03_pipeline

September 21, 2025

[1]:
```
import pandas as pd
import numpy as np
from datetime import datetime
from functions import *

class ClimateRiskPipeline:
    """Pipeline complet pour l'analyse des risques climatiques sectoriels"""

    def __init__(self, output_dir='./'):
        self.output_dir = output_dir
        self.df_sectors = None
        self.financial_data = None
        self.df_complete = None
        self.analysis_results = None
        self.correlations = None
        self.risk_stats = None

    def run_complete_analysis(self, fetch_live_data=True, save_results=True):
        """Exécute le pipeline d'analyse complet"""
        print("PIPELINE ANALYSE RISQUES CLIMATIQUES - DÉBUT")
        print("=" * 60)

        start_time = datetime.now()

        # Étape 1: Chargement des données de base
        print("\nÉTAPE 1: Chargement des données sectorielles de base")
        self._load_base_data()

        # Étape 2: Récupération des données financières
        print("\nÉTAPE 2: Récupération des données financières")
        self._fetch_financial_data(fetch_live_data)

        # Étape 3: Calcul des scores de risque climatique
        print("\nÉTAPE 3: Calcul des scores de risque climatique")
        self._calculate_climate_scores()

        # Étape 4: Fusion des données (AVANT classification)
```

```
        print("\nÉTAPE 4: Fusion des données climatiques et financières")
        self._merge_datasets()

        # Étape 5: Classification et enrichissement (APRÈS fusion)
        print("\nÉTAPE 5: Classification des niveaux de risque")
        self._classify_and_enrich()

        # Étape 6: Analyse des corrélations
        print("\nÉTAPE 6: Analyse des corrélations et statistiques")
        self._analyze_correlations()

        # Étape 7: Génération du rapport
        print("\nÉTAPE 7: Génération du rapport d'analyse")
        self._generate_analysis_report()

        # Étape 8: Sauvegarde
        if save_results:
            print("\nÉTAPE 8: Sauvegarde des résultats")
            self._save_results()

        end_time = datetime.now()
        execution_time = (end_time - start_time).total_seconds()

        print(f"\nPIPELINE TERMINÉ en {execution_time:.1f}s")
        print("=" * 60)

        return self.df_complete

    def _load_base_data(self):
        """Charge les données sectorielles de base"""
        self.df_sectors = load_sector_base_data()
        print(f"  {len(self.df_sectors)} secteurs chargés")
        print(f"  Colonnes: {list(self.df_sectors.columns)}")

    def _fetch_financial_data(self, fetch_live=True):
        """Récupère les données financières"""
        if fetch_live:
            print("  Récupération des données financières en temps réel...")
            self.financial_data = fetch_financial_data(365)
            success_count = len([v for v in self.financial_data.values() if␣
↪v['Annual_Return'] != 0])
            print(f"  Données récupérées pour {success_count}/{len(self.
↪financial_data)} secteurs")
        else:
            print("  Mode simulation: données financières par défaut")
            sectors = self.df_sectors['Sector'].tolist()
            self.financial_data = {
```

```
                sector: {'Annual_Return': np.random.normal(8, 15), 'Volatility':
↪ np.random.normal(20, 5)}
                for sector in sectors
            }
            print(f"  Données simulées pour {len(self.financial_data)}␣
↪secteurs")

    def _calculate_climate_scores(self):
        """Calcule les scores de risque climatique"""
        self.df_sectors = calculate_climate_risk_score(self.df_sectors)
        avg_score = self.df_sectors['Climate_Risk_Score'].mean()
        high_risk_count = len(self.df_sectors[self.
↪df_sectors['Climate_Risk_Score'] >= 70])
        print(f"  Scores calculés - Moyenne: {avg_score:.1f}, Risque élevé:␣
↪{high_risk_count} secteurs")

    def _merge_datasets(self):
        """Fusionne les données climatiques et financières"""
        self.df_complete = merge_climate_financial_data(self.df_sectors, self.
↪financial_data)
        print(f"  Datasets fusionnés - Shape finale: {self.df_complete.shape}")
        print(f"   Colonnes finales: {len(self.df_complete.columns)}")

    def _classify_and_enrich(self):
        """Classifie les niveaux de risque et ajoute les métriques"""
        # Maintenant on peut utiliser df_complete qui contient Annual_Return
        self.df_complete = classify_risk_levels(self.df_complete)
        risk_distribution = self.df_complete['Risk_Level'].value_counts()
        print(f"  Classification terminée:")
        for level, count in risk_distribution.items():
            print(f"    {level}: {count} secteurs")

    def _analyze_correlations(self):
        """Analyse les corrélations entre risque et performance"""
        self.correlations, self.risk_stats = analyze_correlations(self.
↪df_complete)
        print(f"  Corrélations calculées:")
        print(f"   Risque vs Rendement: {self.correlations['climate_return']:.
↪3f}")
        print(f"   Risque vs Volatilité: {self.
↪correlations['climate_volatility']:.3f}")

    def _generate_analysis_report(self):
        """Génère le rapport d'analyse complet"""
        self.analysis_results = generate_risk_report(self.df_complete)
        print(f"  Rapport généré:")
```

```
        print(f"  Secteurs à haut risque: {self.
↪analysis_results['high_risk_count']}")
        print(f"  Score moyen: {self.analysis_results['avg_risk_score']:.1f}")

    def _save_results(self):
        """Sauvegarde les résultats d'analyse"""
        output_file = f"{self.output_dir}climate_risk_analysis.csv"
        result = export_analysis_results(self.df_complete, output_file)
        print(f"  {result}")

    def get_top_risk_sectors(self, n=3):
        """Retourne les secteurs les plus risqués"""
        if self.df_complete is None:
            return None
        return self.df_complete.nlargest(n, 'Climate_Risk_Score')[['Sector',␣
↪'Climate_Risk_Score', 'Annual_Return']]

    def get_low_risk_sectors(self, n=3):
        """Retourne les secteurs les moins risqués"""
        if self.df_complete is None:
            return None
        return self.df_complete.nsmallest(n, 'Climate_Risk_Score')[['Sector',␣
↪'Climate_Risk_Score', 'Annual_Return']]

    def calculate_portfolio_risk_score(self, portfolio_weights):
        """Calcule le score de risque climatique d'un portefeuille"""
        if self.df_complete is None:
            raise ValueError("Pipeline non exécuté - Lancer␣
↪run_complete_analysis() d'abord")

        portfolio_analysis = calculate_portfolio_risk(portfolio_weights, self.
↪df_complete)
        return portfolio_analysis

    def print_summary_report(self):
        """Affiche un rapport de synthèse"""
        if self.analysis_results is None:
            print("Aucune analyse disponible")
            return

        print("\n" + "=" * 60)
        print("RAPPORT DE SYNTHÈSE - RISQUES CLIMATIQUES SECTORIELS")
        print("=" * 60)

        print(f"\nVUE D'ENSEMBLE:")
        print(f"• Secteurs analysés: {self.analysis_results['total_sectors']}")
```

```
        print(f"• Score moyen de risque: {self.
↪analysis_results['avg_risk_score']:.1f}/100")
        print(f"• Secteurs à haut risque: {self.
↪analysis_results['high_risk_count']}")
        print(f"• Corrélation risque-rendement: {self.
↪analysis_results['correlation_climate_return']:.3f}")

        print(f"\nSECTEURS À HAUT RISQUE CLIMATIQUE:")
        for i, sector in enumerate(self.analysis_results['top_risk_sectors'],␣
↪1):
            print(f"{i}. {sector['Sector']}: Score␣
↪{sector['Climate_Risk_Score']:.1f}")

        print(f"\nSECTEURS À FAIBLE RISQUE CLIMATIQUE:")
        for i, sector in enumerate(self.analysis_results['low_risk_sectors'],␣
↪1):
            print(f"{i}. {sector['Sector']}: Score␣
↪{sector['Climate_Risk_Score']:.1f}")

        print("\nRECOMMANDATIONS:")
        if self.analysis_results['correlation_climate_return'] < -0.3:
            print("   Corrélation négative forte: Les secteurs à haut risque␣
↪climatique sous-performent")
        elif self.analysis_results['correlation_climate_return'] > 0.3:
            print("  Paradoxe: Les secteurs à haut risque climatique␣
↪surperforment (temporaire?)")
        else:
            print("  Pas de corrélation claire risque-performance")

        high_risk_pct = (self.analysis_results['high_risk_count'] / self.
↪analysis_results['total_sectors']) * 100
        if high_risk_pct > 40:
            print("  Plus de 40% des secteurs présentent un risque climatique␣
↪élevé")
        elif high_risk_pct > 20:
            print("  Risque climatique modéré dans le portefeuille sectoriel")
        else:
            print("  Profil de risque climatique globalement acceptable")

        print("=" * 60)

    def get_dashboard_data(self):
        """Retourne les données formatées pour le dashboard Streamlit"""
        if self.df_complete is None:
            return None
```

```
        return {
            'dataframe': self.df_complete,
            'correlations': self.correlations,
            'risk_stats': self.risk_stats,
            'analysis_results': self.analysis_results
        }
```

[2]:
```
if __name__ == "__main__":
    print("LANCEMENT DU PIPELINE RISQUES CLIMATIQUES")

    try:
        # Initialisation du pipeline
        pipeline = ClimateRiskPipeline(output_dir='./')

        # Exécution complète
        df_results = pipeline.run_complete_analysis(
            fetch_live_data=True,
            save_results=True
        )

        # Affichage du rapport de synthèse
        pipeline.print_summary_report()

        print("\nPIPELINE TERMINÉ AVEC SUCCÈS!")
        print("Données prêtes pour le dashboard Streamlit")

    except Exception as e:
        print(f"ERREUR PIPELINE: {e}")
        raise
```

```
LANCEMENT DU PIPELINE RISQUES CLIMATIQUES
PIPELINE ANALYSE RISQUES CLIMATIQUES - DÉBUT
============================================================

ÉTAPE 1: Chargement des données sectorielles de base
  11 secteurs chargés
  Colonnes: ['Sector', 'CO2_Intensity', 'Water_Risk_Score', 'Regulatory_Risk',
'Physical_Risk_Exposure']

ÉTAPE 2: Récupération des données financières
  Récupération des données financières en temps réel…

/Users/markus/Downloads/Projets/Exercices/Risques_Climatiques/functions.py:55:
FutureWarning: YF.download() has changed argument auto_adjust default to True
  data = yf.download(ticker, start=start_date, end=end_date, progress=False)
/Users/markus/Downloads/Projets/Exercices/Risques_Climatiques/functions.py:55:
FutureWarning: YF.download() has changed argument auto_adjust default to True
  data = yf.download(ticker, start=start_date, end=end_date, progress=False)
```

```
/Users/markus/Downloads/Projets/Exercices/Risques_Climatiques/functions.py:55:
FutureWarning: YF.download() has changed argument auto_adjust default to True
  data = yf.download(ticker, start=start_date, end=end_date, progress=False)
/Users/markus/Downloads/Projets/Exercices/Risques_Climatiques/functions.py:55:
FutureWarning: YF.download() has changed argument auto_adjust default to True
  data = yf.download(ticker, start=start_date, end=end_date, progress=False)
/Users/markus/Downloads/Projets/Exercices/Risques_Climatiques/functions.py:55:
FutureWarning: YF.download() has changed argument auto_adjust default to True
  data = yf.download(ticker, start=start_date, end=end_date, progress=False)
/Users/markus/Downloads/Projets/Exercices/Risques_Climatiques/functions.py:55:
FutureWarning: YF.download() has changed argument auto_adjust default to True
  data = yf.download(ticker, start=start_date, end=end_date, progress=False)
/Users/markus/Downloads/Projets/Exercices/Risques_Climatiques/functions.py:55:
FutureWarning: YF.download() has changed argument auto_adjust default to True
  data = yf.download(ticker, start=start_date, end=end_date, progress=False)
/Users/markus/Downloads/Projets/Exercices/Risques_Climatiques/functions.py:55:
FutureWarning: YF.download() has changed argument auto_adjust default to True
  data = yf.download(ticker, start=start_date, end=end_date, progress=False)
/Users/markus/Downloads/Projets/Exercices/Risques_Climatiques/functions.py:55:
FutureWarning: YF.download() has changed argument auto_adjust default to True
  data = yf.download(ticker, start=start_date, end=end_date, progress=False)
/Users/markus/Downloads/Projets/Exercices/Risques_Climatiques/functions.py:55:
FutureWarning: YF.download() has changed argument auto_adjust default to True
  data = yf.download(ticker, start=start_date, end=end_date, progress=False)
/Users/markus/Downloads/Projets/Exercices/Risques_Climatiques/functions.py:55:
FutureWarning: YF.download() has changed argument auto_adjust default to True
  data = yf.download(ticker, start=start_date, end=end_date, progress=False)

 Données récupérées pour 0/11 secteurs

ÉTAPE 3: Calcul des scores de risque climatique
 Scores calculés - Moyenne: 42.9, Risque élevé: 2 secteurs

ÉTAPE 4: Fusion des données climatiques et financières
 Datasets fusionnés - Shape finale: (11, 8)
  Colonnes finales: 8

ÉTAPE 5: Classification des niveaux de risque
 Classification terminée:
  Faible: 6 secteurs
  Modéré: 3 secteurs
  Élevé: 2 secteurs

ÉTAPE 6: Analyse des corrélations et statistiques
 Corrélations calculées:
  Risque vs Rendement: nan
  Risque vs Volatilité: nan
```

```
ÉTAPE 7: Génération du rapport d'analyse
  Rapport généré:
   Secteurs à haut risque: 2
   Score moyen: 42.9

ÉTAPE 8: Sauvegarde des résultats
  Analyse sauvegardée: ./climate_risk_analysis.csv

PIPELINE TERMINÉ en 2.5s
============================================================

============================================================
RAPPORT DE SYNTHÈSE - RISQUES CLIMATIQUES SECTORIELS
============================================================

VUE D'ENSEMBLE:
• Secteurs analysés: 11
• Score moyen de risque: 42.9/100
• Secteurs à haut risque: 2
• Corrélation risque-rendement: nan

SECTEURS À HAUT RISQUE CLIMATIQUE:
1. Energy: Score 84.3
2. Utilities: Score 70.9
3. Materials: Score 63.9

SECTEURS À FAIBLE RISQUE CLIMATIQUE:
1. Health Care: Score 19.1
2. Information Technology: Score 19.5
3. Communication Services: Score 24.4

RECOMMANDATIONS:
  Pas de corrélation claire risque-performance
  Profil de risque climatique globalement acceptable
============================================================

PIPELINE TERMINÉ AVEC SUCCÈS!
Données prêtes pour le dashboard Streamlit

/Users/markus/miniconda3/lib/python3.12/site-
packages/numpy/lib/function_base.py:2897: RuntimeWarning: invalid value
encountered in divide
  c /= stddev[:, None]
/Users/markus/miniconda3/lib/python3.12/site-
packages/numpy/lib/function_base.py:2898: RuntimeWarning: invalid value
encountered in divide
  c /= stddev[None, :]
/Users/markus/miniconda3/lib/python3.12/site-
```

```
packages/numpy/lib/function_base.py:2897: RuntimeWarning: invalid value
encountered in divide
  c /= stddev[:, None]
/Users/markus/miniconda3/lib/python3.12/site-
packages/numpy/lib/function_base.py:2898: RuntimeWarning: invalid value
encountered in divide
  c /= stddev[None, :]
```