

Tableaux de Bord Interactifs avec Python

Séance 12

Plan du Cours & Objectifs

Introduction

1

Outils Python

2

Principes de Visualisation

3

Installation & 1er Dashboard

4

Données & Interactivité

5

Design & Cas Pratiques

6

Objectifs Pédagogiques

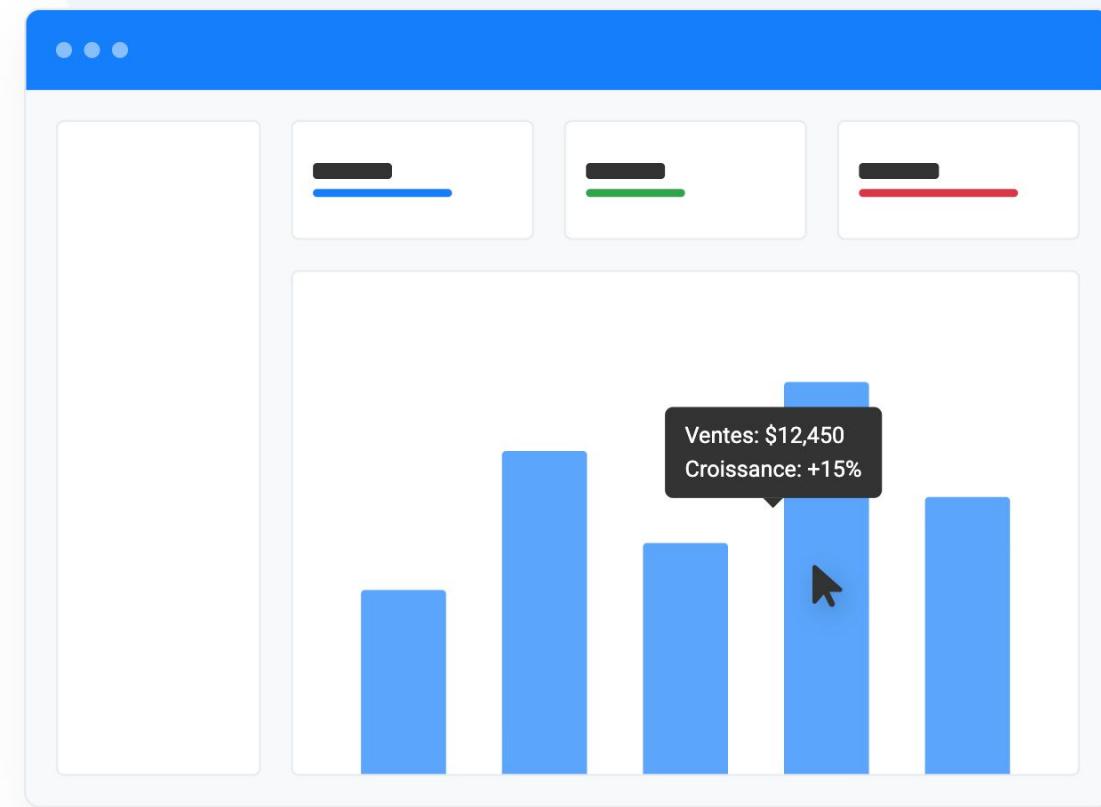
- ✓ Comprendre les principes de conception de tableaux de bord efficaces.
- ✓ Manipuler et préparer les données avec Pandas pour la visualisation.
- ✓ Maîtriser le framework Dash (Layouts & Callbacks) pour l'interactivité.
- ✓ Déployer et optimiser des applications de données professionnelles.

Qu'est-ce qu'un tableau de bord interactif ?

Un tableau de bord interactif est une interface visuelle dynamique permettant aux utilisateurs d'explorer et d'analyser des données complexes en temps réel. Contrairement aux rapports statiques (PDF, Excel figé), il offre une expérience utilisateur fluide pour naviguer dans l'information.

Grâce à des fonctionnalités comme le filtrage, le zoom, le survol (tooltips) et la sélection croisée, l'utilisateur passe du statut de spectateur passif à celui d'analyste actif, capable de découvrir des insights par lui-même.

◎ **Avantages Clés :** Prise de décision accélérée, suivi des KPI en temps réel, collaboration facilitée et narration visuelle (Data Storytelling).



Pourquoi les Dashboards ?

De la donnée brute à la décision éclairée

Dans un environnement professionnel saturé d'informations, la capacité à transformer rapidement des données complexes en insights exploitables est un avantage concurrentiel majeur. Les tableaux de bord interactifs répondent directement à ce besoin critique.

⚠ Problématiques Actuelles

Sans outils de visualisation adaptés, les entreprises font face à des défis majeurs :

- ✖ **Infobésité & Complexité** : Volume de données trop important pour être analysé manuellement via Excel.
- ✖ **Données Silotées** : Informations dispersées dans différents systèmes (ERP, CRM, Analytics) difficiles à croiser.
- ✖ **Latence Décisionnelle** : Rapports statiques mensuels obsolètes dès leur publication.
- ✖ **Manque d'Autonomie** : Dépendance forte aux équipes IT pour la moindre extraction de données.

✓ La Réponse Dashboarding

Les tableaux de bord interactifs apportent des solutions concrètes et immédiates :

- ✓ **Centralisation des KPI** : Vue unifiée des indicateurs clés (Revenus, CAC, Marge) en temps réel.
- ✓ **Diagnostic Rapide** : Identification instantanée des anomalies et tendances grâce à la visualisation.
- ✓ **Self-Service BI** : Autonomie des métiers pour explorer les données (filtres, drill-down).

CAS D'USAGE FRÉQUENTS

Production

Commercial

Web Analytics

Panorama des Outils Python

Dash

Framework web complet basé sur Flask, React et Plotly. Idéal pour des applications "Enterprise-grade", complexes et hautement personnalisables. Il offre un contrôle total sur l'interface (Layout) et la logique réactive (Callbacks).

Streamlit

Solution "low-code" conçue pour le prototypage rapide. Transforme des scripts de données en applications web en quelques lignes de code. Parfait pour les POCs (Preuve de Concept) et les outils internes simples, mais moins flexible que Dash.

Plotly & Pandas

Usage : Visualisation & Manipulation

Le moteur graphique et analytique. **Pandas** gère le chargement et la transformation des données (ETL), tandis que **Plotly** génère les graphiques interactifs (zoom, survol) qui seront intégrés dans Dash ou Streamlit.

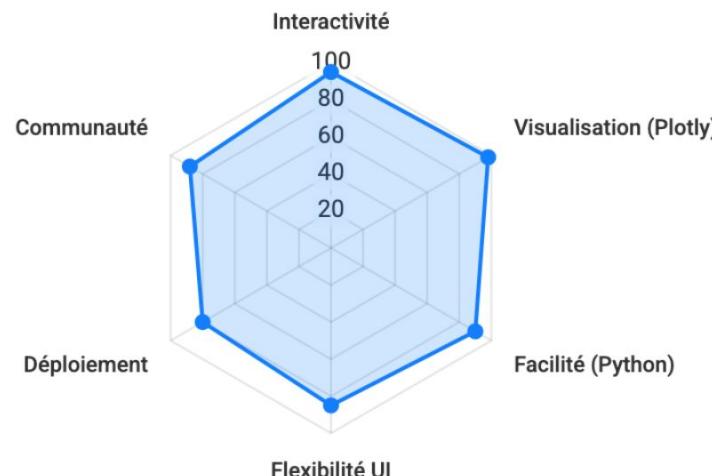
L'écosystème Python offre une variété de bibliothèques puissantes pour la visualisation de données. Le choix de l'outil dépend de la complexité du projet, du besoin de personnalisation et du temps de développement. Voici les trois piliers principaux que nous utiliserons.

Focus sur Dash

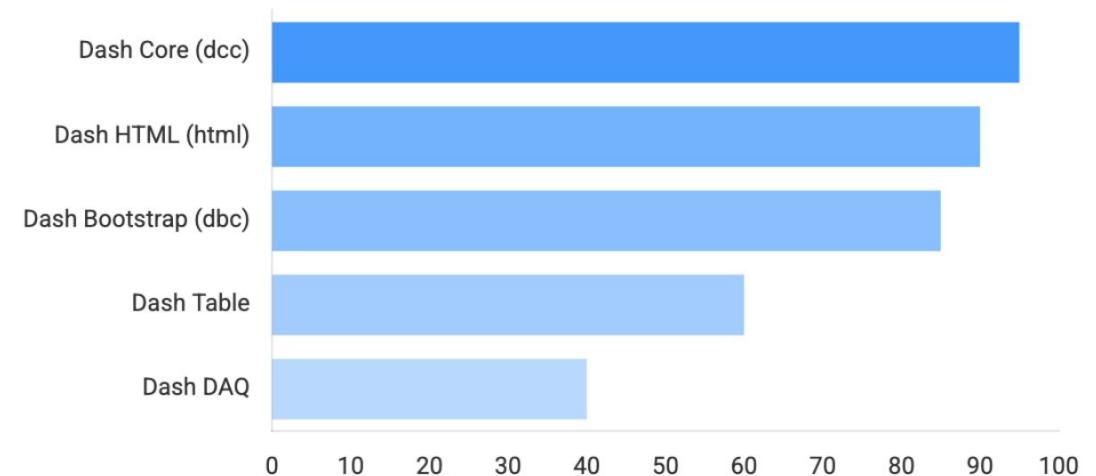
Dash est le framework Python de référence pour la création d'applications web analytiques professionnelles. Basé sur [Flask](#), [Plotly.js](#) et [React.js](#), il permet de construire des interfaces complexes sans écrire de JavaScript. Son architecture repose sur une séparation claire entre le layout (structure visuelle) et les callbacks (logique interactive).

</> Anatomie : `app = Dash()` pour l'initialisation, `app.layout` pour l'interface, et `@app.callback` pour l'interactivité.

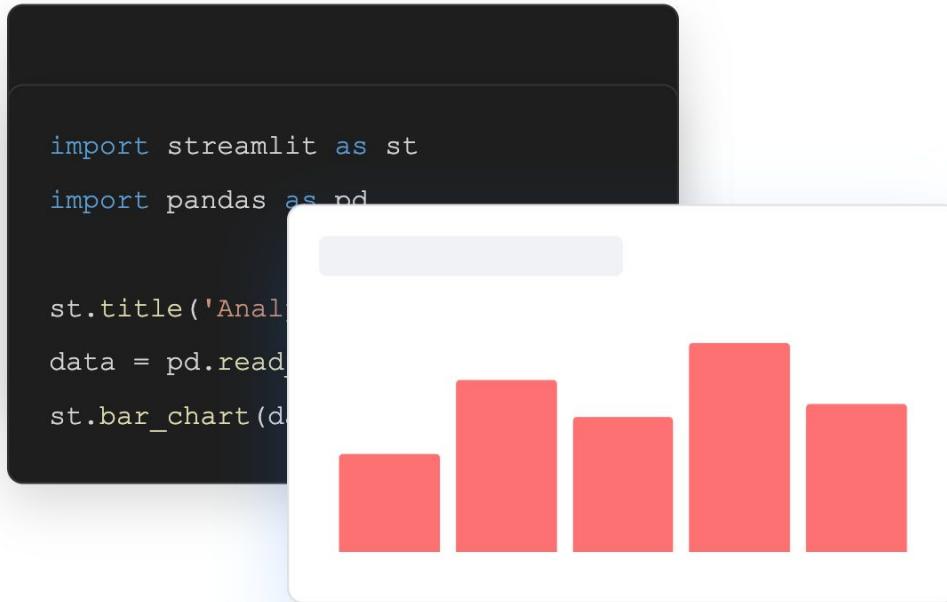
Forces du Framework Dash



Bibliothèques de Composants Clés



Tableaux de Bord Interactifs



Streamlit : L'Alternative Rapide

Pourquoi Streamlit ?

Streamlit permet de transformer des scripts de données en applications web partageables en quelques minutes. Avec une syntaxe 100% Python simple (ex: `st.write`), il élimine le besoin de compétences front-end (HTML/CSS/JS) et offre un chargement instantané ("hot-reload").

Cas d'Usage & Limites

C'est l'outil idéal pour les POCs (Preuves de Concept), l'exploration rapide et les démos internes. Bien que très rapide à développer, il offre moins de contrôle sur la mise en page fine et les interactions complexes comparé à Dash.

Plotly et Pandas

Le Duo Gagnant

L'intégration étroite entre **Pandas** et **Plotly Express** constitue le cœur du workflow de visualisation en Python. Pandas prépare le terrain, Plotly le sublime.

Pandas : Manipulation

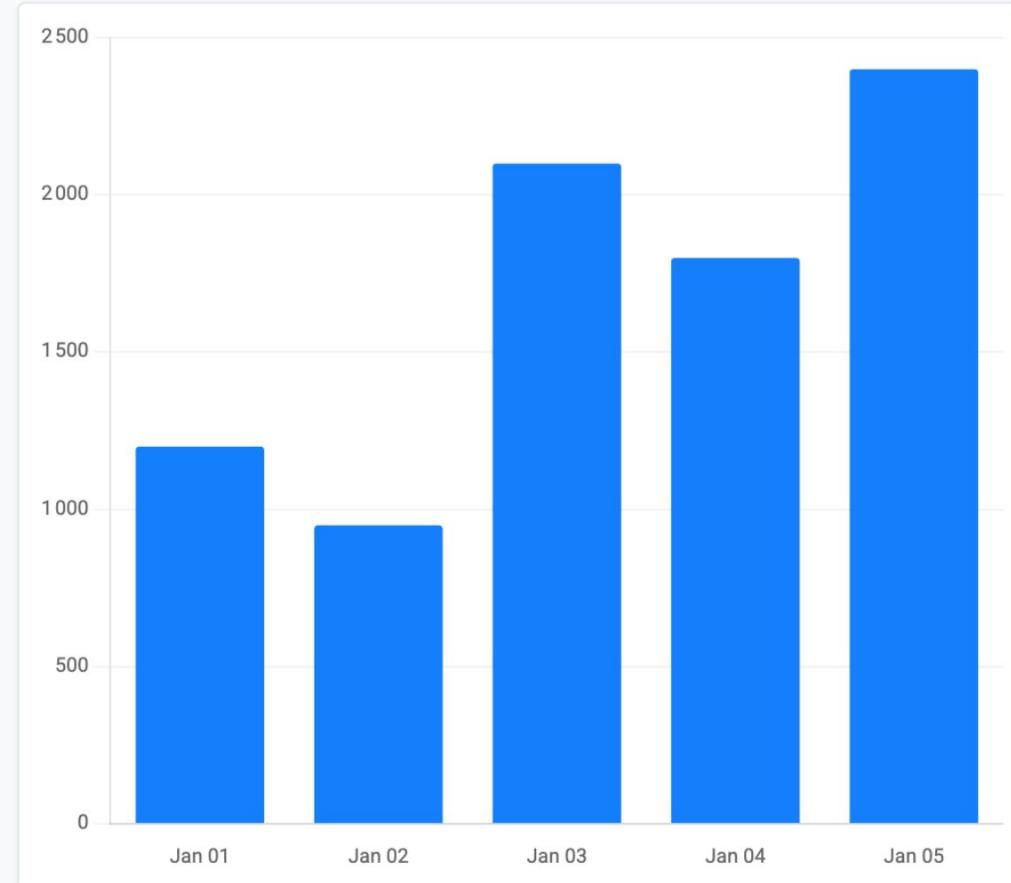
Chargement (CSV, SQL), nettoyage (dropna), agrégation (groupby) et transformation des données temporelles.

Plotly Express : Visualisation

Une syntaxe concise (px.bar, px.scatter) qui accepte directement les DataFrames pour générer des graphiques interactifs instantanés.

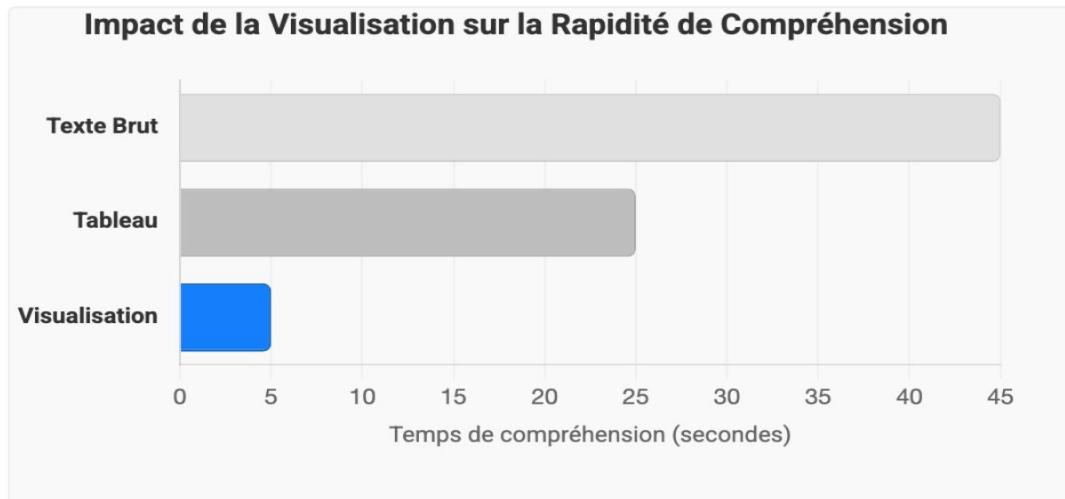
Date	Region	Sales	Profit
2023-01-01	North	1200	300
2023-01-02	South	950	150
2023-01-03	East	2100	500
...

fig = px.bar(df, x='Date', y='Sales')



Principes de Visualisation

Une visualisation efficace ne se limite pas à l'esthétique ; elle doit avant tout communiquer l'information de manière claire et précise. Le principe fondamental "**Clarté > Décor**" implique d'éliminer le bruit graphique (chart junk) pour maximiser le ratio données/encre. Un bon graphique raconte une histoire sans ambiguïté et réduit la charge cognitive de l'utilisateur.



Hiérarchie & Couleur

L'information la plus importante doit être la plus visible. Utilisez la couleur avec intention (catégorisation ou mise en valeur d'alertes), jamais pour la simple décoration. Assurez un contraste suffisant pour l'accessibilité et pensez aux utilisateurs daltoniens en variant aussi les formes ou les motifs.

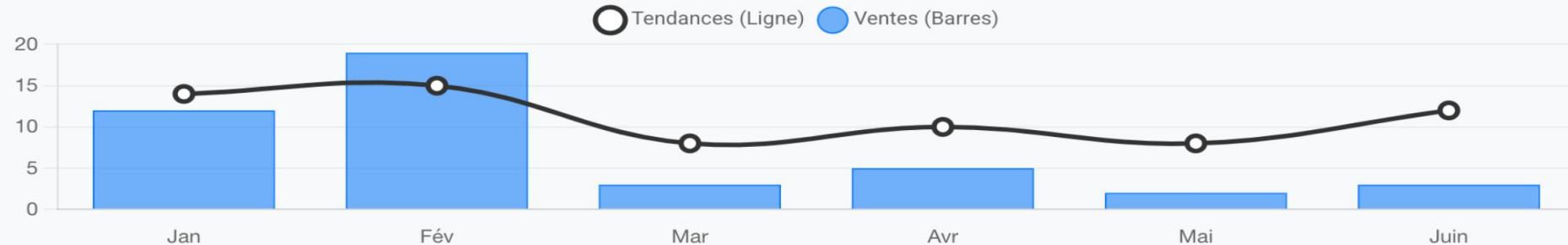
Intégrité Graphique

Ne trompez pas le lecteur. Les axes doivent impérativement commencer à zéro pour les diagrammes en barres afin de ne pas exagérer les différences. Les proportions visuelles doivent correspondre aux valeurs numériques réelles. Évitez les graphiques en 3D (comme les camemberts explosés) qui distordent la perception des volumes.

Simplicité (KISS)

Types de graphiques : Quand et pourquoi les utiliser

Exemple : Comparaison et Tendance Combinées



Le choix de la représentation visuelle est déterminant pour l'interprétation des données. Un bon graphique doit correspondre à la relation que vous souhaitez mettre en évidence : comparaison, évolution, distribution ou corrélation.

Comparison & Trend

Idéal pour comparer des valeurs entre catégories ou suivre une évolution temporelle.

- **Bar Chart** : Comparaison catégorielle
- **Line Chart** : Séries temporelles

Composition & Distribution

Utilisé pour montrer comment des parties forment un tout ou comment les données sont réparties.

- **Pie/Donut** : Parts d'un tout (max 5)
- **Histogramme** : Fréquence et distribution

Relation & Géographie

Permet d'analyser les corrélations entre variables complexes ou la répartition spatiale.

- **Scatter Plot** : Corrélation X/Y
- **Heatmap** : Densité et matrices

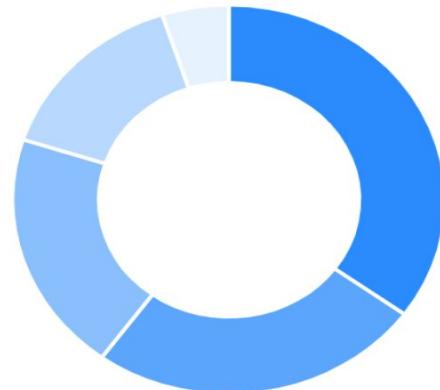
Valeur de l'Interactivité

L'interactivité transforme un rapport statique en un **outil d'exploration** puissant. Elle permet à l'utilisateur de devenir acteur de son analyse en manipulant directement les données pour répondre à ses questions spécifiques, sans nécessiter de nouvelles requêtes techniques. Cette autonomie accélère la prise de décision et favorise la découverte d'insights cachés (Data Discovery).

- Actions Clés : **Filtrer** pour isoler l'information pertinente, **Zoomer** pour analyser les détails temporels ou géographiques, et **Survoler (Toolips)** pour obtenir des valeurs précises instantanément.

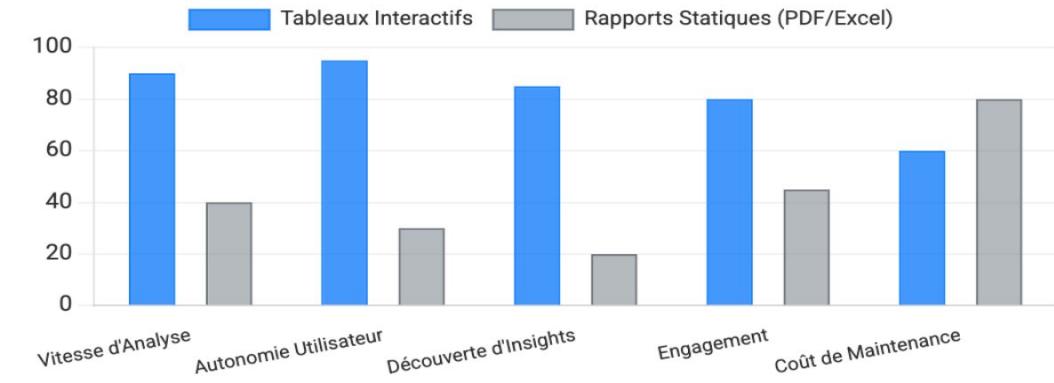
⚠ Attention cependant à la "surcharge cognitive" : trop d'interactivité peut parfois complexifier l'expérience utilisateur.

Répartition des Interactions Utilisateur



- Filtrage (Slicing)
- Forage (Drill-down)
- Survol (Toolips)
- Zoom / Pan
- Export Données

Comparatif Impact : Interactif vs Statique



Installation de l'environnement

Pour développer des tableaux de bord interactifs robustes, une configuration propre et isolée est essentielle. Suivez ces étapes pour préparer votre poste de travail efficacement.



Python + Venv + Dash/Plotly

1. Installer Python

Téléchargez la version 3.10+ depuis python.org.

Sur Windows, cochez impérativement l'option :

Add Python to PATH

> 2. Environnement Virtuel

Créez un dossier pour votre projet et isolez les dépendances :

```
# Création
python -m venv mon_env
# Activation (Windows / Mac-Linux)
mon_env\Scripts\activate | source
mon_env/bin/activate
```

3. Installer les Libs

Installez le framework Dash et les outils de données :

```
pip install dash plotly pandas numpy
```

Bibliothèques et Versions Requises

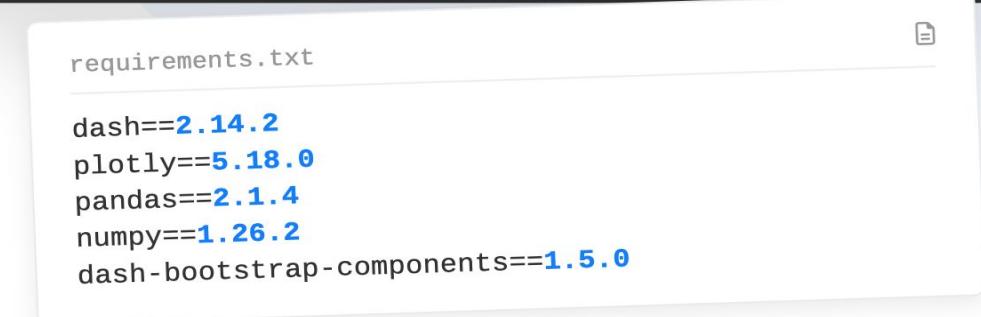
Pour construire des applications de données robustes, nous nous appuyons sur un écosystème de bibliothèques complémentaires. L'installation propre de ces paquets est la première étape critique.

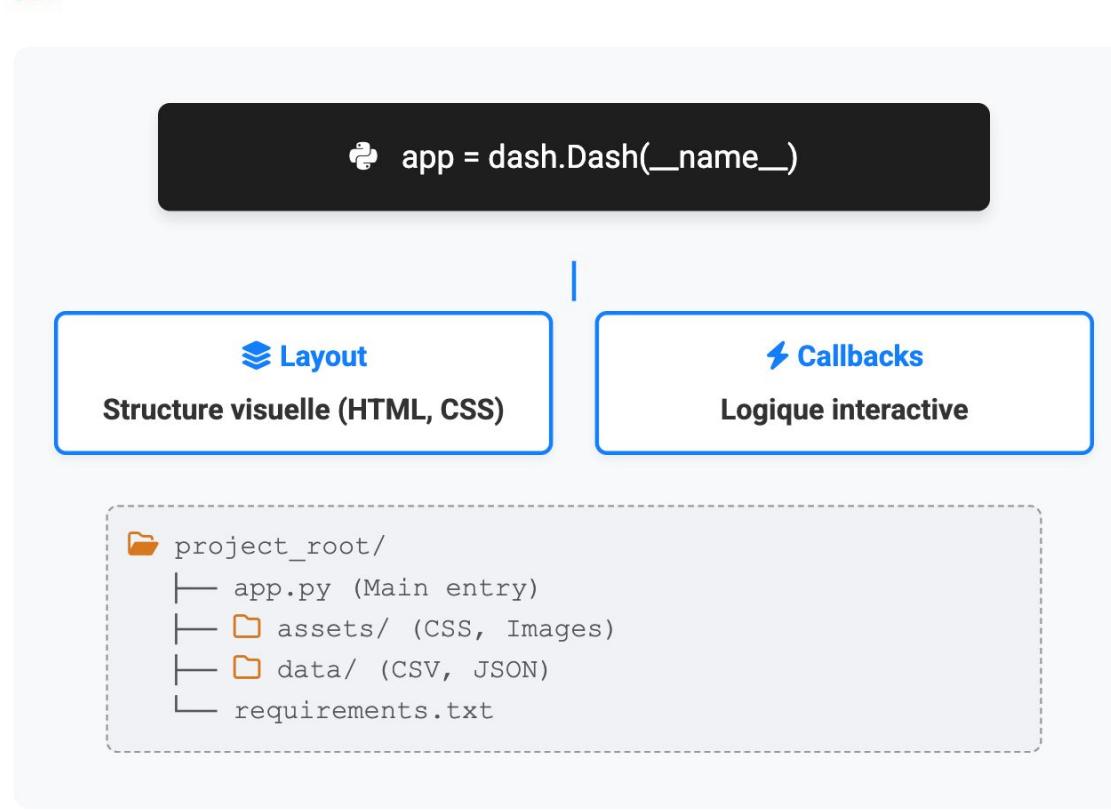
-  **Dash** : Le framework d'application web
-  **Plotly** : Moteur graphique interactif
-  **Pandas** : Manipulation et nettoyage de données
-  **Dash Bootstrap Components** : Styles CSS

✓ **Bonne Pratique :** Utilisez un environnement virtuel (venv) et figez toujours vos versions dans un fichier requirements.txt pour garantir la reproductibilité.

```
user@dev:~$ python -m venv .venv
user@dev:~$ source .venv/bin/activate
# Installation des paquets principaux
(.venv) $ pip install dash plotly pandas numpy dash-bootstrap-components
Installing collected packages... Successfully
installed.

# Sauvegarde des versions
(.venv) $ pip freeze > requirements.txt
```





Architecture d'une Application Dash

Composants Fondamentaux

Une application Dash repose sur deux piliers : le **Layout**, qui définit l'apparence visuelle via `dash.html` et `dash.dcc`, et les **Callbacks**, qui gèrent l'interactivité. L'initialisation se fait via l'objet `app = Dash(__name__)` qui sert de conteneur principal.

Structure de Fichiers

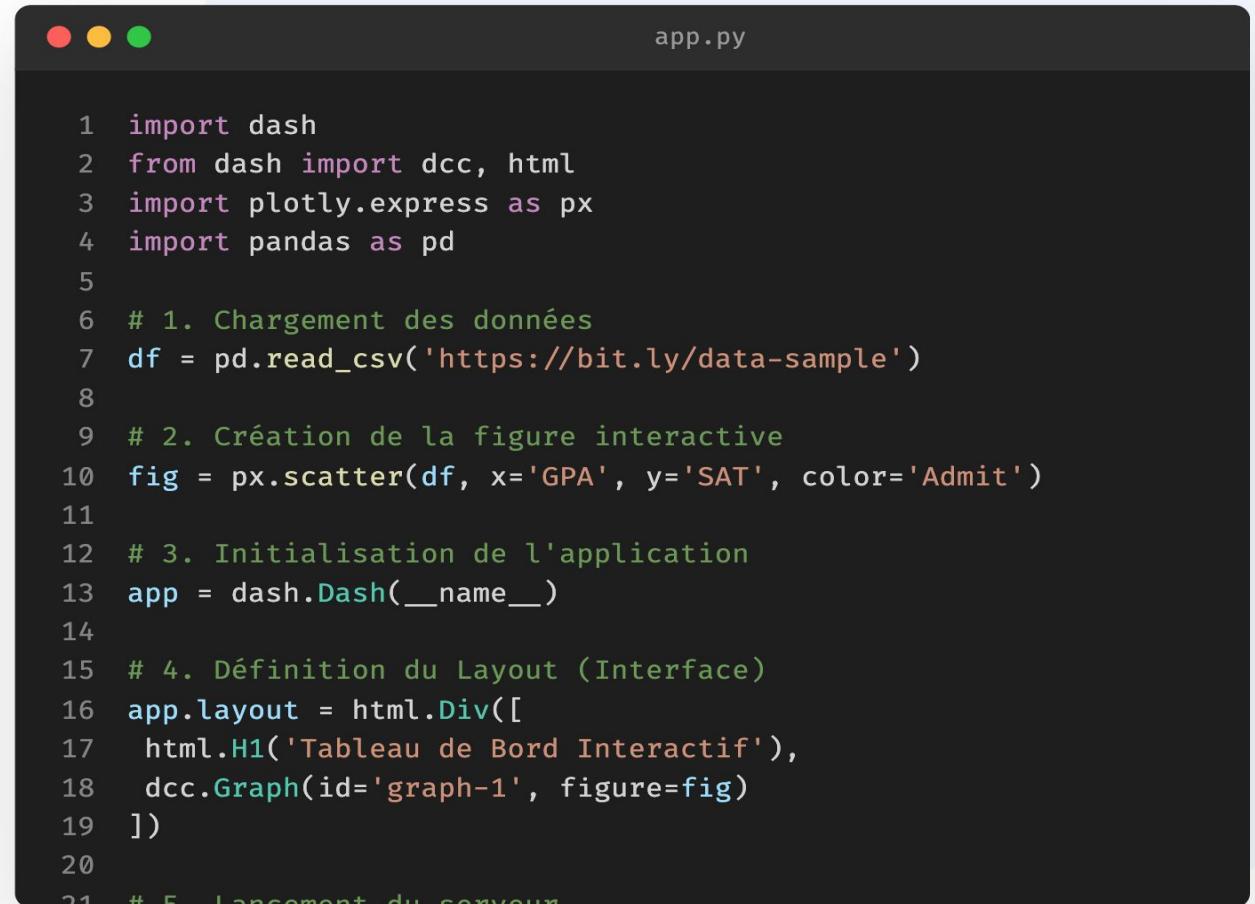
Pour une organisation optimale, séparez votre logique : un fichier principal `app.py` pour le lancement, un dossier `assets/` pour les fichiers statiques (CSS, logos) qui sont chargés automatiquement, et un dossier `data/` pour vos sources de données. Cette structure facilite la maintenance et l'évolution du projet.

Premier dashboard Code Minimal

Créer une application web analytique complète ne nécessite que quelques lignes de code avec Dash. Ce framework abstrait la complexité du web (HTML, CSS, JavaScript) pour vous permettre de vous concentrer sur la logique Python.

L'exemple ci-contre montre la structure fondamentale : importation des librairies, chargement des données via Pandas, création d'un graphique Plotly, et définition de la mise en page (Layout).

⚡ Point clé : En moins de 15 lignes, vous obtenez une application web interactive complète, prête à être déployée, intégrant visualisation de données et serveur web.



```
app.py

1 import dash
2 from dash import dcc, html
3 import plotly.express as px
4 import pandas as pd
5
6 # 1. Chargement des données
7 df = pd.read_csv('https://bit.ly/data-sample')
8
9 # 2. Création de la figure interactive
10 fig = px.scatter(df, x='GPA', y='SAT', color='Admit')
11
12 # 3. Initialisation de l'application
13 app = dash.Dash(__name__)
14
15 # 4. Définition du Layout (Interface)
16 app.layout = html.Div([
17     html.H1('Tableau de Bord Interactif'),
18     dcc.Graph(id='graph-1', figure=fig)
19 ])
20
21 # 5. Lancement du serveur
```

Anatomie du Code Dash

Comprendre la structure d'une application

Une application Dash repose sur une architecture claire séparant la logique de données, la structure visuelle (layout) et l'interactivité. Analysons les composants clés du script minimal que nous venons de voir.

Architecture & Composants

1. Imports Stratégiques

`dash` (coeur), `dcc` (composants graphiques), `html` (structure DOM) et `plotly.express` (visualisation).

2. Données & Figures

Chargement via `pandas` et création de l'objet `fig` en amont. Séparer la donnée du layout améliore la lisibilité.

3. Layout (Mise en page)

`app.layout` définit l'arbre HTML. C'est ici qu'on imbrique titres, textes et le composant `dcc.Graph(figure=fig)`.

Exécution & Points de Vigilance

Initialisation & Serveur

`app = Dash(__name__)` initialise le serveur Flask.
`app.run_server(debug=True)` lance l'app avec rechargement automatique.

Écueil : Conflit de Port

Par défaut sur le port **8050**. Si une instance tourne déjà, l'erreur "Address already in use" survient. Spécifiez `port=8051` si besoin.

Écueil : Chemins de Fichiers

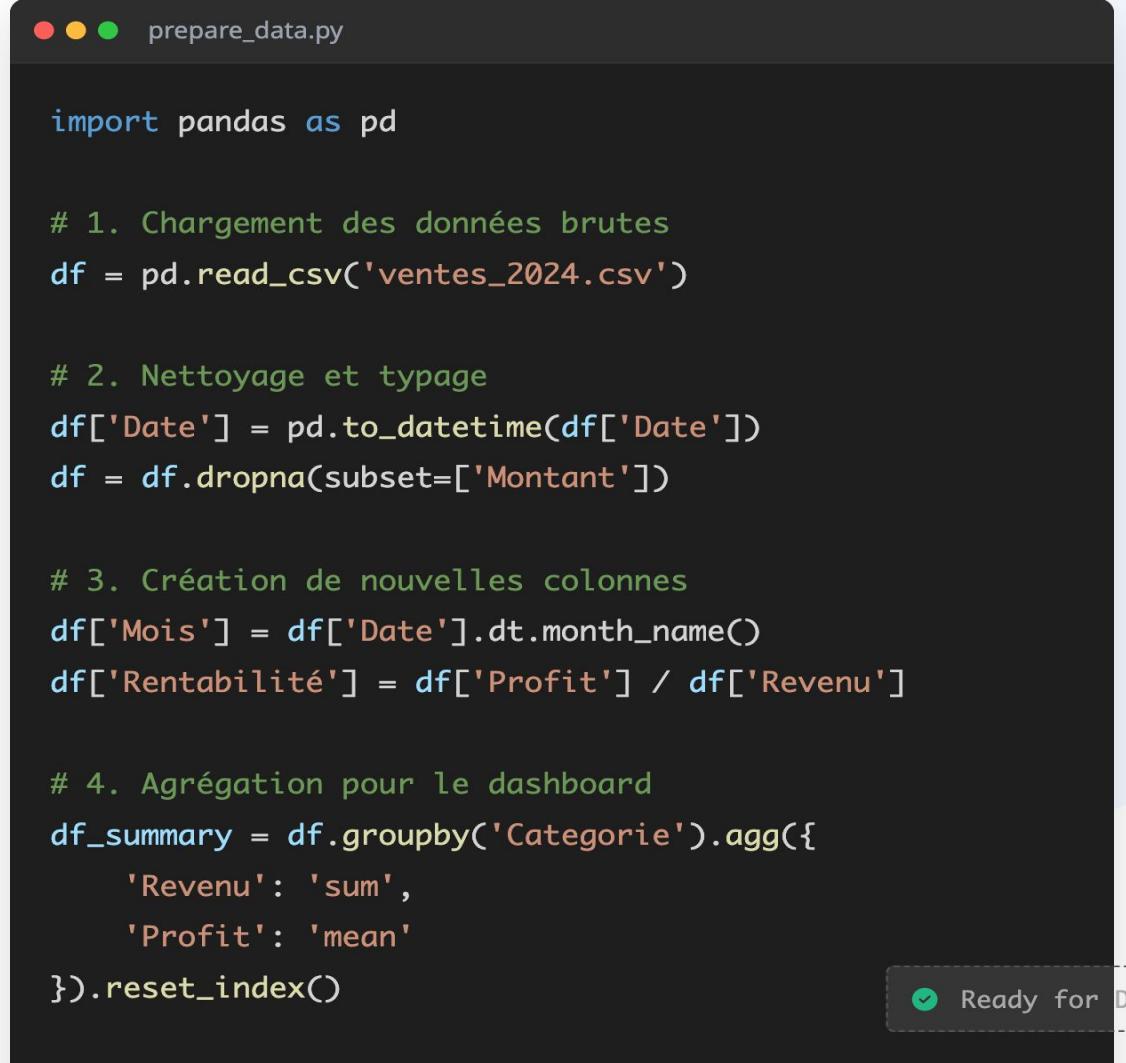
Utilisez des chemins relatifs corrects pour `read_csv`. Vérifiez votre dossier de travail ("cwd") lors de l'exécution du script.

Pandas : Chargement et Préparation

La bibliothèque **Pandas** est le moteur de manipulation de données pour Dash. Avant toute visualisation, les données doivent être chargées depuis des sources variées (CSV, Excel, SQL) et structurées correctement.

Les étapes clés incluent le nettoyage des valeurs manquantes, la conversion des types (notamment les dates pour les séries temporelles) et l'agrégation pour optimiser les performances d'affichage.

 **Astuce Pro :** Effectuez les calculs lourds (`groupby`, `merge`) avec Pandas *avant* de passer les données au graphique Plotly pour réduire la latence de l'application.



```
prepare_data.py

import pandas as pd

# 1. Chargement des données brutes
df = pd.read_csv('ventes_2024.csv')

# 2. Nettoyage et typage
df['Date'] = pd.to_datetime(df['Date'])
df = df.dropna(subset=['Montant'])

# 3. Création de nouvelles colonnes
df['Mois'] = df['Date'].dt.month_name()
df['Rentabilité'] = df['Profit'] / df['Revenu']

# 4. Agrégation pour le dashboard
df_summary = df.groupby('Categorie').agg({
    'Revenu': 'sum',
    'Profit': 'mean'
}).reset_index()

Ready for Dash
```

Pandas : Opérations Clés

▼ 1. Filtrage (Selection)

L'opération la plus courante pour l'interactivité. Elle permet de sélectionner un sous-ensemble de données selon des critères logiques (seuil, catégorie, date). C'est la base des **Callbacks**.

```
# Garder ventes > 1000€  
df_filtered = df[df['ventes'] >  
1000]  
  
# Filtre multiple  
mask = (df['region'] == 'Nord')  
df_nord = df[mask]
```

▀ 2. Agrégation (Groupby)

Indispensable pour résumer les données brutes. Permet de regrouper les lignes par catégorie et d'appliquer une fonction mathématique (somme, moyenne, compte) pour créer des KPI ou des graphiques synthétiques.

```
# CA total par catégorie  
df_agg = df.groupby('cat')\n    .agg({'ca': 'sum'})  
  
# Résultat : Index unique par cat
```

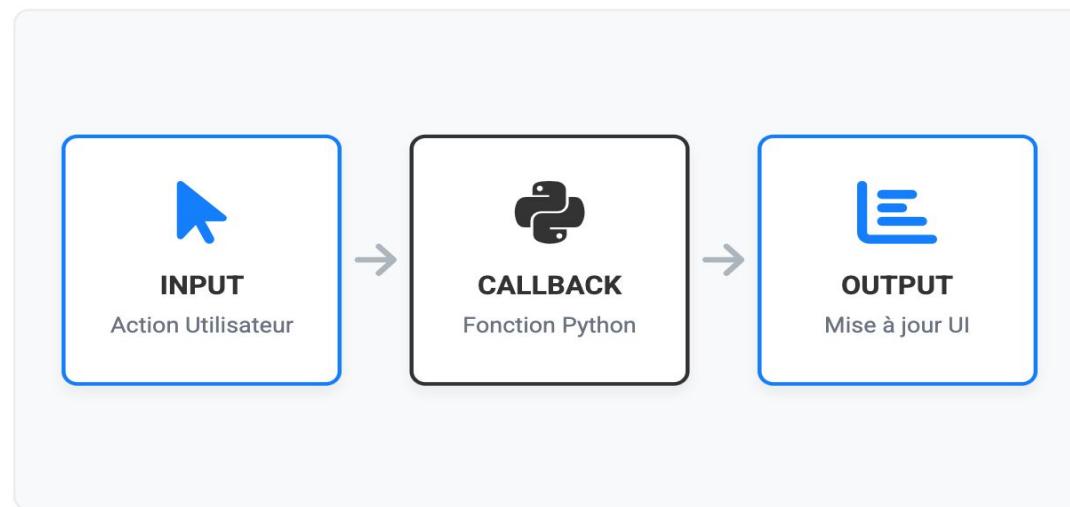
▀ 3. Transformation

La création de nouvelles colonnes ("Feature Engineering") pour enrichir l'analyse. Pandas permet des opérations vectorisées ultra-rapides pour calculer des ratios, extraire des dates ou nettoyer du texte.

```
# Calcul de marge  
df['marge_%'] = (df['profit'] /  
df['ca']) * 100  
  
# Extraction année  
df['annee'] = df['date'].dt.year
```

Les Callbacks : Cœur de la Réactivité

Les **Callbacks** sont des fonctions Python spéciales qui sont automatiquement appelées par Dash chaque fois qu'une propriété d'un composant d'entrée change. C'est ce mécanisme qui transforme une page web statique en une application analytique dynamique, permettant de mettre à jour les graphiques en temps réel selon les actions de l'utilisateur.



Le Décorateur @app.callback

C'est la Liaison "colle" qui lie l'interface graphique à la logique Python. Il déclare explicitement quelles propriétés (ex: `value` d'un Dropdown) déclenchent l'exécution et quelles propriétés (ex: `figure` d'un Graph) seront modifiées par la valeur de retour de la fonction.

Inputs & Outputs

Chaque callback nécessite au moins un **Input** et un **Output**.

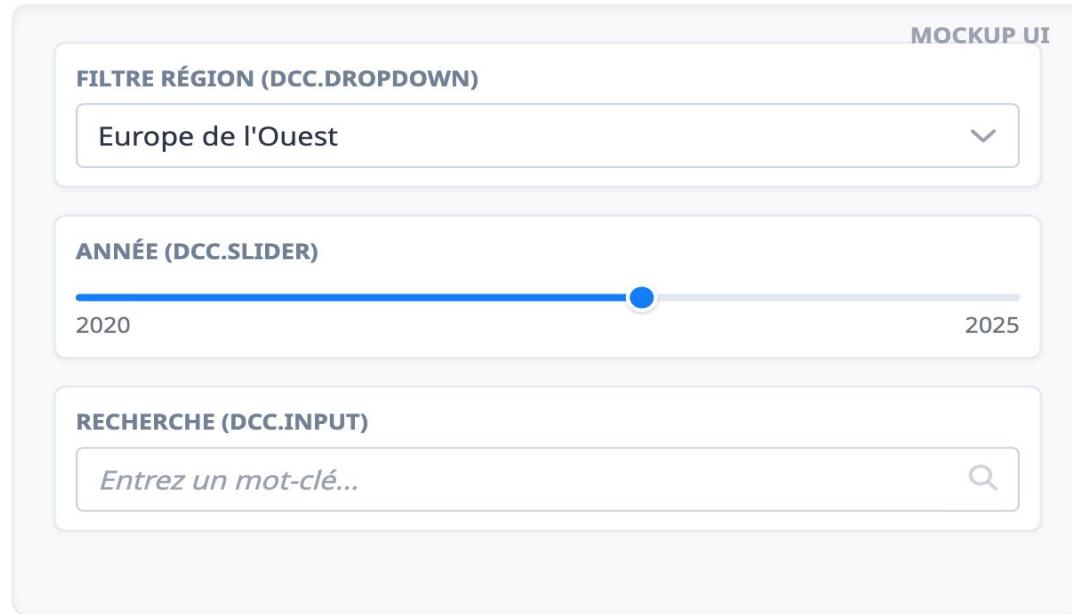
- **Input** : L'élément déclencheur. Dès qu'il change, la fonction s'exécute.
- **Output** : L'élément cible qui recevra le résultat.
- **State** : (Optionnel) Permet de lire une valeur sans déclencher le callback.

Fonctions Pures

Les fonctions de callback doivent être "pures" (stateless). Elles ne doivent jamais modifier de variables globales. Pour les mêmes entrées, elles doivent toujours renvoyer la même sortie. Cela garantit que l'application fonctionne correctement même avec plusieurs utilisateurs simultanés.

Composants Interactifs (dcc)

Le module `dash.dcc` (Dash Core Components) fournit les éléments d'interface essentiels pour capturer les actions des utilisateurs. Ces composants deviennent les **Inputs** de vos callbacks.



1. Menus Déroulants (Dropdown)

Permet la sélection unique ou multiple (`multi=True`). Indispensable pour le filtrage de catégories.

```
dcc.Dropdown(  
    options=[{'label': 'A', 'value': 'a'}],  
    value='a', multi=True  
)
```

2. Curseurs (Slider)

Sélection de valeurs numériques ou plages (RangeSlider). Configurez min, max, pas et marqueurs.

```
dcc.Slider(  
    min=0, max=10, step=1,  
    value=5, marks={0: '0°C', 10: '10°C'}  
)
```

3. Saisie & Choix

Champs texte, boutons radio, cases à cocher pour des interactions précises.

```
dcc.Input(id='my-input', type='text',  
placeholder='Recherche...')  
dcc.RadioItems(options=['A', 'B'], value='A')
```

Callbacks : Exemple Complet

```
● ● ●  
@app.callback(  
    Output('graph-ventes', 'figure'),  
    [Input('dropdown-region', 'value')])  
  
def update_metrics(selected_region):  
    # 1. Filtrage des données  
    df_filtered = df[df.region == selected_region]  
  
    # 2. Création du graphique  
    fig = px.bar(df_filtered, x='date', y='ca')  
  
    return fig
```

Structure & Syntaxe

Le décorateur `@app.callback` est le cœur de l'interactivité. Il définit la relation de cause à effet : les **Inputs** (actions utilisateur) déclenchent la fonction Python, et la valeur renournée est envoyée aux **Outputs** (composants à mettre à jour).

Logique de Traitement

À l'intérieur de la fonction, Pandas joue un rôle clé. On utilise la valeur de l'Input (ex: une région sélectionnée) pour filtrer le DataFrame principal. Un nouvel objet graphique (Figure Plotly) est ensuite généré dynamiquement sur ce sous-ensemble de données.

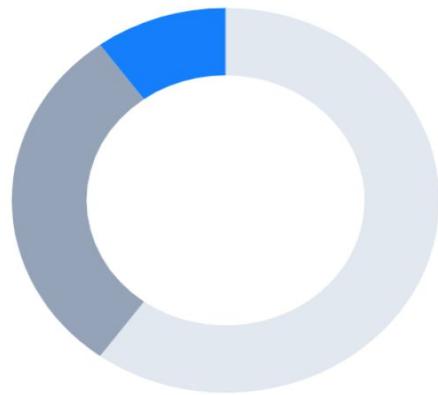
↳ Tableaux de Bord Interactifs

Design & UX/UI

La qualité d'un tableau de bord ne réside pas uniquement dans la complexité de son code, mais dans sa capacité à transmettre l'information instantanément. Une interface efficace réduit la charge cognitive de l'utilisateur grâce à une **hiérarchie visuelle stricte**, un alignement rigoureux des composants et une utilisation stratégique des espaces blancs (whitespace).

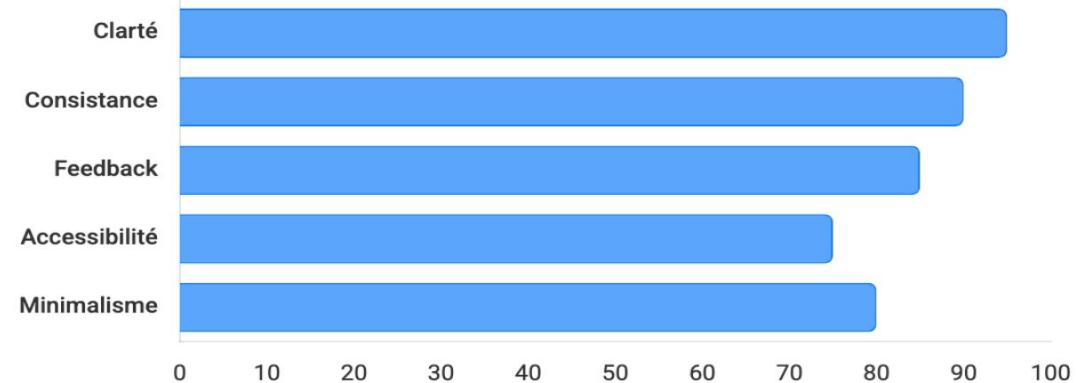
💡 **Règle d'Or :** Un tableau de bord doit répondre à une question clé en moins de **5 secondes**. Limitez chaque vue à 3-5 KPIs majeurs et utilisez le principe du "Less is More" pour éviter la surcharge d'information.

Règle des Couleurs (60-30-10)



- Neutre (Fond/Texte)
- Secondaire (Structure)
- Accent (Data/Action)

Piliers de l'Expérience Utilisateur



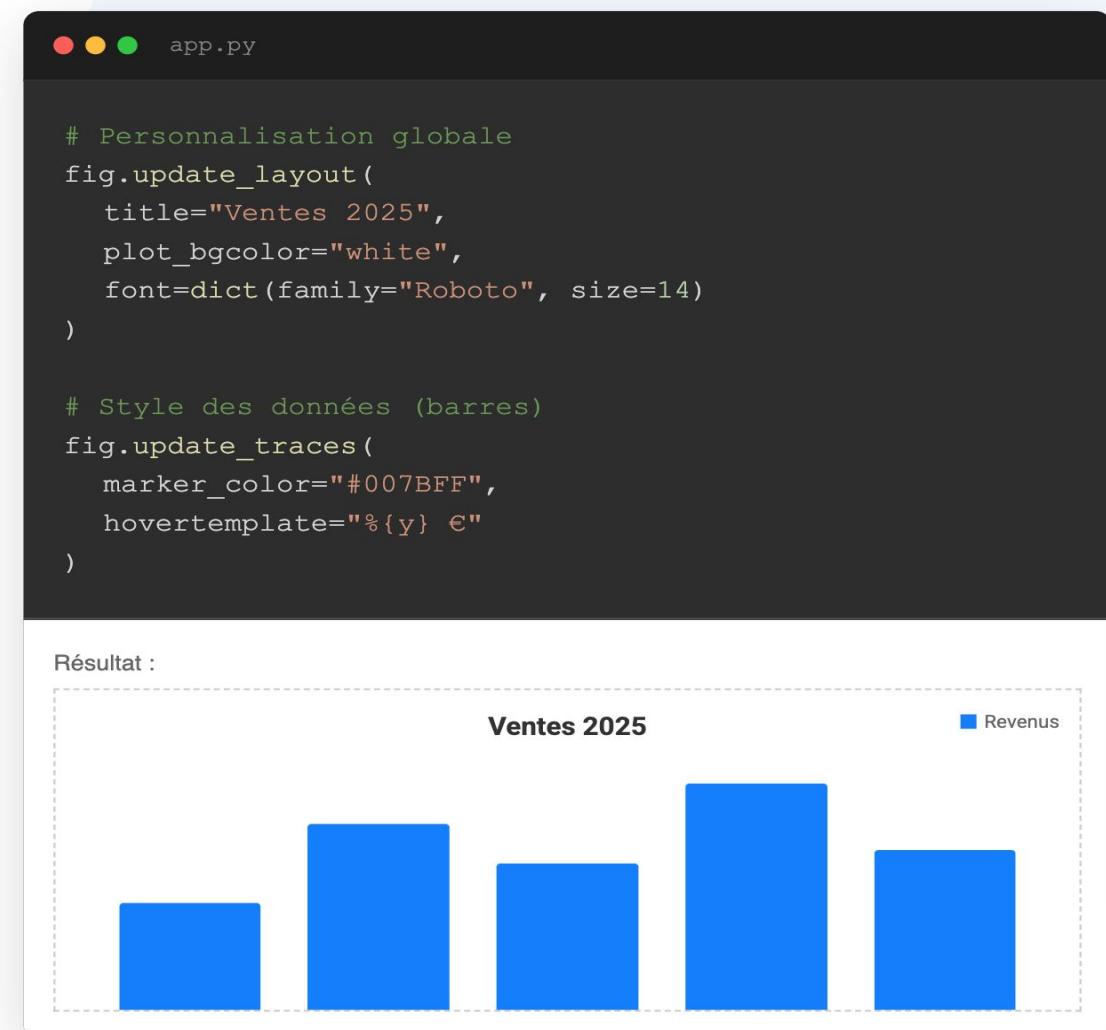
Personnalisation avec Plotly

La puissance de Plotly réside dans sa capacité de personnalisation quasi illimitée. Pour transformer un graphique brut en une visualisation professionnelle, deux méthodes principales sont utilisées :

1. `update_layout()` : Modifie l'apparence globale du "conteneur" du graphique. C'est ici que l'on définit le titre, les marges, la légende, les axes, la couleur de fond ou encore les polices de caractères.

2. `update_traces()` : Cible les propriétés spécifiques des données affichées (les "traces"). Cela permet de changer la couleur des barres, l'épaisseur des lignes, les symboles des marqueurs ou le format des infobulles (`hovertemplate`).

Astuce Pro : Utilisez `template='plotly_white'` ou créez votre propre thème d'entreprise pour assurer une cohérence visuelle instantanée sur tous vos dashboards.



```
app.py
```

```
# Personnalisation globale
fig.update_layout(
    title="Ventes 2025",
    plot_bgcolor="white",
    font=dict(family="Roboto", size=14)
)

# Style des données (barres)
fig.update_traces(
    marker_color="#007BFF",
    hovertemplate="%{y} €"
)
```

Résultat :

Ventes 2025

■ Revenus

Catégorie	Revenus (€)
Produit A	100
Produit B	150
Produit C	120
Produit D	200
Produit E	130

Optimisation des Performances

Gérer les grands volumes de données

Lorsque le volume de données augmente, la réactivité des tableaux de bord peut se dégrader considérablement. L'optimisation n'est pas optionnelle : elle est cruciale pour l'expérience utilisateur (UX) et l'adoption de l'outil.

⚠ Goulots d'étranglement

Les causes principales de ralentissement dans les applications Dash :

☒ Latence Réseau & I/O

Transfert de gros fichiers CSV (>50MB) ou requêtes SQL lentes à chaque interaction.

☒ Callbacks Coûteux

Calculs Pandas complexes (groupby, merge) exécutés en temps réel sur le thread principal.

☒ Surcharge du Navigateur

Tentative de rendu de >10k points dans un graphique SVG classique (le navigateur fige).

⚡ Stratégies d'Optimisation

Techniques éprouvées pour restaurer la fluidité :

⌚ Agrégation Côté Serveur

Ne jamais envoyer toutes les lignes au navigateur. Filtrer et agréger avant le rendu.

⌚ Caching & Stockage

Utiliser `dcc.Store` pour partager des données entre callbacks ou `Flask-Caching` pour mémoriser les résultats.

⌚ Rendu WebGL

Utiliser `Scattergl` au lieu de `Scatter` pour exploiter l'accélération GPU (>100k points fluides).

Outils recommandés

 [Flask-Caching](#)

 [Parquet Format](#)

 [Clientside Callbacks](#)

Cas d'Utilisation Professionnels

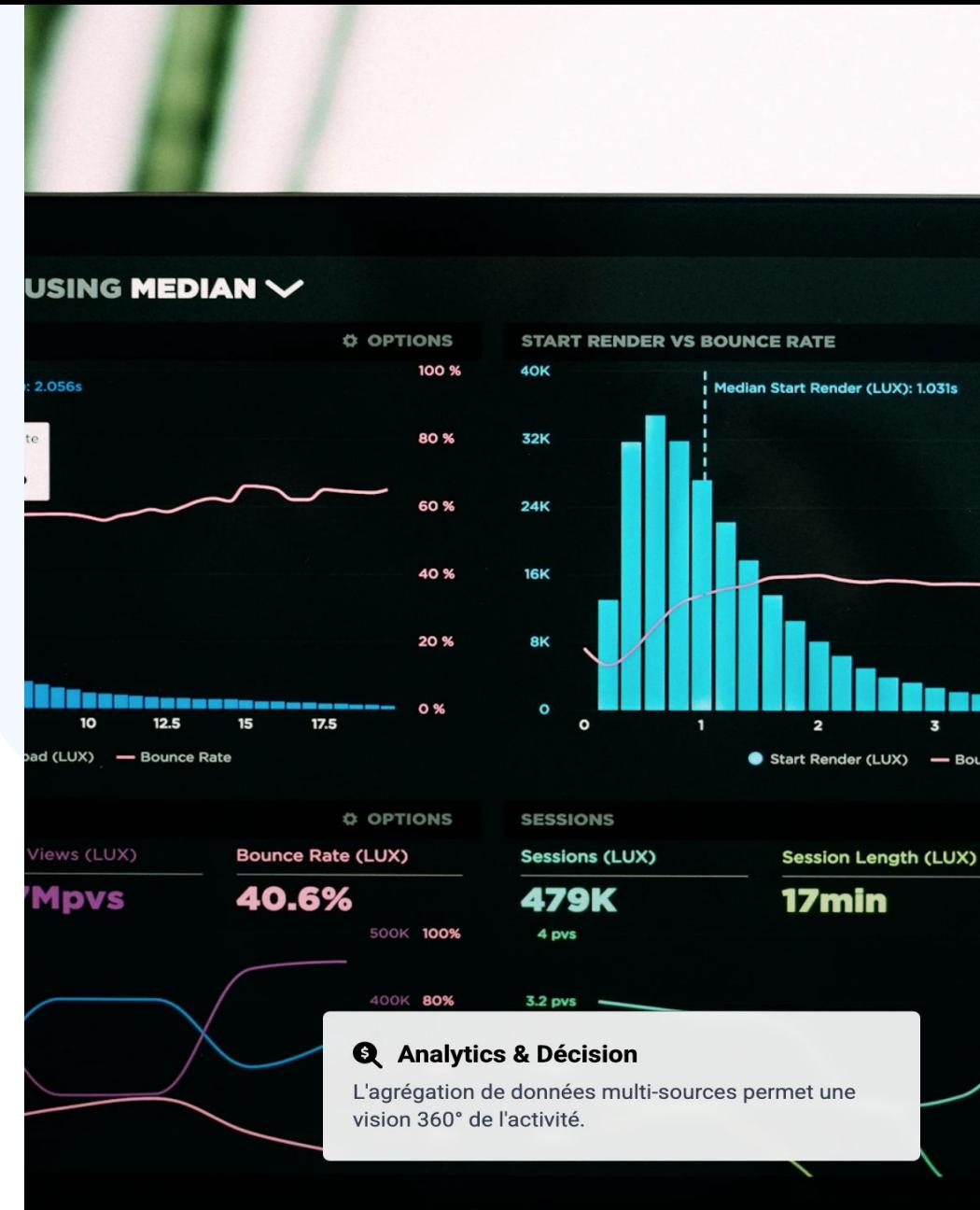
Ventes & Finance

Suivi dynamique du chiffre d'affaires, analyse des marges et pipeline commercial. Visualisation des KPIs critiques (CAC, LTV, ROI) et prévisions budgétaires pour une prise de décision financière rapide et éclairée.

Production & Industrie

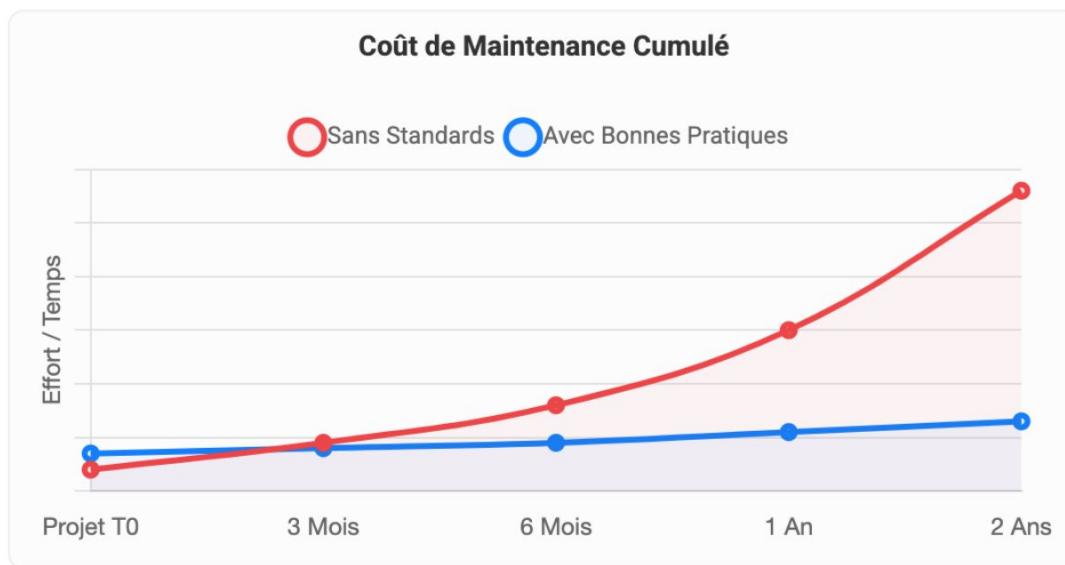
Monitoring en temps réel des lignes de production (OEE, TRS) et détection d'anomalies via l'IoT. Tableaux de bord de maintenance prédictive pour minimiser les arrêts et optimiser la chaîne logistique.

Les tableaux de bord ne se limitent pas au reporting passif : ils deviennent des outils opérationnels permettant d'agir directement sur les processus métier grâce à l'interactivité.



Bonnes pratiques & Ressources

Pour garantir la pérennité de vos applications Dash, l'adoption de standards de développement est cruciale. Une architecture propre réduit la dette technique et facilite la collaboration.



💡 Développement & Architecture

- ✓ **Structure modulaire** : Séparez le layout, les callbacks et les données dans des fichiers distincts (ex: MVC).
- ✓ **Versionning** : Utilisez Git pour suivre l'historique et faciliter le travail en équipe.
- ✓ **Performance** : Utilisez dcc.Store pour partager les données entre callbacks au lieu de variables globales.

💡 Ressources Essentielles

- 🔗 **Documentation officielle** : dash.plotly.com - La référence absolue pour les composants.
- 🔗 **Dash App Gallery** : Une collection d'exemples open-source pour l'inspiration UI/UX.
- 🔗 **Forum Community Plotly** : Une communauté très active pour résoudre les bugs spécifiques.

🎓 Tableaux de Bord Interactif avec Python - Conclusion

Synthèse des acquis

Fondamentaux Techniques

- Architecture MVC avec **Dash** (Layout, Callbacks)
- Visualisation avancée avec l'écosystème **Plotly**
- Intégration fluide du pipeline de données **Pandas**
- Gestion de l'interactivité via les Callbacks réactifs

Qualité & Performance

- Principes de design UX/UI pour dashboards clairs
- Optimisation des performances pour grands datasets
- Bonnes pratiques de code et structure de projet
- Adaptation aux cas d'usage réels (Finance, Industrie)

Votre feuille de route

1. Consolidation

Cloner les exemples du cours, modifier les datasets et personnaliser les graphiques pour maîtriser la syntaxe.

2. Projet Personnel

Créer un dashboard complet à partir d'un dataset public (ex: Kaggle), incluant filtrage avancé et layouts complexes.

3. Professionnalisation

Explorer le déploiement (Docker, Heroku), l'authentification et l'intégration de composants Dash Enterprise.