Système de Prédiction de Performance des Employés

Documentation Détaillée

**Version 1.0**

# Table des Matières

1. Introduction

1.1 Objectif du Projet

1.2 Technologies Utilisées

1.3 Architecture Globale

2. Spécifications Techniques

2.1 Configuration Requise

2.2 Installation

2.3 Structure du Projet

3. Base de Données

3.1 Schéma de la Base de Données

3.2 Relations entre les Tables

3.3 Requêtes SQL Principales

4. Backend (Python)

4.1 Structure des Fichiers

4.2 Modules Principaux

4.3 Fonctions Clés

4.4 Gestion des Utilisateurs

5. Frontend (Streamlit)

5.1 Interface Utilisateur

5.2 Composants Principaux

5.3 Navigation

6. Fonctionnalités

6.1 Authentification

6.2 Tableau de Bord Admin

6.3 Tableau de Bord Manager

6.4 Tableau de Bord Employé

6.5 Système de Feedback

7. Machine Learning

7.1 Modèle de Prédiction

7.2 Features Utilisées

7.3 Entraînement et Évaluation

8. Sécurité

8.1 Authentification

8.2 Hachage des Mots de Passe

8.3 Contrôle d'Accès

9. Tests et Validation

9.1 Tests Unitaires

9.2 Tests d'Intégration

10. Cas d'Utilisation

10.1 Scénarios Administrateur

10.2 Scénarios Manager

10.3 Scénarios Employé

11. Maintenance et Évolution

11.1 Mises à Jour

11.2 Sauvegarde

11.3 Extensions Futures

12. Guide de Déploiement

12.1 Prérequis

12.2 Étapes d'Installation

12.3 Déploiement en Production

12.4 Configuration de Sécurité

12.5 Monitoring et Maintenance

13. Stack Technologique

14. Explication Détaillée du Code

# 1. Introduction

Le Système de Prédiction de Performance des Employés est une application web moderne conçue pour aider les entreprises   
 à suivre, analyser et prédire les performances de leurs employés. Cette solution complète combine une interface utilisateur intuitive avec   
 des analyses avancées et des capacités de machine learning.

## 1.1 Objectif du Projet

* Suivre les performances des employés en temps réel
* Prédire les tendances futures de performance
* Faciliter la communication via un système de feedback
* Fournir des analyses détaillées aux managers
* Automatiser l'évaluation des performances
* Identifier les domaines d'amélioration

## 1.2 Technologies Utilisées

### Backend

* Python 3.8+
* SQLite3 (Base de données)
* Pandas (Analyse de données)
* Scikit-learn (Machine Learning)

### Frontend

* Streamlit (Interface utilisateur)
* Plotly (Visualisations)
* Streamlit-extras (Composants additionnels)

### Sécurité

* Hashlib (Hachage des mots de passe)
* Session management
* SQLite3 (Sécurité des données)

## Code Source

### main.py - Point d'entrée de l'application

import streamlit as st  
from admin import admin\_dashboard, manager\_dashboard  
from employee import employee\_dashboard  
import sqlite3  
import hashlib  
  
# Initialize session state  
if 'logged\_in' not in st.session\_state:  
 st.session\_state['logged\_in'] = False  
  
def check\_password(username, password):  
 conn = sqlite3.connect('database/app.db')  
 cursor = conn.cursor()  
   
 # Hash the password for comparison  
 hashed\_password = hashlib.sha256(password.encode()).hexdigest()  
   
 cursor.execute("""  
 SELECT id, role FROM users   
 WHERE username = ? AND password = ?  
 """, (username, hashed\_password))  
   
 result = cursor.fetchone()  
 conn.close()  
   
 return result  
  
def login():  
 st.title("🔐 Login")  
   
 username = st.text\_input("Username")  
 password = st.text\_input("Password", type="password")  
   
 if st.button("Login"):  
 result = check\_password(username, password)  
 if result:  
 user\_id, role = result  
 st.session\_state['logged\_in'] = True  
 st.session\_state['user'] = (user\_id, role)  
 st.success("Logged in successfully!")  
 st.rerun()  
 else:  
 st.error("Invalid username or password")  
  
def main():  
 if not st.session\_state['logged\_in']:  
 login()  
 else:  
 # Show logout button in sidebar  
 if st.sidebar.button("Logout"):  
 st.session\_state['logged\_in'] = False  
 st.rerun()  
   
 # Get user role  
 role = st.session\_state['user'][1]  
   
 if role == 'admin':  
 # Admin can switch between Admin and Manager dashboards  
 dashboard = st.selectbox("Select Dashboard", ["Admin Dashboard", "Manager Dashboard"])  
 if dashboard == "Admin Dashboard":  
 admin\_dashboard()  
 else:  
 manager\_dashboard()  
 elif role == 'employee':  
 employee\_dashboard()  
 elif role == 'manager':  
 manager\_dashboard()  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 main()

### admin.py - Fonctionnalités administrateur

import streamlit as st  
import pandas as pd  
import joblib  
from sqlite3 import connect  
import hashlib # Import pour le hachage des mots de passe  
import sqlite3  
import plotly.express as px  
import plotly.graph\_objects as go  
from datetime import datetime, timedelta  
from sklearn.preprocessing import StandardScaler  
from sklearn.ensemble import RandomForestRegressor  
import seaborn as sns  
import matplotlib.pyplot as plt  
  
DB\_PATH = "database/app.db"  
  
  
# Fonction de hachage du mot de passe  
def hash\_password(password):  
 return hashlib.sha256(password.encode()).hexdigest()  
  
  
# Charger les modèles enregistrés  
linear\_model = joblib.load('ml\_models/linear\_model.pkl')  
rf\_model = joblib.load('ml\_models/rf\_model.pkl')  
xgb\_model = joblib.load('ml\_models/xgb\_model.pkl')  
  
  
def load\_data():  
 conn = sqlite3.connect(DB\_PATH)  
 df = pd.read\_sql\_query("SELECT \* FROM performance\_data", conn)  
 return df  
  
  
def predict\_performance(model, tasks\_completed, hours\_worked, feedback\_score, punctuality\_rate, employee\_id):  
 input\_data = pd.DataFrame({  
 "tasks\_completed": [tasks\_completed],  
 "hours\_worked": [hours\_worked],  
 "feedback\_score": [feedback\_score],  
 "punctuality\_rate": [punctuality\_rate]  
 })  
 prediction = model.predict(input\_data)  
   
 # Store prediction in history  
 conn = sqlite3.connect(DB\_PATH)  
 cursor = conn.cursor()  
 cursor.execute(  
 "INSERT INTO performance\_history (employee\_id, prediction\_date, performance\_score) VALUES (?, datetime('now'), ?)",  
 (employee\_id, prediction[0])  
 )  
 conn.commit()  
 conn.close()  
   
 return prediction[0]  
  
  
def add\_user(username, password, role, manager\_id=None, name=None, department=None):  
 conn = sqlite3.connect(DB\_PATH)  
 cursor = conn.cursor()  
 try:  
 # Vérifier si l'utilisateur existe déjà  
 cursor.execute("SELECT \* FROM users WHERE username = ?", (username,))  
 existing\_user = cursor.fetchone()  
 if existing\_user:  
 print(f"User {username} already exists.")  
 return False  
  
 # Hacher le mot de passe avant de l'insérer dans la base de données  
 hashed\_password = hash\_password(password)  
  
 # Insérer un utilisateur dans la table 'users'  
 cursor.execute("INSERT INTO users (username, password, role) VALUES (?, ?, ?)",  
 (username, hashed\_password, role))  
 conn.commit()  
  
 # Si l'utilisateur est un employé, l'ajouter à la table 'employees'  
 if role == "employee":  
 user\_id = cursor.lastrowid # Obtenir l'ID du nouvel utilisateur  
 if name and department and manager\_id:  
 cursor.execute("INSERT INTO employees (user\_id, name, department, manager\_id) VALUES (?, ?, ?, ?)",  
 (user\_id, name, department, manager\_id))  
 conn.commit()  
 else:  
 raise ValueError("Name, department, and manager are required for an employee.")  
  
 return True  
 except sqlite3.IntegrityError as e:  
 print(f"IntegrityError: {e}")  
 return False  
 except sqlite3.OperationalError as e:  
 print(f"OperationalError: {e}")  
 return False  
 except Exception as e:  
 print(f"Error: {e}")  
 return False  
 finally:  
 conn.close() # Toujours fermer la connexion à la base de données  
  
  
def load\_team\_data():  
 """Load all team data from database"""  
 conn = connect(DB\_PATH)  
 query = """  
 SELECT   
 e.user\_id,  
 e.department,  
 u.username,  
 ph.prediction\_date,  
 ph.performance\_score,  
 COUNT(ph.id) OVER (PARTITION BY e.user\_id) as total\_evaluations,  
 AVG(ph.performance\_score) OVER (PARTITION BY e.department) as dept\_avg  
 FROM employees e  
 JOIN users u ON e.user\_id = u.id  
 LEFT JOIN performance\_history ph ON ph.employee\_id = u.id  
 WHERE u.role = 'employee'  
 """  
 df = pd.read\_sql\_query(query, conn)  
 conn.close()  
 return df  
  
  
def predict\_future\_performance(employee\_id, history\_days=30, future\_days=7):  
 """Predict future performance using ML model"""  
 conn = connect(DB\_PATH)  
   
 # Get historical data  
 query = """  
 SELECT   
 prediction\_date,  
 performance\_score,  
 CAST(  
 (JulianDay(prediction\_date) - JulianDay(  
 (SELECT MIN(prediction\_date) FROM performance\_history WHERE employee\_id = ?)  
 )) AS INTEGER  
 ) as day\_number  
 FROM performance\_history   
 WHERE employee\_id = ?  
 ORDER BY prediction\_date  
 """  
 df = pd.read\_sql\_query(query, conn, params=(employee\_id, employee\_id))  
 conn.close()  
   
 if len(df) < 5: # Need minimum data points  
 return None  
   
 # Prepare features  
 df['prediction\_date'] = pd.to\_datetime(df['prediction\_date'])  
 df['dayofweek'] = df['prediction\_date'].dt.dayofweek  
 df['month'] = df['prediction\_date'].dt.month  
   
 # Create training data  
 X = df[['day\_number', 'dayofweek', 'month']].values  
 y = df['performance\_score'].values  
   
 # Train model  
 model = RandomForestRegressor(n\_estimators=100, random\_state=42)  
 model.fit(X, y)  
   
 # Prepare future dates  
 last\_date = df['prediction\_date'].max()  
 future\_dates = pd.date\_range(last\_date + timedelta(days=1), periods=future\_days)  
 future\_X = pd.DataFrame({  
 'day\_number': range(len(df), len(df) + future\_days),  
 'dayofweek': future\_dates.dayofweek,  
 'month': future\_dates.month  
 })  
   
 # Make predictions  
 predictions = model.predict(future\_X)  
   
 return pd.DataFrame({  
 'date': future\_dates,  
 'predicted\_score': predictions  
 })  
  
  
def get\_team\_insights(df):  
 """Generate team performance insights"""  
 insights = []  
   
 # Top performers  
 top\_performers = df.groupby('username')['performance\_score'].mean().nlargest(3)  
 insights.append(("🌟 Top Performers", top\_performers))  
   
 # Most improved  
 df['date'] = pd.to\_datetime(df['prediction\_date'])  
 recent\_scores = df[df['date'] >= df['date'].max() - timedelta(days=7)]  
 improved = recent\_scores.groupby('username')['performance\_score'].mean() - \  
 df.groupby('username')['performance\_score'].mean()  
 most\_improved = improved.nlargest(3)  
 insights.append(("📈 Most Improved (Last 7 Days)", most\_improved))  
   
 # Department performance  
 dept\_perf = df.groupby('department')['performance\_score'].agg(['mean', 'std']).round(2)  
 insights.append(("🏢 Department Performance", dept\_perf))  
   
 return insights  
  
  
def admin\_dashboard():  
 st.subheader("📊 Gestion des utilisateurs")  
  
 conn = sqlite3.connect(DB\_PATH)  
 cursor = conn.cursor()  
  
 # --- Filtrage ---  
 st.markdown("### 🎯 Filtres")  
 role\_filter = st.selectbox("Filtrer par rôle", ["Tous", "admin", "manager", "employee"])  
  
 cursor.execute(  
 "SELECT DISTINCT m.username FROM users u JOIN employees e ON u.id = e.user\_id JOIN users m ON e.manager\_id = m.id WHERE u.role = 'employee'")  
 managers\_list = [row[0] for row in cursor.fetchall()]  
 manager\_filter = st.selectbox("Filtrer par manager", ["Tous"] + managers\_list)  
  
 # --- Requête principale ---  
 query = """  
 SELECT u.id, u.username, u.role, e.name, e.department, m.username AS manager\_name  
 FROM users u  
 LEFT JOIN employees e ON u.id = e.user\_id  
 LEFT JOIN users m ON e.manager\_id = m.id  
 """  
 df\_users = pd.read\_sql\_query(query, conn)  
  
 if role\_filter != "Tous":  
 df\_users = df\_users[df\_users["role"] == role\_filter]  
 if manager\_filter != "Tous":  
 df\_users = df\_users[df\_users["manager\_name"] == manager\_filter]  
  
 st.dataframe(df\_users)  
 # --- Export CSV ---  
 csv = df\_users.to\_csv(index=False).encode('utf-8')  
 st.download\_button(  
 label="📥 Exporter les utilisateurs en CSV",  
 data=csv,  
 file\_name='utilisateurs.csv',  
 mime='text/csv'  
 )  
 # --- Modification et suppression ---  
 st.subheader("✏️ Modifier / Supprimer un utilisateur")  
 user\_ids = df\_users['id'].tolist()  
 if not user\_ids:  
 st.info("Aucun utilisateur trouvé avec ces filtres.")  
 return  
  
 selected\_user\_id = st.selectbox("Sélectionner un utilisateur", user\_ids)  
 selected\_user = df\_users[df\_users['id'] == selected\_user\_id].iloc[0]  
  
 with st.form("edit\_user\_form"):  
 username = st.text\_input("Nom d'utilisateur", value=selected\_user["username"])  
 role = st.selectbox("Rôle", ["admin", "manager", "employee"],  
 index=["admin", "manager", "employee"].index(selected\_user["role"]))  
 new\_password = st.text\_input("Nouveau mot de passe (laisse vide pour ne pas changer)", type="password")  
  
 name, department, manager\_id = None, None, None  
 if role == "employee":  
 name = st.text\_input("Nom complet", value=selected\_user["name"] or "")  
 department = st.text\_input("Département", value=selected\_user["department"] or "")  
 cursor.execute("SELECT id, username FROM users WHERE role = 'manager'")  
 managers = cursor.fetchall()  
 manager\_id = st.selectbox("Manager", [m[0] for m in managers],  
 format\_func=lambda x: [m[1] for m in managers if m[0] == x][0],  
 index=next(  
 (i for i, m in enumerate(managers) if m[1] == selected\_user["manager\_name"]),  
 0) if managers else 0)  
  
 col1, col2 = st.columns(2)  
 with col1:  
 update\_btn = st.form\_submit\_button("Mettre à jour")  
 with col2:  
 delete\_btn = st.form\_submit\_button("Supprimer")  
  
 if update\_btn:  
 try:  
 cursor.execute("UPDATE users SET username = ?, role = ? WHERE id = ?",  
 (username, role, selected\_user\_id))  
 if new\_password:  
 hashed\_pwd = hash\_password(new\_password)  
 cursor.execute("UPDATE users SET password = ? WHERE id = ?", (hashed\_pwd, selected\_user\_id))  
  
 if role == "employee":  
 cursor.execute("SELECT \* FROM employees WHERE user\_id = ?", (selected\_user\_id,))  
 if cursor.fetchone():  
 cursor.execute("""  
 UPDATE employees  
 SET name = ?, department = ?, manager\_id = ?  
 WHERE user\_id = ?  
 """, (name, department, manager\_id, selected\_user\_id))  
 else:  
 cursor.execute("""  
 INSERT INTO employees (user\_id, name, department, manager\_id)  
 VALUES (?, ?, ?, ?)  
 """, (selected\_user\_id, name, department, manager\_id))  
 else:  
 cursor.execute("DELETE FROM employees WHERE user\_id = ?", (selected\_user\_id,))  
 conn.commit()  
 st.success("✅ Utilisateur mis à jour.")  
 except Exception as e:  
 st.error(f"⚠️ Erreur lors de la mise à jour : {e}")  
  
 if delete\_btn:  
 try:  
 cursor.execute("DELETE FROM employees WHERE user\_id = ?", (selected\_user\_id,))  
 cursor.execute("DELETE FROM users WHERE id = ?", (selected\_user\_id,))  
 conn.commit()  
 st.success("🗑️ Utilisateur supprimé.")  
 except Exception as e:  
 st.error(f"⚠️ Erreur lors de la suppression : {e}")  
  
 # --- Ajout d’un utilisateur ---  
 st.subheader("➕ Ajouter un utilisateur")  
 with st.form("add\_user\_form"):  
 new\_username = st.text\_input("Username")  
 new\_password = st.text\_input("Password", type="password")  
 new\_role = st.selectbox("Role", ["admin", "manager", "employee"])  
  
 name, department, manager\_id = None, None, None  
 if new\_role == "employee":  
 name = st.text\_input("Nom de l'employé")  
 department = st.text\_input("Département")  
 cursor.execute("SELECT id, username FROM users WHERE role = 'manager'")  
 managers = cursor.fetchall()  
 if managers:  
 manager\_id = st.selectbox("Manager", [m[0] for m in managers],  
 format\_func=lambda x: [m[1] for m in managers if m[0] == x][0])  
 else:  
 st.warning("Aucun manager trouvé. Créez un manager d'abord.")  
  
 create\_btn = st.form\_submit\_button("Créer l'utilisateur")  
 if create\_btn:  
 if new\_username and new\_password:  
 if new\_role == "employee":  
 if not name or not department or not manager\_id:  
 st.warning("Remplissez tous les champs pour l'employé.")  
 else:  
 if add\_user(new\_username, new\_password, new\_role, manager\_id, name, department):  
 st.success(f"✅ Utilisateur '{new\_username}' créé.")  
 else:  
 st.error("⚠️ Erreur lors de la création.")  
 else:  
 if add\_user(new\_username, new\_password, new\_role):  
 st.success(f"✅ Utilisateur '{new\_username}' créé.")  
 else:  
 st.error("⚠️ Erreur lors de la création.")  
 else:  
 st.warning("Tous les champs sont requis.")  
  
 conn.close()  
  
  
def get\_team\_feedback(department=None):  
 conn = sqlite3.connect('database/app.db')  
   
 query = """  
 SELECT   
 u.username,  
 f.feedback\_date,  
 f.feedback\_text,  
 f.category,  
 f.satisfaction\_level,  
 u.department  
 FROM feedback f  
 JOIN users u ON f.employee\_id = u.id  
 """  
   
 if department:  
 query += " WHERE u.department = ?"  
 df = pd.read\_sql\_query(query, conn, params=(department,))  
 else:  
 df = pd.read\_sql\_query(query, conn)  
   
 conn.close()  
 return df  
  
def display\_feedback\_analytics(feedback\_df):  
 if feedback\_df.empty:  
 st.info("Aucun feedback disponible pour le moment.")  
 return  
   
 st.subheader("📊 Analyse des Feedbacks")  
   
 # Convert dates  
 feedback\_df['feedback\_date'] = pd.to\_datetime(feedback\_df['feedback\_date'])  
   
 # Satisfaction moyenne par département  
 col1, col2 = st.columns(2)  
   
 with col1:  
 avg\_satisfaction = feedback\_df.groupby('department')['satisfaction\_level'].mean()  
 fig = px.bar(  
 avg\_satisfaction,  
 title="Satisfaction Moyenne par Département",  
 labels={'value': 'Niveau de Satisfaction', 'department': 'Département'}  
 )  
 st.plotly\_chart(fig)  
   
 with col2:  
 category\_counts = feedback\_df['category'].value\_counts()  
 fig = px.pie(  
 values=category\_counts.values,  
 names=category\_counts.index,  
 title="Distribution des Catégories de Feedback"  
 )  
 st.plotly\_chart(fig)  
   
 # Évolution temporelle  
 st.subheader("📈 Évolution de la Satisfaction")  
 time\_trend = feedback\_df.groupby([pd.Grouper(key='feedback\_date', freq='W')])['satisfaction\_level'].mean()  
 fig = px.line(  
 x=time\_trend.index,  
 y=time\_trend.values,  
 labels={'x': 'Date', 'y': 'Satisfaction Moyenne'},  
 title="Tendance de Satisfaction (Moyenne Hebdomadaire)"  
 )  
 st.plotly\_chart(fig)  
   
 # Feedbacks récents  
 st.subheader("📝 Feedbacks Récents")  
 recent\_feedback = feedback\_df.sort\_values('feedback\_date', ascending=False).head(5)  
 for \_, row in recent\_feedback.iterrows():  
 with st.expander(f"{row['username']} - {row['category']} - {row['feedback\_date'].strftime('%Y-%m-%d %H:%M')}"):  
 st.write(row['feedback\_text'])  
 st.write(f"Satisfaction: {'⭐' \* row['satisfaction\_level']}")  
  
def manager\_dashboard():  
 st.title("👥 Tableau de Bord Manager")  
   
 # Get department selection  
 departments = get\_departments()  
 selected\_department = st.selectbox(  
 "Sélectionner un département",  
 ["Tous"] + departments  
 )  
   
 department = None if selected\_department == "Tous" else selected\_department  
   
 # Create tabs  
 tab1, tab2, tab3 = st.tabs(["Performance", "Feedback", "Prédictions"])  
   
 with tab1:  
 st.subheader("📊 Performance de l'Équipe")  
 team\_data = get\_team\_data(department)  
 display\_team\_performance(team\_data)  
   
 with tab2:  
 st.subheader("💭 Feedback de l'Équipe")  
 feedback\_data = get\_team\_feedback(department)  
 display\_feedback\_analytics(feedback\_data)  
   
 with tab3:  
 st.subheader("🔮 Prédictions")  
 display\_predictions(department)  
  
def main():  
 st.title("👋 Bienvenue dans l'application de gestion des performances")  
 st.write("Cette application permet aux administrateurs de gérer les performances des employés.")  
   
 # Afficher les dashboards  
 dashboard = st.selectbox("Sélectionner un dashboard", ["Administrateur", "Manager"])  
   
 if dashboard == "Administrateur":  
 admin\_dashboard()  
 elif dashboard == "Manager":  
 manager\_dashboard()  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 main()

### employee.py - Fonctionnalités employé

import streamlit as st  
import pandas as pd  
import sqlite3  
import plotly.express as px  
from datetime import datetime, timedelta  
import numpy as np  
from scipy import stats  
  
DB\_PATH = "database/app.db"  
  
def connect\_db():  
 return sqlite3.connect(DB\_PATH)  
  
def load\_data():  
 conn = connect\_db()  
 df = pd.read\_sql\_query("SELECT \* FROM performance\_data", conn)  
 return df  
  
  
def view\_performance\_history(employee\_id):  
 conn = connect\_db()  
 df = pd.read\_sql\_query(  
 """  
 SELECT prediction\_date, performance\_score   
 FROM performance\_history   
 WHERE employee\_id = ?  
 ORDER BY prediction\_date DESC  
 """,   
 conn,   
 params=(employee\_id,)  
 )  
 conn.close()  
   
 if not df.empty:  
 st.line\_chart(df.set\_index('prediction\_date'))  
 else:  
 st.write("No performance history available")  
   
 return df  
  
  
def get\_team\_average(employee\_id):  
 conn = connect\_db()  
   
 # Get employee's department  
 cursor = conn.cursor()  
 cursor.execute("""  
 SELECT e.department   
 FROM employees e   
 WHERE e.user\_id = ?  
 """, (employee\_id,))  
 department = cursor.fetchone()[0]  
   
 # Get team members in same department  
 team\_df = pd.read\_sql\_query(  
 """  
 SELECT ph.\*   
 FROM performance\_history ph  
 JOIN users u ON ph.employee\_id = u.id  
 JOIN employees e ON e.user\_id = u.id  
 WHERE e.department = ?  
 ORDER BY ph.prediction\_date  
 """,  
 conn,  
 params=(department,)  
 )  
 conn.close()  
   
 if not team\_df.empty:  
 return team\_df.groupby('prediction\_date')['performance\_score'].mean().reset\_index()  
 return pd.DataFrame()  
  
  
def calculate\_performance\_stats(scores):  
 """Calculate detailed performance statistics."""  
 if len(scores) < 2:  
 return None  
   
 stats\_dict = {  
 'mean': np.mean(scores),  
 'median': np.median(scores),  
 'std': np.std(scores),  
 'min': np.min(scores),  
 'max': np.max(scores),  
 'trend': np.polyfit(range(len(scores)), scores, 1)[0],  
 'consistency': 1 - (np.std(scores) / np.mean(scores)) # coefficient of variation  
 }  
   
 # Calculate percentile ranks  
 stats\_dict['percentile\_25'] = np.percentile(scores, 25)  
 stats\_dict['percentile\_75'] = np.percentile(scores, 75)  
   
 return stats\_dict  
  
def analyze\_performance\_trend(scores):  
 """Analyze the trend in performance scores."""  
 if len(scores) < 2:  
 return None  
   
 # Calculate moving average  
 window = min(5, len(scores) // 2)  
 moving\_avg = np.convolve(scores, np.ones(window)/window, mode='valid')  
   
 # Calculate trend  
 trend = np.polyfit(range(len(scores)), scores, 1)[0]  
   
 # Perform statistical test for trend significance  
 correlation = stats.pearsonr(range(len(scores)), scores)[0]  
   
 return {  
 'trend': trend,  
 'correlation': correlation,  
 'moving\_avg': moving\_avg  
 }  
  
def get\_performance\_insights(individual\_scores, team\_scores):  
 """Generate insights by comparing individual and team performance."""  
 if len(individual\_scores) < 2 or len(team\_scores) < 2:  
 return None  
   
 insights = []  
   
 # Compare means  
 ind\_mean = np.mean(individual\_scores)  
 team\_mean = np.mean(team\_scores)  
 diff = ind\_mean - team\_mean  
   
 # Perform t-test  
 t\_stat, p\_value = stats.ttest\_ind(individual\_scores, team\_scores)  
   
 # Calculate effect size (Cohen's d)  
 pooled\_std = np.sqrt((np.var(individual\_scores) + np.var(team\_scores)) / 2)  
 cohens\_d = diff / pooled\_std  
   
 return {  
 'mean\_difference': diff,  
 'significance': p\_value < 0.05,  
 'effect\_size': cohens\_d,  
 'p\_value': p\_value  
 }  
  
def submit\_feedback(employee\_id, feedback\_text, category, satisfaction\_level):  
 conn = connect\_db()  
 cursor = conn.cursor()  
   
 cursor.execute("""  
 INSERT INTO feedback (employee\_id, feedback\_text, category, satisfaction\_level)  
 VALUES (?, ?, ?, ?)  
 """, (employee\_id, feedback\_text, category, satisfaction\_level))  
   
 conn.commit()  
 conn.close()  
  
def get\_employee\_feedback\_history(employee\_id):  
 conn = connect\_db()  
 query = """  
 SELECT feedback\_date, feedback\_text, category, satisfaction\_level  
 FROM feedback  
 WHERE employee\_id = ?  
 ORDER BY feedback\_date DESC  
 """  
 df = pd.read\_sql\_query(query, conn, params=(employee\_id,))  
 conn.close()  
 return df  
  
def feedback\_section(employee\_id):  
 st.subheader("📝 Feedback")  
   
 # Feedback form  
 with st.form("feedback\_form"):  
 feedback\_text = st.text\_area("Votre feedback:", height=100)  
 col1, col2 = st.columns(2)  
   
 with col1:  
 category = st.selectbox(  
 "Catégorie:",  
 ["Général", "Environnement de Travail", "Management", "Projets", "Formation", "Autre"]  
 )  
   
 with col2:  
 satisfaction = st.slider(  
 "Niveau de satisfaction:",  
 min\_value=1,  
 max\_value=5,  
 value=3,  
 help="1 = Très insatisfait, 5 = Très satisfait"  
 )  
   
 submit = st.form\_submit\_button("Soumettre le Feedback")  
   
 if submit and feedback\_text:  
 submit\_feedback(employee\_id, feedback\_text, category, satisfaction)  
 st.success("Feedback soumis avec succès!")  
 st.rerun()  
   
 # Show feedback history  
 st.subheader("Historique des Feedbacks")  
 feedback\_history = get\_employee\_feedback\_history(employee\_id)  
   
 if not feedback\_history.empty:  
 # Convert feedback\_date to datetime if it's not already  
 feedback\_history['feedback\_date'] = pd.to\_datetime(feedback\_history['feedback\_date'])  
   
 # Display satisfaction trend  
 fig = px.line(  
 feedback\_history,  
 x='feedback\_date',  
 y='satisfaction\_level',  
 title='Évolution de la Satisfaction'  
 )  
 st.plotly\_chart(fig)  
   
 # Display feedback history in a table  
 for \_, row in feedback\_history.iterrows():  
 with st.expander(f"{row['category']} - {row['feedback\_date'].strftime('%Y-%m-%d %H:%M')}"):  
 st.write(row['feedback\_text'])  
 st.write(f"Satisfaction: {'⭐' \* row['satisfaction\_level']}")  
 else:  
 st.info("Aucun feedback soumis pour le moment.")  
  
def employee\_dashboard():  
 st.subheader("🙋‍♂️ Employee Dashboard")  
   
 # Get employee data  
 df = load\_data()  
 employee\_data = df[df["employee\_id"] == st.session\_state["user"][0]]  
   
 # Performance Overview  
 st.subheader("📊 Performance Overview")  
 col1, col2 = st.columns(2)  
   
 with col1:  
 if not employee\_data.empty:  
 latest\_score = employee\_data['performance\_score'].iloc[-1]  
 st.metric("Latest Performance Score", f"{latest\_score:.2f}/5.0")  
   
 # Performance History Section  
 st.subheader("📈 Your Performance History")  
 history = view\_performance\_history(st.session\_state["user"][0])  
   
 if not history.empty:  
 # Convert dates to datetime  
 history['prediction\_date'] = pd.to\_datetime(history['prediction\_date'])  
   
 # Add date filter  
 min\_date = history['prediction\_date'].min().date()  
 max\_date = history['prediction\_date'].max().date()  
   
 dates = st.date\_input(  
 "Select Date Range",  
 value=(min\_date, max\_date),  
 min\_value=min\_date,  
 max\_value=max\_date  
 )  
   
 if len(dates) == 2:  
 start\_date, end\_date = dates  
 filtered\_history = history[  
 (history['prediction\_date'].dt.date >= start\_date) &   
 (history['prediction\_date'].dt.date <= end\_date)  
 ]  
   
 # Calculate detailed statistics  
 scores = filtered\_history['performance\_score'].values  
 stats\_dict = calculate\_performance\_stats(scores)  
 trend\_analysis = analyze\_performance\_trend(scores)  
   
 if stats\_dict and trend\_analysis:  
 # Display Statistics  
 st.subheader("📊 Performance Statistics")  
 col1, col2, col3 = st.columns(3)  
   
 with col1:  
 st.metric("Average Score", f"{stats\_dict['mean']:.2f}")  
 st.metric("Consistency", f"{stats\_dict['consistency']\*100:.1f}%")  
   
 with col2:  
 st.metric("Highest Score", f"{stats\_dict['max']:.2f}")  
 st.metric("Lowest Score", f"{stats\_dict['min']:.2f}")  
   
 with col3:  
 trend\_icon = "↗️" if trend\_analysis['trend'] > 0 else "↘️"  
 st.metric("Score Trend", f"{trend\_icon} {abs(trend\_analysis['trend']):.3f}/day")  
 st.metric("Variability", f"±{stats\_dict['std']:.2f}")  
   
 # Performance Distribution  
 st.subheader("📊 Score Distribution")  
 fig\_col1, fig\_col2 = st.columns(2)  
   
 with fig\_col1:  
 # Create histogram  
 hist\_data = [scores]  
 group\_labels = ['Scores']  
 import plotly.figure\_factory as ff  
 fig1 = ff.create\_distplot(hist\_data, group\_labels, bin\_size=0.2)  
 fig1.update\_layout(title="Score Distribution")  
 st.plotly\_chart(fig1, use\_container\_width=True)  
   
 with fig\_col2:  
 # Create box plot  
 import plotly.graph\_objects as go  
 fig2 = go.Figure()  
 fig2.add\_trace(go.Box(y=scores, name="Performance Scores"))  
 fig2.update\_layout(title="Score Range Analysis")  
 st.plotly\_chart(fig2, use\_container\_width=True)  
   
 # Team Comparison with Enhanced Analysis  
 st.subheader("👥 Team Comparison")  
 team\_avg = get\_team\_average(st.session\_state["user"][0])  
   
 if not team\_avg.empty:  
 # Convert team dates to datetime  
 team\_avg['prediction\_date'] = pd.to\_datetime(team\_avg['prediction\_date'])  
 filtered\_team = team\_avg[  
 (team\_avg['prediction\_date'].dt.date >= start\_date) &   
 (team\_avg['prediction\_date'].dt.date <= end\_date)  
 ]  
   
 # Analyze individual vs team performance  
 team\_scores = filtered\_team['performance\_score'].values  
 performance\_insights = get\_performance\_insights(scores, team\_scores)  
   
 if performance\_insights:  
 # Display comparison insights  
 st.subheader("🎯 Performance Insights")  
   
 # Effect size interpretation  
 effect\_size = abs(performance\_insights['effect\_size'])  
 if effect\_size < 0.2:  
 effect\_text = "minimal"  
 elif effect\_size < 0.5:  
 effect\_text = "moderate"  
 else:  
 effect\_text = "substantial"  
   
 # Display insights based on statistical analysis  
 if performance\_insights['significance']:  
 if performance\_insights['mean\_difference'] > 0:  
 st.success(f"🌟 Your performance is significantly higher than the team average (p < {performance\_insights['p\_value']:.3f})")  
 st.info(f"The difference is {effect\_text} (Cohen's d = {effect\_size:.2f})")  
 else:  
 st.warning(f"📈 Your performance is significantly lower than the team average (p < {performance\_insights['p\_value']:.3f})")  
 st.info(f"The gap is {effect\_text} (Cohen's d = {effect\_size:.2f})")  
 else:  
 st.info("Your performance is statistically similar to the team average")  
   
 # Plot comparison  
 comparison = filtered\_history.merge(  
 filtered\_team,  
 on='prediction\_date',  
 suffixes=('\_individual', '\_team')  
 )  
   
 st.line\_chart(  
 comparison.set\_index('prediction\_date')[  
 ['performance\_score\_individual', 'performance\_score\_team']  
 ].rename(columns={  
 'performance\_score\_individual': 'Your Score',  
 'performance\_score\_team': 'Team Average'  
 }),  
 use\_container\_width=True  
 )  
 else:  
 st.info("No performance history available yet. Check back after your first evaluation!")  
   
 # Create tabs for different sections  
 tab1, tab2, tab3 = st.tabs(["Performance", "Feedback", "Statistiques"])  
   
 with tab1:  
 st.subheader("📈 Vos Performances")  
 view\_performance\_history(st.session\_state["user"][0])  
   
 with tab2:  
 feedback\_section(st.session\_state["user"][0])  
   
 with tab3:  
 st.subheader("📊 Statistiques")  
 # display\_employee\_stats(st.session\_state["user"][0])

## Schéma de la Base de Données

CREATE TABLE IF NOT EXISTS users (  
 id INTEGER PRIMARY KEY AUTOINCREMENT,  
 username TEXT UNIQUE NOT NULL,  
 password TEXT NOT NULL,  
 role TEXT CHECK(role IN ('admin', 'manager', 'employee')) NOT NULL  
);  
  
CREATE TABLE IF NOT EXISTS employees (  
 id INTEGER PRIMARY KEY AUTOINCREMENT,  
 user\_id INTEGER,  
 name TEXT,  
 department TEXT,  
 FOREIGN KEY (user\_id) REFERENCES users(id)  
);  
  
CREATE TABLE IF NOT EXISTS performance\_data (  
 id INTEGER PRIMARY KEY AUTOINCREMENT,  
 employee\_id INTEGER,  
 hours\_worked REAL,  
 tasks\_completed INTEGER,  
 feedback\_score REAL,  
 punctuality\_rate REAL,  
 performance\_score REAL,  
 date TEXT,  
 FOREIGN KEY (employee\_id) REFERENCES employees(id)  
);  
  
ALTER TABLE employees ADD COLUMN manager\_id INTEGER;  
ALTER TABLE performance\_data ADD COLUMN manager\_id INTEGER;

# 10. Cas d'Utilisation

## 10.1 Scénarios Administrateur

### Création d'un Compte Manager

#### Étapes:

* Se connecter avec les identifiants admin
* Accéder au tableau de bord admin
* Cliquer sur "Gestion des Utilisateurs"
* Sélectionner "Ajouter un Manager"
* Remplir les informations requises
* Valider la création du compte

#### Résultat Attendu:

Compte manager créé avec succès

### Analyse des Performances Globales

#### Étapes:

* Accéder au tableau de bord admin
* Consulter les métriques globales
* Analyser les tendances par département
* Exporter les rapports de performance

#### Résultat Attendu:

Vue complète des performances de l'entreprise

## 10.2 Scénarios Manager

### Suivi des Performances d'Équipe

#### Étapes:

* Se connecter en tant que manager
* Accéder au tableau de bord équipe
* Filtrer par département
* Analyser les tendances
* Identifier les points d'amélioration

#### Résultat Attendu:

Vue détaillée des performances de l'équipe

### Gestion des Feedbacks

#### Étapes:

* Accéder à la section feedback
* Consulter les feedbacks récents
* Analyser les tendances de satisfaction
* Répondre aux feedbacks critiques

#### Résultat Attendu:

Amélioration de la communication d'équipe

## 10.3 Scénarios Employé

### Consultation des Performances

#### Étapes:

* Se connecter à son compte
* Accéder au tableau de bord personnel
* Visualiser les scores de performance
* Consulter les prédictions futures

#### Résultat Attendu:

Compréhension claire de sa performance

### Soumission de Feedback

#### Étapes:

* Accéder à la section feedback
* Choisir une catégorie
* Rédiger le feedback
* Noter le niveau de satisfaction
* Soumettre le feedback

#### Résultat Attendu:

Feedback enregistré et visible par le management

# 7. Machine Learning

## 7.1 Modèle de Prédiction

Le système utilise un modèle Random Forest Regressor pour prédire les performances futures des employés.   
 Ce choix est motivé par la capacité du modèle à :

* Gérer des données non linéaires
* Prendre en compte de multiples features
* Résister au surapprentissage
* Fournir des prédictions robustes

## 7.2 Features Utilisées

### Temporelles

* Jour de la semaine
* Mois
* Trimestre
* Ancienneté de l'employé

### Performance

* Scores historiques
* Moyenne mobile
* Écart-type des performances
* Tendance récente

### Contextuelles

* Département
* Rôle
* Niveau d'expérience
* Formation suivie

# 12. Guide de Déploiement

## 12.1 Prérequis

### Système

* Python 3.8 ou supérieur
* pip (gestionnaire de paquets Python)
* Git
* Espace disque: minimum 500 MB

### Base de données

* SQLite3
* Permissions d'écriture pour le dossier database/

### Dépendances Python

* streamlit==1.24.0
* pandas==2.0.0
* numpy==1.24.0
* scikit-learn==1.2.0
* plotly==5.13.0
* python-docx==1.0.0

## 12.2 Étapes d'Installation

### 1. Cloner le Repository

```bash  
git clone https://github.com/votre-repo/PFA-main.git  
cd PFA-main  
```

### 2. Créer un Environnement Virtuel

# Windows  
python -m venv venv  
venv\Scripts\activate  
  
# Linux/MacOS  
python3 -m venv venv  
source venv/bin/activate

### 3. Installer les Dépendances

```bash  
pip install -r requirements.txt  
```

### 4. Initialiser la Base de Données

python scripts/init\_db.py  
python scripts/create\_admin.py # Crée le compte admin par défaut

### 5. Lancer l'Application

streamlit run main.py

## 12.3 Déploiement en Production

### Option 1: Déploiement avec Docker

#### Dockerfile

FROM python:3.8-slim  
  
WORKDIR /app  
COPY . .  
  
RUN pip install -r requirements.txt  
RUN python scripts/init\_db.py  
RUN python scripts/create\_admin.py  
  
EXPOSE 8501  
  
CMD ["streamlit", "run", "main.py"]

#### Build & Run

# Build l'image  
docker build -t employee-predictor .  
  
# Lancer le conteneur  
docker run -d -p 8501:8501 employee-predictor

### Option 2: Déploiement sur le Cloud

#### Heroku

* Créer un compte Heroku
* Installer Heroku CLI
* heroku login
* heroku create employee-predictor
* git push heroku main

#### Google Cloud Run

* Installer Google Cloud SDK
* gcloud init
* gcloud builds submit --tag gcr.io/PROJECT\_ID/employee-predictor
* gcloud run deploy --image gcr.io/PROJECT\_ID/employee-predictor

#### AWS Elastic Beanstalk

* Installer AWS CLI
* Configurer les credentials AWS
* Créer une application Elastic Beanstalk
* eb init -p python-3.8 employee-predictor
* eb create production

## 12.4 Configuration de Sécurité

### Variables d'Environnement

* SECRET\_KEY: Clé secrète pour les sessions
* DATABASE\_URL: URL de la base de données
* ADMIN\_PASSWORD: Mot de passe admin initial
* LOG\_LEVEL: Niveau de logging

### HTTPS

* Configurer un certificat SSL
* Rediriger HTTP vers HTTPS
* Configurer HSTS

### Base de Données

* Sauvegardes automatiques
* Chiffrement des données sensibles
* Rotation des logs

## 12.5 Monitoring et Maintenance

### Logs

* Configurer la rotation des logs
* Centraliser les logs avec ELK Stack
* Alertes sur erreurs critiques

### Métriques

* Temps de réponse API
* Utilisation mémoire/CPU
* Nombre d'utilisateurs actifs
* Taux d'erreurs

### Sauvegardes

* Backup quotidien de la base
* Rétention sur 30 jours
* Test de restauration mensuel

# 13. Stack Technologique

## 13.1 Langages de Programmation

### Python

* Version: 3.8+
* Rôle: Backend, logique métier, ML
* Utilisé pour: Traitement des données, API, prédictions

### SQL

* Version: SQLite3
* Rôle: Gestion des données
* Utilisé pour: Stockage, requêtes, relations

### HTML/CSS

* Via: Streamlit
* Rôle: Interface utilisateur
* Utilisé pour: Mise en page, style, composants

## 13.2 Bibliothèques et Frameworks

### Interface Utilisateur

#### Streamlit

* Version: 1.24.0
* Rôle: Framework web pour data apps
* Composants: st.write, st.plotly\_chart, st.form

### Analyse de Données

#### Pandas

* Version: 2.0.0
* Rôle: Manipulation de données
* Fonctions: DataFrame, groupby, merge

#### NumPy

* Version: 1.24.0
* Rôle: Calculs numériques
* Fonctions: array, mean, std

### Visualisation

#### Plotly

* Version: 5.13.0
* Rôle: Graphiques interactifs
* Types: line, bar, scatter, pie

### Machine Learning

#### Scikit-learn

* Version: 1.2.0
* Rôle: Modèles prédictifs
* Algorithmes: RandomForestRegressor, train\_test\_split

### Base de Données

#### SQLite3

* Version: Intégrée à Python
* Rôle: Base de données légère
* Fonctions: connect, execute, cursor

### Sécurité

#### Hashlib

* Version: Intégrée à Python
* Rôle: Cryptographie
* Algorithmes: SHA-256

# 14. Explication Détaillée du Code

## 14.1 main.py - Point d'Entrée

### Code Source:

import streamlit as st  
from admin import admin\_dashboard, manager\_dashboard  
from employee import employee\_dashboard  
import sqlite3  
import hashlib  
  
# Initialize session state  
if 'logged\_in' not in st.session\_state:  
 st.session\_state['logged\_in'] = False  
  
def check\_password(username, password):  
 conn = sqlite3.connect('database/app.db')  
 cursor = conn.cursor()  
   
 # Hash the password for comparison  
 hashed\_password = hashlib.sha256(password.encode()).hexdigest()  
   
 cursor.execute("""  
 SELECT id, role FROM users   
 WHERE username = ? AND password = ?  
 """, (username, hashed\_password))  
   
 result = cursor.fetchone()  
 conn.close()  
   
 return result  
  
def login():  
 st.title("🔐 Login")  
   
 username = st.text\_input("Username")  
 password = st.text\_input("Password", type="password")  
   
 if st.button("Login"):  
 result = check\_password(username, password)  
 if result:  
 user\_id, role = result  
 st.session\_state['logged\_in'] = True  
 st.session\_state['user'] = (user\_id, role)  
 st.success("Logged in successfully!")  
 st.rerun()  
 else:  
 st.error("Invalid username or password")  
  
def main():  
 if not st.session\_state['logged\_in']:  
 login()  
 else:  
 # Show logout button in sidebar  
 if st.sidebar.button("Logout"):  
 st.session\_state['logged\_in'] = False  
 st.rerun()  
   
 # Get user role  
 role = st.session\_state['user'][1]  
   
 if role == 'admin':  
 # Admin can switch between Admin and Manager dashboards  
 dashboard = st.selectbox("Select Dashboard", ["Admin Dashboard", "Manager Dashboard"])  
 if dashboard == "Admin Dashboard":  
 admin\_dashboard()  
 else:  
 manager\_dashboard()  
 elif role == 'employee':  
 employee\_dashboard()  
 elif role == 'manager':  
 manager\_dashboard()  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 main()

### Explication des Composants:

#### Imports

* streamlit: Framework web pour l'interface
* sqlite3: Connexion base de données
* hashlib: Hachage des mots de passe

#### Fonction check\_password

* Rôle: Vérifie les identifiants utilisateur
* Processus: Hash le mot de passe et compare avec la base
* Retour: Tuple (user\_id, role) si succès, None sinon

#### Fonction login

* Rôle: Gère l'interface de connexion
* Composants: Formulaire de login Streamlit
* Actions: Authentification et redirection

#### Fonction main

* Rôle: Point d'entrée de l'application
* Logique: Gestion des sessions et routage
* Affichage: Dashboard selon le rôle

## 14.2 admin.py - Fonctionnalités Admin

### Code Source:

import streamlit as st  
import pandas as pd  
import joblib  
from sqlite3 import connect  
import hashlib # Import pour le hachage des mots de passe  
import sqlite3  
import plotly.express as px  
import plotly.graph\_objects as go  
from datetime import datetime, timedelta  
from sklearn.preprocessing import StandardScaler  
from sklearn.ensemble import RandomForestRegressor  
import seaborn as sns  
import matplotlib.pyplot as plt  
  
DB\_PATH = "database/app.db"  
  
  
# Fonction de hachage du mot de passe  
def hash\_password(password):  
 return hashlib.sha256(password.encode()).hexdigest()  
  
  
# Charger les modèles enregistrés  
linear\_model = joblib.load('ml\_models/linear\_model.pkl')  
rf\_model = joblib.load('ml\_models/rf\_model.pkl')  
xgb\_model = joblib.load('ml\_models/xgb\_model.pkl')  
  
  
def load\_data():  
 conn = sqlite3.connect(DB\_PATH)  
 df = pd.read\_sql\_query("SELECT \* FROM performance\_data", conn)  
 return df  
  
  
def predict\_performance(model, tasks\_completed, hours\_worked, feedback\_score, punctuality\_rate, employee\_id):  
 input\_data = pd.DataFrame({  
 "tasks\_completed": [tasks\_completed],  
 "hours\_worked": [hours\_worked],  
 "feedback\_score": [feedback\_score],  
 "punctuality\_rate": [punctuality\_rate]  
 })  
 prediction = model.predict(input\_data)  
   
 # Store prediction in history  
 conn = sqlite3.connect(DB\_PATH)  
 cursor = conn.cursor()  
 cursor.execute(  
 "INSERT INTO performance\_history (employee\_id, prediction\_date, performance\_score) VALUES (?, datetime('now'), ?)",  
 (employee\_id, prediction[0])  
 )  
 conn.commit()  
 conn.close()  
   
 return prediction[0]  
  
  
def add\_user(username, password, role, manager\_id=None, name=None, department=None):  
 conn = sqlite3.connect(DB\_PATH)  
 cursor = conn.cursor()  
 try:  
 # Vérifier si l'utilisateur existe déjà  
 cursor.execute("SELECT \* FROM users WHERE username = ?", (username,))  
 existing\_user = cursor.fetchone()  
 if existing\_user:  
 print(f"User {username} already exists.")  
 return False  
  
 # Hacher le mot de passe avant de l'insérer dans la base de données  
 hashed\_password = hash\_password(password)  
  
 # Insérer un utilisateur dans la table 'users'  
 cursor.execute("INSERT INTO users (username, password, role) VALUES (?, ?, ?)",  
 (username, hashed\_password, role))  
 conn.commit()  
  
 # Si l'utilisateur est un employé, l'ajouter à la table 'employees'  
 if role == "employee":  
 user\_id = cursor.lastrowid # Obtenir l'ID du nouvel utilisateur  
 if name and department and manager\_id:  
 cursor.execute("INSERT INTO employees (user\_id, name, department, manager\_id) VALUES (?, ?, ?, ?)",  
 (user\_id, name, department, manager\_id))  
 conn.commit()  
 else:  
 raise ValueError("Name, department, and manager are required for an employee.")  
  
 return True  
 except sqlite3.IntegrityError as e:  
 print(f"IntegrityError: {e}")  
 return False  
 except sqlite3.OperationalError as e:  
 print(f"OperationalError: {e}")  
 return False  
 except Exception as e:  
 print(f"Error: {e}")  
 return False  
 finally:  
 conn.close() # Toujours fermer la connexion à la base de données  
  
  
def load\_team\_data():  
 """Load all team data from database"""  
 conn = connect(DB\_PATH)  
 query = """  
 SELECT   
 e.user\_id,  
 e.department,  
 u.username,  
 ph.prediction\_date,  
 ph.performance\_score,  
 COUNT(ph.id) OVER (PARTITION BY e.user\_id) as total\_evaluations,  
 AVG(ph.performance\_score) OVER (PARTITION BY e.department) as dept\_avg  
 FROM employees e  
 JOIN users u ON e.user\_id = u.id  
 LEFT JOIN performance\_history ph ON ph.employee\_id = u.id  
 WHERE u.role = 'employee'  
 """  
 df = pd.read\_sql\_query(query, conn)  
 conn.close()  
 return df  
  
  
def predict\_future\_performance(employee\_id, history\_days=30, future\_days=7):  
 """Predict future performance using ML model"""  
 conn = connect(DB\_PATH)  
   
 # Get historical data  
 query = """  
 SELECT   
 prediction\_date,  
 performance\_score,  
 CAST(  
 (JulianDay(prediction\_date) - JulianDay(  
 (SELECT MIN(prediction\_date) FROM performance\_history WHERE employee\_id = ?)  
 )) AS INTEGER  
 ) as day\_number  
 FROM performance\_history   
 WHERE employee\_id = ?  
 ORDER BY prediction\_date  
 """  
 df = pd.read\_sql\_query(query, conn, params=(employee\_id, employee\_id))  
 conn.close()  
   
 if len(df) < 5: # Need minimum data points  
 return None  
   
 # Prepare features  
 df['prediction\_date'] = pd.to\_datetime(df['prediction\_date'])  
 df['dayofweek'] = df['prediction\_date'].dt.dayofweek  
 df['month'] = df['prediction\_date'].dt.month  
   
 # Create training data  
 X = df[['day\_number', 'dayofweek', 'month']].values  
 y = df['performance\_score'].values  
   
 # Train model  
 model = RandomForestRegressor(n\_estimators=100, random\_state=42)  
 model.fit(X, y)  
   
 # Prepare future dates  
 last\_date = df['prediction\_date'].max()  
 future\_dates = pd.date\_range(last\_date + timedelta(days=1), periods=future\_days)  
 future\_X = pd.DataFrame({  
 'day\_number': range(len(df), len(df) + future\_days),  
 'dayofweek': future\_dates.dayofweek,  
 'month': future\_dates.month  
 })  
   
 # Make predictions  
 predictions = model.predict(future\_X)  
   
 return pd.DataFrame({  
 'date': future\_dates,  
 'predicted\_score': predictions  
 })  
  
  
def get\_team\_insights(df):  
 """Generate team performance insights"""  
 insights = []  
   
 # Top performers  
 top\_performers = df.groupby('username')['performance\_score'].mean().nlargest(3)  
 insights.append(("🌟 Top Performers", top\_performers))  
   
 # Most improved  
 df['date'] = pd.to\_datetime(df['prediction\_date'])  
 recent\_scores = df[df['date'] >= df['date'].max() - timedelta(days=7)]  
 improved = recent\_scores.groupby('username')['performance\_score'].mean() - \  
 df.groupby('username')['performance\_score'].mean()  
 most\_improved = improved.nlargest(3)  
 insights.append(("📈 Most Improved (Last 7 Days)", most\_improved))  
   
 # Department performance  
 dept\_perf = df.groupby('department')['performance\_score'].agg(['mean', 'std']).round(2)  
 insights.append(("🏢 Department Performance", dept\_perf))  
   
 return insights  
  
  
def admin\_dashboard():  
 st.subheader("📊 Gestion des utilisateurs")  
  
 conn = sqlite3.connect(DB\_PATH)  
 cursor = conn.cursor()  
  
 # --- Filtrage ---  
 st.markdown("### 🎯 Filtres")  
 role\_filter = st.selectbox("Filtrer par rôle", ["Tous", "admin", "manager", "employee"])  
  
 cursor.execute(  
 "SELECT DISTINCT m.username FROM users u JOIN employees e ON u.id = e.user\_id JOIN users m ON e.manager\_id = m.id WHERE u.role = 'employee'")  
 managers\_list = [row[0] for row in cursor.fetchall()]  
 manager\_filter = st.selectbox("Filtrer par manager", ["Tous"] + managers\_list)  
  
 # --- Requête principale ---  
 query = """  
 SELECT u.id, u.username, u.role, e.name, e.department, m.username AS manager\_name  
 FROM users u  
 LEFT JOIN employees e ON u.id = e.user\_id  
 LEFT JOIN users m ON e.manager\_id = m.id  
 """  
 df\_users = pd.read\_sql\_query(query, conn)  
  
 if role\_filter != "Tous":  
 df\_users = df\_users[df\_users["role"] == role\_filter]  
 if manager\_filter != "Tous":  
 df\_users = df\_users[df\_users["manager\_name"] == manager\_filter]  
  
 st.dataframe(df\_users)  
 # --- Export CSV ---  
 csv = df\_users.to\_csv(index=False).encode('utf-8')  
 st.download\_button(  
 label="📥 Exporter les utilisateurs en CSV",  
 data=csv,  
 file\_name='utilisateurs.csv',  
 mime='text/csv'  
 )  
 # --- Modification et suppression ---  
 st.subheader("✏️ Modifier / Supprimer un utilisateur")  
 user\_ids = df\_users['id'].tolist()  
 if not user\_ids:  
 st.info("Aucun utilisateur trouvé avec ces filtres.")  
 return  
  
 selected\_user\_id = st.selectbox("Sélectionner un utilisateur", user\_ids)  
 selected\_user = df\_users[df\_users['id'] == selected\_user\_id].iloc[0]  
  
 with st.form("edit\_user\_form"):  
 username = st.text\_input("Nom d'utilisateur", value=selected\_user["username"])  
 role = st.selectbox("Rôle", ["admin", "manager", "employee"],  
 index=["admin", "manager", "employee"].index(selected\_user["role"]))  
 new\_password = st.text\_input("Nouveau mot de passe (laisse vide pour ne pas changer)", type="password")  
  
 name, department, manager\_id = None, None, None  
 if role == "employee":  
 name = st.text\_input("Nom complet", value=selected\_user["name"] or "")  
 department = st.text\_input("Département", value=selected\_user["department"] or "")  
 cursor.execute("SELECT id, username FROM users WHERE role = 'manager'")  
 managers = cursor.fetchall()  
 manager\_id = st.selectbox("Manager", [m[0] for m in managers],  
 format\_func=lambda x: [m[1] for m in managers if m[0] == x][0],  
 index=next(  
 (i for i, m in enumerate(managers) if m[1] == selected\_user["manager\_name"]),  
 0) if managers else 0)  
  
 col1, col2 = st.columns(2)  
 with col1:  
 update\_btn = st.form\_submit\_button("Mettre à jour")  
 with col2:  
 delete\_btn = st.form\_submit\_button("Supprimer")  
  
 if update\_btn:  
 try:  
 cursor.execute("UPDATE users SET username = ?, role = ? WHERE id = ?",  
 (username, role, selected\_user\_id))  
 if new\_password:  
 hashed\_pwd = hash\_password(new\_password)  
 cursor.execute("UPDATE users SET password = ? WHERE id = ?", (hashed\_pwd, selected\_user\_id))  
  
 if role == "employee":  
 cursor.execute("SELECT \* FROM employees WHERE user\_id = ?", (selected\_user\_id,))  
 if cursor.fetchone():  
 cursor.execute("""  
 UPDATE employees  
 SET name = ?, department = ?, manager\_id = ?  
 WHERE user\_id = ?  
 """, (name, department, manager\_id, selected\_user\_id))  
 else:  
 cursor.execute("""  
 INSERT INTO employees (user\_id, name, department, manager\_id)  
 VALUES (?, ?, ?, ?)  
 """, (selected\_user\_id, name, department, manager\_id))  
 else:  
 cursor.execute("DELETE FROM employees WHERE user\_id = ?", (selected\_user\_id,))  
 conn.commit()  
 st.success("✅ Utilisateur mis à jour.")  
 except Exception as e:  
 st.error(f"⚠️ Erreur lors de la mise à jour : {e}")  
  
 if delete\_btn:  
 try:  
 cursor.execute("DELETE FROM employees WHERE user\_id = ?", (selected\_user\_id,))  
 cursor.execute("DELETE FROM users WHERE id = ?", (selected\_user\_id,))  
 conn.commit()  
 st.success("🗑️ Utilisateur supprimé.")  
 except Exception as e:  
 st.error(f"⚠️ Erreur lors de la suppression : {e}")  
  
 # --- Ajout d’un utilisateur ---  
 st.subheader("➕ Ajouter un utilisateur")  
 with st.form("add\_user\_form"):  
 new\_username = st.text\_input("Username")  
 new\_password = st.text\_input("Password", type="password")  
 new\_role = st.selectbox("Role", ["admin", "manager", "employee"])  
  
 name, department, manager\_id = None, None, None  
 if new\_role == "employee":  
 name = st.text\_input("Nom de l'employé")  
 department = st.text\_input("Département")  
 cursor.execute("SELECT id, username FROM users WHERE role = 'manager'")  
 managers = cursor.fetchall()  
 if managers:  
 manager\_id = st.selectbox("Manager", [m[0] for m in managers],  
 format\_func=lambda x: [m[1] for m in managers if m[0] == x][0])  
 else:  
 st.warning("Aucun manager trouvé. Créez un manager d'abord.")  
  
 create\_btn = st.form\_submit\_button("Créer l'utilisateur")  
 if create\_btn:  
 if new\_username and new\_password:  
 if new\_role == "employee":  
 if not name or not department or not manager\_id:  
 st.warning("Remplissez tous les champs pour l'employé.")  
 else:  
 if add\_user(new\_username, new\_password, new\_role, manager\_id, name, department):  
 st.success(f"✅ Utilisateur '{new\_username}' créé.")  
 else:  
 st.error("⚠️ Erreur lors de la création.")  
 else:  
 if add\_user(new\_username, new\_password, new\_role):  
 st.success(f"✅ Utilisateur '{new\_username}' créé.")  
 else:  
 st.error("⚠️ Erreur lors de la création.")  
 else:  
 st.warning("Tous les champs sont requis.")  
  
 conn.close()  
  
  
def get\_team\_feedback(department=None):  
 conn = sqlite3.connect('database/app.db')  
   
 query = """  
 SELECT   
 u.username,  
 f.feedback\_date,  
 f.feedback\_text,  
 f.category,  
 f.satisfaction\_level,  
 u.department  
 FROM feedback f  
 JOIN users u ON f.employee\_id = u.id  
 """  
   
 if department:  
 query += " WHERE u.department = ?"  
 df = pd.read\_sql\_query(query, conn, params=(department,))  
 else:  
 df = pd.read\_sql\_query(query, conn)  
   
 conn.close()  
 return df  
  
def display\_feedback\_analytics(feedback\_df):  
 if feedback\_df.empty:  
 st.info("Aucun feedback disponible pour le moment.")  
 return  
   
 st.subheader("📊 Analyse des Feedbacks")  
   
 # Convert dates  
 feedback\_df['feedback\_date'] = pd.to\_datetime(feedback\_df['feedback\_date'])  
   
 # Satisfaction moyenne par département  
 col1, col2 = st.columns(2)  
   
 with col1:  
 avg\_satisfaction = feedback\_df.groupby('department')['satisfaction\_level'].mean()  
 fig = px.bar(  
 avg\_satisfaction,  
 title="Satisfaction Moyenne par Département",  
 labels={'value': 'Niveau de Satisfaction', 'department': 'Département'}  
 )  
 st.plotly\_chart(fig)  
   
 with col2:  
 category\_counts = feedback\_df['category'].value\_counts()  
 fig = px.pie(  
 values=category\_counts.values,  
 names=category\_counts.index,  
 title="Distribution des Catégories de Feedback"  
 )  
 st.plotly\_chart(fig)  
   
 # Évolution temporelle  
 st.subheader("📈 Évolution de la Satisfaction")  
 time\_trend = feedback\_df.groupby([pd.Grouper(key='feedback\_date', freq='W')])['satisfaction\_level'].mean()  
 fig = px.line(  
 x=time\_trend.index,  
 y=time\_trend.values,  
 labels={'x': 'Date', 'y': 'Satisfaction Moyenne'},  
 title="Tendance de Satisfaction (Moyenne Hebdomadaire)"  
 )  
 st.plotly\_chart(fig)  
   
 # Feedbacks récents  
 st.subheader("📝 Feedbacks Récents")  
 recent\_feedback = feedback\_df.sort\_values('feedback\_date', ascending=False).head(5)  
 for \_, row in recent\_feedback.iterrows():  
 with st.expander(f"{row['username']} - {row['category']} - {row['feedback\_date'].strftime('%Y-%m-%d %H:%M')}"):  
 st.write(row['feedback\_text'])  
 st.write(f"Satisfaction: {'⭐' \* row['satisfaction\_level']}")  
  
def manager\_dashboard():  
 st.title("👥 Tableau de Bord Manager")  
   
 # Get department selection  
 departments = get\_departments()  
 selected\_department = st.selectbox(  
 "Sélectionner un département",  
 ["Tous"] + departments  
 )  
   
 department = None if selected\_department == "Tous" else selected\_department  
   
 # Create tabs  
 tab1, tab2, tab3 = st.tabs(["Performance", "Feedback", "Prédictions"])  
   
 with tab1:  
 st.subheader("📊 Performance de l'Équipe")  
 team\_data = get\_team\_data(department)  
 display\_team\_performance(team\_data)  
   
 with tab2:  
 st.subheader("💭 Feedback de l'Équipe")  
 feedback\_data = get\_team\_feedback(department)  
 display\_feedback\_analytics(feedback\_data)  
   
 with tab3:  
 st.subheader("🔮 Prédictions")  
 display\_predictions(department)  
  
def main():  
 st.title("👋 Bienvenue dans l'application de gestion des performances")  
 st.write("Cette application permet aux administrateurs de gérer les performances des employés.")  
   
 # Afficher les dashboards  
 dashboard = st.selectbox("Sélectionner un dashboard", ["Administrateur", "Manager"])  
   
 if dashboard == "Administrateur":  
 admin\_dashboard()  
 elif dashboard == "Manager":  
 manager\_dashboard()  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 main()

### Explication des Composants:

#### Classe RandomForestModel

* Rôle: Modèle de prédiction ML
* Features: Historique, temps, département
* Méthodes: train, predict, evaluate

#### Fonction admin\_dashboard

* Rôle: Interface administrateur
* Visualisations: Métriques globales
* Actions: Gestion utilisateurs, rapports

#### Fonction predict\_performance

* Rôle: Prédiction performances futures
* Algorithme: Random Forest
* Données: Historique sur 30 jours

## 14.3 employee.py - Fonctionnalités Employé

### Code Source:

import streamlit as st  
import pandas as pd  
import sqlite3  
import plotly.express as px  
from datetime import datetime, timedelta  
import numpy as np  
from scipy import stats  
  
DB\_PATH = "database/app.db"  
  
def connect\_db():  
 return sqlite3.connect(DB\_PATH)  
  
def load\_data():  
 conn = connect\_db()  
 df = pd.read\_sql\_query("SELECT \* FROM performance\_data", conn)  
 return df  
  
  
def view\_performance\_history(employee\_id):  
 conn = connect\_db()  
 df = pd.read\_sql\_query(  
 """  
 SELECT prediction\_date, performance\_score   
 FROM performance\_history   
 WHERE employee\_id = ?  
 ORDER BY prediction\_date DESC  
 """,   
 conn,   
 params=(employee\_id,)  
 )  
 conn.close()  
   
 if not df.empty:  
 st.line\_chart(df.set\_index('prediction\_date'))  
 else:  
 st.write("No performance history available")  
   
 return df  
  
  
def get\_team\_average(employee\_id):  
 conn = connect\_db()  
   
 # Get employee's department  
 cursor = conn.cursor()  
 cursor.execute("""  
 SELECT e.department   
 FROM employees e   
 WHERE e.user\_id = ?  
 """, (employee\_id,))  
 department = cursor.fetchone()[0]  
   
 # Get team members in same department  
 team\_df = pd.read\_sql\_query(  
 """  
 SELECT ph.\*   
 FROM performance\_history ph  
 JOIN users u ON ph.employee\_id = u.id  
 JOIN employees e ON e.user\_id = u.id  
 WHERE e.department = ?  
 ORDER BY ph.prediction\_date  
 """,  
 conn,  
 params=(department,)  
 )  
 conn.close()  
   
 if not team\_df.empty:  
 return team\_df.groupby('prediction\_date')['performance\_score'].mean().reset\_index()  
 return pd.DataFrame()  
  
  
def calculate\_performance\_stats(scores):  
 """Calculate detailed performance statistics."""  
 if len(scores) < 2:  
 return None  
   
 stats\_dict = {  
 'mean': np.mean(scores),  
 'median': np.median(scores),  
 'std': np.std(scores),  
 'min': np.min(scores),  
 'max': np.max(scores),  
 'trend': np.polyfit(range(len(scores)), scores, 1)[0],  
 'consistency': 1 - (np.std(scores) / np.mean(scores)) # coefficient of variation  
 }  
   
 # Calculate percentile ranks  
 stats\_dict['percentile\_25'] = np.percentile(scores, 25)  
 stats\_dict['percentile\_75'] = np.percentile(scores, 75)  
   
 return stats\_dict  
  
def analyze\_performance\_trend(scores):  
 """Analyze the trend in performance scores."""  
 if len(scores) < 2:  
 return None  
   
 # Calculate moving average  
 window = min(5, len(scores) // 2)  
 moving\_avg = np.convolve(scores, np.ones(window)/window, mode='valid')  
   
 # Calculate trend  
 trend = np.polyfit(range(len(scores)), scores, 1)[0]  
   
 # Perform statistical test for trend significance  
 correlation = stats.pearsonr(range(len(scores)), scores)[0]  
   
 return {  
 'trend': trend,  
 'correlation': correlation,  
 'moving\_avg': moving\_avg  
 }  
  
def get\_performance\_insights(individual\_scores, team\_scores):  
 """Generate insights by comparing individual and team performance."""  
 if len(individual\_scores) < 2 or len(team\_scores) < 2:  
 return None  
   
 insights = []  
   
 # Compare means  
 ind\_mean = np.mean(individual\_scores)  
 team\_mean = np.mean(team\_scores)  
 diff = ind\_mean - team\_mean  
   
 # Perform t-test  
 t\_stat, p\_value = stats.ttest\_ind(individual\_scores, team\_scores)  
   
 # Calculate effect size (Cohen's d)  
 pooled\_std = np.sqrt((np.var(individual\_scores) + np.var(team\_scores)) / 2)  
 cohens\_d = diff / pooled\_std  
   
 return {  
 'mean\_difference': diff,  
 'significance': p\_value < 0.05,  
 'effect\_size': cohens\_d,  
 'p\_value': p\_value  
 }  
  
def submit\_feedback(employee\_id, feedback\_text, category, satisfaction\_level):  
 conn = connect\_db()  
 cursor = conn.cursor()  
   
 cursor.execute("""  
 INSERT INTO feedback (employee\_id, feedback\_text, category, satisfaction\_level)  
 VALUES (?, ?, ?, ?)  
 """, (employee\_id, feedback\_text, category, satisfaction\_level))  
   
 conn.commit()  
 conn.close()  
  
def get\_employee\_feedback\_history(employee\_id):  
 conn = connect\_db()  
 query = """  
 SELECT feedback\_date, feedback\_text, category, satisfaction\_level  
 FROM feedback  
 WHERE employee\_id = ?  
 ORDER BY feedback\_date DESC  
 """  
 df = pd.read\_sql\_query(query, conn, params=(employee\_id,))  
 conn.close()  
 return df  
  
def feedback\_section(employee\_id):  
 st.subheader("📝 Feedback")  
   
 # Feedback form  
 with st.form("feedback\_form"):  
 feedback\_text = st.text\_area("Votre feedback:", height=100)  
 col1, col2 = st.columns(2)  
   
 with col1:  
 category = st.selectbox(  
 "Catégorie:",  
 ["Général", "Environnement de Travail", "Management", "Projets", "Formation", "Autre"]  
 )  
   
 with col2:  
 satisfaction = st.slider(  
 "Niveau de satisfaction:",  
 min\_value=1,  
 max\_value=5,  
 value=3,  
 help="1 = Très insatisfait, 5 = Très satisfait"  
 )  
   
 submit = st.form\_submit\_button("Soumettre le Feedback")  
   
 if submit and feedback\_text:  
 submit\_feedback(employee\_id, feedback\_text, category, satisfaction)  
 st.success("Feedback soumis avec succès!")  
 st.rerun()  
   
 # Show feedback history  
 st.subheader("Historique des Feedbacks")  
 feedback\_history = get\_employee\_feedback\_history(employee\_id)  
   
 if not feedback\_history.empty:  
 # Convert feedback\_date to datetime if it's not already  
 feedback\_history['feedback\_date'] = pd.to\_datetime(feedback\_history['feedback\_date'])  
   
 # Display satisfaction trend  
 fig = px.line(  
 feedback\_history,  
 x='feedback\_date',  
 y='satisfaction\_level',  
 title='Évolution de la Satisfaction'  
 )  
 st.plotly\_chart(fig)  
   
 # Display feedback history in a table  
 for \_, row in feedback\_history.iterrows():  
 with st.expander(f"{row['category']} - {row['feedback\_date'].strftime('%Y-%m-%d %H:%M')}"):  
 st.write(row['feedback\_text'])  
 st.write(f"Satisfaction: {'⭐' \* row['satisfaction\_level']}")  
 else:  
 st.info("Aucun feedback soumis pour le moment.")  
  
def employee\_dashboard():  
 st.subheader("🙋‍♂️ Employee Dashboard")  
   
 # Get employee data  
 df = load\_data()  
 employee\_data = df[df["employee\_id"] == st.session\_state["user"][0]]  
   
 # Performance Overview  
 st.subheader("📊 Performance Overview")  
 col1, col2 = st.columns(2)  
   
 with col1:  
 if not employee\_data.empty:  
 latest\_score = employee\_data['performance\_score'].iloc[-1]  
 st.metric("Latest Performance Score", f"{latest\_score:.2f}/5.0")  
   
 # Performance History Section  
 st.subheader("📈 Your Performance History")  
 history = view\_performance\_history(st.session\_state["user"][0])  
   
 if not history.empty:  
 # Convert dates to datetime  
 history['prediction\_date'] = pd.to\_datetime(history['prediction\_date'])  
   
 # Add date filter  
 min\_date = history['prediction\_date'].min().date()  
 max\_date = history['prediction\_date'].max().date()  
   
 dates = st.date\_input(  
 "Select Date Range",  
 value=(min\_date, max\_date),  
 min\_value=min\_date,  
 max\_value=max\_date  
 )  
   
 if len(dates) == 2:  
 start\_date, end\_date = dates  
 filtered\_history = history[  
 (history['prediction\_date'].dt.date >= start\_date) &   
 (history['prediction\_date'].dt.date <= end\_date)  
 ]  
   
 # Calculate detailed statistics  
 scores = filtered\_history['performance\_score'].values  
 stats\_dict = calculate\_performance\_stats(scores)  
 trend\_analysis = analyze\_performance\_trend(scores)  
   
 if stats\_dict and trend\_analysis:  
 # Display Statistics  
 st.subheader("📊 Performance Statistics")  
 col1, col2, col3 = st.columns(3)  
   
 with col1:  
 st.metric("Average Score", f"{stats\_dict['mean']:.2f}")  
 st.metric("Consistency", f"{stats\_dict['consistency']\*100:.1f}%")  
   
 with col2:  
 st.metric("Highest Score", f"{stats\_dict['max']:.2f}")  
 st.metric("Lowest Score", f"{stats\_dict['min']:.2f}")  
   
 with col3:  
 trend\_icon = "↗️" if trend\_analysis['trend'] > 0 else "↘️"  
 st.metric("Score Trend", f"{trend\_icon} {abs(trend\_analysis['trend']):.3f}/day")  
 st.metric("Variability", f"±{stats\_dict['std']:.2f}")  
   
 # Performance Distribution  
 st.subheader("📊 Score Distribution")  
 fig\_col1, fig\_col2 = st.columns(2)  
   
 with fig\_col1:  
 # Create histogram  
 hist\_data = [scores]  
 group\_labels = ['Scores']  
 import plotly.figure\_factory as ff  
 fig1 = ff.create\_distplot(hist\_data, group\_labels, bin\_size=0.2)  
 fig1.update\_layout(title="Score Distribution")  
 st.plotly\_chart(fig1, use\_container\_width=True)  
   
 with fig\_col2:  
 # Create box plot  
 import plotly.graph\_objects as go  
 fig2 = go.Figure()  
 fig2.add\_trace(go.Box(y=scores, name="Performance Scores"))  
 fig2.update\_layout(title="Score Range Analysis")  
 st.plotly\_chart(fig2, use\_container\_width=True)  
   
 # Team Comparison with Enhanced Analysis  
 st.subheader("👥 Team Comparison")  
 team\_avg = get\_team\_average(st.session\_state["user"][0])  
   
 if not team\_avg.empty:  
 # Convert team dates to datetime  
 team\_avg['prediction\_date'] = pd.to\_datetime(team\_avg['prediction\_date'])  
 filtered\_team = team\_avg[  
 (team\_avg['prediction\_date'].dt.date >= start\_date) &   
 (team\_avg['prediction\_date'].dt.date <= end\_date)  
 ]  
   
 # Analyze individual vs team performance  
 team\_scores = filtered\_team['performance\_score'].values  
 performance\_insights = get\_performance\_insights(scores, team\_scores)  
   
 if performance\_insights:  
 # Display comparison insights  
 st.subheader("🎯 Performance Insights")  
   
 # Effect size interpretation  
 effect\_size = abs(performance\_insights['effect\_size'])  
 if effect\_size < 0.2:  
 effect\_text = "minimal"  
 elif effect\_size < 0.5:  
 effect\_text = "moderate"  
 else:  
 effect\_text = "substantial"  
   
 # Display insights based on statistical analysis  
 if performance\_insights['significance']:  
 if performance\_insights['mean\_difference'] > 0:  
 st.success(f"🌟 Your performance is significantly higher than the team average (p < {performance\_insights['p\_value']:.3f})")  
 st.info(f"The difference is {effect\_text} (Cohen's d = {effect\_size:.2f})")  
 else:  
 st.warning(f"📈 Your performance is significantly lower than the team average (p < {performance\_insights['p\_value']:.3f})")  
 st.info(f"The gap is {effect\_text} (Cohen's d = {effect\_size:.2f})")  
 else:  
 st.info("Your performance is statistically similar to the team average")  
   
 # Plot comparison  
 comparison = filtered\_history.merge(  
 filtered\_team,  
 on='prediction\_date',  
 suffixes=('\_individual', '\_team')  
 )  
   
 st.line\_chart(  
 comparison.set\_index('prediction\_date')[  
 ['performance\_score\_individual', 'performance\_score\_team']  
 ].rename(columns={  
 'performance\_score\_individual': 'Your Score',  
 'performance\_score\_team': 'Team Average'  
 }),  
 use\_container\_width=True  
 )  
 else:  
 st.info("No performance history available yet. Check back after your first evaluation!")  
   
 # Create tabs for different sections  
 tab1, tab2, tab3 = st.tabs(["Performance", "Feedback", "Statistiques"])  
   
 with tab1:  
 st.subheader("📈 Vos Performances")  
 view\_performance\_history(st.session\_state["user"][0])  
   
 with tab2:  
 feedback\_section(st.session\_state["user"][0])  
   
 with tab3:  
 st.subheader("📊 Statistiques")  
 # display\_employee\_stats(st.session\_state["user"][0])

### Explication des Composants:

#### Fonction employee\_dashboard

* Rôle: Interface employé
* Affichage: Performances personnelles
* Fonctionnalités: Feedback, statistiques

#### Fonction submit\_feedback

* Rôle: Soumission de feedback
* Données: Texte, catégorie, satisfaction
* Stockage: Table feedback dans SQLite

#### Fonction view\_performance\_history

* Rôle: Historique des performances
* Visualisation: Graphiques temporels
* Calculs: Moyennes, tendances

## 14.4 Schéma Base de Données

### Code Source:

CREATE TABLE IF NOT EXISTS users (  
 id INTEGER PRIMARY KEY AUTOINCREMENT,  
 username TEXT UNIQUE NOT NULL,  
 password TEXT NOT NULL,  
 role TEXT CHECK(role IN ('admin', 'manager', 'employee')) NOT NULL  
);  
  
CREATE TABLE IF NOT EXISTS employees (  
 id INTEGER PRIMARY KEY AUTOINCREMENT,  
 user\_id INTEGER,  
 name TEXT,  
 department TEXT,  
 FOREIGN KEY (user\_id) REFERENCES users(id)  
);  
  
CREATE TABLE IF NOT EXISTS performance\_data (  
 id INTEGER PRIMARY KEY AUTOINCREMENT,  
 employee\_id INTEGER,  
 hours\_worked REAL,  
 tasks\_completed INTEGER,  
 feedback\_score REAL,  
 punctuality\_rate REAL,  
 performance\_score REAL,  
 date TEXT,  
 FOREIGN KEY (employee\_id) REFERENCES employees(id)  
);  
  
ALTER TABLE employees ADD COLUMN manager\_id INTEGER;  
ALTER TABLE performance\_data ADD COLUMN manager\_id INTEGER;

### Explication des Tables:

#### users

* Rôle: Stockage utilisateurs
* Colonnes: id, username, password, role
* Relations: Référencé par feedback, performance

#### performance\_history

* Rôle: Historique performances
* Colonnes: id, employee\_id, date, score
* Relations: Lié à users via employee\_id

#### feedback

* Rôle: Stockage feedbacks
* Colonnes: id, employee\_id, text, category
* Relations: Lié à users via employee\_id