# Personality and Interest Patterns Analysis

**Introduction**

This document explains the step-by-step process carried out in the given Python script, which performs data preprocessing, feature engineering, model training, evaluation, and insights generation using different machine learning models.

**1. Importing Libraries**

The script starts by importing essential Python libraries for:

* Data manipulation (pandas, numpy)
* Data visualization (matplotlib, seaborn)
* Machine learning modeling and evaluation (sklearn)
* # importing librarys
* import pandas as pd
* import numpy as np
* import matplotlib.pyplot as plt
* import seaborn as sns
* from sklearn.model\_selection import train\_test\_split
* from sklearn.preprocessing import LabelEncoder, StandardScaler
* from sklearn.ensemble import RandomForestClassifier
* from sklearn.tree import DecisionTreeClassifier
* from sklearn.svm import SVC
* from sklearn.metrics import accuracy\_score, confusion\_matrix, classification\_report, f1\_score

**2. Importing Dataset**

* Reads the dataset from a CSV file.
* Displays the first few rows and checks the data types.
* # import dataset
* df=pd.read\_csv("data.csv")
* df.head()

**3. Data Preprocessing & Exploration**

**Handling Missing Values:**

* Replaces invalid values (e.g., 'Unknown') with NaN.
* Fills missing values using the mode of the column.
* # replace invalid value with nan and then with mode or mean/median as per data dypes
* df['Interest']=df['Interest'].replace('Unknown',np.nan)
* df['Interest']=df['Interest'].fillna(df['Interest'].mode()[0],axis=0)

**Feature Engineering:**

* Identifies continuous numerical columns.
* Creates a new categorical feature Age\_Group by binning the Age column.
* Generates a new feature Score\_Sum, which sums all numerical columns.
* # Feature engineering: Create new features
* continuous\_cols=[]
* for i in df.columns:
* data\_typ=df[i].dtypes
* if data\_typ in ['float64','int64']:
* continuous\_cols.append(i)
* df['Age\_Group'] = pd.cut(df['Age'], bins=[0, 20, 30, 40], labels=['<20', '20-30', '30-40'])
* df['Score\_Sum'] = df[continuous\_cols].sum(axis=1)
* print("Feature Engineering: Added 'Age\_Group' and 'Score\_Sum'.")

**Data Type Adjustments:**

* Converts Age\_Group to categorical type.
* df['Age\_Group'] = df['Age\_Group'].astype('object')
* df.dtypes

**4. Encoding Categorical Variables**

* Uses LabelEncoder to convert categorical columns into numerical values for machine learning compatibility.
* # Encode categorical variables
* for i in df.columns:
* data\_typ=df[i].dtypes
* if data\_typ=='object':
* encoder=LabelEncoder()
* df[i]=encoder.fit\_transform(df[i])

**5. Feature Scaling (Normalization)**

* Standardizes continuous numerical features using StandardScaler to normalize the data distribution.
* # Normalize continuous variables
* for i in df.columns:
* data\_typ=df[i].dtypes
* if data\_typ in ['float64','int64']:
* scaler = StandardScaler()
* df[[i]] = scaler.fit\_transform(df[[i]])

**6. Data Visualization**

**Correlation Heatmap:**

* Creates a heatmap to visualize correlations between continuous variables.
* # Visualize distributions
* plt.figure(figsize=(8, 6))
* sns.heatmap(df[continuous\_cols].corr(), annot=True, cmap='coolwarm')
* plt.title('Correlation Heatmap of Personality Scores')
* plt.show()

**Histograms:**

* Generates histograms to explore feature distributions.
* # Visualize distributions
* plt.figure(figsize=(10, 6))
* df[continuous\_cols].hist(bins=20, figsize=(10, 8))
* plt.suptitle('Distributions of Personality Scores')
* plt.show()

**7. Model Building and Evaluation**

**Data Splitting:**

* Splits data into training and testing sets (80% training, 20% testing).
* # Split the data
* X = df.drop(columns=['Personality'])
* y = df['Personality']
* X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

**Model Training:**

* Trains three machine learning models:
  + Decision Tree Classifier
  + Random Forest Classifier
  + Support Vector Machine (SVM)
* # Train models
* models = {
* 'Decision Tree': DecisionTreeClassifier(random\_state=42),
* 'Random Forest': RandomForestClassifier(random\_state=42),
* 'SVM': SVC(random\_state=42)}

**Model Evaluation:**

* Evaluates each model using:
  + **Accuracy Score** (percentage of correctly predicted labels)
  + **F1 Score** (weighted measure of precision and recall)
* Stores results and identifies the best model.
* results = {}
* results = {}
* for name, model in models.items():
* model.fit(X\_train, y\_train) # training the model in each iteration
* y\_pred = model.predict(X\_test) # predicting for each model
* accuracy = accuracy\_score(y\_test, y\_pred) \* 100  # checking the accuracy and f1 score for each model and store it in results
* f1 = f1\_score(y\_test, y\_pred, average='weighted') \* 100
* results[name] = {'Accuracy': accuracy, 'F1 Score': f1}
* print(f"{name}:\nAccuracy: {accuracy}, F1 Score: {f1}\n")

**Confusion Matrix:**

* Generates a confusion matrix to visualize classification performance of the best model.
* # Confusion matrix for the best model
* best\_model\_name = max(results, key=lambda x: results[x]['Accuracy'])
* best\_model = models[best\_model\_name]
* best\_model
* y\_pred\_best = best\_model.predict(X\_test)
* conf\_matrix = confusion\_matrix(y\_test, y\_pred\_best)
* conf\_matrix

**8. Feature Importance Analysis (For Random Forest Model)**

* If the best-performing model is Random Forest, it extracts and displays feature importance scores to interpret which features contributed most to predictions.
* if best\_model\_name == 'Random Forest':
* feature\_importances = best\_model.feature\_importances\_\*100
* feature\_importances\_df = pd.DataFrame({'Feature': X.columns, 'Importance': feature\_importances})
* feature\_importances\_df = feature\_importances\_df.sort\_values(by='Importance', ascending=False)
* print(feature\_importances\_df)

**9. Insights and Reporting**

* Summarizes the best-performing model and its evaluation metrics.
* Provides a breakdown of model results for decision-making.
* # Insights and reporting
* print("Insights:")
* print(f"Best Model: {best\_model\_name}")
* print(f"Results: {results}")
* output
* Insights:
* Best Model: Random Forest
* Results:
* {'Decision Tree': {'Accuracy': 85.15987974856517, 'F1 Score': 85.16118352142654},
* 'Random Forest': {'Accuracy': 88.0607504001874, 'F1 Score': 88.02230244068426},
* 'SVM': {'Accuracy': 86.08128684652326, 'F1 Score': 85.98158219883398}}

**Conclusion**

This script provides a structured pipeline for:

* Data cleaning and preprocessing
* Feature engineering
* Training multiple machine learning models()
* Evaluating and selecting the best model
* Extracting insights from model performance

The best model is identified based on accuracy, and further analysis helps in understanding feature importance and overall predictive capabilities.