02_functions

September 21, 2025

[ ]:
```
import pandas as pd
import numpy as np
import yfinance as yf
import matplotlib.pyplot as plt
import seaborn as sns
from datetime import datetime, timedelta
import streamlit as st
import plotly.express as px
import plotly.graph_objects as go
```

[ ]:
```
def load_sector_base_data():
    """Charge les données sectorielles de base avec indicateurs climatiques"""
    sectors_data = {
        'Sector': [
            'Energy', 'Materials', 'Industrials', 'Consumer Discretionary',
            'Consumer Staples', 'Health Care', 'Financials', 'Information␣
 ↪Technology',
            'Communication Services', 'Utilities', 'Real Estate'
        ],
        'CO2_Intensity': [850, 420, 180, 120, 95, 45, 35, 25, 40, 520, 85],
        'Water_Risk_Score': [8.5, 7.2, 6.1, 4.3, 5.8, 2.1, 1.5, 2.8, 3.2, 7.8,␣
 ↪4.9],
        'Regulatory_Risk': [9.2, 7.8, 6.5, 5.1, 4.2, 2.8, 6.8, 3.5, 4.1, 8.1, 5.
 ↪7],
        'Physical_Risk_Exposure': [7.8, 8.1, 7.2, 5.5, 6.3, 3.2, 4.1, 2.9, 3.8,␣
 ↪8.7, 6.8]
    }
    return pd.DataFrame(sectors_data)
```

[ ]:
```
def fetch_financial_data(lookback_days=365):
    """Récupère les données financières sectorielles via yfinance"""
    sector_etfs = {
        'Energy': 'XLE', 'Materials': 'XLB', 'Industrials': 'XLI',
        'Consumer Discretionary': 'XLY', 'Consumer Staples': 'XLP',
        'Health Care': 'XLV', 'Financials': 'XLF', 'Information Technology':␣
 ↪'XLK',
```

```
        'Communication Services': 'XLC', 'Utilities': 'XLU', 'Real Estate':␣
↪'XLRE'
    }

    end_date = datetime.now()
    start_date = end_date - timedelta(days=lookback_days)

    sector_returns = {}
    for sector, ticker in sector_etfs.items():
        try:
            data = yf.download(ticker, start=start_date, end=end_date,␣
↪progress=False)
            prices = data.get('Adj Close', data['Close'])
            if isinstance(prices, pd.DataFrame):
                prices = prices.squeeze()  # DataFrame 1-col → Series
            prices = prices.dropna()

            if len(prices) >= 2:
                returns = (prices.iloc[-1] / prices.iloc[0] - 1) * 100
                volatility = prices.pct_change().dropna().std() * np.sqrt(252)␣
↪* 100
                sector_returns[sector] = {
                    'Annual_Return': round(float(returns), 2),
                    'Volatility': round(float(volatility), 2)
                }
        except:
            sector_returns[sector] = {'Annual_Return': 0, 'Volatility': 20}

    return sector_returns
```

[ ]:
```
def calculate_climate_risk_score(df):
    """Calcule le score composite de risque climatique"""
    def climate_score_formula(row):
        co2_norm = min(row['CO2_Intensity'] / 100, 10)
        water_norm = row['Water_Risk_Score']
        reg_norm = row['Regulatory_Risk']
        phys_norm = row['Physical_Risk_Exposure']

        composite_score = (
            co2_norm * 0.4 +
            phys_norm * 0.3 +
            reg_norm * 0.2 +
            water_norm * 0.1
        ) * 10

        return min(composite_score, 100)
```

```
    df['Climate_Risk_Score'] = df.apply(climate_score_formula, axis=1)
    return df
```

[ ]:
```
def classify_risk_levels(df):
    """Classifie les niveaux de risque et ajoute les métriques dérivées"""
    def risk_level_classifier(score):
        if score >= 70:
            return 'Élevé'
        elif score >= 40:
            return 'Modéré'
        else:
            return 'Faible'

    df['Risk_Level'] = df['Climate_Risk_Score'].apply(risk_level_classifier)
    df['Risk_Adjusted_Return'] = df['Annual_Return'] /␣
 ↪(df['Climate_Risk_Score'] / 10)
    df['ESG_Ready'] = (df['Climate_Risk_Score'] < 50).astype(int)

    return df
```

[ ]:
```
def merge_climate_financial_data(df_sectors, financial_data):
    """Fusionne les données climatiques et financières"""
    financial_list = []
    for _, row in df_sectors.iterrows():
        sector = row['Sector']
        if sector in financial_data:
            financial_list.append(financial_data[sector])
        else:
            financial_list.append({'Annual_Return': 0, 'Volatility': 20})

    df_financial = pd.DataFrame(financial_list)
    return pd.concat([df_sectors, df_financial], axis=1)
```

[ ]:
```
def analyze_correlations(df):
    """Analyse les corrélations entre risque climatique et performance"""
    correlations = {
        'climate_return': df['Climate_Risk_Score'].corr(df['Annual_Return']),
        'climate_volatility': df['Climate_Risk_Score'].corr(df['Volatility'])
    }

    risk_stats = df.groupby('Risk_Level').agg({
        'Annual_Return': ['mean', 'std'],
        'Volatility': 'mean',
        'Climate_Risk_Score': 'mean'
    }).round(2)

    return correlations, risk_stats
```

[ ]:
```python
def create_risk_heatmap(df):
    """Crée une heatmap des composantes de risque pour Streamlit"""
    risk_components = df[['CO2_Intensity', 'Water_Risk_Score',
                          'Regulatory_Risk', 'Physical_Risk_Exposure']]

    fig = px.imshow(risk_components.T,
                    x=df['Sector'],
                    y=['CO2 Intensity', 'Water Risk', 'Regulatory Risk',␣
 ↪'Physical Risk'],
                    color_continuous_scale='Reds',
                    title='Matrice des Risques Climatiques par Secteur')

    fig.update_layout(height=400)
    return fig
```

[ ]:
```python
def create_risk_scatter_plot(df):
    """Crée un scatter plot risque vs rendement pour Streamlit"""
    color_map = {'Élevé': 'red', 'Modéré': 'orange', 'Faible': 'green'}

    fig = px.scatter(df,
                     x='Climate_Risk_Score',
                     y='Annual_Return',
                     color='Risk_Level',
                     color_discrete_map=color_map,
                     size='Volatility',
                     hover_data=['Sector'],
                     title='Risque Climatique vs Performance Financière')

    fig.update_layout(
        xaxis_title='Score de Risque Climatique',
        yaxis_title='Rendement Annuel (%)',
        height=500
    )

    return fig
```

[ ]:
```python
def create_sector_bar_chart(df):
    """Crée un graphique en barres des scores par secteur"""
    df_sorted = df.sort_values('Climate_Risk_Score', ascending=True)

    color_discrete_map = {'Élevé': 'red', 'Modéré': 'orange', 'Faible': 'green'}

    fig = px.bar(df_sorted,
                 x='Climate_Risk_Score',
                 y='Sector',
                 color='Risk_Level',
                 color_discrete_map=color_discrete_map,
```

```
                orientation='h',
                title='Score de Risque Climatique par Secteur')

    fig.add_vline(x=40, line_dash="dash", line_color="orange",
                  annotation_text="Seuil Modéré", annotation_position="top")
    fig.add_vline(x=70, line_dash="dash", line_color="red",
                  annotation_text="Seuil Élevé", annotation_position="top")

    fig.update_layout(height=600, xaxis_title='Score de Risque Climatique')
    return fig
```

[ ]:
```
def calculate_portfolio_risk(portfolio_weights, df):
    """Calcule le risque climatique d'un portefeuille pondéré"""
    if len(portfolio_weights) != len(df):
        return None

    weights_array = np.array(portfolio_weights) / 100  # Conversion en décimales
    portfolio_risk = np.sum(df['Climate_Risk_Score'].values * weights_array)
    portfolio_return = np.sum(df['Annual_Return'].values * weights_array)

    return {
        'risk_score': portfolio_risk,
        'expected_return': portfolio_return,
        'risk_level': 'Élevé' if portfolio_risk >= 70 else 'Modéré' if␣
 ↪portfolio_risk >= 40 else 'Faible'
    }
```

[ ]:
```
def generate_risk_report(df, portfolio_analysis=None):
    """Génère un rapport de risque climatique"""
    report = {
        'total_sectors': len(df),
        'high_risk_count': len(df[df['Risk_Level'] == 'Élevé']),
        'correlation_climate_return': df['Climate_Risk_Score'].
 ↪corr(df['Annual_Return']),
        'avg_risk_score': df['Climate_Risk_Score'].mean(),
        'top_risk_sectors': df.nlargest(3, 'Climate_Risk_Score')[['Sector',␣
 ↪'Climate_Risk_Score']].to_dict('records'),
        'low_risk_sectors': df.nsmallest(3, 'Climate_Risk_Score')[['Sector',␣
 ↪'Climate_Risk_Score']].to_dict('records')
    }

    if portfolio_analysis:
        report['portfolio'] = portfolio_analysis

    return report
```

[ ]:
```python
def export_analysis_results(df, output_path='climate_risk_analysis.csv'):
    """Sauvegarde les résultats d'analyse"""
    df.to_csv(output_path, index=False)
    return f"Analyse sauvegardée: {output_path}"
```