Comment j'ai créé un dashboard politique interactif avec Python

Retour d'expérience sur le développement d'une application d'analyse des corrélations entre prénoms et tendances de vote en France

Introduction : Quand les données révèlent l'inattendu

Saviez-vous que votre prénom pourrait révéler vos tendances politiques ? C'est la question fascinante que j'ai explorée dans le cadre de mon projet de Master en Informatique à SUPINFO. L'objectif : créer un dashboard interactif permettant d'analyser les corrélations entre les prénoms des citoyens français et leurs comportements électoraux.

Ce projet, baptisé "Qui Vote Quoi", combine data science, géovisualisation et développement web pour offrir une expérience utilisateur moderne et intuitive. Dans cet article, je partage mon retour d'expérience sur les choix techniques, les défis rencontrés et les solutions apportées.

Le défi : Croiser démographie et politique

Problématique de départ

L'idée est née d'une observation simple : les prénoms reflètent souvent l'époque, la région et le milieu social des parents. Ces mêmes facteurs influencent-ils les choix politiques ? Pour le vérifier, j'ai décidé de croiser deux sources de données massives :

- **Données INSEE** : Prénoms par département et année de naissance (plusieurs millions d'enregistrements)
- **Données électorales** : Résultats des élections par département

L'approche méthodologique

1. **Identifier le département le plus représentatif** pour un prénom donné
2. **Analyser les résultats électoraux** de ce département
3. **Visualiser les corrélations** sur une interface interactive
4. **Permettre l'exploration** par période générationnelle

Stack technique : Pourquoi ces choix ?

Backend : Python + PostgreSQL + Docker

PostgreSQL s'est imposé pour plusieurs raisons :

- **Performance** sur les requêtes complexes avec JOINs multiples
- **Robustesse** pour les calculs de ratios et agrégations
- **Extensibilité** avec les types de données avancés

Docker pour l'orchestration :

```yaml
# docker-compose.yml simplifié
services:
  postgres:
    image: postgres:15
    environment:
      POSTGRES_DB: voters
      POSTGRES_USER: dev_user
    volumes:
      - postgres_data:/var/lib/postgresql/data

  pgadmin:
    image: dpage/pgadmin4
    ports:
      - "8080:80"
```

Frontend : Le pari Streamlit

Pourquoi Streamlit ?

Au lieu d'un framework web traditionnel (Django, Flask), j'ai choisi **Streamlit** pour plusieurs raisons stratégiques :

- **Rapidité de développement** : Dashboard complet en quelques heures
- **Intégration Python native** : Pas de translation backend/frontend
- **Widgets interactifs** intégrés : Formulaires, graphiques, cartes
- **Déploiement simplifié** : Une seule commande pour lancer

```python
# Example de la simplicité Streamlit
import streamlit as st
import plotly.express as px

st.title("🗳️ Qui Vote Quoi")
prenom = st.text_input("Entrez un prénom:")

if prenom:
    results = analyze_prenom(prenom)
    fig = px.bar(results, x='candidat', y='votes')
    st.plotly_chart(fig)
```

Géovisualisation : Folium pour les cartes

Pour la carte interactive de France, **Folium** s'est révélé être le choix optimal :

python

```python
import folium
from streamlit_folium import st_folium

# Création de la carte avec géolocalisation
m = folium.Map(location=[46.6, 1.9], zoom_start=6)

# Ajout des départements avec style conditionnel
folium.GeoJson(
    departements_geojson,
    style_function=lambda x: {
        'fillColor': 'red' if x['properties']['code'] == dept_highlight else 'blue',
        'weight': 3 if x['properties']['code'] == dept_highlight else 1
    }
).add_to(m)

# Intégration dans Streamlit
st_folium(m, width=700, height=500)
```

Défis techniques rencontrés

1. Le cas particulier de la Corse

Problème : La Corse était le département "20" jusqu'en 1976, puis divisée en "2A" (Corse-du-Sud) et "2B" (Haute-Corse) . Les données INSEE conservent l'ancien code, mais les données géographiques utilisent les nouveaux.

Solution : Mapping intelligent avec gestion des cas spéciaux :

```python
def convert_dept_code_for_map(dept_code):
    if dept_code == "20":
        return ["2A", "2B"]  # Illuminer les deux départements
    elif dept_code.isdigit() and len(dept_code) == 1:
        return [f"0{dept_code}"]  # Padding des codes courts
    return [dept_code]
```

2. Performance des requêtes complexes

Problème : Requêtes SQL impliquant plusieurs JOINs sur millions d'enregistrements (prénoms × départements × années).

Solution : Cache multi-niveau et optimisation SQL :

```python
# Cache en mémoire Python
_query_cache = {}

def get_cached_result(cache_key, query_func):
    if cache_key in _query_cache:
        print("🚀 Cache hit!")
        return _query_cache[cache_key]

    result = query_func()
    _query_cache[cache_key] = result
    return result

# Session State Streamlit pour persistance
if 'search_results' not in st.session_state:
    st.session_state.search_results = None
```

3. Gestion des faux positifs

Problème : Certains prénoms très rares donnaient des résultats aberrants (ratios artificiellement élevés).

Solution : Validation statistique avec seuils :

```python
def validate_result(dept_info, threshold_ratio=0.0001, min_occurrences=10):
    """Filtrer les résultats non significatifs"""
    if not dept_info:
        return False

    ratio = dept_info.get('ratio', 0)
    occurrences = dept_info.get('prenom_total', 0)

    return ratio >= threshold_ratio and occurrences >= min_occurrences
```

Résultats et métriques

Performance obtenue

Métrique	Valeur	Commentaire
Temps de recherche moyen	0.8s	Acceptable pour l'interactivité
Taux de cache hit	85%	Excellent pour les prénoms populaires
Données traitées	2.5M+ enregistrements	INSEE + élections
Départements couverts	96/101	Métropole + quelques DOM-TOM

Fonctionnalités réalisées

- **Recherche par prénom** avec autocomplétion
- **Filtrage par période** (Baby Boomers, Génération X, etc.)
- **Carte interactive** avec départements colorés
- **Graphiques dynamiques** des résultats électoraux
- **Export de données** (CSV, JSON)

Historique des recherches avec métriques
Mode comparaison multi-départements

Exemples de résultats intéressants

- **MARIE** (1960-1990) → Corse (département historique 20)
- **JEAN** (1946-1964) → Nord (département 59)
- **KEVIN** (1981-1996) → Seine-Saint-Denis (département 93)

Interface utilisateur : UX/UI moderne

Design System

J'ai opté pour un design moderne avec :

```python
# CSS personnalisé pour professionnaliser l'interface
st.markdown("""
<style>
.stMetric {
    background-color: #f0f2f6;
    padding: 1rem;
    border-radius: 0.5rem;
    border-left: 4px solid #1f77b4;
}
.success-message {
    background-color: #d4edda;
    color: #155724;
    padding: 1rem;
    border-radius: 0.5rem;
}
</style>
""", unsafe_allow_html=True)
```

Composants clés

1. **Sidebar paramétrable** : Recherche, filtres, historique
2. **Layout 2/3 - 1/3** : Carte principale + résultats détaillés
3. **Métriques temps réel** : Performance, statistiques
4. **Messages contextuels** : Success, warning, error avec emojis

Leçons apprises

Ce qui a bien fonctionné

1. **Streamlit pour le prototypage rapide** : Gain de temps énorme
2. **PostgreSQL pour les données complexes** : Requêtes robustes
3. **Cache intelligent** : Performance acceptable malgré la complexité
4. **Session State** : Persistance des données utilisateur

Ce que je ferais différemment

1. **Prévoir la scalabilité** : Redis pour un cache distribué
2. **Tests automatisés** : Couverture des cas edge dès le début
3. **Validation des données** : Contrôles plus stricts à l'import
4. **Documentation** : README et docstrings plus détaillés

Points d'amélioration futurs

- **Machine Learning** : Prédiction des tendances avec algorithmes
- **API REST** : Séparation backend/frontend pour la réutilisabilité
- **Données temps réel** : Intégration avec les API électorales officielles
- **Mobile-first** : Optimisation pour smartphones et tablettes

Impact et perspectives

Valeur ajoutée du projet

Ce dashboard démontre plusieurs compétences clés :

- **Full-Stack Development** : De la base de données à l'interface
- **Data Science** : Analyse statistique de données massives
- **Product Management** : Identification du besoin et solution UX
- **DevOps** : Containerisation et déploiement

Applications potentielles

1. **Recherche sociologique** : Analyse des corrélations socio-politiques
2. **Marketing politique** : Ciblage démographique des campagnes
3. **Études de marché** : Segmentation client par profil démographique
4. **Outil pédagogique** : Sensibilisation aux biais statistiques

💡 Conseils pour reproduire le projet

Prérequis techniques

```bash
# Installation de l'environnement
poetry install
docker compose up -d

# Import des données
poetry run python3 main.py import

# Lancement du dashboard
poetry run python3 main.py dashboard
```

Points d'attention

1. **Qualité des données** : Nettoyage et validation essentiels
2. **Performance** : Index et cache dès la conception
3. **UX** : Feedback utilisateur constant (loading, erreurs)
4. **Sécurité** : Validation des inputs, protection SQL injection

Conclusion

Ce projet illustre parfaitement la puissance de Python pour la data science appliquée. En quelques semaines, j'ai pu créer un dashboard professionnel analysant des millions de données avec une interface moderne et intuitive.

Streamlit s'est révélé être un choix judicieux pour le prototypage rapide, bien que ses limitations apparaissent pour des applications plus complexes. La combinaison **PostgreSQL + Docker** a apporté la robustesse nécessaire pour gérer la complexité des données.

Au-delà de l'aspect technique, ce projet démontre l'importance de la **validation statistique** et de la **gestion des biais** dans l'analyse de données. Les corrélations observées doivent être interprétées avec prudence et contextualisation.

Prochaines étapes

1. **Open Source** : Publication du code sur GitHub
2. **Article scientifique** : Formalisation de la méthodologie
3. **Déploiement public** : Mise en ligne sur Streamlit Cloud
4. **Extension** : Intégration d'autres sources de données

Contact : Pour questions techniques ou collaborations

Cet article illustre comment la data science peut révéler des insights surprenants sur notre société, tout en montrant l'importance d'une approche technique rigoureuse et d'une interface utilisateur soignée.