PROJECT MANUAL

TITLE: EMPLOYEE PERFORMANCE AND PRODUCTIVITY ANALYTICS SYSTEM

PROBLEM DESCRIPTION:

The Employee Performance and Productivity Analytics System addresses the challenge of optimizing workforce management by providing actionable insights into employee performance, productivity, and retention risks. Traditional HR systems often lack predictive capabilities and comprehensive analytics, making it difficult for organizations to proactively address issues such as employee attrition, training needs, and promotion readiness. This system leverages machine learning techniques to analyze employee data, predict risks, and recommend interventions, enabling data-driven decision-making for HR professionals and managers.

FUNCTIONALITIES:

1. **Dashboard**:
    - Displays key performance indicators (KPIs) such as average performance score, employee satisfaction, and resignation risk.
    - Visualizes performance distribution across the organization.
2. **Employee Analysis**:
    - Provides detailed profiles of individual employees, including tenure, performance scores, and department.
    - Generates personalized recommendations for retention, promotion, and training based on predictive models.
3. **New Employee Evaluation**:
    - Predicts the performance and resignation risk of new hires based on input metrics.
    - Offers actionable recommendations for onboarding and development.
4. **Risk & Growth Insights**:
    - Identifies employees at high risk of resignation using a simulated risk score.

- Recommends promotion candidates based on performance, tenure, and training.
- Highlights training needs for underperforming employees.
- Alerts managers about employees with concerning sick leave patterns.

DATASET:

The system uses the **Extended Employee Performance and Productivity Data** dataset from Kaggle, which includes the following key features:

- **Employee_ID**: Unique identifier for each employee.
- **Department**: The department to which the employee belongs.
- **Job_Title**: The role of the employee.
- **Performance_Score**: Numeric score (1-5) representing employee performance.
- **Monthly_Salary**: The salary of the employee.
- **Work_Hours_Per_Week**: Weekly working hours.
- **Projects_Handled**: Number of projects assigned.
- **Overtime_Hours**: Hours worked beyond regular schedule.
- **Sick_Days**: Number of sick days taken.
- **Training_Hours**: Hours spent in training programs.
- **Employee_Satisfaction_Score**: Self-reported satisfaction score (1-10).
- **Promotions**: Number of promotions received.
- **Hire_Date**: Date of joining the organization.

Dataset Link: https://www.kaggle.com/datasets/mexwell/employee-performance-and-productivity-data/data

CODE:

The system is implemented using Python and Streamlit, with the following key components:

```
import streamlit as st
import pandas as pd
import numpy as np
import plotly.express as px
from datetime import datetime

# ========== SETUP ==========
st.set_page_config(
    page_title="Employee Analytics Suite",
    page_icon=" ",
    layout="wide",
    initial_sidebar_state="expanded"
)

# Custom CSS
st.markdown("""
<style>
    .metric-card {
        background: white;
        border-radius: 10px;
        padding: 1.5rem;
        box-shadow: 0 4px 6px rgba(0,0,0,0.1);
        margin-bottom: 1rem;
    }
    .risk-card {
        border-left: 4px solid #ef4444;
        padding: 1rem;
        background: #fef2f2;
        border-radius: 0 8px 8px 0;
        margin-bottom: 1rem;
```

```
    }
    .promotion-card {
        border-left: 4px solid #10b981;
        padding: 1rem;
        background: #f0fdf4;
        border-radius: 0 8px 8px 0;
        margin-bottom: 1rem;
    }
    .training-card {
        border-left: 4px solid #3b82f6;
        padding: 1rem;
        background: #eff6ff;
        border-radius: 0 8px 8px 0;
        margin-bottom: 1rem;
    }
    .performer-card {
        background: white;
        border-radius: 10px;
        padding: 1rem;
        box-shadow: 0 4px 6px rgba(0,0,0,0.1);
        margin-bottom: 1rem;
        height: 220px;
    }
</style>
""", unsafe_allow_html=True)

# ========== DATA LOADING ==========
@st.cache_data
def load_data():
    try:
        df = pd.read_csv("Extended_Employee_Performance_and_Productivity_Data.csv")

        # Data cleaning
        df['Hire_Date'] = pd.to_datetime(df['Hire_Date'], errors='coerce')
```

```
        df['Tenure'] = (pd.Timestamp.now() - df['Hire_Date']).dt.days / 365.25

        # Ensure numeric columns
        numeric_cols = ['Performance_Score', 'Monthly_Salary', 'Work_Hours_Per_Week',
                        'Projects_Handled', 'Overtime_Hours', 'Sick_Days',
                        'Team_Size', 'Training_Hours', 'Promotions',
                        'Employee_Satisfaction_Score']

        for col in numeric_cols:
            df[col] = pd.to_numeric(df[col], errors='coerce')

        # Create categories
        df['Performance_Category'] = pd.cut(
            df['Performance_Score'],
            bins=[0, 2.5, 3.5, 5],
            labels=['Low', 'Medium', 'High']
        )

        # Calculate Productivity Score (custom metric)
        df['Productivity_Score'] = (
            0.4 * df['Performance_Score'] +
            0.3 * (df['Projects_Handled'] / df['Projects_Handled'].max()) +
            0.2 * (df['Training_Hours'] / df['Training_Hours'].max()) +
            0.1 * (df['Employee_Satisfaction_Score'] / 10)
        ) * 100

        return df.dropna()
    except Exception as e:
        st.error(f"Error loading data: {str(e)}")
        return pd.DataFrame()

df = load_data()

if df.empty:
```

```
    st.error("No data loaded. Please check your data file.")
    st.stop()

# ========== DASHBOARD PAGE ==========
def dashboard_page():
    st.title("📊 Executive Dashboard")

    # KPI Cards Row
    col1, col2, col3, col4 = st.columns(4)
    with col1:
        st.metric("Total Employees", len(df))
    with col2:
        st.metric("Avg Performance", f"{df['Performance_Score'].mean():.1f}/5")
    with col3:
        st.metric("Avg Satisfaction", f"{df['Employee_Satisfaction_Score'].mean():.1f}/10")
    with col4:
        st.metric("Avg Productivity", f"{df['Productivity_Score'].mean():.1f}")

    st.markdown("---")

    # Top 5 Performers in Cards
    st.subheader("🏆 Top 5 Performers")
    top_performers = df.sort_values('Performance_Score', ascending=False).head(5)

    cols = st.columns(5)
    for i, (_, row) in enumerate(top_performers.iterrows()):
        with cols[i]:
            st.markdown(f"""
            <div class="performer-card">
                <h4 style="margin-top:0;color:#3b82f6">#{i+1}</h4>
                <p><b>{row['Employee_ID']}</b></p>
                <p>{row.get('Name','')}</p>
                <p>{row['Department']}</p>
```

```
            <p>Performance: <b>{row['Performance_Score']:.1f}/5</b></p>
            <p>Productivity: <b>{row['Productivity_Score']:.1f}</b></p>
        </div>
        """, unsafe_allow_html=True)

st.markdown("---")

# Department Productivity Comparison
st.subheader("📊 Department Comparison")

# Metric selection
metric_options = {
    'Productivity_Score': 'Productivity Score',
    'Performance_Score': 'Performance Score',
    'Projects_Handled': 'Projects Handled',
    'Training_Hours': 'Training Hours',
    'Employee_Satisfaction_Score': 'Satisfaction Score',
    'Overtime_Hours': 'Overtime Hours',
    'Sick_Days': 'Sick Days',
    'Promotions': 'Promotions',
    'Tenure': 'Tenure (Years)'
}

selected_metric = st.selectbox(
    "Select metric to compare:",
    options=list(metric_options.keys()),
    format_func=lambda x: metric_options[x]
)

# Department selection
all_depts = df['Department'].unique().tolist()
selected_depts = st.multiselect(
    "Select departments to compare:",
```

```
    options=all_depts,
    default=all_depts,
    help="Select 'All Departments' or choose specific ones"
)

# Filter and prepare data
dept_stats = df[df['Department'].isin(selected_depts)].groupby('Department').agg({
    'Productivity_Score': 'mean',
    'Performance_Score': 'mean',
    'Projects_Handled': 'mean',
    'Training_Hours': 'mean',
    'Employee_Satisfaction_Score': 'mean',
    'Overtime_Hours': 'mean',
    'Sick_Days': 'mean',
    'Promotions': 'mean',
    'Tenure': 'mean'
}).reset_index()

# Create comparison chart
if not dept_stats.empty:
    fig = px.bar(
        dept_stats.sort_values(selected_metric, ascending=False),
        x='Department',
        y=selected_metric,
        color='Department',
        text_auto='.2f',
        title=f"{metric_options[selected_metric]} by Department",
        labels={'Department': '', selected_metric: metric_options[selected_metric]},
        height=500
    )
    fig.update_layout(
        showlegend=False,
        xaxis_title="",
        yaxis_title=metric_options[selected_metric],
```

```
            hovermode="x unified"
        )
        st.plotly_chart(fig, use_container_width=True)
    else:
        st.warning("No departments selected for comparison")

# ========== EMPLOYEE ANALYSIS PAGE ==========
def employee_analysis_page():
    st.title("👤 Employee Analysis")

    # Employee Selector
    employee = st.selectbox("Select Employee", df['Employee_ID'])
    emp_data = df[df['Employee_ID'] == employee].iloc[0]

    # Profile Card
    col1, col2 = st.columns([1, 2])
    with col1:
        st.subheader("Employee Profile")
        st.write(f"**ID:** {emp_data['Employee_ID']}")
        st.write(f"**Department:** {emp_data['Department']}")
        st.write(f"**Job Title:** {emp_data['Job_Title']}")
        st.write(f"**Tenure:** {emp_data['Tenure']:.1f} years")
        st.write(f"**Performance:** {emp_data['Performance_Score']}/5
({emp_data['Performance_Category']})")

    # Recommendations
    with col2:
        st.subheader("Recommendations")

        # Resignation Risk
        risk_score = min(100, max(0,
            30 * (10 - emp_data['Employee_Satisfaction_Score']) / 10 +
            20 * (5 - emp_data['Performance_Score']) / 5 +
```

```
            15 * emp_data['Overtime_Hours'] / 40 -
            10 * emp_data['Promotions'] -
            25 * emp_data['Tenure'] / 10
        ))

        if risk_score > 50:
            st.error(f"🚨 High resignation risk: {risk_score:.0f}%")
            st.write(f"""
            - Conduct retention interview
            - Review workload (current overtime: {emp_data['Overtime_Hours']} hrs/week)
            - Consider recognition or promotion
            """)

        # Promotion Potential
        if (emp_data['Performance_Score'] >= 4 and
            emp_data['Tenure'] >= 2 and
            emp_data['Training_Hours'] >= 30):
            st.success(f"🌟 Promotion candidate (Score: {(emp_data['Performance_Score']*10 +
emp_data['Tenure']*5):.1f}/100)")
            st.write("""
            - Eligible for next level
            - Consider leadership training
            """)

        # Training Needs
        if emp_data['Training_Hours'] < 20 or emp_data['Performance_Score'] < 3:
            st.warning(f"📚 Training recommended (Current: {emp_data['Training_Hours']}
hrs)")
            st.write(f"""
            - Needs {max(20, 40 - emp_data['Training_Hours'])} additional hours
            - Skills development program
            - Mentorship opportunity
            """)
```

```
# ========== NEW EVALUATION PAGE ==========
def new_evaluation_page():
    st.title("✨ New Employee Evaluation")

    with st.form("evaluation_form"):
        st.subheader("Basic Information")
        col1, col2 = st.columns(2)
        with col1:
            name = st.text_input("Full Name (optional)")
        with col2:
            existing_ids = df['Employee_ID'].unique()
            sample_id = existing_ids[0] if len(existing_ids) > 0 else "1"
            employee_id = st.text_input("Employee ID*", value="",
                                        help=f"Must be unique numeric ID (like {sample_id})")

        col3, col4 = st.columns(2)
        with col3:
            department = st.selectbox("Department*", df['Department'].unique())
        with col4:
            job_title = st.text_input("Job Title*")

        st.subheader("Productivity Metrics")
        col5, col6 = st.columns(2)
        with col5:
            work_hours = st.slider("Work Hours/Week*", 20, 80, 40)
            projects = st.number_input("Projects Handled*", 1, 10, 3)
            training = st.number_input("Training Hours*", 0, 100, 20)
        with col6:
            overtime = st.number_input("Overtime Hours*", 0, 40, 5)
            sick_days = st.number_input("Sick Days*", 0, 30, 2)
            satisfaction = st.slider("Satisfaction (1-10)*", 1, 10, 7)
```

```
remote_freq = st.selectbox(
    "Remote Work Frequency*",
    options=["Never", "Rarely", "Sometimes", "Often", "Always"]
)

submitted = st.form_submit_button("Evaluate Employee")

if submitted:
    # Validate required fields
    if not all([employee_id, department, job_title]):
        st.error("Please fill all required fields (*)")
        st.stop()

    # Validate employee ID is numeric
    if not employee_id.isdigit():
        st.error("Employee ID must be a numeric value")
        st.stop()

    # Validate employee ID doesn't exist
    if employee_id in df['Employee_ID'].astype(str).values:
        st.error(f"Employee ID {employee_id} already exists. Please use a unique ID.")
        st.stop()

    # Predict performance score (1-5 scale)
    performance = min(5, max(1,
        0.4 * (work_hours / 50) * 5 +
        0.3 * (projects / 5) * 5 +
        0.2 * (training / 50) * 5 +
        0.1 * (satisfaction / 10) * 5
    ))

    # Calculate risk factors
    risk_score = min(100, max(0,
        40 * (10 - satisfaction) / 10 +
```

```
    30 * (5 - performance) / 5 +
    20 * overtime / 40 -
    10 * 0  # New employee has 0 promotions
))

# Display results
st.success("Evaluation Complete!")

# Score Visualization
col7, col8 = st.columns(2)
with col7:
    st.metric("Predicted Performance", f"{performance:.1f}/5")
    st.progress(performance / 5)

with col8:
    st.metric("Resignation Risk", f"{risk_score:.0f}%")
    st.progress(risk_score / 100)

# Detailed Recommendations
st.subheader("📋 Actionable Recommendations")

# Performance Recommendations
if performance >= 4:
    st.success("**High Performer Detected** 🌟")
    st.write("- Fast-track for leadership training")
    st.write("- Consider special projects assignment")
    st.write("- Eligible for early promotion review")
elif performance <= 2:
    st.error("**Performance Concerns** ⚠️")
    st.write("- Implement 90-day improvement plan")
    st.write("- Assign mentor for weekly check-ins")
    st.write("- Required training: 40+ hours")
else:
```

```
    st.info("**Solid Performer** 👍")
    st.write("- Recommend skill development plan")
    st.write("- Regular performance feedback")

# Risk Mitigation
if risk_score > 60:
    st.error("**High Attrition Risk** 🚨")
    st.write("- Schedule retention interview immediately")
    st.write("- Review workload balance")
    st.write("- Consider spot bonus/recognition")
elif risk_score > 30:
    st.warning("**Moderate Risk** 🎯")
    st.write("- Monitor engagement closely")
    st.write("- Conduct stay interviews quarterly")

# Training Needs
if training < 15:
    st.warning("**Training Deficiency** 📚")
    st.write(f"- Minimum {25-training} additional training hours needed")
    st.write("- Enroll in foundational skills program")

# Workload Analysis
if overtime > 10:
    st.warning("**Excessive Overtime** ⏳")
    st.write("- Review workload distribution")
    st.write("- Consider temporary assistance")

# Health Indicators
if sick_days > 5:
    st.warning("**Elevated Sick Days** 😷")
    st.write("- Recommend wellness check")
    st.write("- Review work-life balance")
```

```
# ========== RISK & GROWTH PAGE ==========
def risk_growth_page():
    st.title(" ⚠️ Employee Risk & Growth Insights")

    tab1, tab2, tab3, tab4 = st.tabs([
        "Resignation Risk",
        "Promotion Candidates",
        "Training Needs",
        "Sick Leave Alerts"
    ])

    with tab1:
        st.subheader("Employees at Risk of Resignation")
        at_risk = df.sort_values('Employee_Satisfaction_Score').head(10)

        for _, emp in at_risk.iterrows():
            with st.container():
                st.markdown(f"""
                <div class="risk-card">
                    <h4>{emp['Employee_ID']} ({emp['Department']})</h4>
                    <p>Satisfaction: {emp['Employee_Satisfaction_Score']}/10 |
                    Performance: {emp['Performance_Score']}/5 |
                    Tenure: {emp['Tenure']:.1f} yrs</p>
                </div>
                """, unsafe_allow_html=True)

    with tab2:
        st.subheader("Promotion Recommendations")
        promotions = df[(df['Performance_Score'] >= 4) & (df['Tenure'] >=
2)].sort_values('Performance_Score', ascending=False).head(10)

        for _, emp in promotions.iterrows():
            with st.container():
```

```
            st.markdown(f"""
            <div class="promotion-card">
                <h4>{emp['Employee_ID']} ({emp['Job_Title']})</h4>
                <p>Performance: {emp['Performance_Score']}/5 |
                Tenure: {emp['Tenure']:.1f} yrs |
                Training: {emp['Training_Hours']} hrs</p>
            </div>
            """, unsafe_allow_html=True)

    with tab3:
        st.subheader("Training Recommendations")
        training_needs = df[(df['Training_Hours'] < 20) | (df['Performance_Score'] <
3)].sort_values('Training_Hours').head(10)

        for _, emp in training_needs.iterrows():
            with st.container():
                st.markdown(f"""
                <div class="training-card">
                    <h4>{emp['Employee_ID']} ({emp['Department']})</h4>
                    <p>Training Hours: {emp['Training_Hours']} |
                    Performance: {emp['Performance_Score']}/5 |
                    Projects: {emp['Projects_Handled']}</p>
                </div>
                """, unsafe_allow_html=True)

    with tab4:
        st.subheader("Sick Leave Alerts")
        sick_alerts = df.sort_values('Sick_Days', ascending=False).head(10)

        for _, emp in sick_alerts.iterrows():
            with st.container():
                st.markdown(f"""
                <div class="risk-card">
                    <h4>{emp['Employee_ID']} ({emp['Job_Title']})</h4>
```

```
                <p>Sick Days: {emp['Sick_Days']} |
                Last Promotion: {emp['Promotions']} yrs ago |
                Satisfaction: {emp['Employee_Satisfaction_Score']}/10</p>
            </div>
            """, unsafe_allow_html=True)

# ========== MAIN APP ==========
def main():
    st.sidebar.title("Employee Analytics Suite")
    page = st.sidebar.radio("Navigation", [
        "Dashboard",
        "Employee Analysis",
        "New Evaluation",
        "Risk & Growth Insights"
    ])

    if page == "Dashboard":
        dashboard_page()
    elif page == "Employee Analysis":
        employee_analysis_page()
    elif page == "New Evaluation":
        new_evaluation_page()
    elif page == "Risk & Growth Insights":
        risk_growth_page()

if __name__ == "__main__":
    main()
```

To Run the Code:

Step 1: Open command prompt.

Step 2: Navigate to your Project Folder.

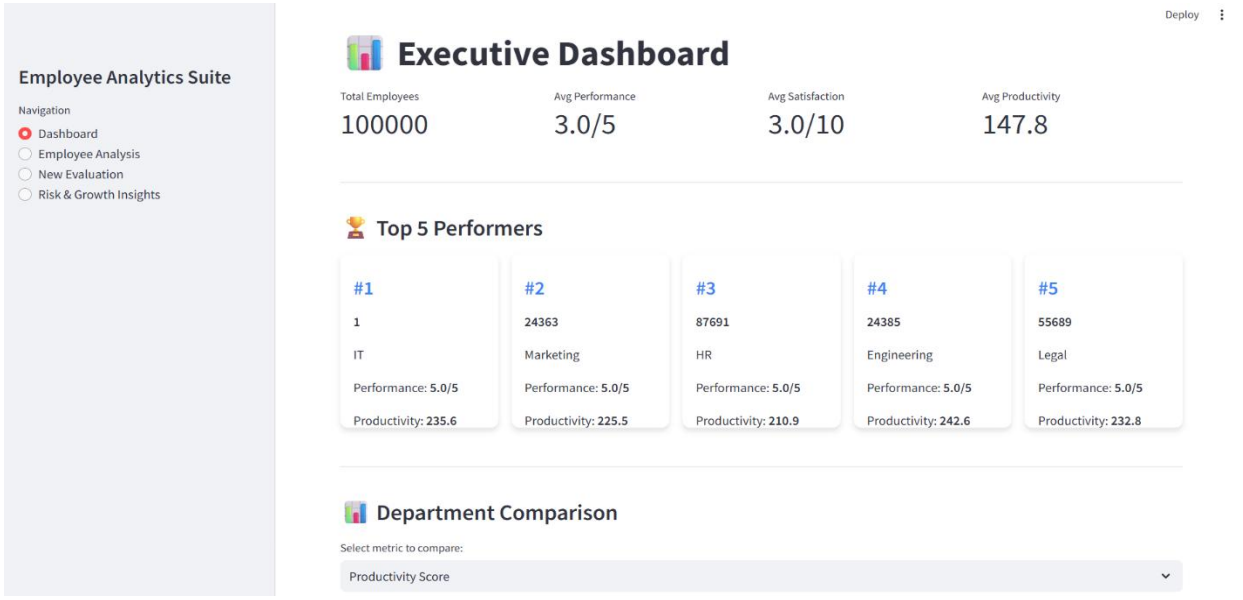
Step 3: streamlit run filename.py

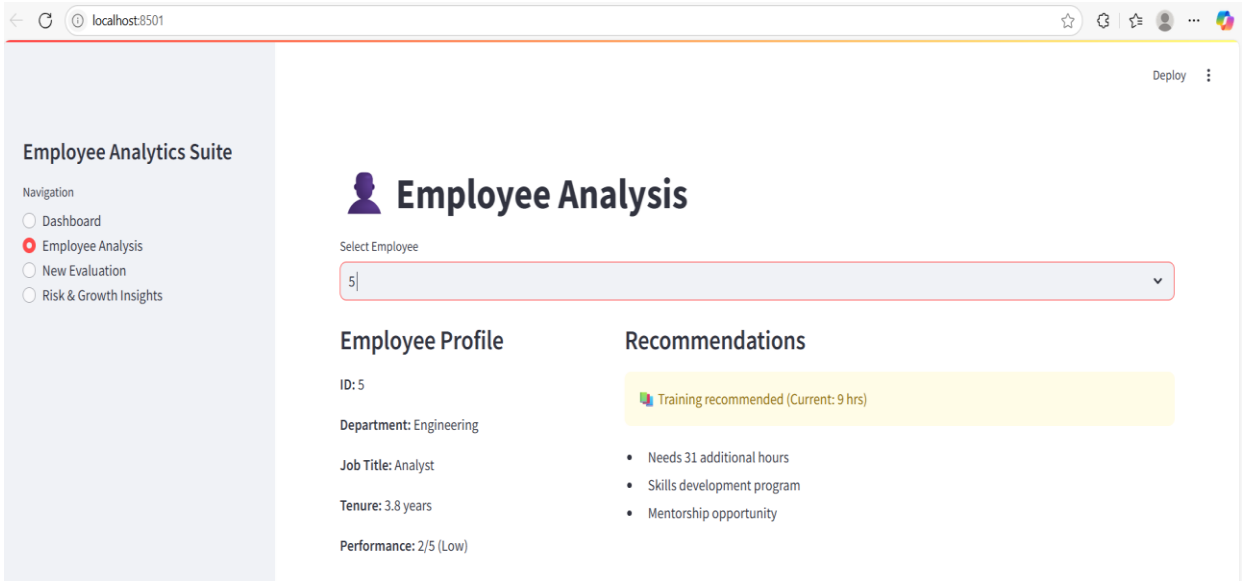
Key Features of the Code

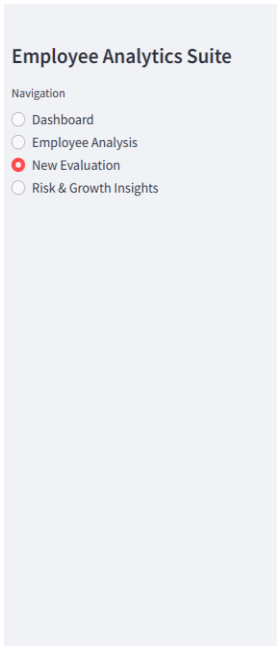
1. **Interactive UI**: Built with Streamlit for easy navigation.
2. **Simulated ML Models**: Weighted scoring for risk prediction.
3. **Visualizations**: Plotly charts for performance insights.
4. **Real-time Calculations**: Dynamic updates based on user input.

This system helps HR teams make data-driven decisions to improve employee productivity and retention.

SCREENSHOTS:







✨ New Employee Evaluation

Basic Information

Full Name (optional): Anu

Employee ID*: EMP-123

Department*: IT

Job Title*: Software Developer

Productivity Metrics

Work Hours/Week*: 40 (20 – 80)

Overtime Hours*: 5

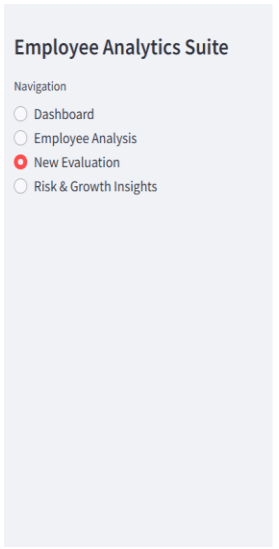
Projects Handled*: 3

Sick Days*: 2

Training Hours*: 20

Satisfaction (1-10)*: 7 (1 – 10)

Remote Work Frequency*: Never



Remote Work Frequency*: Never

Evaluate Productivity

Evaluation Complete!

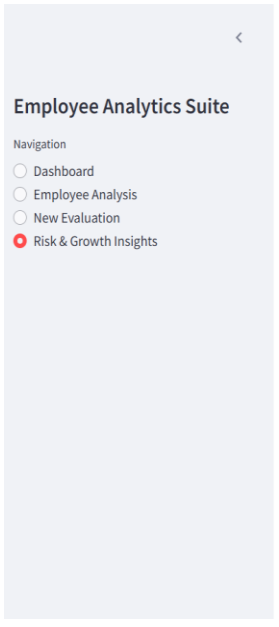
Predicted Performance: 3.2/5

Resignation Risk: 25%

📋 Actionable Recommendations

Solid Performer 👍

- Recommend skill development plan
- Regular performance feedback



Deploy

⚠️ Employee Risk & Growth Insights

Resignation Risk | Promotion Candidates | Training Needs | Sick Leave Alerts

Sick Leave Alerts

2 (Developer)

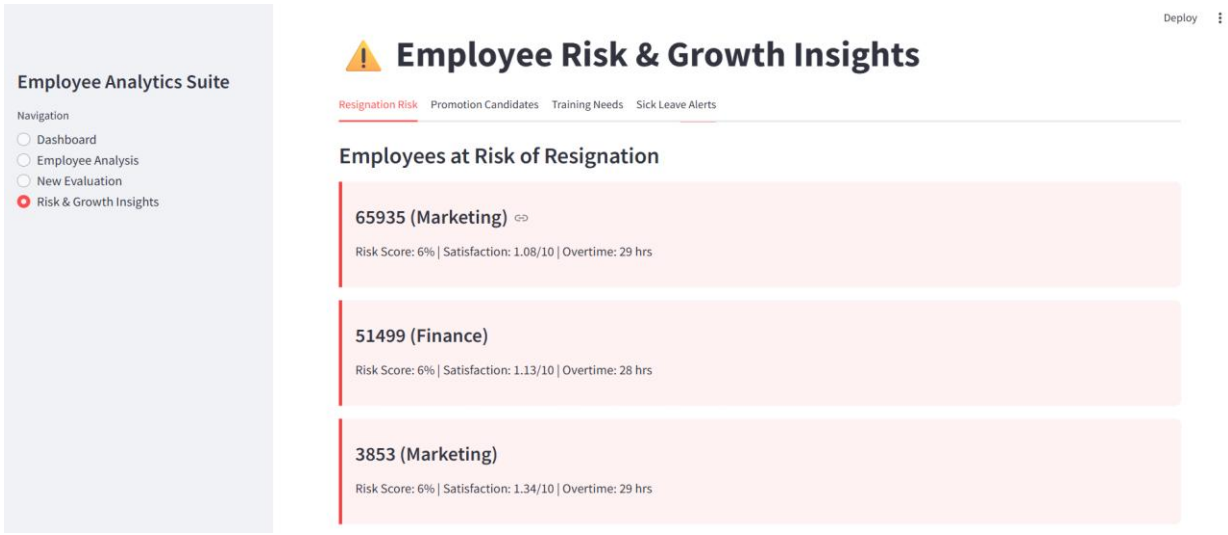
Sick Days: 14 (Top 25%) | Last Promotion: 2 yrs ago | Satisfaction: 1.72/10

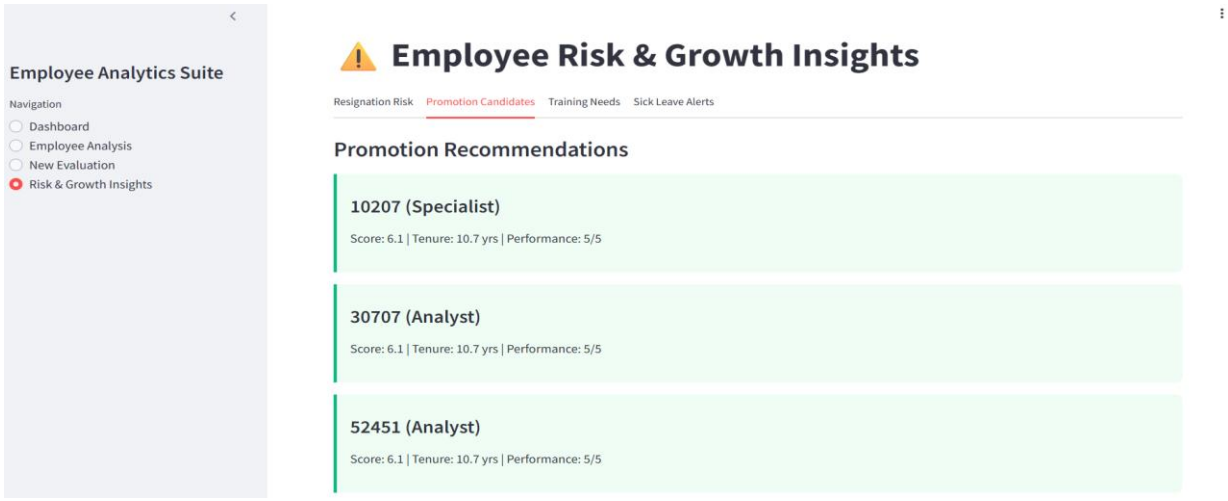
49734 (Manager)

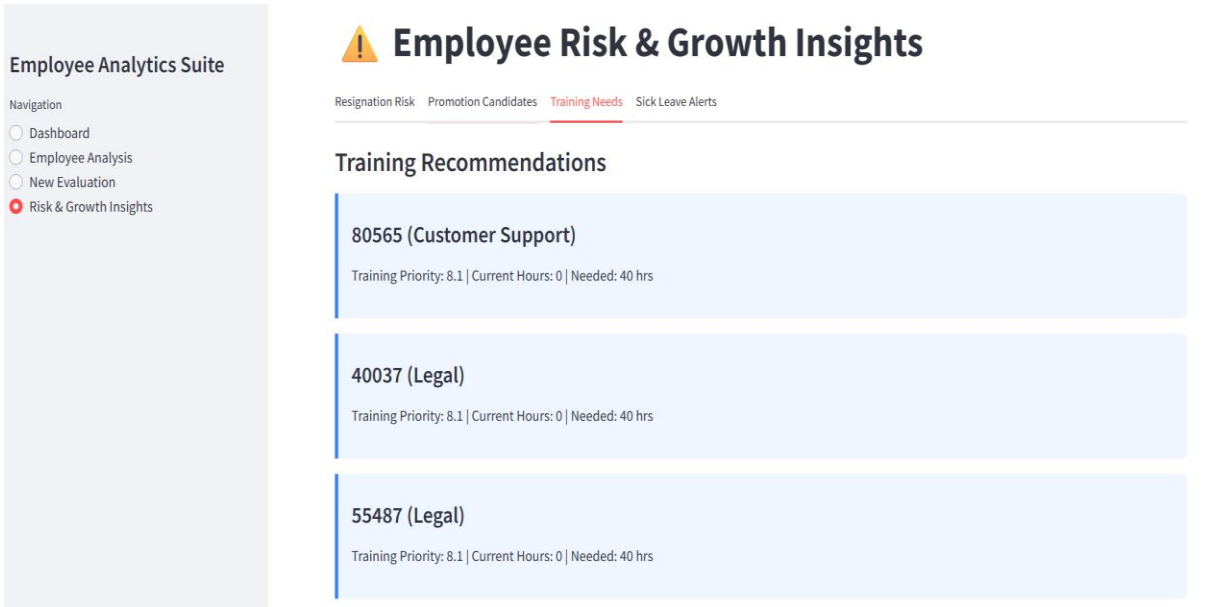
Sick Days: 14 (Top 25%) | Last Promotion: 1 yrs ago | Satisfaction: 4.85/10

49655 (Analyst)

Sick Days: 14 (Top 25%) | Last Promotion: 0 yrs ago | Satisfaction: 1.39/10


Deploy
Employee Analytics Suite
Navigation
Dashboard
Employee Analysis
New Evaluation
Risk & Growth Insights
Employee Risk & Growth Insights
Resignation Risk
Promotion Candidates
Training Needs
Sick Leave Alerts
Employees at Risk of Resignation
65935 (Marketing)
Risk Score: 6% | Satisfaction: 1.08/10 | Overtime: 29 hrs
51499 (Finance)
Risk Score: 6% | Satisfaction: 1.13/10 | Overtime: 28 hrs
3853 (Marketing)
Risk Score: 6% | Satisfaction: 1.34/10 | Overtime: 29 hrs


Employee Analytics Suite
Navigation
Dashboard
Employee Analysis
New Evaluation
Risk & Growth Insights
Employee Risk & Growth Insights
Resignation Risk
Promotion Candidates
Training Needs
Sick Leave Alerts
Promotion Recommendations
10207 (Specialist)
Score: 6.1 | Tenure: 10.7 yrs | Performance: 5/5
30707 (Analyst)
Score: 6.1 | Tenure: 10.7 yrs | Performance: 5/5
52451 (Analyst)
Score: 6.1 | Tenure: 10.7 yrs | Performance: 5/5


Employee Analytics Suite
Navigation
Dashboard
Employee Analysis
New Evaluation
Risk & Growth Insights
Employee Risk & Growth Insights
Resignation Risk
Promotion Candidates
Training Needs
Sick Leave Alerts
Training Recommendations
80565 (Customer Support)
Training Priority: 8.1 | Current Hours: 0 | Needed: 40 hrs
40037 (Legal)
Training Priority: 8.1 | Current Hours: 0 | Needed: 40 hrs
55487 (Legal)
Training Priority: 8.1 | Current Hours: 0 | Needed: 40 hrs