

Ministère de l'enseignement

Supérieur et de la recherche

UCAO-UUT



République Togolaise

Travail-Liberté-Patrie



Ecole Supérieure de l'Ingénierie

(ESI)

Licence 3

Programmation Python

Rapport du TP : Analyse des
ventes d'un magasin

Présenté par : **MAHAMAT**
Ahmat Alkher

Chargé du cours :
M. LOUKOUUME

2024-2025

Table des matières

Introduction.....	3
2.ExPLICATION DU SCRIPT DE CRÉATION DE LA BASE DE DONNÉES	3
3. EXPLICATION DU SCRIPT D'ANALYSE DE VENTES	6
4. EXPLICATION DU SCRIPT DE LA VISUALISATION.....	12
5. EXPLICATION DU SCRIPT DU TABLEAU DE BORD.....	16
6. EXPLICATION DU SCRIPT DE GÉNÉRATION DE RAPPORT PDF	35
7. EXPLICATION DU SCRIPT DU REPORTGENERATOR.....	39
Conclusion	47

Introduction

Ce projet s'inscrit dans le cadre d'un travail pratique visant à maîtriser les outils Python pour l'analyse et la visualisation de données. À travers ce travail, nous avons développé une application complète permettant d'explorer des données commerciales, de générer des indicateurs clés et de produire des rapports automatisés.

L'objectif était de concevoir une solution fonctionnelle intégrant plusieurs composants : un système d'authentification sécurisé, un tableau de bord interactif, des visualisations dynamiques et un générateur de rapports PDF. Pour y parvenir, nous avons utilisé des bibliothèques Python telles que **Streamlit** pour l'interface, **Pandas** pour le traitement des données, **SQLite** pour le stockage, ainsi que **Plotly** et **Matplotlib** pour les graphiques.

Ce document présente notre démarche, depuis la modélisation des données jusqu'à la réalisation des différentes fonctionnalités. Il met en lumière les défis techniques rencontrés, les solutions apportées et les enseignements tirés de ce projet, qui allie développement logiciel et analyse décisionnelle.

2. Explication du script de création de la base de données

Ce script a pour objectif de générer une base de données SQLite contenant des données simulées relatives aux ventes d'un magasin. Il commence par importer les bibliothèques nécessaires : sqlite3 pour interagir avec la base de données, os pour manipuler les chemins de fichiers, datetime et timedelta pour gérer les dates, et random pour la génération aléatoire de données.

Le chemin vers la base de données est défini dynamiquement à l'aide de os.path, afin d'assurer une compatibilité quel que soit l'environnement d'exécution du script. Le fichier de base de données vente.db est situé dans un dossier data, qui sera automatiquement créé s'il n'existe pas encore.

La fonction create_db() commence par établir une connexion à la base de données. Ensuite, elle supprime les tables existantes (ventes, produits, et clients) afin de garantir une création propre et réinitialisée à chaque exécution. Puis, elle définit la structure des trois tables principales. La table produite contient des informations sur les articles en vente (nom et prix), la table clients stocke les noms des clients, et la table ventes enregistre chaque transaction, associant un produit à un client avec une date et une quantité.

Après la création des tables, le script insère des données d'exemple. Une liste de huit produits est ajoutée, avec des prix réalistes pour des articles électroniques. Ensuite, deux clients sont insérés dans la base : "Jean Dupont" et "Marie Martin".

La partie suivante du code génère automatiquement cinquante ventes fictives. Pour chaque vente, un produit et un client sont choisis aléatoirement. La date de la vente est aussi générée

```
import sqlite3
import os
from datetime import datetime, timedelta
import random

# Chemin absolu vers la base
db_path = os.path.join(os.path.dirname(__file__), '../data/vente.db')
def create_db(): 1usage
    # Création du dossier si inexistant
    os.makedirs(os.path.dirname(db_path), exist_ok=True)
    # Connexion/Création de la base
    conn = sqlite3.connect(db_path)
    cursor = conn.cursor()
    # Nettoyage des tables existantes
    cursor.execute("DROP TABLE IF EXISTS ventes")
    cursor.execute("DROP TABLE IF EXISTS produits")
    cursor.execute("DROP TABLE IF EXISTS clients")
    # Création des tables
    cursor.execute("""
CREATE TABLE produits (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    nom TEXT NOT NULL,
    prix REAL NOT NULL
)""")

    cursor.execute("""
CREATE TABLE clients (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    nom TEXT NOT NULL
)""")
```

de façon aléatoire, dans une période allant jusqu'à un an en arrière par rapport à la date actuelle. De plus, une quantité aléatoire entre 1 et 3 est attribuée à chaque vente.

```

cursor.execute("""
CREATE TABLE ventes (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    produit_id INTEGER,
    client_id INTEGER,
    date TEXT NOT NULL,
    quantite INTEGER NOT NULL,
    FOREIGN KEY (produit_id) REFERENCES produits(id),
    FOREIGN KEY (client_id) REFERENCES clients(id)
)""")

# Insertion de données de test
produits = [
    ("Ordinateur portable", 999.99),
    ("Téléphone", 599.99),
    ("Casque audio", 99.99),
    ("Souris sans fil", 25.99), # Nouveau
    ("Clavier mécanique", 89.99), # Nouveau
    ("Écran 4K", 299.99), # Nouveau
    ("Disque dur SSD", 120.50), # Nouveau
    ("Webcam HD", 75.00) # Nouveau
]
cursor.executemany(__sql: "INSERT INTO produits (nom, prix) VALUES (?, ?)", produits)

clients = [
    ("Jean Dupont",),
    ("Marie Martin",)
]
cursor.executemany(__sql: "INSERT INTO clients (nom) VALUES (?)", clients)

```

```

# Génération de 50 ventes aléatoires
for _ in range(50):
    cursor.execute(
        __sql: """INSERT INTO ventes
        (produit_id, client_id, date, quantite)
        VALUES (?, ?, ?, ?)""",
        __parameters: (
            random.randint(a: 1, b: 3), # produit_id
            random.randint(a: 1, b: 2), # client_id
            (datetime.now() - timedelta(days=random.randint(a: 0, b: 365))).strftime("%Y-%m-%d"),
            random.randint(a: 1, b: 3) # quantite
        )
    )

    conn.commit()
    conn.close()
    print(f"Base créée avec succès : {db_path}")

if __name__ == "__main__":
    create_db()

```

Enfin, les modifications sont enregistrées avec conn.commit(), puis la connexion à la base est fermée. Un message de confirmation est affiché pour indiquer que la base de données a été créée avec succès.

3. Explication du Script d'Analyse de Ventes

Le script Python présenté permet de réaliser une analyse complète des ventes à partir d'une base de données SQLite. Il commence par importer les bibliothèques nécessaires comme sqlite3 pour accéder à la base de données, pandas pour la manipulation des données, ainsi que des modules comme os, datetime, pathlib et logging pour gérer les chemins, les fichiers journaux et les dates. Le taux de conversion de l'euro vers le franc CFA est fixé à 655.96, ce qui permet d'exprimer les résultats financiers dans les deux devises. Le script établit ensuite une connexion sécurisée à la base de données et vérifie que les trois tables attendues (ventes, produits, clients) existent et contiennent les colonnes nécessaires. Une fois la connexion établie, il effectue le calcul des principaux indicateurs de performance : chiffre d'affaires total (en euros et en FCFA), nombre de clients uniques, panier moyen par client, ainsi que le top 5 des produits les plus vendus, dont il détermine aussi la part de marché. Les résultats sont ensuite exportés sous forme de fichiers CSV, ce qui facilite leur exploitation dans d'autres outils comme Excel. Le script génère aussi un rapport textuel structuré, contenant les indicateurs clés et un résumé des meilleures ventes. Enfin, l'ensemble du processus est consigné dans un fichier de log pour assurer une traçabilité complète. Ce script constitue donc un outil efficace, fiable et automatisé pour analyser les performances commerciales d'une entreprise.

```
import sqlite3
import pandas as pd
import os
from datetime import datetime
from pathlib import Path
from typing import Tuple, Optional
import logging

# Configuration
TAUX_EURO_CFA = 655.96 # Taux de conversion BCEAO
```

```
BASE_DIR = Path(__file__).parent
DB_PATH = BASE_DIR / "../data/vente.db"
OUTPUT_DIR = BASE_DIR / "../output"
os.makedirs(OUTPUT_DIR, exist_ok=True)

# Configuration du logging
logging.basicConfig(
    level=logging.INFO,
    format='%(asctime)s - %(levelname)s - %(message)s',
    handlers=[
        logging.FileHandler(OUTPUT_DIR / 'analyse.log'),
        logging.StreamHandler()
    ]
)
logger = logging.getLogger(__name__)

class DatabaseError(Exception):
    """Exception personnalisée pour les erreurs de base de données"""
    pass


def verify_database_schema(conn: sqlite3.Connection) -> None:
    """Vérifie que le schéma de la base correspond aux attentes"""
    required_tables = {
        'ventes': {'id', 'produit_id', 'client_id', 'date', 'quantite'},
        'produits': {'id', 'nom', 'prix'},
        'clients': {'id', 'nom'}
    }

    cursor = conn.cursor()

    try:
        # Vérification des tables
        cursor.execute("SELECT name FROM sqlite_master WHERE
type='table';")
        existing_tables = {row[0] for row in cursor.fetchall()}

        missing_tables = set(required_tables.keys()) - existing_tables
        if missing_tables:
            raise DatabaseError(f"Tables manquantes: {missing_tables}")
    
```

```

# Vérification des colonnes
for table, columns in required_tables.items():
    cursor.execute(f"PRAGMA table_info({table});")
    existing_columns = {row[1] for row in cursor.fetchall()}

missing_columns = columns - existing_columns
if missing_columns:
    raise DatabaseError(f"Colonnes manquantes dans '{table}': {missing_columns}")

except sqlite3.Error as e:
    raise DatabaseError(f"Erreur de vérification du schéma: {e}")
finally:
    cursor.close()

def get_db_connection() -> sqlite3.Connection:
    """Établit une connexion sécurisée à la base SQLite"""
    try:
        conn = sqlite3.connect(DB_PATH)
        conn.row_factory = sqlite3.Row
        verify_database_schema(conn)
        return conn
    except sqlite3.Error as e:
        logger.error(f"Erreur de connexion à la base: {e}")
        raise DatabaseError(f"Impossible de se connecter à la base: {e}")

def calculate_kpis(conn: sqlite3.Connection) -> Tuple[pd.DataFrame,
pd.DataFrame]:
    """Calcule les indicateurs clés de performance"""
    try:
        # CA Total
        ca_query = """
        SELECT
            SUM(p.prix * v.quantite) as ca_eur,
            SUM(p.prix * v.quantite) * ? as ca_cfa,
            COUNT(DISTINCT v.client_id) as clients_uniques,
            AVG(p.prix * v.quantite) as panier_moyen_eur
        FROM ventes v
        """

```

```

        JOIN produits p ON v.produit_id = p.id
"""

ca_total = pd.read_sql(ca_query, conn, params=(TAUX_EURO_CFA,))

# Validation des données
if ca_total.isnull().any().any():
    logger.warning("Certaines valeurs KPIs sont nulles - vérifiez
les données source")

# Top produits
top_produits_query = """
SELECT
    p.nom as produit,
    SUM(v.quantite) as quantite,
    SUM(p.prix * v.quantite) as ca_eur,
    SUM(p.prix * v.quantite) * ? as ca_cfa,
    ROUND(SUM(p.prix * v.quantite) * 100.0 /
    (SELECT SUM(p.prix * v.quantite) FROM ventes v JOIN
produits p ON v.produit_id = p.id), 2) as part_marche
FROM ventes v
JOIN produits p ON v.produit_id = p.id
GROUP BY p.nom
ORDER BY ca_eur DESC
LIMIT 5
"""

top_produits = pd.read_sql(top_produits_query, conn,
params=(TAUX_EURO_CFA,))

return ca_total, top_produits

except sqlite3.Error as e:
    logger.error(f"Erreur lors du calcul des KPIs: {e}")
    raise DatabaseError(f"Erreur d'exécution des requêtes: {e}")


def export_results(df: pd.DataFrame, filename: str, output_dir: Path =
OUTPUT_DIR) -> None:
    """Exporte les résultats en CSV avec gestion robuste des erreurs"""
    try:
        filepath = output_dir / filename
        df.to_csv(

```

```

        filepath,
        index=False,
        encoding='utf-8-sig', # Pour une meilleure compatibilité Excel
        date_format='%Y-%m-%d' # Format standard pour les dates
    )
    logger.info(f"Fichier exporté avec succès: {filepath}")
except PermissionError:
    logger.error(f"Permission refusée pour écrire dans {filepath}")
    raise
except Exception as e:
    logger.error(f"Erreur inattendue lors de l'export: {e}")
    raise


def generate_report(ca_total: pd.DataFrame, top_produits: pd.DataFrame) -> str:
    """Génère un rapport textuel des résultats"""
    report = []
    report.append("\n==== RAPPORT D'ANALYSE ===")
    report.append(f"Date: {datetime.now().strftime('%Y-%m-%d %H:%M')}")

    if not ca_total.empty:
        report.append("\n==== INDICATEURS CLÉS ===")
        report.append(f"• CA Total: {ca_total.iloc[0]['ca_eur']:.2f} €"
                     f"\n({ca_total.iloc[0]['ca_cfa']:.0f} FCFA)")
        report.append(f"• Clients uniques: {ca_total.iloc[0]['clients_uniques']}")

        report.append(f"• Panier moyen: {ca_total.iloc[0]['panier_moyen_eur']:.2f} €")

        if not top_produits.empty:
            report.append("\n==== TOP 5 PRODUITS ===")
            with pd.option_context('display.float_format', '{:.2f}'.format):
                report.append(top_produits.to_string(index=False))

    return "\n".join(report)


def analyser_ventes() -> Optional[bool]:
    """Workflow principal d'analyse avec gestion complète des erreurs"""
    conn = None

```

```
try:
    logger.info("Début de l'analyse des ventes")

    # 1. Connexion et vérification
    conn = get_db_connection()
    logger.info("Connexion à la base établie avec succès")

    # 2. Calcul des indicateurs
    ca_total, top_produits = calculate_kpis(conn)
    logger.info("Calcul des indicateurs terminé")

    # 3. Génération et affichage du rapport
    report = generate_report(ca_total, top_produits)
    print(report)

    # 4. Export des résultats
    export_results(top_produits, 'top_produits.csv')
    export_results(ca_total, 'ca_total.csv')

    # 5. Export du rapport texte
    with open(OUTPUT_DIR / 'rapport_analyse.txt', 'w', encoding='utf-
8') as f:
        f.write(report)

    logger.info("Analyse terminée avec succès")
    return True

except DatabaseError as e:
    logger.error(f"Erreur de base de données: {e}")
    return False
except Exception as e:
    logger.critical(f"Erreur inattendue: {e}", exc_info=True)
    return False
finally:
    if conn:
        conn.close()
    logger.info("Connexion à la base fermée")

if __name__ == "__main__":
    print("==== DÉBUT DE L'ANALYSE ===")
```

```

success = analyser_ventes()
status = "SUCCÈS" if success else "ÉCHEC"
print(f"\n==== ANALYSE TERMINÉE - {status} ====")
if not success:
    print("Consultez les logs pour plus de détails")

```

4. Explication du Script de la visualisation

Ce script Python est destiné à visualiser les résultats d'analyse des ventes, en particulier le chiffre d'affaires (CA) généré par les différents produits. Il utilise les bibliothèques **Pandas**, **Matplotlib** et **Seaborn** pour lire les données, configurer les graphiques et produire des visualisations esthétiques. Il commence par définir un style graphique clair avec une grille blanche et des paramètres de taille et de police adaptés à la lisibilité. Ensuite, il charge un fichier CSV appelé `top_produits.csv`, qui doit contenir au minimum deux colonnes : le nom du produit (produit) et le chiffre d'affaires en franc CFA (ca_cfa). Il vérifie que ces colonnes existent et trie les produits par chiffre d'affaires décroissant.

La fonction principale de visualisation crée deux graphiques côte à côté : un **graphique à barres** montrant le montant du chiffre d'affaires par produit, et un **diagramme circulaire (camembert)** représentant la part relative de chaque produit dans le chiffre d'affaires total. Les barres sont annotées avec les montants, et le camembert est enrichi d'une légende détaillant les montants et pourcentages de chaque produit. Une fois générée, la figure est sauvegardée dans un fichier image nommé `repartition_ca.png` dans le dossier `output`. Ce script permet donc de mieux comprendre visuellement quels produits contribuent le plus aux ventes, et peut être intégré à un tableau de bord ou un rapport d'analyse commerciale.

```

import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import os
import sys
from pathlib import Path

# Configuration

```

```

output_dir = Path(__file__).parent.parent / 'output'
os.makedirs(output_dir, exist_ok=True)

def setup_plot_style():
    """Configure le style des graphiques"""
    try:
        sns.set_theme(style="whitegrid")
        plt.rcParams.update({
            'figure.figsize': (16, 8),
            'font.size': 12,
            'axes.titlesize': 16,
            'axes.labelpad': 15,
            'savefig.facecolor': 'white'
        })
    except Exception as e:
        print(f"Warning: Could not set plot style - {str(e)}",
file=sys.stderr)

def load_and_validate_data():
    """Charge et valide les données d'entrée"""
    input_file = output_dir / 'top_produits.csv'
    if not input_file.exists():
        raise FileNotFoundError(f"Fichier {input_file} introuvable.
Exécutez d'abord 02_analyse.py")

    df = pd.read_csv(input_file)
    print("Colonnes détectées:", list(df.columns))

    required_cols = ['produit', 'ca_cfa']
    missing_cols = [col for col in required_cols if col not in df.columns]
    if missing_cols:
        raise KeyError(f"Colonnes manquantes: {missing_cols}. Colonnes
disponibles: {list(df.columns)}")

    return df.sort_values('ca_cfa', ascending=False)

def create_visualizations(df):
    """Crée les visualisations"""

```

```

# Calcul des pourcentages
df['pourcentage'] = df['ca_cfa'] / df['ca_cfa'].sum() * 100

# Création de la figure
fig, (ax1, ax2) = plt.subplots(1, 2)
fig.suptitle('Analyse du Chiffre d\'Affaires', y=1.05, fontsize=18)

# Graphique à barres
bar_plot = sns.barplot(
    data=df,
    x='produit',
    y='ca_cfa',
    palette='viridis',
    ax=ax1
)
ax1.set(title='CA par Produit (FCFA)', xlabel='', ylabel='Montant en FCFA')
ax1.tick_params(axis='x', rotation=45)

# Ajout des valeurs sur les barres
for p in bar_plot.patches:
    ax1.annotate(
        f'{p.get_height():.0f}',
        (p.get_x() + p.get_width() / 2., p.get_height()),
        ha='center', va='center',
        xytext=(0, 10),
        textcoords='offset points'
    )

# Camembert
pie_wedges, _, _ = ax2.pie(
    df['ca_cfa'],
    labels=df['produit'],
    autopct='%1.1f%%',
    startangle=90,
    colors=sns.color_palette('pastel'),
    textprops={'fontsize': 10},
    wedgeprops={'linewidth': 1, 'edgecolor': 'white'}
)
ax2.set_title('Répartition du CA')

```

```

# Légende
ax2.legend(
    pie_wedges,
    [f"{n}: {v:.0f} FCFA ({p:.1f}%)" for n, v, p in zip(df['produit'],
df['ca_cfa'], df['pourcentage'])],
    title='Détail par produit',
    loc='center left',
    bbox_to_anchor=(1, 0.5)
)

plt.tight_layout()
return fig
}

def visualiser_cfa():
    """Fonction principale"""
    try:
        print("== DÉBUT DE LA VISUALISATION ===")

        # Configuration
        setup_plot_style()

        # Données
        df = load_and_validate_data()

        # Visualisation
        fig = create_visualizations(df)

        # Sauvegarde
        output_file = output_dir / 'repartition_ca.png'
        fig.savefig(output_file, dpi=300, bbox_inches='tight')
        print(f"Visualisation sauvegardée dans {output_file}")
        plt.close(fig)

        print("== VISUALISATION TERMINÉE AVEC SUCCÈS ===")
        return True

    except Exception as e:
        print(f"ERREUR: {str(e)}", file=sys.stderr)
        return False

```

```
if __name__ == "__main__":
    success = visualiser_cfa()
    sys.exit(0 if success else 1)
```

5. Explication du Script du tableau de bord

Ce tableau de bord est une application web interactive conçue pour analyser et visualiser des données commerciales. Il utilise Streamlit comme framework principal pour l'interface utilisateur et Plotly pour les visualisations graphiques avancées. L'accès au tableau de bord est sécurisé par un système d'authentification qui vérifie les identifiants des utilisateurs dans une base de données SQLite locale. Les mots de passe sont stockés de manière sécurisée grâce à un hachage SHA-256.

L'application se compose de plusieurs sections clés. La partie authentification gère la connexion des utilisateurs et vérifie leurs permissions. Une fois connecté, l'utilisateur accède à un tableau de bord principal organisé autour de quatre grandes fonctionnalités : des indicateurs clés de performance, des visualisations interactives, un tableau de données filtrables et des outils d'export. Les données proviennent d'une base SQLite séparée contenant les informations sur les ventes, les produits et les clients.

La section d'analyse propose différents types de graphiques permettant d'explorer les données sous plusieurs angles. Un graphique en camembert montre la répartition du chiffre d'affaires par produit, une courbe linéaire illustre l'évolution mensuelle des ventes, et des barres groupées permettent de comparer les performances entre produits. Tous ces éléments sont rendus interactifs grâce à la bibliothèque Plotly.

Pour les administrateurs, une section spéciale permet de gérer les comptes utilisateurs. Ils peuvent visualiser la liste complète des utilisateurs, créer de nouveaux comptes et modifier les permissions. Cette fonctionnalité n'est accessible qu'aux comptes ayant le rôle "admin".

Le système inclut également un générateur de rapports PDF qui permet de créer des documents professionnels à partir des données analysées. Les utilisateurs peuvent personnaliser le contenu du rapport en choisissant d'inclure ou non certains éléments comme les graphiques ou les tableaux détaillés.

En termes de robustesse, l'application comprend de nombreuses vérifications et gestions d'erreurs. Elle vérifie systématiquement la présence des colonnes nécessaires dans les données avant de tenter des opérations d'analyse. Des messages d'erreur clairs guident l'utilisateur en cas de problème.

L'interface est conçue pour être intuitive et responsive. La barre latérale contient les outils de navigation et les informations sur la session en cours, tandis que la zone principale affiche le contenu sélectionné. Les différentes fonctionnalités sont accessibles via un système d'onglets qui permet de naviguer facilement entre les vues.

```
# 1. Importations standards
from datetime import datetime
import sys
from pathlib import Path
import hashlib
import sqlite3
import tempfile

# 2. Importations tierces
import streamlit as st
import pandas as pd
import plotly.express as px

# 3. Importations locales (vos modules)
from report_generator import ReportGenerator
# Configuration des chemins
output_dir = Path(__file__).parent.parent / 'output'
DB_PATH = Path(__file__).parent.parent / 'data' / 'users.db'

# ---- PARTIE AUTHENTIFICATION ----
def init_auth_db():
    """Initialise la base de données d'authentification avec le bon
    schéma"""
    try:
        conn = sqlite3.connect(str(DB_PATH))
        cursor = conn.cursor()

        # Vérifie si la table existe déjà
        cursor.execute("SELECT name FROM sqlite_master WHERE type='table'"
```

```

AND name='users'")

    if not cursor.fetchone():

        # Crée la table avec les colonnes exactes
        cursor.execute("""
CREATE TABLE users (
    username TEXT PRIMARY KEY,
    password_hash TEXT NOT NULL,
    full_name TEXT,
    role TEXT DEFAULT 'user'
)
""")

# Insertion du compte admin avec toutes les colonnes
password_hash = hashlib.sha256('admin123'.encode()).hexdigest()
cursor.execute("""
INSERT INTO users (username, password_hash, full_name, role)
VALUES (?, ?, ?, ?)
""", ('admin', password_hash, 'Administrateur', 'admin'))

conn.commit()

except Exception as e:
    st.error(f"Erreur d'initialisation de la base: {e}")
finally:
    if 'conn' in locals():
        conn.close()

def hash_password(password):
    """Hash le mot de passe avec SHA-256"""
    return hashlib.sha256(password.encode()).hexdigest()

def verify_user(username, password):
    """Vérifie les identifiants de l'utilisateur"""
    try:
        conn = sqlite3.connect(str(DB_PATH))
        cursor = conn.cursor()
        cursor.execute(
            "SELECT password_hash, role, full_name FROM users WHERE
username = ?",
            (username,))

```

```

        )

    result = cursor.fetchone()
    if result:
        stored_hash, role, full_name = result
        if stored_hash == hash_password(password):
            return True, role, full_name
    except sqlite3.Error as e:
        st.error(f"Erreur de base de données: {e}")
finally:
    if 'conn' in locals():
        conn.close()

return False, None, None


def login_page():
    """Affiche la page de connexion"""
    st.title("🔑 Connexion au Dashboard")

    with st.form("login_form"):
        username = st.text_input("Nom d'utilisateur")
        password = st.text_input("Mot de passe", type="password")
        submit = st.form_submit_button("Se connecter")

        if submit:
            is_valid, role, full_name = verify_user(username, password)
            if is_valid:
                st.session_state.update({
                    'authenticated': True,
                    'username': username,
                    'role': role,
                    'full_name': full_name
                })
                st.success("Connexion réussie!")
                st.rerun() # <-- Changement ici (remplacement de
experimental_rerun)
            else:
                st.error("Identifiants incorrects")

# ----- PARTIE DASHBOARD -----

def load_data():
    """

```

```

Charge les données depuis la base SQLite et les prépare pour l'analyse.
Retourne un DataFrame consolidé avec les ventes, produits et clients.

"""

try:
    db_path = Path(__file__).parent / "../data/vente.db"
    conn = sqlite3.connect(db_path)

    # Chargement des tables avec des alias pour éviter les conflits de
    noms
    df_ventes = pd.read_sql("SELECT * FROM ventes", conn)
    df_produits = pd.read_sql("SELECT id as produit_id, nom, prix FROM
    produits", conn)
    df_clients = pd.read_sql("SELECT id as client_id, nom FROM
    clients", conn)

    # Fusion des DataFrames en vérifiant les colonnes
    if not df_ventes.empty:
        df = pd.merge(df_ventes, df_produits, on="produit_id")
        df = pd.merge(df, df_clients, on="client_id")

        # Conversion des dates et calculs
        df['date'] = pd.to_datetime(df['date'])
        df['mois'] = df['date'].dt.strftime('%Y-%m')
        df['chiffre_affaires'] = df['quantite'] * df['prix']

    # Suppression uniquement des colonnes existantes
    cols_to_drop = [col for col in ['id', 'id_vente'] if col in
    df.columns]
    if cols_to_drop:
        df = df.drop(columns=cols_to_drop)

    conn.close()
    return df

else:
    st.warning("Aucune donnée de vente trouvée dans la base.")
    return pd.DataFrame()

except sqlite3.Error as e:
    st.error(f"Erreur de base de données : {str(e)}")
    return pd.DataFrame()
except Exception as e:

```

```

        st.error(f"Erreur de traitement : {str(e)}")
        return pd.DataFrame()

def display_metrics(df):
    """Affiche les indicateurs clés"""
    if df.empty:
        st.warning("Aucune donnée disponible pour afficher les métriques")
        return

    st.header("📊 Vue d'Ensemble")
    cols = st.columns(4)

    try:
        metrics = {
            "CA Total": f"{df['ca_cfa'].sum():,.0f} FCFA",
            "Produit Phare": df.loc[df['ca_cfa'].idxmax(), 'produit'],
            "Panier Moyen": f"{df['ca_cfa'].sum() /
df['quantite'].sum():,.0f} FCFA",
            "Transactions": f"{df['quantite'].sum():,.0f}"
        }

        for col, (name, value) in zip(cols, metrics.items()):
            with col:
                st.markdown(f"""
<div style="background: white;
border-radius: 10px;
padding: 15px;
box-shadow: 0 4px 6px rgba(0,0,0,0.1);
text-align: center;>
<h3 style="margin:0;color:#555;">{name}</h3>
<p style="font-size:24px;margin:10px 0;font-weight:bold;">{value}</p>
</div>
""", unsafe_allow_html=True)
    except Exception as e:
        st.error(f"Erreur dans l'affichage des métriques: {e}")

def display_charts(df):

```

```

"""Affiche les visualisations"""
if df.empty:
    st.warning("Aucune donnée disponible pour afficher les graphiques")
    return

st.header("📈 Analyse des Ventes")
tab1, tab2, tab3 = st.tabs(["Répartition", "Évolution", "Comparaison"])

with tab1:
    try:
        fig = px.pie(
            df,
            values='ca_cfa',
            names='produit',
            title='Répartition du CA par Produit',
            hole=0.4,
            color_discrete_sequence=px.colors.sequential.Viridis
        )
        fig.update_layout(legend=dict(orientation="h",
yanchor="bottom", y=-0.2))
        st.plotly_chart(fig, use_container_width=True)
    except Exception as e:
        st.error(f"Erreur dans le graphique de répartition: {e}")

with tab2:
    try:
        if 'mois' in df.columns:
            monthly = df.groupby('mois', as_index=False).agg({'ca_cfa': 'sum', 'quantite': 'sum'})
            fig = px.line(
                monthly,
                x='mois',
                y='ca_cfa',
                title='Évolution Mensuelle du CA',
                markers=True,
                labels={'ca_cfa': 'CA (FCFA)', 'mois': 'Mois'}
            )
            st.plotly_chart(fig, use_container_width=True)
        else:
            st.warning("Données temporelles non disponibles")
    except Exception as e:

```

```

        st.error(f"Erreur dans le graphique d'évolution: {e}")

    with tab3:
        try:
            fig = px.bar(
                df.sort_values('ca_cfa', ascending=False),
                x='produit',
                y=['ca_cfa', 'quantite'],
                barmode='group',
                title='Comparaison Produits',
                labels={'value': 'Valeur', 'variable': 'Métrique'},
                color_discrete_sequence=['#636EFA', '#00CC96']
            )
            st.plotly_chart(fig, use_container_width=True)
        except Exception as e:
            st.error(f"Erreur dans le graphique de comparaison: {e}")

def display_data_table(df):
    """Affiche le tableau de données avec filtres"""
    if df.empty:
        st.warning("Aucune donnée disponible pour afficher le tableau")
        return

    st.header("🔍 Données Détailées")
    with st.expander("🔎 Filtres", expanded=True):
        col1, col2 = st.columns(2)
        with col1:
            min_ca = st.slider("CA Minimum (FCFA)", 0,
int(df['ca_cfa'].max()), 0)
        with col2:
            selected = st.multiselect(
                "Produits",
                df['produit'].unique(),
                default=df['produit'].unique()
            )

        filtered = df[(df['ca_cfa'] >= min_ca) &
(df['produit'].isin(selected))]

        try:

```

```

        st.dataframe(
            filtered.style
                .background_gradient(subset=['ca_cfa'], cmap='Blues')
                .format({'ca_cfa': '{:,.0f} FCFA'}),
            height=500,
            use_container_width=True
        )

        st.download_button(
            "📤 Exporter les données",
            filtered.to_csv(index=False).encode('utf-8'),
            "export_ventes.csv",
            "text/csv"
        )
    except Exception as e:
        st.error(f"Erreur dans l'affichage du tableau: {e}")

def admin_section():
    """Section réservée à l'administrateur"""
    if st.session_state.get('role') != 'admin':
        return

    with st.expander("⚙️ Administration", expanded=False):
        st.subheader("Gestion des Utilisateurs")

        # Afficher la liste des utilisateurs
        try:
            conn = sqlite3.connect(str(DB_PATH))
            users_df = pd.read_sql("SELECT username, full_name, role FROM
users", conn)
            st.dataframe(users_df, hide_index=True)
        except Exception as e:
            st.error(f"Erreur lors du chargement des utilisateurs: {e}")
        finally:
            if 'conn' in locals():
                conn.close()

        # Formulaire d'ajout d'utilisateur
        with st.form("add_user"):
            st.write("Ajouter un nouvel utilisateur")

```

```

new_user = st.text_input("Nom d'utilisateur*")
new_pass = st.text_input("Mot de passe*", type="password")
full_name = st.text_input("Nom complet")
is_admin = st.checkbox("Administrateur")

if st.form_submit_button("Créer"):
    if not new_user or not new_pass:
        st.error("Les champs marqués d'un * sont obligatoires")
    else:
        try:
            conn = sqlite3.connect(str(DB_PATH))
            conn.execute(
                "INSERT INTO users (username, password_hash,
full_name, role) VALUES (?, ?, ?, ?)",
                (new_user, hash_password(new_pass), full_name,
                'admin' if is_admin else 'user')
            )
            conn.commit()
            st.success(f"Utilisateur {new_user} créé avec
succès!")
            st.rerun()
        except sqlite3.IntegrityError:
            st.error("Ce nom d'utilisateur existe déjà")
        except Exception as e:
            st.error(f"Erreur lors de la création: {e}")
        finally:
            if 'conn' in locals():
                conn.close()

def main_dashboard():
    """Page principale du dashboard avec gestion intégrée des rapports
PDF"""
    # Configuration de la page
    st.set_page_config(
        page_title="Tableau de Bord Commercial",
        page_icon="📊",
        layout="wide"
    )

    # Barre latérale améliorée

```

```

with st.sidebar:

    # Section utilisateur
    st.markdown(f"""
        <div style="text-align:center; margin-bottom:30px;">
            <h2 style="color:#2c3e50;">Tableau de Bord</h2>
            <div style="background:#f8f9fa; padding:15px; border-radius:10px;">
                <p>👤 <strong>{st.session_state.get('full_name',
st.session_state.get('username', 'Inconnu'))}</strong></p>
                <p style="color:{'#e74c3c' if st.session_state.get('role') == 'admin' else '#2ecc50'}; font-weight:bold;">
                    {st.session_state.get('role', 'user').upper()}</p>
                </p>
            </div>
        </div>
    """", unsafe_allow_html=True)

    # Bouton de déconnexion
    if st.button("Logout Se déconnecter", use_container_width=True,
key="logout_btn"):
        for key in list(st.session_state.keys()):
            del st.session_state[key]
        st.rerun()

    st.markdown("---")

    # Section administration
    admin_section()

    # Menu de navigation
    st.markdown("### 📁 Navigation")
    page_options = ["Tableau de bord", "Gestion PDF"]
    selected_page = st.radio("", page_options,
label_visibility="collapsed")

    # Chargement des données (une seule fois)
    df = load_data()

    if df.empty:

```

```

        st.warning("Aucune donnée disponible pour l'analyse")
        return

    # Contenu principal conditionnel
    if selected_page == "Tableau de bord":
        display_dashboard_content(df)
    else:
        display_pdf_tools(df)
        display_export_section(df)    # Ajoutez cette ligne


def display_dashboard_content(df):
    """Affiche le contenu principal du dashboard avec gestion des erreurs"""
    if df.empty:
        st.warning("Aucune donnée disponible pour l'analyse")
        return

    # Vérification et renommage des colonnes
    if 'nom_x' in df.columns and 'nom_y' in df.columns:
        df = df.rename(columns={'nom_x': 'produit', 'nom_y': 'client'})
    elif 'nom_x' in df.columns:
        df = df.rename(columns={'nom_x': 'produit'})

    # Création de la colonne ca_cfa si elle n'existe pas
    if 'chiffre_affaires' in df.columns:
        df['ca_cfa'] = df['chiffre_affaires']    # Utilisation du même champ
    si la devise est déjà en FCFA
    else:
        st.error("Colonne 'chiffre_affaires' manquante - impossible de
        calculer le CA")
        return

    # Section des métriques
    st.write("📈 Analyse des Ventes")

try:
    # Calcul des indicateurs clés
    total_ca = df['ca_cfa'].sum()
    avg_ca = df['ca_cfa'].mean()
    nb_transactions = len(df)

```

```

    col1, col2, col3 = st.columns(3)
    col1.metric("CA Total", f"{total_ca:,.0f} FCFA")
    col2.metric("CA Moyen", f"{avg_ca:,.0f} FCFA")
    col3.metric("Nombre de ventes", nb_transactions)

except Exception as e:
    st.error(f"Erreur dans l'affichage des métriques: {str(e)}")

# Section des visualisations
tab1, tab2, tab3 = st.tabs(["Répartition", "Évolution", "Comparaison"])

with tab1:
    try:
        if 'produit' in df.columns:
            st.bar_chart(df.groupby('produit')['ca_cfa'].sum())
        else:
            st.error("Colonne 'produit' manquante pour la répartition")
    except Exception as e:
        st.error(f"Erreur dans le graphique de répartition: {str(e)}")

with tab2:
    try:
        if 'mois' in df.columns:
            st.line_chart(df.groupby('mois')['ca_cfa'].sum())
        else:
            st.error(f"Erreur dans le graphique d'évolution: {str(e)}")
    except Exception as e:
        st.error(f"Erreur dans le graphique d'évolution: {str(e)}")

with tab3:
    try:
        # Ajoutez ici vos visualisations de comparaison
        pass
    except Exception as e:
        st.error(f"Erreur dans les comparaisons: {str(e)}")

# Section des données détaillées
st.write("👁️ Données Détailées")
try:
    display_data_table(df)
except Exception as e:
    st.error(f"Erreur dans l'affichage des données: {str(e)}")

```

```

def display_data_table(df):
    """Affiche le tableau de données avec filtres"""
    st.write("👁️ Filtres")

    # Vérification des colonnes avant filtrage
    available_columns = df.columns.tolist()

    col1, col2 = st.columns(2)

    with col1:
        if 'produit' in available_columns:
            produits = df['produit'].unique()
            sel_produit = st.multiselect("Produits", produits,
default=produits)
        else:
            st.warning("Colonne 'produit' non disponible")

    with col2:
        if 'ca_cfa' in available_columns:
            min_ca = st.slider("CA Minimum (FCFA)", 0,
int(df['ca_cfa'].max()), 0)
        else:
            st.warning("Colonne 'ca_cfa' non disponible")

    # Application des filtres
    filtered_df = df.copy()

    if 'produit' in available_columns and len(sel_produit) > 0:
        filtered_df = filtered_df[filtered_df['produit'].isin(sel_produit)]

    if 'ca_cfa' in available_columns:
        filtered_df = filtered_df[filtered_df['ca_cfa'] >= min_ca]

    st.dataframe(filtered_df)

def display_pdf_tools(df):
    """Interface de génération de rapports PDF"""
    st.header("📄 Outils de Rapport PDF")

```

```

with st.expander("⚙️ Paramètres du rapport", expanded=True):
    # Paramètres personnalisables
    report_title = st.text_input("Titre du rapport", "Rapport d'Analyse
Commerciale")
    include_details = st.checkbox("Inclure les détails complets", True)
    include_charts = st.checkbox("Inclure les graphiques", True)

    # Bouton de génération
    if st.button("🖨 Générer le PDF", type="primary"):
        with st.spinner("Création du rapport..."):
            # Génération du PDF
            # Initialisez d'abord l'instance
            report_gen = ReportGenerator()

            # Puis utilisez la nouvelle méthode
            pdf_buffer = report_gen.generate_report(
                df=df,
                title="Mon Rapport Personnalisé",
                report_type="full" # ou "summary" pour le résumé
            )

            if pdf_buffer:
                st.success("Rapport généré avec succès!")
            else:
                st.error("Erreur lors de la génération du rapport")

def display_export_section(df):
    """Affiche la section d'export des données"""
    st.write("## 📥 Export des Données")

```

```

with st.expander("Options d'Export", expanded=True):
    col1, col2 = st.columns([1, 2])

    with col1:
        export_format = st.radio(
            "Format d'export:",
            ["CSV", "Excel", "JSON"],
            index=0,
            horizontal=True
        )

    with col2:
        export_name = st.text_input(
            "Nom du fichier (sans extension)",
            value="export_ventes",
            help="Choisissez un nom pour votre fichier exporté"
        )

if st.button("动生成 l'Export", type="primary"):
    try:
        if export_format == "CSV":
            data = df.to_csv(index=False)
            file_ext = "csv"
            mime_type = "text/csv"

        elif export_format == "Excel":
            output = io.BytesIO()
            with pd.ExcelWriter(output, engine='openpyxl') as writer:
                df.to_excel(writer, index=False,
                           sheet_name="Ventes")
                data = output.getvalue()
                file_ext = "xlsx"
                mime_type = "application/vnd.openxmlformats-
officedocument.spreadsheetml.sheet"

        else: # JSON
            data = df.to_json(indent=2, orient="records")
            file_ext = "json"
            mime_type = "application/json"

        # Bouton de téléchargement
        st.download_button(

```

```

        label=f"⬇ Télécharger {export_format}",
        data=data,
        file_name=f"{export_name}.{file_ext}",
        mime=mime_type,
        help=f"Cliquez pour sauvegarder au format
{export_format}"
    )

    st.success("Export généré avec succès!")

except Exception as e:
    st.error(f"Erreur lors de l'export: {str(e)}")

import io # Pour utiliser un buffer en mémoire

def display_export_options(df: pd.DataFrame, default_filename: str =
"export") -> None:
    """
    Affiche les options d'exportation et permet le téléchargement des
données.

    Args:
        df: DataFrame à exporter
        default_filename: Nom de fichier par défaut (sans extension)
    """
    st.write("### ⬇ Options d'exportation")

    # Options d'export
    col1, col2 = st.columns(2)
    with col1:
        export_format = st.selectbox(
            "Format d'export :",
            ["CSV", "Excel", "JSON"],
            help="Sélectionnez le format de fichier pour l'export"
        )

    with col2:
        filename = st.text_input(
            "Nom du fichier :",
            value=default_filename,

```

```

        help="Nom du fichier (sans extension)"
    ).strip()

# Bouton d'export
if st.button("⬇️ Générer l'export", type="primary"):
    try:
        if export_format == "CSV":
            export_data = df.to_csv(index=False)
            file_extension = "csv"
            mime_type = "text/csv"

        elif export_format == "Excel":
            excel_buffer = io.BytesIO()
            with pd.ExcelWriter(excel_buffer, engine='openpyxl') as writer:
                df.to_excel(writer, index=False, sheet_name="Données")
                export_data = excel_buffer.getvalue()
                file_extension = "xlsx"
                mime_type = "application/vnd.openxmlformats-officedocument.spreadsheetml.sheet"

        elif export_format == "JSON":
            export_data = df.to_json(indent=2, orient="records",
force_ascii=False)
            file_extension = "json"
            mime_type = "application/json"

        # Téléchargement
        st.download_button(
            label=f"⬇️ Télécharger {export_format}",
            data=export_data,
            file_name=f"{filename}.{file_extension}",
            mime=mime_type,
            help=f"Cliquez pour télécharger le fichier {export_format}"
        )

        st.success(f"Export {export_format} prêt !")

    except Exception as e:
        st.error(f"Erreur lors de l'export : {str(e)}")
        st.exception(e)

```

```

# ----- EXECUTION -----
if __name__ == "__main__":
    # Initialisation de la base de données
    init_auth_db()

    # Vérification de l'authentification
    if not st.session_state.get('authenticated'):
        login_page()
    else:
        main_dashboard()

```



6. Explication du Script de génération de rapport PDF

Ce script Python crée un rapport PDF professionnel à partir de données commerciales analysées. Il utilise la bibliothèque FPDF pour générer le document et pandas pour manipuler les données. Le rapport est structuré en plusieurs parties et inclut des métriques clés, des graphiques et un tableau détaillé des produits.

Le script commence par configurer un système de journalisation (logging) pour suivre son exécution. La classe PDFReport hérite de FPDF et personnalise l'apparence du document avec un en-tête contenant le titre et la date de génération, ainsi qu'un pied de page numéroté. Les sections du rapport sont clairement identifiées par des titres mis en valeur.

La fonction principale generer_rapport() orchestre toute la génération du document. Elle commence par vérifier la présence du fichier CSV contenant les données analysées dans le répertoire "output". Si le fichier est trouvé, les données sont chargées dans un DataFrame pandas avec vérification des colonnes obligatoires (produit, quantité et chiffre d'affaires).

Le rapport présente d'abord les indicateurs clés calculés à partir des données : le chiffre d'affaires total, le produit phare (celui ayant généré le plus de revenus) et la quantité totale vendue. Ces métriques sont formatées proprement avec des séparateurs de milliers pour une meilleure lisibilité.

Ensuite, le script intègre automatiquement les graphiques pré-générés (stockés sous forme d'images PNG dans le dossier "output") dans le PDF. Chaque graphique est placé sur une nouvelle page avec un titre descriptif dérivé du nom du fichier image.

La dernière section du rapport affiche un tableau détaillé listant tous les produits avec leurs quantités vendues et chiffre d'affaires correspondant. Le tableau utilise un style professionnel avec des en-têtes colorés et un alignement cohérent des données numériques à droite.

Le document final est sauvegardé dans le même répertoire "output" sous le nom "rapport_ventes.pdf". Tout au long du processus, le script enregistre des messages de journalisation pour faciliter le débogage en cas d'erreur. La gestion des exceptions permet de

remonter clairement les problèmes éventuels tout en fournissant des informations techniques via le système de logging.

```
from fpdf import FPDF
import pandas as pd
from pathlib import Path
import logging
from datetime import datetime

# Configuration du logging
logging.basicConfig(level=logging.INFO, format='%(asctime)s - %(levelname)s - %(message)s')
logger = logging.getLogger(__name__)

class PDFReport(FPDF):
    def __init__(self):
        super().__init__()
        # Utilisation des polices core PDF standard
        self.set_font("helvetica", size=12)

    def header(self):
        """En-tête du document PDF"""
        self.set_font("helvetica", "B", 16)
        self.cell(0, 10, "Rapport d'Analyse des Ventes", new_x="LMARGIN",
new_y="NEXT", align="C")
        self.set_font("helvetica", size=10)
        self.cell(0, 10, f"Genere le {datetime.now().strftime('%d/%m/%Y %H:%M')}", new_x="LMARGIN", new_y="NEXT",
align="C")
        self.ln(10)

    def footer(self):
        """Pied de page du document PDF"""
        self.set_y(-15)
        self.set_font("helvetica", "I", 8)
        self.cell(0, 10, f"Page {self.page_no()}", align="C")

    def add_section_title(self, title):
        """Ajoute un titre de section"""
        self.set_font("helvetica", "B", 14)
```

```

        self.cell(0, 10, title, new_x="LMARGIN", new_y="NEXT")
        self.ln(5)

def generer_rapport():
    """Genere le rapport PDF a partir des donnees analysees"""
    try:
        # Configuration des chemins
        BASE_DIR = Path(__file__).parent
        OUTPUT_DIR = BASE_DIR.parent / "output"
        OUTPUT_DIR.mkdir(exist_ok=True)

        csv_path = OUTPUT_DIR / "top_produits.csv"
        logger.info(f"Recherche du fichier de donnees: {csv_path}")

        if not csv_path.exists():
            raise FileNotFoundError(f"Fichier {csv_path} introuvable")

        # Initialisation PDF
        pdf = PDFReport()
        pdf.add_page()

        # 1. Chargement des donnees
        df = pd.read_csv(csv_path)
        logger.info("Donnees chargees avec succes")

        # Verification des colonnes requises
        required_columns = {'produit', 'quantite', 'ca_cfa'}
        if not required_columns.issubset(df.columns):
            missing = required_columns - set(df.columns)
            raise ValueError(f"Colonnes manquantes: {missing}. Colonnes disponibles: {list(df.columns)}")

        # 2. Metriques principales
        pdf.add_section_title("Resultats Cles")
        pdf.set_font("helvetica", size=12)

        # Calcul des metriques
        ca_total = df['ca_cfa'].sum()
        produit_phare = df.loc[df['ca_cfa'].idxmax(), 'produit']
        quantite_totale = df['quantite'].sum()

```

```

# Utilisation de caracteres simples
texte_metriques = (
    f"- Chiffre d'affaires total : {ca_total:.0f} FCFA\n"
    f"- Produit phare : {produit_phare}\n"
    f"- Quantite totale vendue : {quantite_totale:.0f} unites"
)
pdf.multi_cell(0, 10, texte_metriques)

# 3. Insertion des graphiques
images = ["ca_produits_cfa.png", "repartition_ca.png"]
for img in images:
    img_path = OUTPUT_DIR / img
    if img_path.exists():
        pdf.add_page()
        pdf.add_section_title(img.replace(".png", "").replace("_",
" ").title())
        pdf.image(str(img_path), x=10, y=30, w=180)
        logger.info(f"Image {img} ajoutee au rapport")
    else:
        logger.warning(f"Image {img} non trouvée")

# 4. Details des produits
pdf.add_page()
pdf.add_section_title("Details par Produit")

# En-tetes du tableau
pdf.set_fill_color(200, 220, 255)
pdf.set_font("helvetica", "B", 12)
col_widths = [80, 40, 60]

for col, width in zip(["Produit", "Quantite", "CA (FCFA)"],
col_widths):
    pdf.cell(width, 10, col, border=1, align="C", fill=True)
pdf.ln()

# Donnees du tableau
pdf.set_font("helvetica", size=10)
for _, row in df.iterrows():
    pdf.cell(col_widths[0], 10, row["produit"], border=1)
    pdf.cell(col_widths[1], 10, f"{row['quantite']}:", border=1,

```

```

    align="R")
        pdf.cell(col_widths[2], 10, f"{row['ca_cfa']:.0f}", border=1,
align="R")
    pdf.ln()

    # Sauvegarde du rapport
    rapport_path = OUTPUT_DIR / "rapport_ventes.pdf"
    pdf.output(str(rapport_path))
    logger.info(f"Rapport généré avec succès: {rapport_path}")

except Exception as e:
    logger.error(f"Erreur lors de la génération du rapport: {str(e)}",
exc_info=True)
    raise

if __name__ == "__main__":
    generer_rapport()

```

7. Explication du Script du ReportGenerator

Ce code implémente une classe ReportGenerator qui crée des rapports PDF professionnels à partir de données commerciales. La solution utilise plusieurs bibliothèques Python pour offrir une génération de documents riche et flexible.

La classe commence par définir une configuration de styles pour le document PDF. Ces styles incluent des paramètres typographiques (polices, tailles, couleurs) pour les titres, en-têtes et texte normal. L'initialisation des styles se fait via la méthode `_initialize_styles` qui combine des styles prédéfinis avec des styles personnalisés.

La fonction principale `generate_report` orchestre le processus complet :

1. Elle valide d'abord les données d'entrée pour s'assurer que les colonnes requises sont présentes
2. Crée un buffer mémoire pour stocker le PDF généré
3. Construit les différents éléments du rapport
4. Assemble le document final

Le rapport comprend plusieurs sections :

- Une page de titre avec la date de génération
- Un tableau récapitulatif avec les indicateurs clés (CA total, panier moyen, produit le plus vendu)
- Un graphique horizontal montrant la répartition du chiffre d'affaires par produit
- Pour les rapports complets (type 'full'), une section supplémentaire avec un tableau détaillé des ventes par produit

Les éléments visuels sont soignés :

- Le graphique est généré avec matplotlib dans un style professionnel (ggplot)
- Les tableaux utilisent des couleurs harmonieuses et des bordures discrètes
- Les titres sont hiérarchisés avec des styles différents

La classe inclut également une méthode `generate_sales_report` pour assurer la compatibilité avec d'anciennes versions, permettant une transition en douceur.

La gestion des erreurs est robuste :

- Validation des données en entrée
- Capture des exceptions pendant la génération
- Messages d'erreur clairs via l'interface Streamlit

Les particularités techniques notables incluent :

- Utilisation de `tempfile` pour la gestion temporaire des images
- Formatage professionnel des nombres (séparateurs de milliers)
- Style cohérent sur l'ensemble du document
- Optimisation pour l'intégration avec Streamlit

Ce générateur de rapports est conçu pour être à la fois flexible (acceptant différents types de données) et rigoureux (avec une validation stricte des entrées), tout en produisant des documents PDF esthétiques et informatifs.

```
import io
import tempfile
from datetime import datetime
from typing import Optional, List
import matplotlib.pyplot as plt
import pandas as pd
import streamlit as st
from reportlab.lib import colors
from reportlab.lib.pagesizes import A4
from reportlab.lib.styles import getSampleStyleSheet, ParagraphStyle
from reportlab.platypus import (
    SimpleDocTemplate, Paragraph, Spacer,
    Table, TableStyle, Image, PageBreak
)
from reportlab.lib.units import inch
from reportlab.lib.enums import TA_CENTER

@classmethod
def generate_sales_report(
    cls,
    df: pd.DataFrame,
    title: str = "Rapport d'Analyse des Ventes",
    include_details: bool = False,
    include_charts: bool = True
) -> Optional[io.BytesIO]:
    """Méthode de compatibilité avec l'ancienne interface"""
    instance = cls()
    return instance.generate_report(
        df=df,
        title=title,
        report_type="full" if include_details else "summary"
    )
class ReportGenerator:
    """Générateur de rapports PDF professionnels pour données commerciales"""

    # Configuration des styles
    _STYLES = {
        'title': {
            'fontName': 'Helvetica-Bold',
            'fontSize': 18,
```

```

        'alignment': TA_CENTER,
        'textColor': colors.HexColor("#003366"),
        'spaceAfter': 12
    },
    'header1': {
        'fontName': 'Helvetica-Bold',
        'fontSize': 14,
        'textColor': colors.HexColor("#0066CC"),
        'spaceBefore': 12,
        'spaceAfter': 6
    },
    'normal': {
        'fontName': 'Helvetica',
        'fontSize': 10,
        'leading': 12
    }
}

def __init__(self):
    self._styles = self._initialize_styles()

@staticmethod
def _initialize_styles() -> dict:
    """Initialise et retourne les styles PDF"""
    styles = getSampleStyleSheet()
    custom_styles = {
        name: ParagraphStyle(name, **attrs)
        for name, attrs in ReportGenerator._STYLES.items()
    }
    return {**styles.__dict__, **custom_styles}

def generate_report(
    self,
    df: pd.DataFrame,
    title: str = "Rapport d'Analyse Commerciale",
    report_type: str = "summary"
) -> Optional[io.BytesIO]:
    """
    Génère un rapport PDF à partir des données fournies
    """

    Args:

```

```

    df: DataFrame contenant les données
    title: Titre du rapport
    report_type: 'summary' ou 'full'

>Returns:
    Buffer PDF ou None en cas d'erreur
"""

try:
    self._validate_data(df)

    buffer = io.BytesIO()
    doc = SimpleDocTemplate(buffer, pagesize=A4)
    elements = self._build_report_elements(df, title, report_type)

    doc.build(elements)
    buffer.seek(0)
    return buffer

except Exception as e:
    st.error(f"Erreur de génération: {str(e)}")
    return None

def _validate_data(self, df: pd.DataFrame) -> None:
    """Valide la structure des données"""
    required_columns = {'produit', 'quantite', 'ca_cfa'}
    if not required_columns.issubset(df.columns):
        missing = required_columns - set(df.columns)
        raise ValueError(f"Colonnes requises manquantes: {missing}")

def _build_report_elements(
    self,
    df: pd.DataFrame,
    title: str,
    report_type: str
) -> List[Paragraph]:
    """Construit les éléments du rapport"""
    elements = [
        Paragraph(title, self._styles['Title']),
        Paragraph(f"Généré le {datetime.now().strftime('%d/%m/%Y')}"),
        self._styles['Normal']),
        Spacer(1, 24)

```

```

        ])

# Section Résumé
elements.extend([
    Paragraph("Résumé Analytique", self._styles['Heading1']),
    *self._create_summary_table(df)
])

# Graphique
chart_path = self._create_sales_chart(df)
if chart_path:
    elements.extend([
        Spacer(1, 12),
        Paragraph("Analyse Visuelle", self._styles['Heading1']),
        Image(chart_path, width=6 * inch, height=3 * inch)
    ])

# Section Détail (si full report)
if report_type == "full":
    elements.extend([
        PageBreak(),
        Paragraph("Détail des Ventes", self._styles['Heading1']),
        *self._create_detail_table(df)
    ])

return elements

def _create_summary_table(self, df: pd.DataFrame) -> List[Table]:
    """Crée le tableau de synthèse"""
    total_ca = df['ca_cfa'].sum()
    avg_basket = total_ca / df['quantite'].sum()

    data = [
        ["Indicateur", "Valeur"],
        ["CA Total", f"{total_ca:.2f} FCFA"],
        ["Panier Moyen", f"{avg_basket:.2f} FCFA"],
        ["Produit le Plus Vendu", df['produit'].mode()[0]]
    ]

    table = Table(data)
    table.setStyle(TableStyle([

```

```

        ('BACKGROUND', (0, 0), (-1, 0), colors.HexColor("#003366")),
        ('TEXTCOLOR', (0, 0), (-1, 0), colors.whitesmoke),
        ('ALIGN', (0, 0), (-1, -1), 'CENTER'),
        ('FONTCNAME', (0, 0), (-1, 0), 'Helvetica-Bold'),
        ('FONTSIZE', (0, 0), (-1, 0), 12),
        ('BACKGROUND', (0, 1), (-1, -1), colors.HexColor("#F5F5F5")),
        ('GRID', (0, 0), (-1, -1), 1, colors.lightgrey)
    ))
)

return [table, Spacer(1, 24)]
```

```

def _create_detail_table(self, df: pd.DataFrame) -> List[Table]:
    """Crée le tableau détaillé"""
    summary = df.groupby('produit').agg({
        'quantite': ['sum', 'mean'],
        'ca_cfa': 'sum'
    }).reset_index()

    summary.columns = ['Produit', 'Quantité Totale', 'Moyenne', 'CA
Total']

    summary['Moyenne'] = summary['Moyenne'].round(2)

    data = [summary.columns.tolist()] + summary.values.tolist()

    table = Table(data, repeatRows=1)
    table.setStyle(TableStyle([
        ('BACKGROUND', (0, 0), (-1, 0), colors.HexColor("#003366")),
        ('TEXTCOLOR', (0, 0), (-1, 0), colors.whitesmoke),
        ('ALIGN', (0, 0), (-1, -1), 'CENTER'),
        ('FONTCNAME', (0, 0), (-1, 0), 'Helvetica-Bold'),
        ('BACKGROUND', (0, 1), (-1, -1), colors.beige),
        ('GRID', (0, 0), (-1, -1), 1, colors.lightgrey),
        ('FONTSIZE', (0, 0), (-1, -1), 8)
    ]))

    return [table]
```

```

def _create_sales_chart(self, df: pd.DataFrame) -> Optional[str]:
    """Génère un graphique et retourne son chemin"""
    try:
        plt.style.use('ggplot')
```

```

fig, ax = plt.subplots(figsize=(10, 6))

sales = df.groupby('produit')['ca_cfa'].sum().sort_values()
bars = ax.barh(sales.index.astype(str), sales.values,
color='#4e79a7')

ax.bar_label(bars, fmt='%.0f FCFA', padding=5)
ax.set_title("Répartition du CA par Produit", pad=20)
ax.set_xlabel("Chiffre d'Affaires (FCFA)")

plt.tight_layout()

img_path = tempfile.mktemp(suffix='.png')
fig.savefig(img_path, dpi=300, bbox_inches='tight')
plt.close()

return img_path

except Exception as e:
    st.warning(f"Erreur de création du graphique: {str(e)}")
    return None

```

The screenshot displays a dark-themed web application interface. On the left, a sidebar titled "Tableau de Bord" contains a user profile section with "ADMIN" and a "Se déconnecter" button, and a navigation menu with "Tableau de bord" and "Gestion PDF" selected. The main area features two expandable sections: "Outils de Rapport PDF" and "Export des Données". The "Outils de Rapport PDF" section includes fields for "Titre du rapport" (set to "Rapport d'Analyse Commerciale"), checkboxes for "Inclure les détails complets" and "Inclure les graphiques", and a "Générer le PDF" button. The "Export des Données" section includes fields for "Format d'export" (set to "CSV"), "Nom du fichier (sans extension)" (set to "export_ventes"), and a "Générer l'Export" button.

Conclusion

L'analyse a permis d'identifier les produits les plus populaires, les clients réguliers, ainsi que les périodes les plus actives. Ces informations sont essentielles pour orienter les décisions commerciales : renforcer les stocks des produits à forte demande, récompenser la fidélité client, ou organiser des promotions en période creuse.

Ce TP a été très enrichissant et a montré l'intérêt de la science des données pour appuyer les stratégies marketing et commerciales.