CU CHANDIGARH UNIVERSITY UNIVERSITY INSTITUTE of COMPUTING Asia's Fastest Growing University

NAAC GRADE A+ Accredited University

MACHINE LEARNING

CU CHANDIGARH UNIVERSITY UNIVERSITY INSTITUTE of COMPUTING Asia's Fastest Growing University

PROJECT REPORT ON EMPLOYEE ATTRITION PREDICTOR USING MACHINE LEARNING

SUBMITTED BY:	SUBMITTED TO:
RITINDER KAUR	DR.JAVED ALAM
UID:24MCI10092	E17984
SECTION:24MAM1A	BRANCH:MCA[AI&ML]

CU CHANDIGARH UNIVERSITY UNIVERSITY INSTITUTE of COMPUTING Asia's Fastest Growing University

NAAC GRADE A+ Accredited University

Student Name :Ritinder Kaur **UID: 24MCI10092**
Branch : MCA(AIML) **Section/Group :24MAM-1A**
Semester : 2nd **Date of submission:15/04/25**
Subject Name : Machine Learning Lab **Subject Code : 24CAP-672**

EMPLOYEE ATTRITION PREDICTORUSING MACHINE LEARNING

1. AIM:

Predict whether an employee will leave the company (attrition) based on various HR related features like salary, job satisfaction, work-life balance, etc.

2. PROBLEM STATEMENT:

Given a dataset containing various attributes of employees such as age, job role, education, monthly income, job satisfaction, work-life balance, and more, the goal is to develop a machine learning model that can accurately predict whether an employee is likely to leave the organization.

3. OBJECTIVE:

- Analyze and understand the patterns behind employee attrition.
- Build a predictive model using historical employee data to classify whether an employee will leave the company (Attrition: Yes/No).
- Identify key features contributing to attrition to help HR take preventive actions.

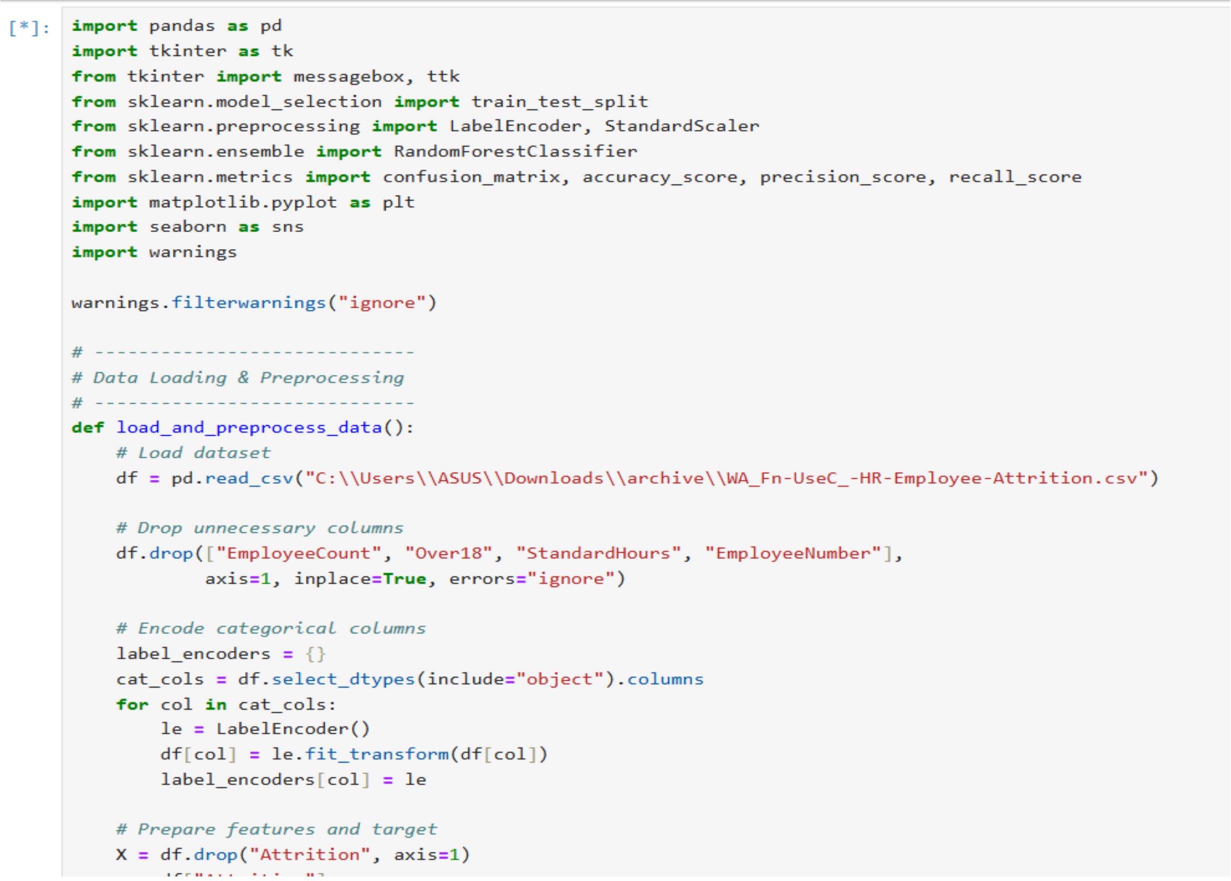
Input: Structured data of employees, including features like:

- Age, Gender, Department, Job Role
- Monthly Income, Overtime, Work-Life Balance
- Years at Company, Job Satisfaction, etc. **Output:** A binary classification:
- 1 → Employee will leave (Attrition = Yes)
- 0 → Employee will stay (Attrition = No) **Scope:**
- Use supervised machine learning models (e.g., Random Forest, Logistic Regression, XGBoost).

4. PROGRAMMING LANGUAGE USED:

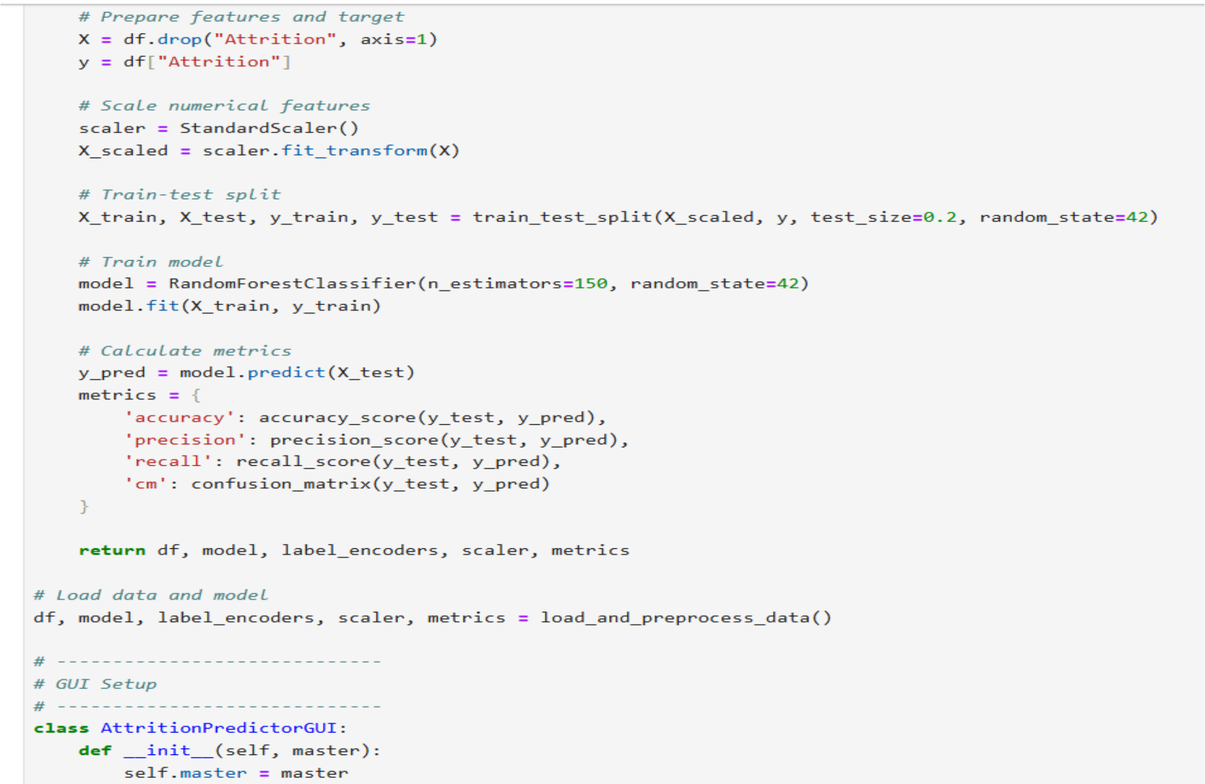
- Python
- Pandas, NumPy
- Matplotlib, Seaborn (for visualization)
- Scikit-learn (ML models)
- XGBoost or RandomForest (for boosting performance)

5. IMPLEMENTATION:

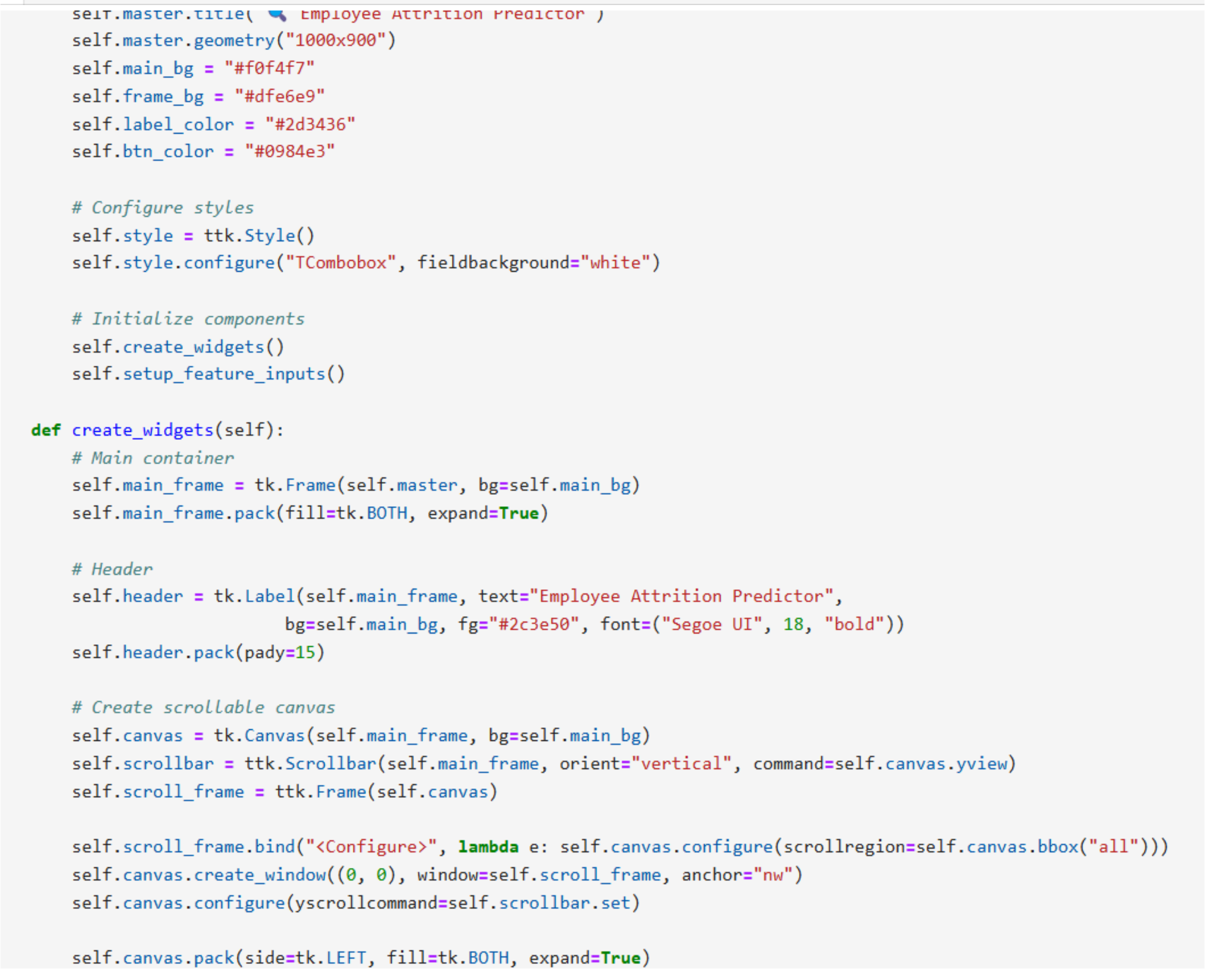


[*]:

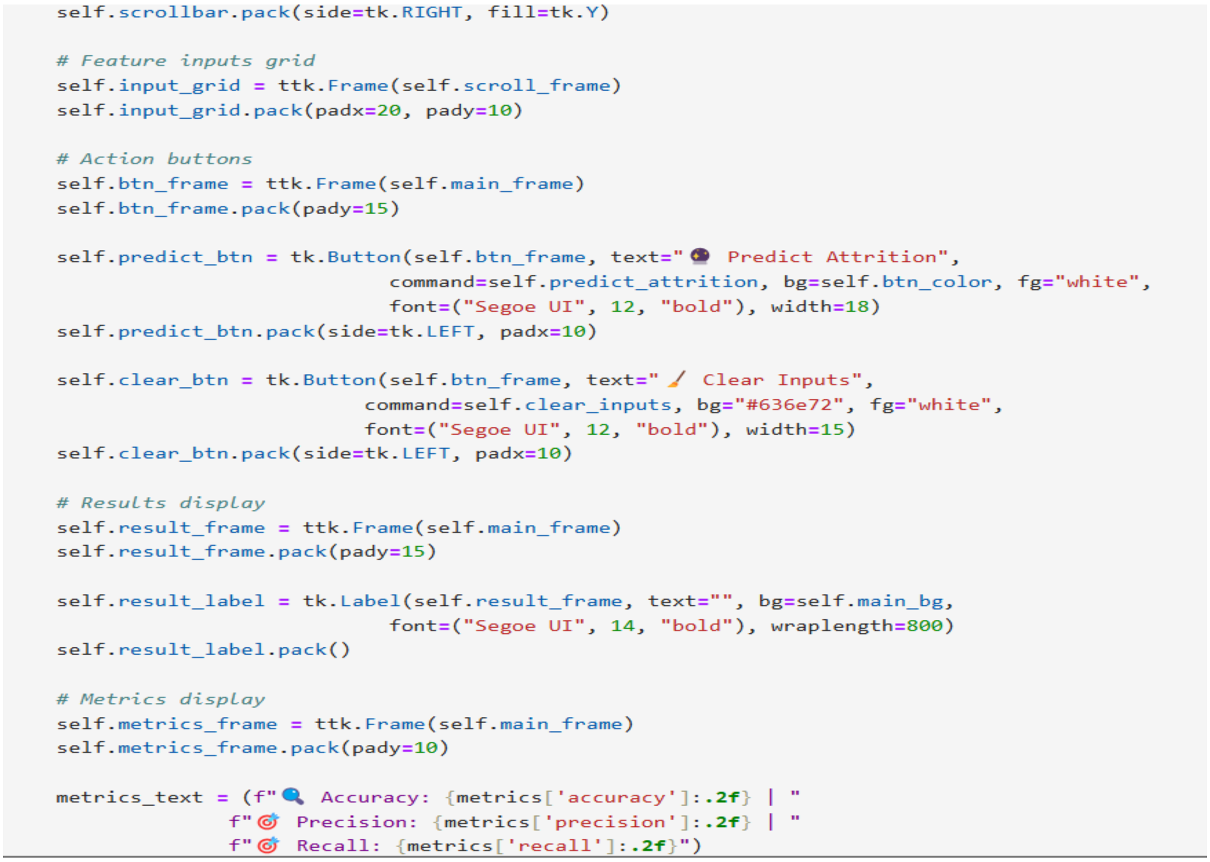
```
import pandas as pd
import tkinter as tk
from tkinter import messagebox, ttk
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import confusion_matrix, accuracy_score, precision_score, recall_score
import matplotlib.pyplot as plt
import seaborn as sns
import warnings

warnings.filterwarnings("ignore")

# ----------------------------
# Data Loading & Preprocessing
# ----------------------------
def load_and_preprocess_data():
    # Load dataset
    df = pd.read_csv("C:\\Users\\ASUS\\Downloads\\archive\\WA_Fn-UseC_-HR-Employee-Attrition.csv")

    # Drop unnecessary columns
    df.drop(["EmployeeCount", "Over18", "StandardHours", "EmployeeNumber"],
            axis=1, inplace=True, errors="ignore")

    # Encode categorical columns
    label_encoders = {}
    cat_cols = df.select_dtypes(include="object").columns
    for col in cat_cols:
        le = LabelEncoder()
        df[col] = le.fit_transform(df[col])
        label_encoders[col] = le

    # Prepare features and target
    X = df.drop("Attrition", axis=1)
```



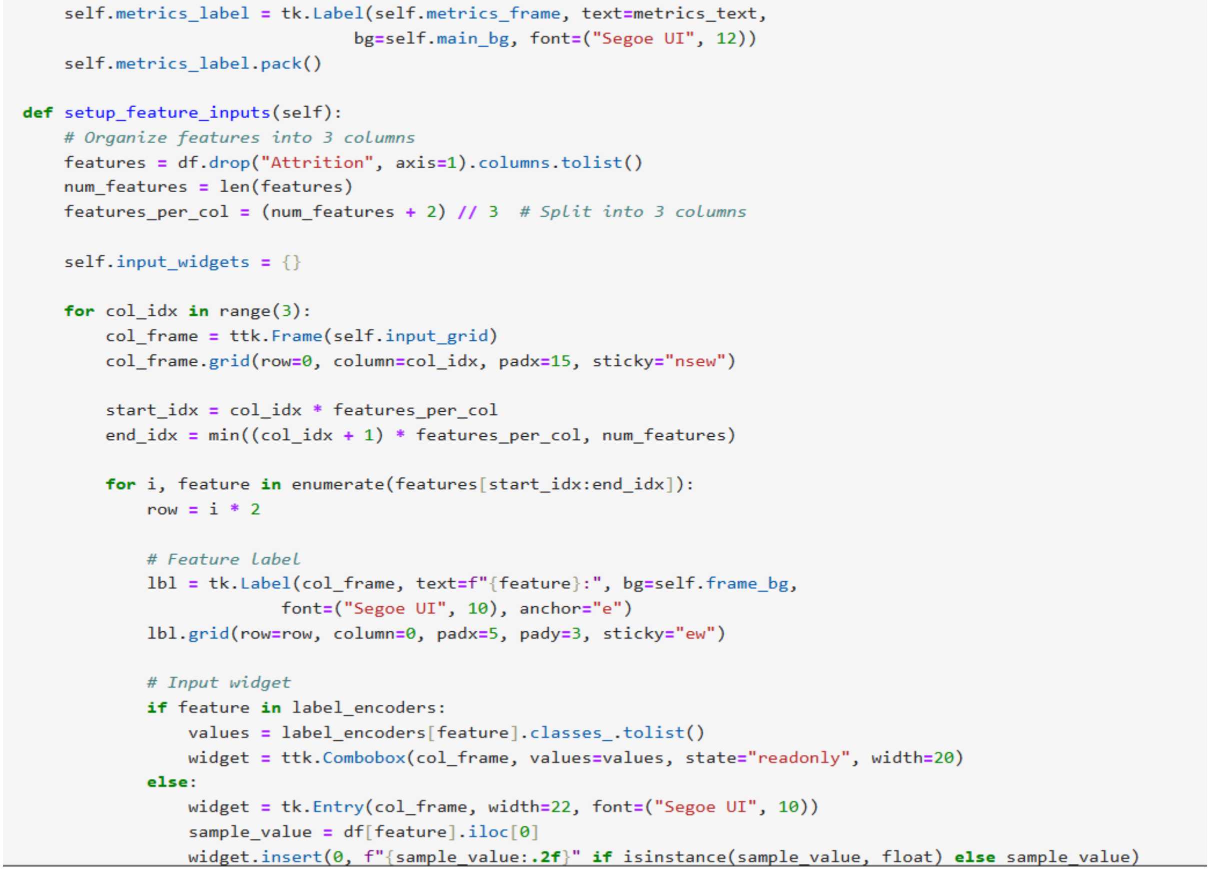
```
    # Prepare features and target
    X = df.drop("Attrition", axis=1)
    y = df["Attrition"]

    # Scale numerical features
    scaler = StandardScaler()
    X_scaled = scaler.fit_transform(X)

    # Train-test split
    X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2, random_state=42)

    # Train model
    model = RandomForestClassifier(n_estimators=150, random_state=42)
    model.fit(X_train, y_train)

    # Calculate metrics
    y_pred = model.predict(X_test)
    metrics = {
        'accuracy': accuracy_score(y_test, y_pred),
        'precision': precision_score(y_test, y_pred),
        'recall': recall_score(y_test, y_pred),
        'cm': confusion_matrix(y_test, y_pred)
    }

    return df, model, label_encoders, scaler, metrics

# Load data and model
df, model, label_encoders, scaler, metrics = load_and_preprocess_data()

# ----------------------------
# GUI Setup
# ----------------------------
class AttritionPredictorGUI:
    def __init__(self, master):
        self.master = master
```



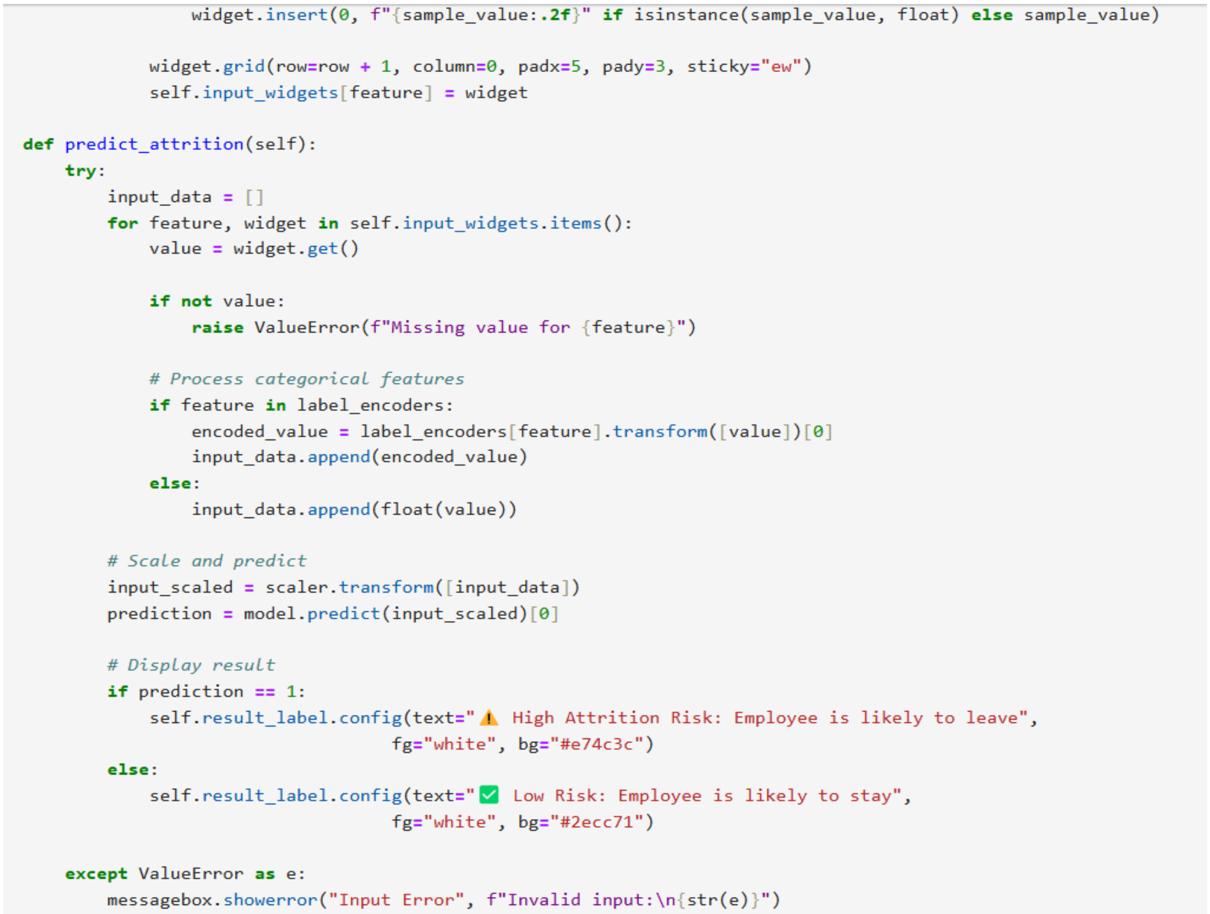
```
        self.master.title("🔍 Employee Attrition Predictor")
        self.master.geometry("1000x900")
        self.main_bg = "#f0f4f7"
        self.frame_bg = "#dfe6e9"
        self.label_color = "#2d3436"
        self.btn_color = "#0984e3"

        # Configure styles
        self.style = ttk.Style()
        self.style.configure("TCombobox", fieldbackground="white")

        # Initialize components
        self.create_widgets()
        self.setup_feature_inputs()

    def create_widgets(self):
        # Main container
        self.main_frame = tk.Frame(self.master, bg=self.main_bg)
        self.main_frame.pack(fill=tk.BOTH, expand=True)

        # Header
        self.header = tk.Label(self.main_frame, text="Employee Attrition Predictor",
                               bg=self.main_bg, fg="#2c3e50", font=("Segoe UI", 18, "bold"))
        self.header.pack(pady=15)

        # Create scrollable canvas
        self.canvas = tk.Canvas(self.main_frame, bg=self.main_bg)
        self.scrollbar = ttk.Scrollbar(self.main_frame, orient="vertical", command=self.canvas.yview)
        self.scroll_frame = ttk.Frame(self.canvas)

        self.scroll_frame.bind("<Configure>", lambda e: self.canvas.configure(scrollregion=self.canvas.bbox("all")))
        self.canvas.create_window((0, 0), window=self.scroll_frame, anchor="nw")
        self.canvas.configure(yscrollcommand=self.scrollbar.set)

        self.canvas.pack(side=tk.LEFT, fill=tk.BOTH, expand=True)
```



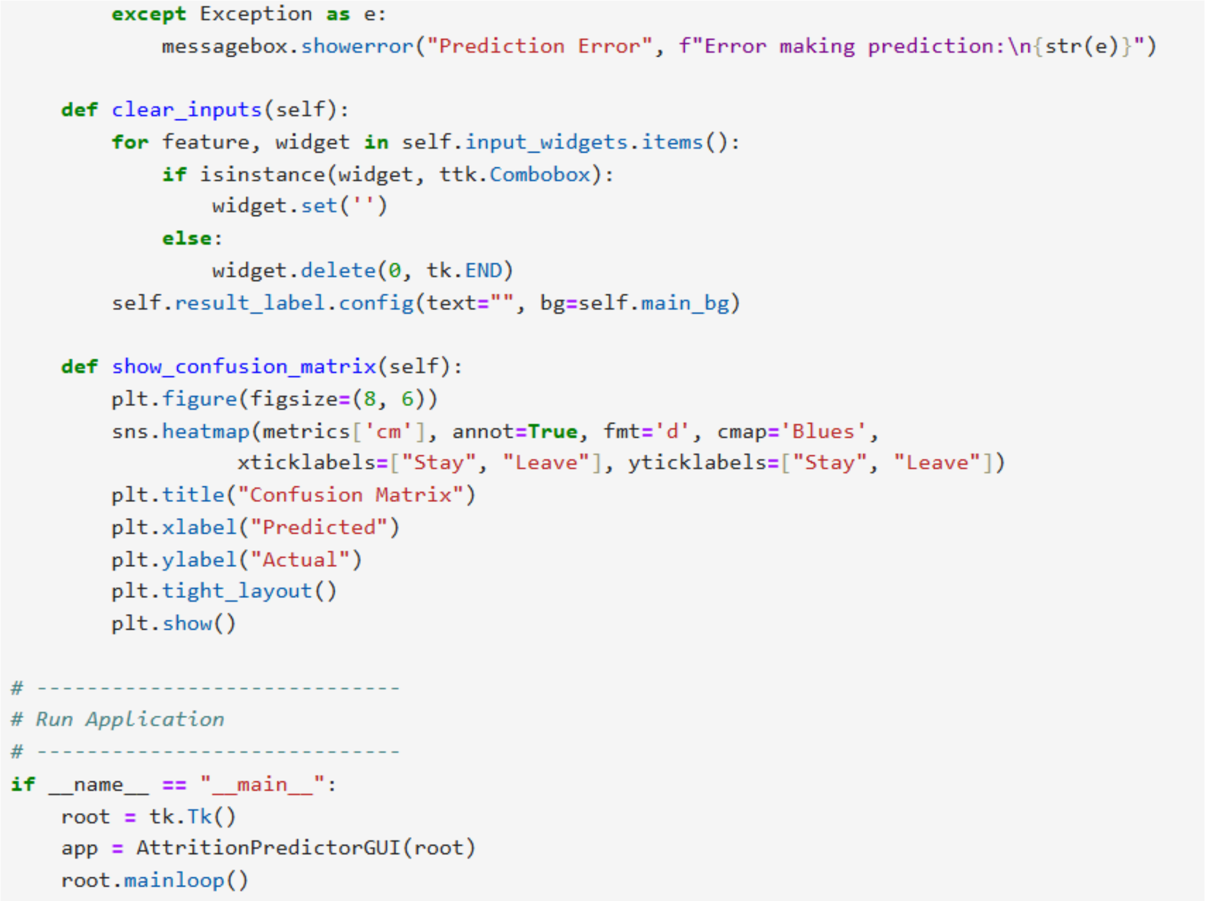
```python
        self.scrollbar.pack(side=tk.RIGHT, fill=tk.Y)

        # Feature inputs grid
        self.input_grid = ttk.Frame(self.scroll_frame)
        self.input_grid.pack(padx=20, pady=10)

        # Action buttons
        self.btn_frame = ttk.Frame(self.main_frame)
        self.btn_frame.pack(pady=15)

        self.predict_btn = tk.Button(self.btn_frame, text="🔮 Predict Attrition",
                                   command=self.predict_attrition, bg=self.btn_color, fg="white",
                                   font=("Segoe UI", 12, "bold"), width=18)
        self.predict_btn.pack(side=tk.LEFT, padx=10)

        self.clear_btn = tk.Button(self.btn_frame, text="🧹 Clear Inputs",
                                 command=self.clear_inputs, bg="#636e72", fg="white",
                                 font=("Segoe UI", 12, "bold"), width=15)
        self.clear_btn.pack(side=tk.LEFT, padx=10)

        # Results display
        self.result_frame = ttk.Frame(self.main_frame)
        self.result_frame.pack(pady=15)

        self.result_label = tk.Label(self.result_frame, text="", bg=self.main_bg,
                                   font=("Segoe UI", 14, "bold"), wraplength=800)
        self.result_label.pack()

        # Metrics display
        self.metrics_frame = ttk.Frame(self.main_frame)
        self.metrics_frame.pack(pady=10)

        metrics_text = (f"🔍 Accuracy: {metrics['accuracy']:.2f} | "
                      f"🎯 Precision: {metrics['precision']:.2f} | "
                      f"🎯 Recall: {metrics['recall']:.2f}")
```



```python
        self.metrics_label = tk.Label(self.metrics_frame, text=metrics_text,
                                    bg=self.main_bg, font=("Segoe UI", 12))
        self.metrics_label.pack()

    def setup_feature_inputs(self):
        # Organize features into 3 columns
        features = df.drop("Attrition", axis=1).columns.tolist()
        num_features = len(features)
        features_per_col = (num_features + 2) // 3  # Split into 3 columns

        self.input_widgets = {}

        for col_idx in range(3):
            col_frame = ttk.Frame(self.input_grid)
            col_frame.grid(row=0, column=col_idx, padx=15, sticky="nsew")

            start_idx = col_idx * features_per_col
            end_idx = min((col_idx + 1) * features_per_col, num_features)

            for i, feature in enumerate(features[start_idx:end_idx]):
                row = i * 2

                # Feature label
                lbl = tk.Label(col_frame, text=f"{feature}:", bg=self.frame_bg,
                             font=("Segoe UI", 10), anchor="e")
                lbl.grid(row=row, column=0, padx=5, pady=3, sticky="ew")

                # Input widget
                if feature in label_encoders:
                    values = label_encoders[feature].classes_.tolist()
                    widget = ttk.Combobox(col_frame, values=values, state="readonly", width=20)
                else:
                    widget = tk.Entry(col_frame, width=22, font=("Segoe UI", 10))
                    sample_value = df[feature].iloc[0]
                    widget.insert(0, f"{sample_value:.2f}" if isinstance(sample_value, float) else sample_value)
```

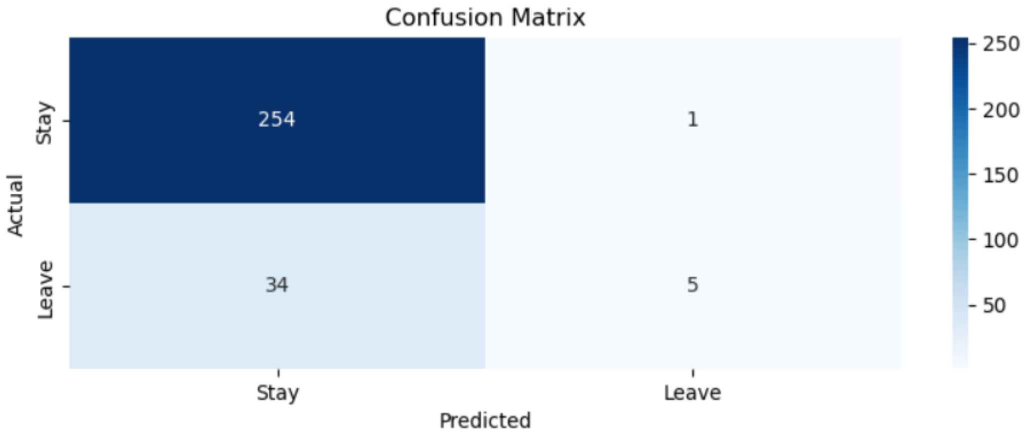


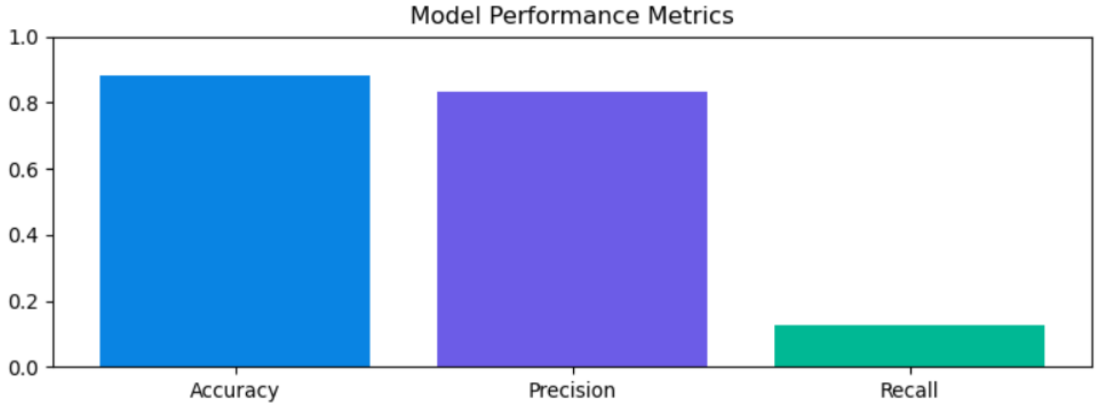
```
                widget.insert(0, f"{sample_value:.2f}" if isinstance(sample_value, float) else sample_value)

            widget.grid(row=row + 1, column=0, padx=5, pady=3, sticky="ew")
            self.input_widgets[feature] = widget

def predict_attrition(self):
    try:
        input_data = []
        for feature, widget in self.input_widgets.items():
            value = widget.get()

            if not value:
                raise ValueError(f"Missing value for {feature}")

            # Process categorical features
            if feature in label_encoders:
                encoded_value = label_encoders[feature].transform([value])[0]
                input_data.append(encoded_value)
            else:
                input_data.append(float(value))

        # Scale and predict
        input_scaled = scaler.transform([input_data])
        prediction = model.predict(input_scaled)[0]

        # Display result
        if prediction == 1:
            self.result_label.config(text="⚠️ High Attrition Risk: Employee is likely to leave",
                                   fg="white", bg="#e74c3c")
        else:
            self.result_label.config(text="✅ Low Risk: Employee is likely to stay",
                                   fg="white", bg="#2ecc71")

    except ValueError as e:
        messagebox.showerror("Input Error", f"Invalid input:\n{str(e)}")
```



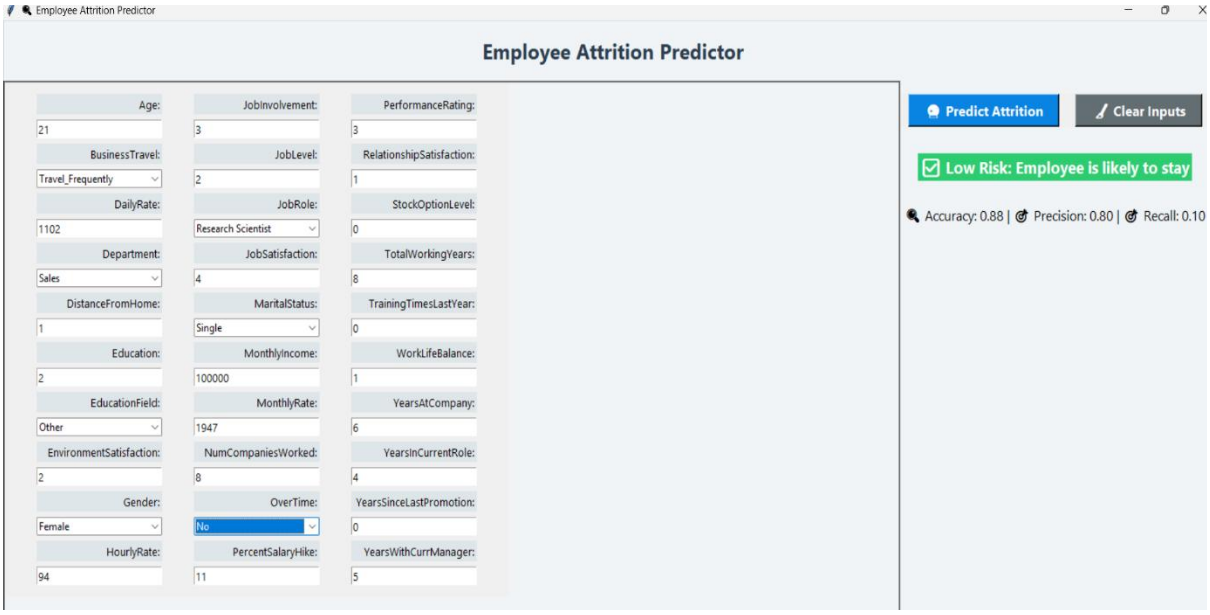
```
        except Exception as e:
            messagebox.showerror("Prediction Error", f"Error making prediction:\n{str(e)}")

    def clear_inputs(self):
        for feature, widget in self.input_widgets.items():
            if isinstance(widget, ttk.Combobox):
                widget.set('')
            else:
                widget.delete(0, tk.END)
        self.result_label.config(text="", bg=self.main_bg)

    def show_confusion_matrix(self):
        plt.figure(figsize=(8, 6))
        sns.heatmap(metrics['cm'], annot=True, fmt='d', cmap='Blues',
                  xticklabels=["Stay", "Leave"], yticklabels=["Stay", "Leave"])
        plt.title("Confusion Matrix")
        plt.xlabel("Predicted")
        plt.ylabel("Actual")
        plt.tight_layout()
        plt.show()

# ----------------------------
# Run Application
# ----------------------------
if __name__ == "__main__":
    root = tk.Tk()
    app = AttritionPredictorGUI(root)
    root.mainloop()
```

6. OUTPUT:





7. GUI OF THE CODE:



Employee Attrition Predictor

Age: 21

BusinessTravel: Travel_Frequently

DailyRate: 1102

Department: Sales

DistanceFromHome: 1

Education: 2

EducationField: Other

EnvironmentSatisfaction: 2

Gender: Female

HourlyRate: 94

JobInvolvement: 3

JobLevel: 2

JobRole: Research Scientist

JobSatisfaction: 4

MaritalStatus: Single

MonthlyIncome: 100000

MonthlyRate: 1947

NumCompaniesWorked: 8

OverTime: No

PercentSalaryHike: 11

PerformanceRating: 3

RelationshipSatisfaction: 1

StockOptionLevel: 0

TotalWorkingYears: 8

TrainingTimesLastYear: 0

WorkLifeBalance: 1

YearsAtCompany: 6

YearsInCurrentRole: 4

YearsSinceLastPromotion: 0

YearsWithCurrManager: 5

☑ Low Risk: Employee is likely to stay

🔍 Accuracy: 0.88 | 🎯 Precision: 0.80 | 🎯 Recall: 0.10

8. LEARNING OUTCOMES:

1. Understand the Business Problem:

- Gain a clear understanding of employee attrition and its impact on organizational performance.
- Translate a real-world HR problem into a machine learning task.

2. Apply Data Preprocessing Techniques:

- Handle missing or irrelevant data.
- Encode categorical variables and normalize numerical features.
- Prepare a dataset suitable for machine learning models.

3. Build and Train Machine Learning Models:

- Implement classification algorithms such as Random Forest, Logistic Regression, and XGBoost.
- Train, tune, and evaluate models using appropriate metrics like accuracy, precision, recall, and F1-score.

4. Evaluate Model Performance:

- Interpret model predictions using confusion matrices and classification reports.
- Compare the effectiveness of different models in predicting attrition.