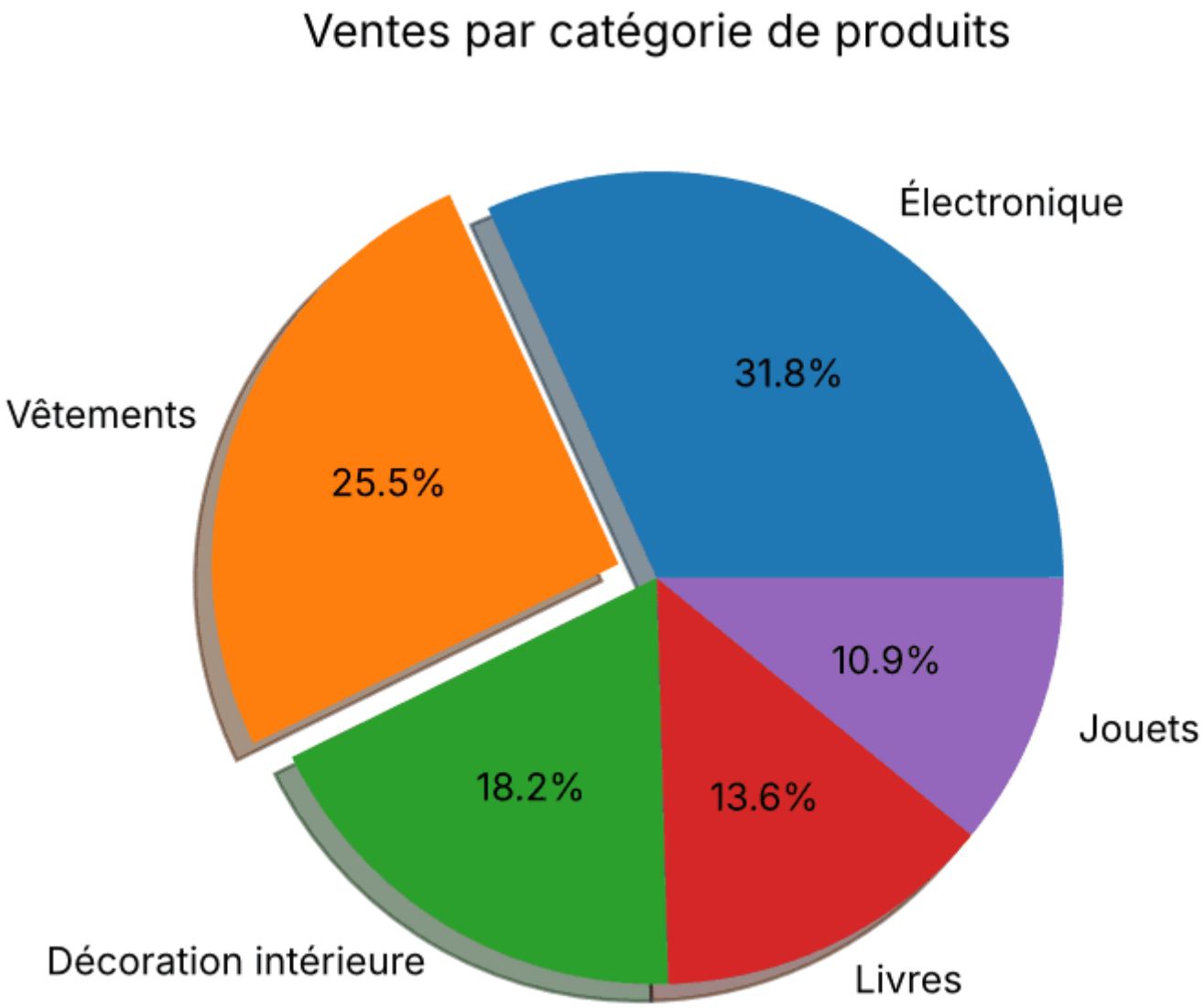


Diagramme en boîte (boîte à moustaches)

Les diagrammes en boîte ont l'air compliqués, mais le principe est simple : ils résument la distribution de données numériques en montrant les quartiles, les valeurs aberrantes et l'éventuelle asymétrie. Ils donnent un aperçu de la dispersion, de la tendance centrale et de la variabilité, et sont particulièrement utiles pour repérer les valeurs aberrantes et comparer des distributions.

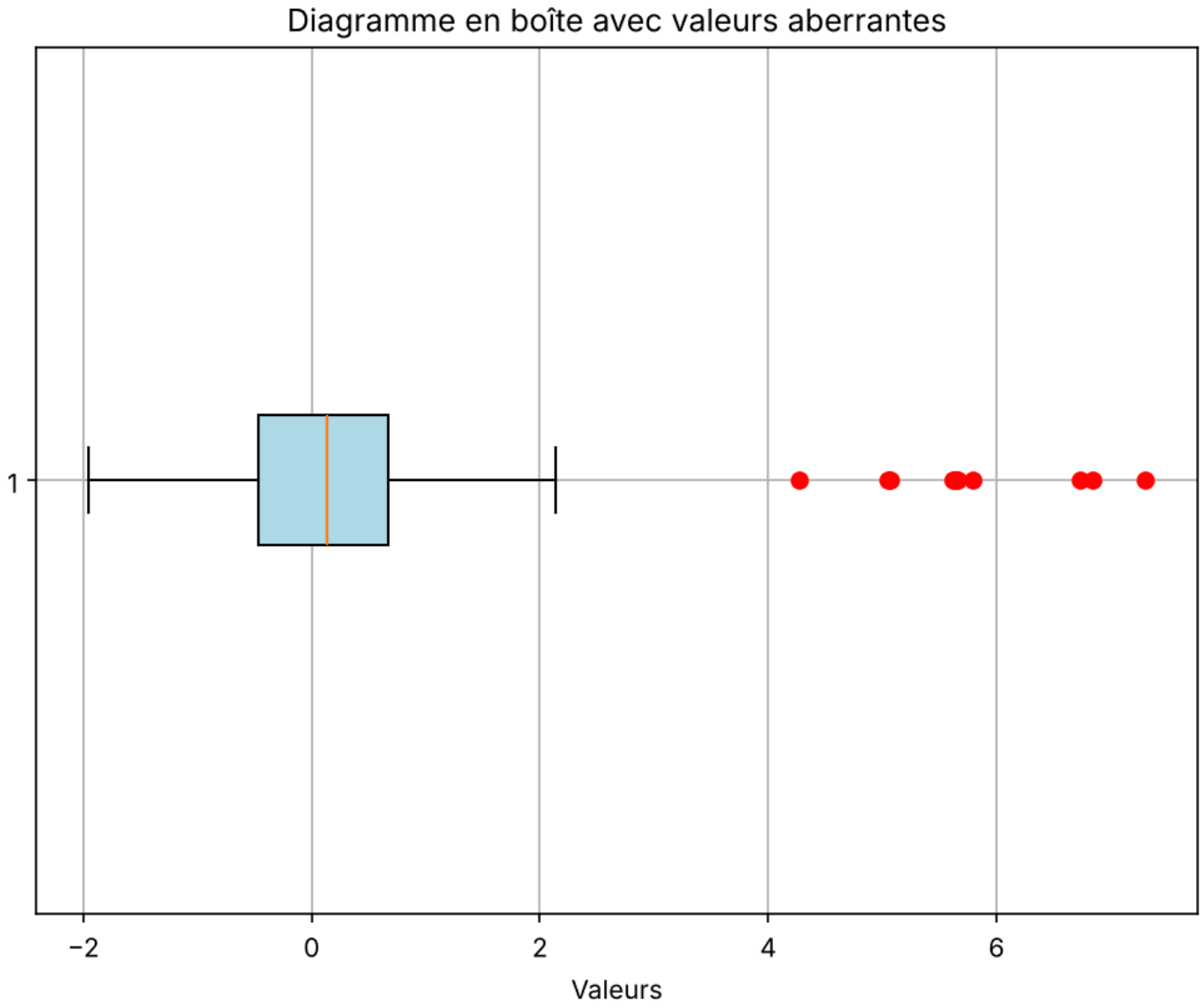
Utilise `plt.boxplot(data)`. Avec `patch_artist=True`, la boîte peut être colorée ; `boxprops` et `flierprops` personnalisent la boîte et les valeurs aberrantes. Pour un diagramme horizontal, utilise `orientation='horizontal'` (Matplotlib 3.10 et plus ; avec les versions plus anciennes, utilise

`vert=False`).

Cas d'usage : analyser la distribution des salaires dans une entreprise, comparer la variabilité des prix de logements par quartier.

```python
# Données aléatoires avec quelques valeurs aberrantes
rng = np.random.default_rng(42)
data = np.concatenate([rng.normal(0, 1, 100), rng.normal(6, 1, 10)])

plt.figure(figsize=(8, 6))
plt.boxplot(data, orientation='horizontal', patch_artist=True,
            boxprops={'facecolor': 'lightblue'},
            flierprops={'marker': 'o', 'markerfacecolor': 'red',
                        'markeredgecolor': 'red'})

plt.xlabel('Valeurs')
plt.title('Diagramme en boîte avec valeurs aberrantes')
plt.grid(True)
plt.show()
```

Python



Carte thermique et affichage d'images

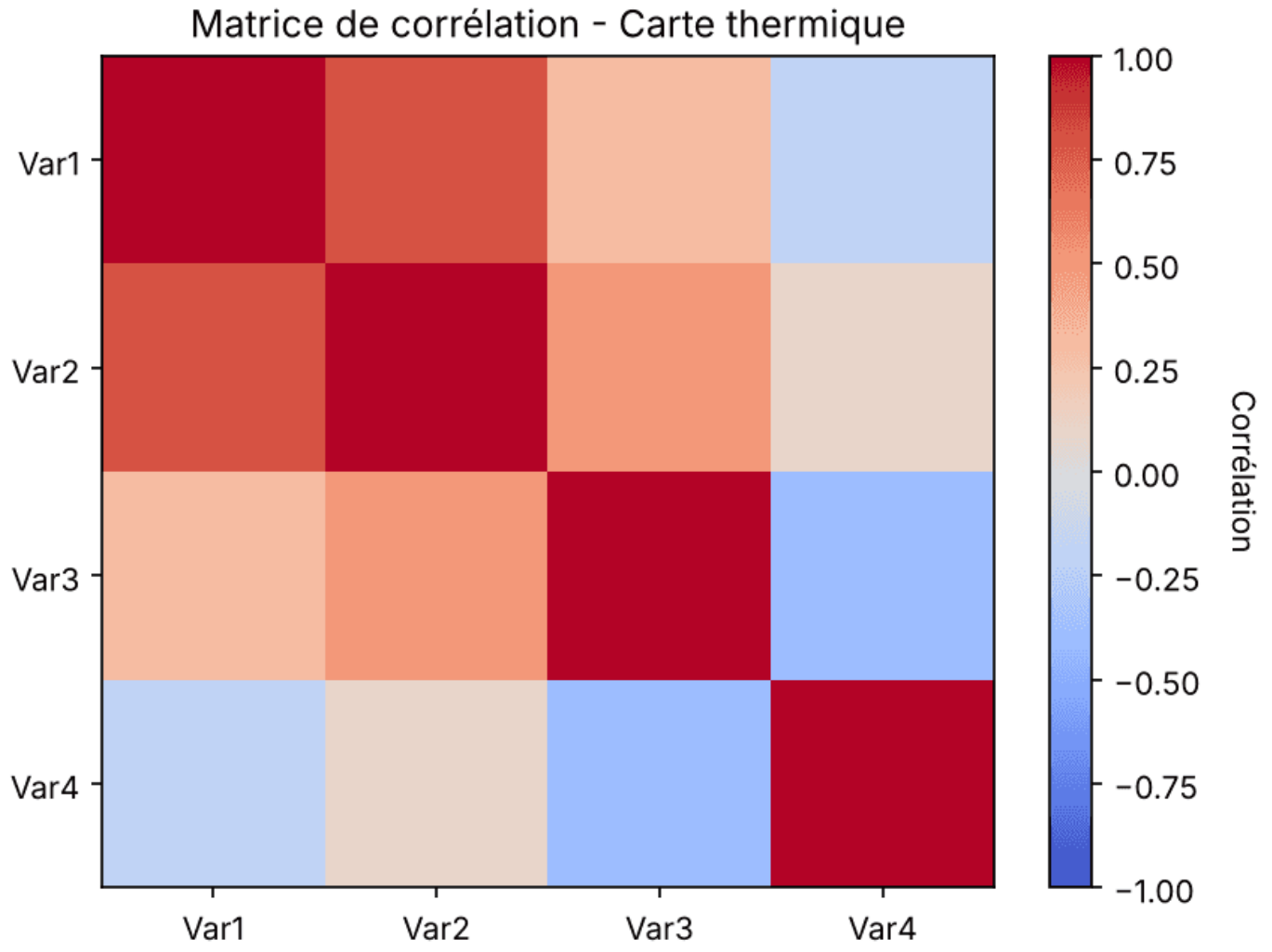
La fonction `plt.imshow()` affiche des données d'image 2D, visualise des tableaux 2D ou montre des images dans différents formats.

imshow pour une carte thermique (heatmap)

Une carte thermique peut visualiser une matrice de corrélation, qui indique le lien entre chaque variable et toutes les autres. Passe la matrice à `imshow`, avec une palette de couleurs (`cmap`) pour rendre la relation visible.

```python
# Exemple de matrice de corrélation
correlation_matrix = np.array([[1.0, 0.8, 0.3, -0.2],
                               [0.8, 1.0, 0.5, 0.1],
                               [0.3, 0.5, 1.0, -0.4],
                               [-0.2, 0.1, -0.4, 1.0]])

# Carte thermique
plt.imshow(correlation_matrix, cmap='coolwarm', vmin=-1, vmax=1, aspect='auto', origin='upper')

# Barre de couleur
cbar = plt.colorbar()
cbar.set_label('Corrélation', rotation=270, labelpad=20)

# Titre et noms des variables sur les axes
plt.title('Matrice de corrélation - Carte thermique')
plt.xticks(range(len(correlation_matrix)), ['Var1', 'Var2', 'Var3', 'Var4'])
plt.yticks(range(len(correlation_matrix)), ['Var1', 'Var2', 'Var3', 'Var4'])
plt.show()
```

Python



imshow pour afficher une image

La fonction `imread` du module `matplotlib.image` lit et charge des images dans différents formats (JPEG, PNG, BMP, TIFF...). Elles s'affichent ensuite avec `imshow`. Si tu as déjà les données de l'image dans un tableau, tu peux l'afficher directement.



```python
import matplotlib.image as mpimg

# Charger l'image
img = mpimg.imread('sample_image.jpg')

# Afficher l'image
plt.imshow(img)

# Masquer les axes
plt.axis('off')
plt.title("Exemple d'image")
plt.show()
```

Python

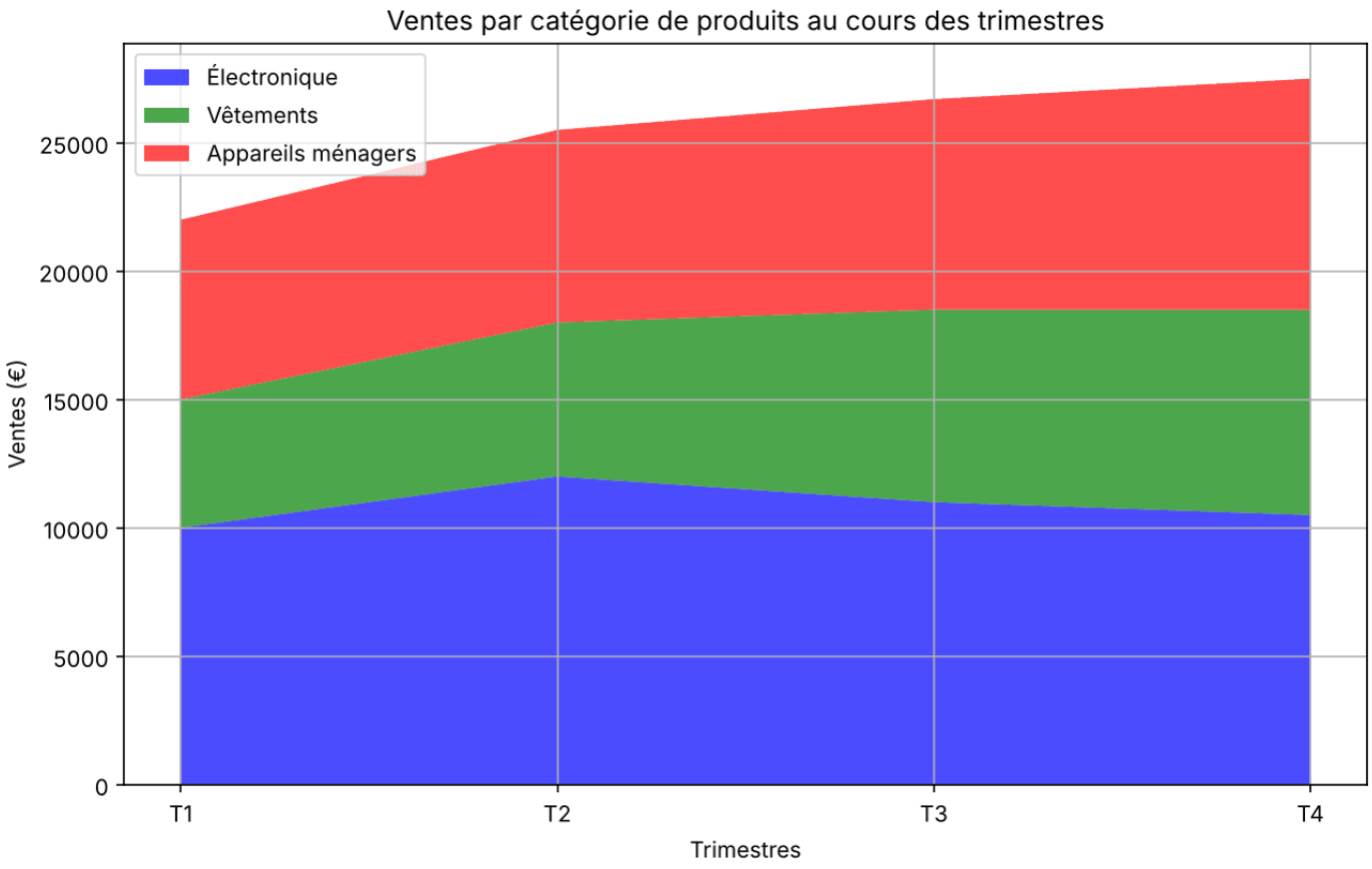
Graphique en pile

Imagine que tu veuilles visualiser la contribution de trois catégories de produits (électronique, vêtements, électroménager) aux ventes totales sur quatre trimestres. Tu peux représenter les ventes de chaque catégorie comme une couche du graphique : tu vois alors leurs contributions et leurs tendances au fil du temps. C'est exactement ce que fait le graphique en pile.

Un graphique en pile, ou graphique en aires empilées, affiche plusieurs jeux de données sous forme de couches empilées, chacune représentant une catégorie ou un composant. Il est particulièrement utile pour voir comment des composants contribuent à un ensemble au cours du temps, ou sur un domaine catégoriel. Utilise `plt.stackplot(x, y1, y2, ...)`, avec autant de couches que tu veux !

```python
# Données pour le graphique en pile
quarters = ['T1', 'T2', 'T3', 'T4']
electronics = [10000, 12000, 11000, 10500]
clothing = [5000, 6000, 7500, 8000]
home_appliances = [7000, 7500, 8200, 9000]

plt.figure(figsize=(10, 6))
plt.stackplot(quarters, electronics, clothing, home_appliances,
              labels=['Électronique', 'Vêtements', 'Appareils ménagers'],
              colors=['blue', 'green', 'red'], alpha=0.7)

plt.xlabel('Trimestres')
plt.ylabel('Ventes (€)')
plt.title('Ventes par catégorie de produits au cours des trimestres')
plt.legend(loc='upper left')
plt.grid(True)
plt.show()
```

Python



La suite de la série porte sur les multiples subplots avec Matplotlib. Documentation : galerie des types de graphiques Matplotlib.

Pour aller plus loin, consulte aussi notre guide des 9 bibliothèques Python pour la data science.

