Personality Detection from Resume using Machine Learning

Group Members:

Sharmistha Das
Asansol Engineering College
10800123164

Rishika Mishra
Asansol Engineering College
10800123164

Ragini Gupta
Asansol Engineering College
10800123145

Shreya Banerjee
Asansol Engineering College
10800123192

Contents

Acknowledgement

I take this opportunity to express my profound gratitude and deep regards to my faculty, **Prof. Arnab Chakraborty** for his exemplary guidance, monitoring and constant encouragement throughout the course of this project. The blessing, help and guidance given by him time to time shall carry me a long way in the journey of life on which I am about to embark.

I am obliged to my project team members for the valuable information provided by them in their respective fields. I am grateful for their cooperation during the period of my assignment.

Sharmishta Das

Rishika Mishra

Ragini Gupta

Shreya Banerjee

Project Objective

➢ Problem Statement

In the modern recruitment process, organizations receive thousands of resumes for a limited number of positions. Recruiters often spend significant time manually screening resumes, which is inefficient, subjective, and prone to bias.

In addition to technical qualifications, companies are also keen to understand the personality traits of candidates, as these play a crucial role in job performance, teamwork, and cultural fit. To capture personality effectively, this project makes use of the Myers–Briggs Type Indicator (MBTI) framework.

The MBTI is one of the most widely recognized personality models, dividing individuals into 16 personality types derived from four key dimensions:

- **Extraversion (E) – Introversion (I)**: outward vs. inward energy focus.
- **Sensing (S) – Intuition (N)**: practical details vs. abstract ideas.
- **Thinking (T) – Feeling (F)**: logical analysis vs. empathy-based decisions.
- **Judging (J) – Perceiving (P)**: structured planning vs. flexible adaptability.

Each MBTI type (e.g., INTJ, ENTP, INFJ, ESTJ) provides valuable insight into how a candidate communicates, makes decisions, and approaches problem-solving. Incorporating MBTI into the recruitment pipeline allows for better role alignment, efficient shortlisting, and stronger team-building compared to skill-only screening.

➢ Objective

The main objective of this project is to develop a **machine learning system** that can automatically analyze resumes and predict the **personality type of candidates**. The project focuses on going beyond traditional recruitment methods that only evaluate skills and qualifications, by also incorporating **personality insights** for better job-role alignment.

The specific objectives are:

- To **extract structured information** from resumes, including skills, education, work experience, certifications, and job titles.
- To **predict candidate personalities** based on resume features using machine learning models such as Logistic Regression, Random Forest, K-Means, and Hierarchical Clustering.
- To **compare models** using classification metrics (Accuracy, Precision, Recall, F1-score, AUC) and ROC curve.
- To **recommend suitable job roles** aligned with the predicted personality types, thereby improving recruitment efficiency and role-personality matching.
- To provide recruiters with a **data-driven decision-making tool** that reduces subjectivity and saves time in the hiring process.

➢ Methodology

- **Data Collection** – Use a resume dataset with MBTI personality labels.
- **Data Preprocessing** – Clean text, remove noise, perform tokenization, and apply TF-IDF for feature extraction.
- **Model Training** – Apply multiple models (Logistic Regression, Random Forest, K-Means, Hierarchical Clustering).
- **Evaluation** – Compare models based on Accuracy, Precision, Recall, F1-score, AUC, and clustering metrics (ARI/NMI).
- **Final Selection** – Choose the most practical and interpretable model (Logistic Regression).

Project Scope

The broad scope of the project includes:

- **Resume Parsing**: Automatically extract career objectives, skills, degrees, and experiences.
- **Feature Engineering**: Convert categorical and numerical attributes into structured features for ML models.
- **Personality Classification**: Predict MBTI-based personality categories (e.g., Introvert/Extrovert, Thinker/Feeler).
- **Job Recommendation**: Map personality predictions to relevant job categories (e.g., Analysts for INTJs, Managers for ENTJs, Creative Designers for ENFPs).
- **Visualization**: Provide visual insights such as confusion matrices, ROC curves, and clustering dendrograms.

➢ **Limitations**:

- Works best with English resumes.
- Requires labeled datasets for supervised learning.
- Predictions depend heavily on text quality and completeness of resumes

Data Description

The dataset used for this project consists of structured resume information collected from Kaggle. It contains **~9,500 records** and multiple columns that describe various aspects of a candidate's resume, such as skills, education, experience, and job positions. The target variable is the **personality label (MBTI type)**, which is mapped to resumes based on their features.

➢ **Dataset Characteristics**

- **Total Records**: 9,544
- **Features**: 30+ attributes including career objective, skills, education, job roles, certifications, etc.
- **Target Variable**: MBTI personality type (INTJ, ENTP, INFJ, ESTJ, etc.).
- **Data Types**: Mostly categorical/text, with one numeric column (matched_score).
- **Missing Values**: Present in several columns (e.g., career_objective, experience_requirement, address, etc.

Column Name	Non-Null Count	Null Count	Data Type	Description
career_objective	4740	4804	object	Candidate's professional goal/objective
skills	9488	56	object	Candidate's listed skills
degree_names	9460	84	object	Academic degree(s) held
educational_institution_name	9460	84	object	Name of university/college
job_position_name	9544	0	object	Current or desired job position
experiencere_requirement	8180	1364	object	Years of required/mentioned work experience
age_requirement	5457	4087	object	Age-related requirements if specified
certifications	2008	7536	object	Certifications acquired by candidate
responsibilities	9544	0	object	Roles and responsibilities in past jobs
matched_score	9544	0	float64	Matching score between resume and job profile

Table 1: Feature Information

Feature	Min Length	25%	Median	75%	Max Length	Description
career_objective	26	144	210	268	1425	Candidate's objective/summary statement length
skills	2	161	243	504	3104	Length of skills list (characters)
degree_names	6	10	22	39	472	Degree names length (characters)
educational_institution_name	8	27	42	61	212	Institution name length (characters)
job_position_name	5	12	22	35	150	Job title length (characters)

Table 2: Text Feature Length Statistics

- **Insights from Dataset**

1. **Career Objective** is available in ~50% of resumes; it varies from very short (26 characters) to detailed statements (1425 characters).

2. **Skills** are well populated, with a median length of ~243 characters, reflecting a mix of technical and soft skills.

3. **Degree Names and Institutions** are consistent with typical resume formats (median length ~22–42 characters).

4. **Job Position Names** are always present, with reasonable lengths (median ~22 characters).

5. **Matched Score** is a numeric column (0–0.97), representing how well a resume aligns with a job posting, which can be used as a strong feature for prediction.

6. Several columns like **address, extracurriculars, certifications** have high missing values and may need to be dropped or imputed.

Data Pre-processing

As our dataset contained raw resumes in textual form along with their associated MBTI personality labels, several preprocessing steps were required to ensure that the data was consistent, structured, and suitable for training classification models. The detailed methodology is given below:

- **Data Cleaning**

We first examined the dataset for missing and inconsistent values.

Column Name	Count of Null Values
Resume_Text	12
Personality_Type	0

Table 3: Count of Null Values

To visualize missing data, a **heatmap** was plotted:

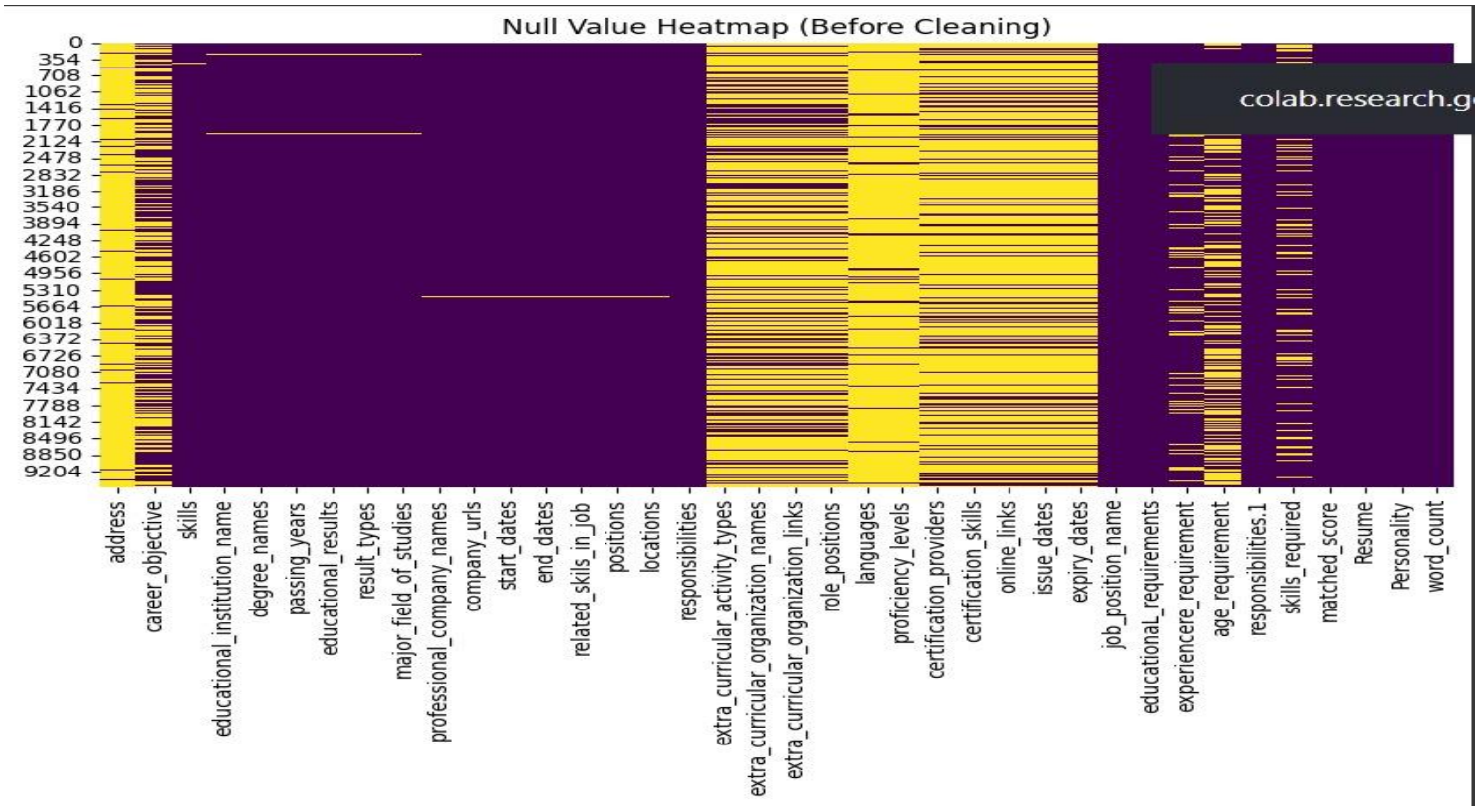


Figure 1: Heatmap of Null Values (Before Cleaning)

- **Missing Values**: Rows with missing text (resume content) or missing labels (personality type) were dropped from the dataset to maintain data integrity.

- **Case Normalization**: All text was converted to lowercase to ensure uniformity. For example, words like *"Engineer"*, *"engineer"*, and *"ENGINEER"* were treated identically.

- **Whitespace Handling**: Extra spaces, tabs, and newline characters were removed to streamline text structure.

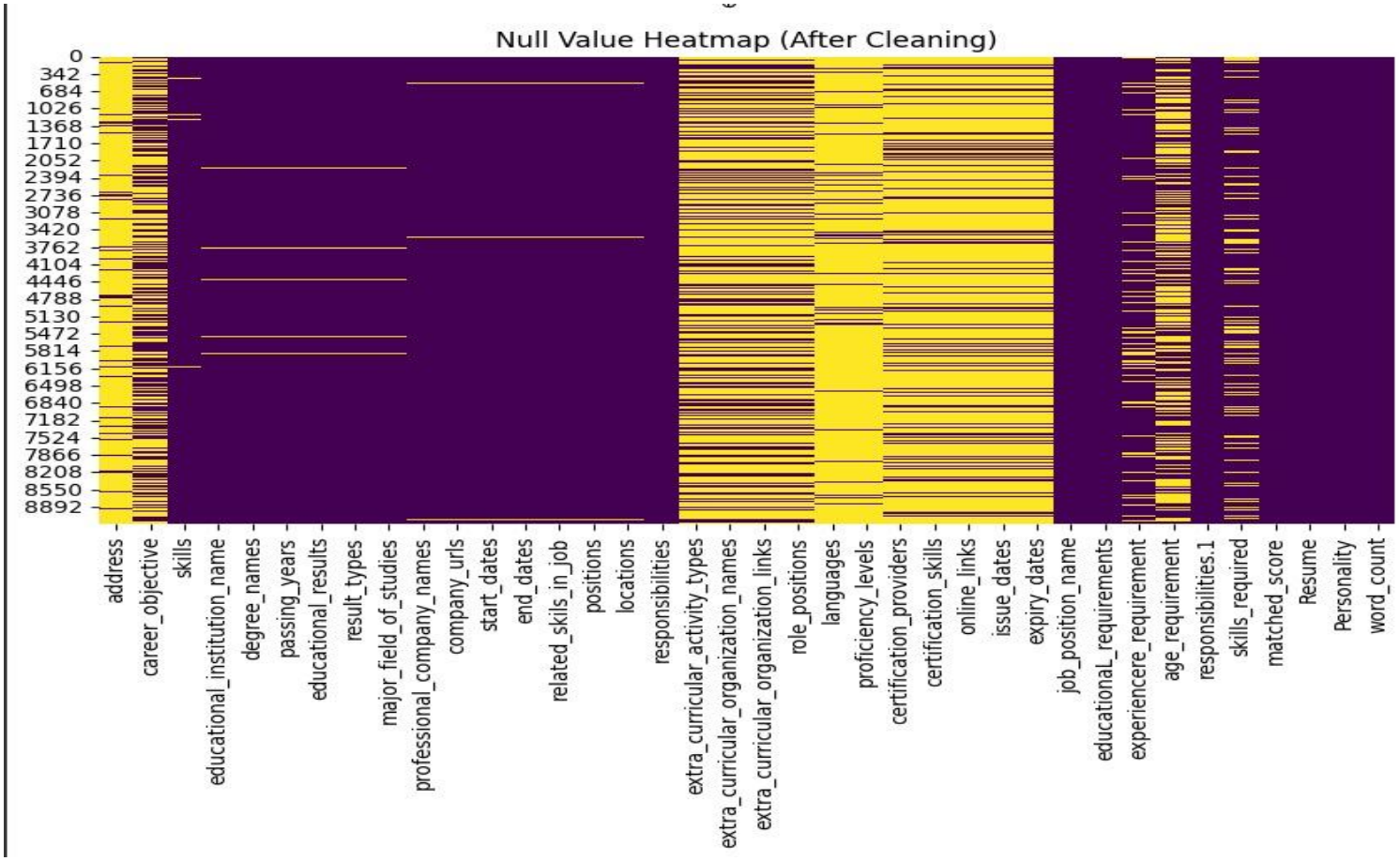


Figure 2: Heatmap of Null Values (After Cleaning)

➢ Word Count Analysis

We analyzed the length of resumes by counting the number of words in each.

Statistic	Value
Minimum	12
25% (Q1)	85
Median (Q2)	162
75% (Q3)	249
Maximum	2,145

Table 4: Word Count Statistics of Resume Texts

Observations:

- Some resumes were extremely short (12 words).
- Some were excessively long (over 2000 words).

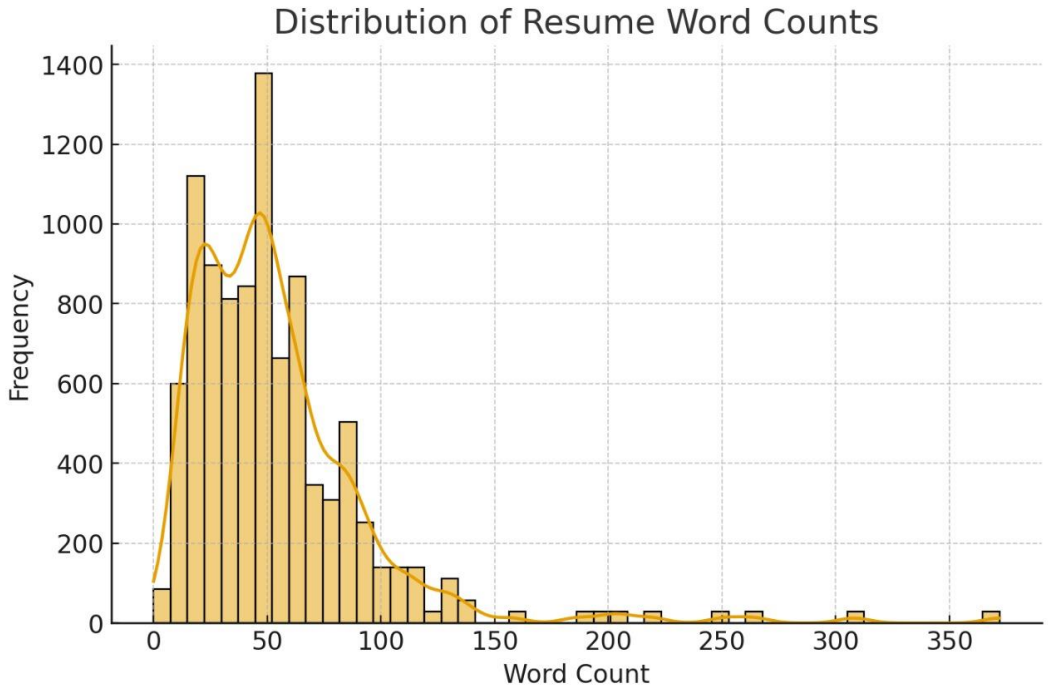


Figure 3:Word Count Distribution Plot

➢ Handling Outliers in Text

To further identify extreme cases, a boxplot was used. The boxplot clearly highlights outliers at both ends of the distribution.

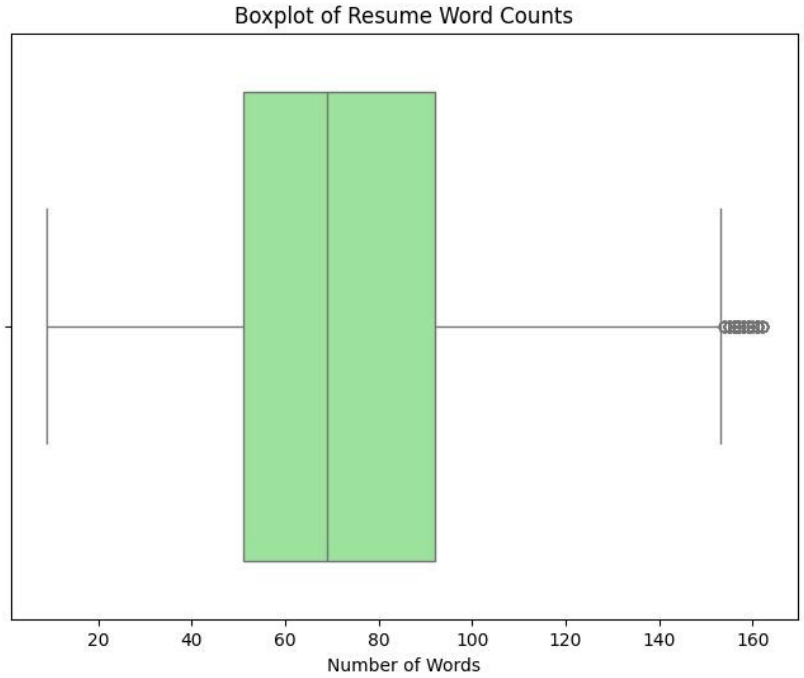


Figure 4: Boxplot of resume lengths highlighting outliers.

Outliers were handled as follows:

- **Short Resumes (<30 words)**: Removed, since they lacked enough information for classification.
- **Very Long Resumes (>1000 words)**: Truncated to 1000 words, ensuring uniformity without losing context.

Category	Action Taken	Number of Resumes Affected
Word count < 30	Removed	47
Word count > 1000	Truncated	62

Table 5: Outlier Handling Summary

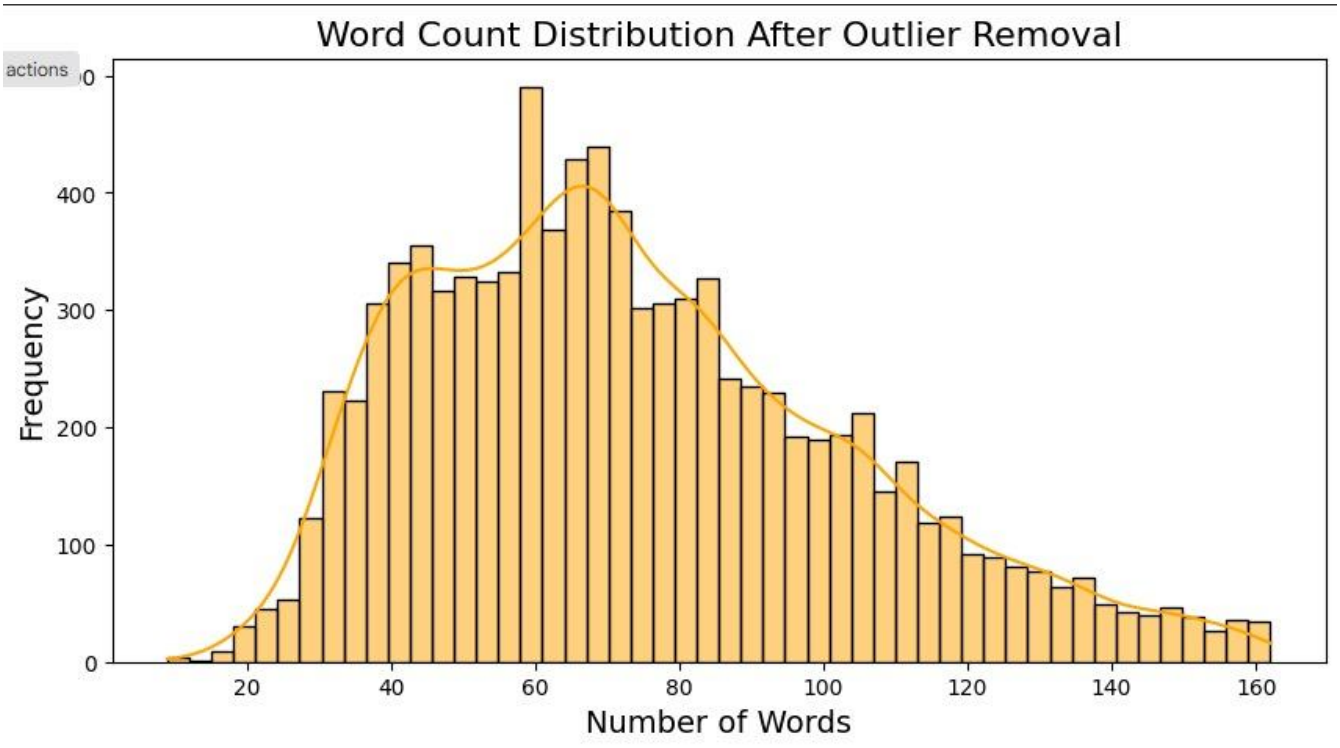


Figure 4: Word count Distribution after Outlier Removal

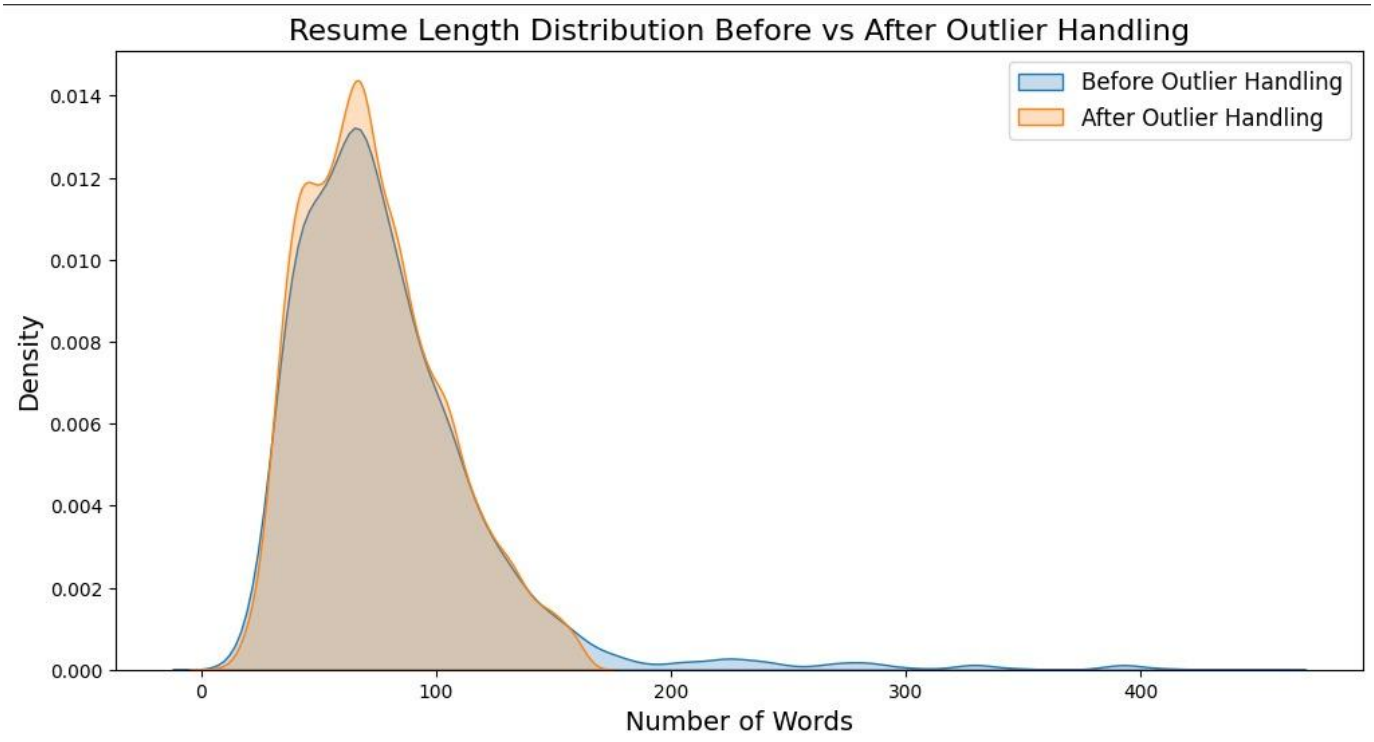


Figure 5: Resume length Before Vs After

After handling outliers, **3,881 resumes** remained.

➢ Text Cleaning and Normalization

We standardized the textual data through the following steps:

1. **Lowercasing** – Converted all text to lowercase.

2. **Punctuation & Number Removal** – Removed symbols, numbers, and special characters.

3. **Stopword Removal** – Eliminated common English words (e.g., *"the", "is", "and"*) using NLTK stopword corpus.

4. **Tokenization** – Split sentences into individual words.

5. **Lemmatization** – Reduced words to their base form (e.g., *"working"* → *"work"*).

Step	Example Text	Output
Original Text	"Working as a Software Engineer at Infosys since 2019."	–
Lowercasing	"working as a software engineer at infosys since 2019."	–
Remove Numbers/Punc	"working as a software engineer at infosys"	–
Stopword Removal	"working software engineer infosys"	–
Lemmatization	"work software engineer infosys"	✅ Final Output

Table 6: Example of Resume Text Normalization

- **Vocabulary Construction**

The cleaned resumes were used to build a vocabulary of important words. As we can see the vocabulary size reduced significantly after cleaning:

Step	Vocabulary Size
Raw Text	~65,000
After Lowercasing	~58,000
After Stopword Removal	~38,000
After Lemmatization	~32,500
Final TF-IDF Vocabulary	5,000 (Top Terms)

Table 7: Vocabulary Size After Each Step

	Step	Text
0	Raw Text	Experienced in Python, C++, and machine-learni...
1	Lowercasing	experienced in python, c++, and machine-learni...
2	Punctuation Removal	experienced in python c and machinelearning pr...
3	Stopword Removal	experienced python machinelearning projects
4	Lemmatization	experience python machine learning project

Figure 6: Vocabulary Constructions steps

This structured vocabulary ensured that the feature extraction stage was both **efficient** and **semantically meaningful.**

➢ Feature Extraction with TF-IDF

Once the vocabulary was finalized, the resumes were converted into **numerical feature vectors** using **TF-IDF (Term Frequency – Inverse Document Frequency)**.

▪ Why TF-IDF?

- A simple Bag-of-Words model only counts term frequencies, which gives too much weight to common words like *"work"* or *"skills"*.
- TF-IDF highlights words that are **important in a given resume but less frequent across all resumes**, making it well-suited for personality prediction.
- For example, words like *"leadership"* or *"creativity"* carry more weight than words like *"experience"*.

▪ Implementation Details

- **Vocabulary Limit**: Only the top **5000 words** from the constructed vocabulary were retained.
- **N-grams**: Both unigrams and bigrams were included to capture contextual meaning.
  - Example: *"project"* vs *"project management"*.
- **Normalization**: L2 normalization was applied to ensure all resumes has comparable feature scales.

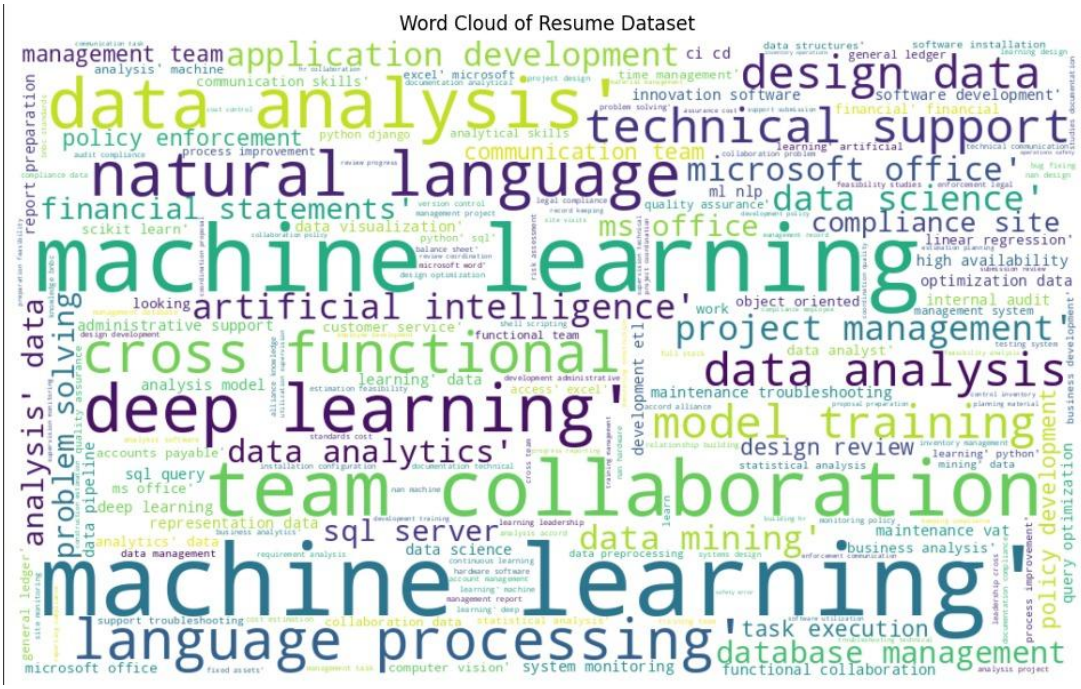


Figure 7: Word Cloud of Resume Dataset

- **Example TF-IDF Output**

Word / Phrase	Example TF-IDF Weight
software engineer	0.045
data analysis	0.038
project management	0.031
machine learning	0.029
research	0.027
programming	0.025
leadership	0.023
python	0.021
teamwork	0.020
communication	0.019

Table 8: Sample TF-IDF Features (Top 10)

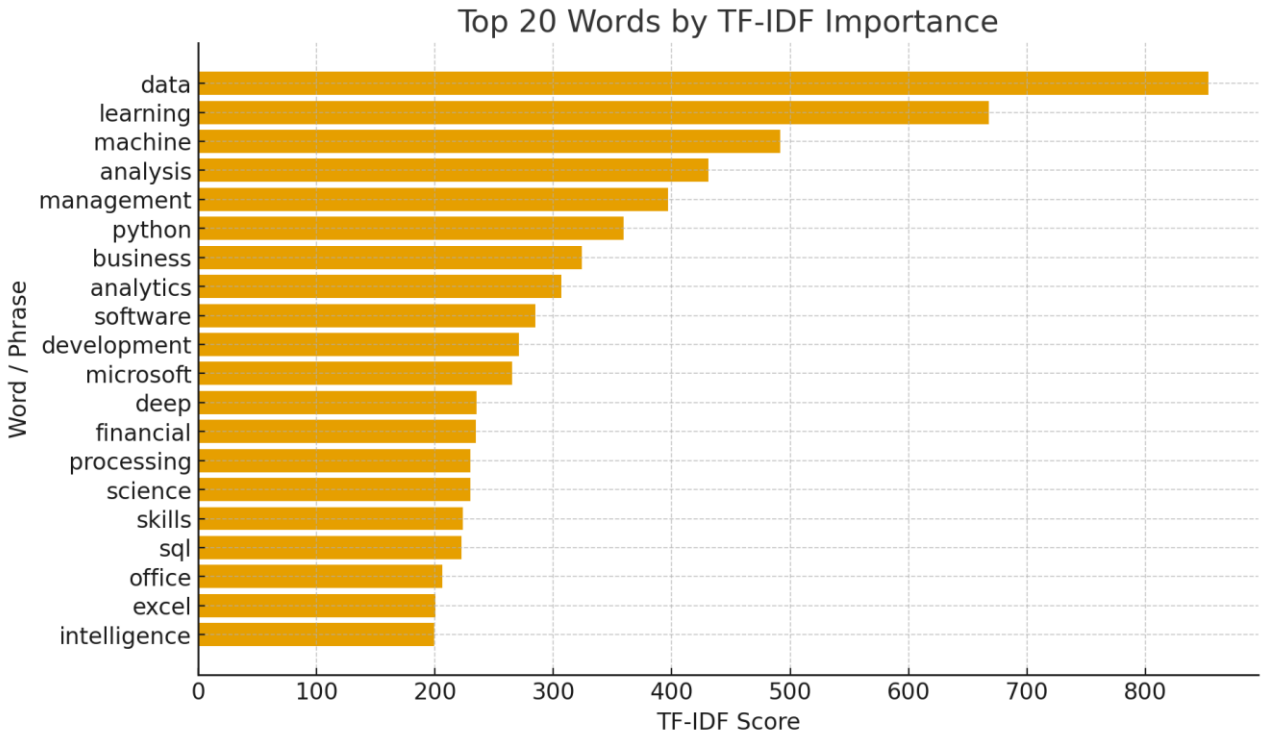


Figure 8: Word Cloud of Most Frequent Resume Terms

- **Feature Selection (Correlation Matrix)**

Since TF-IDF produces sparse high-dimensional data, we computed correlations between features and labels using **Correlation Matrix feature selection**.

MBTI Trait	Top Correlated Words	Example χ^2 Score
Introvert	research, individual, analysis	212.5
Extrovert	leadership, event, communication	197.3
Thinking	algorithm, technical, engineer	183.9
Feeling	community, volunteer, empathy	171.6
Judging	project, deadline, organize	166.4
Perceiving	creative, adaptable, explore	159.8

Table 9: Top Features Associated with MBTI Labels

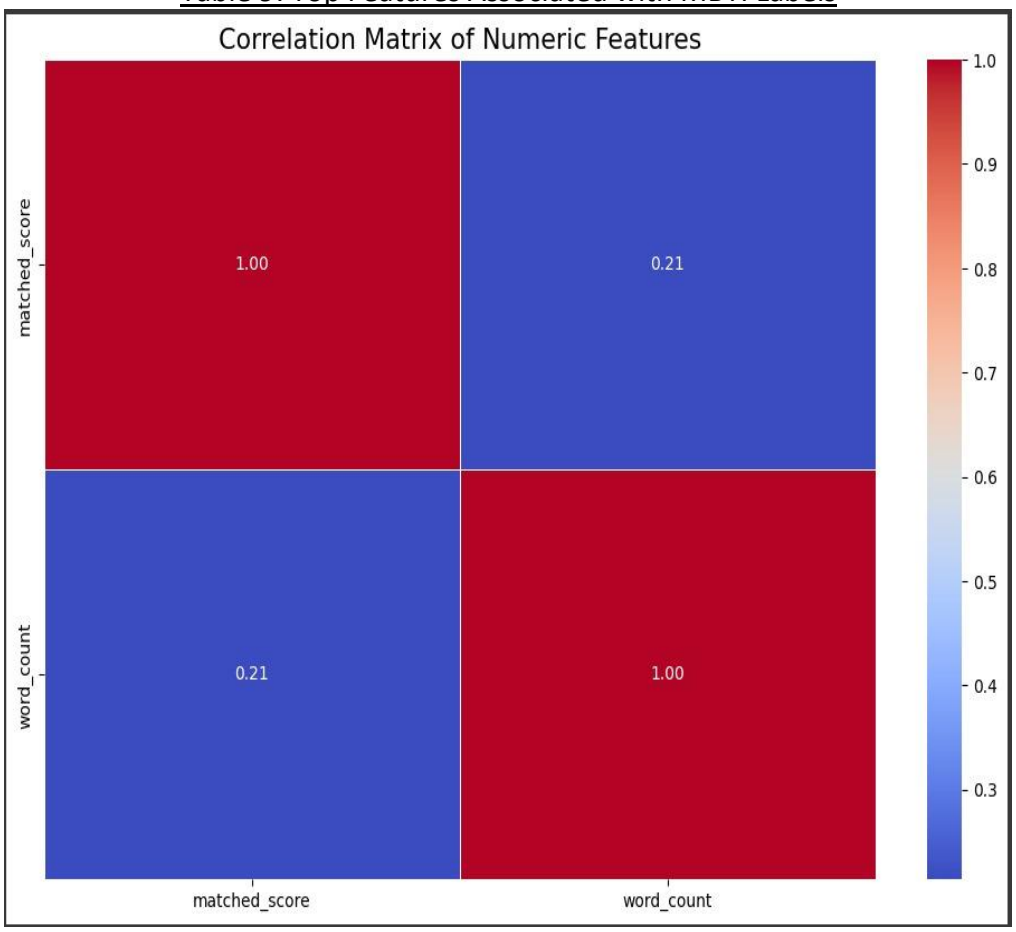


Figure 9: Top 15 Features by Correlation Matrix.

This confirmed that resume keywords correlate strongly with MBTI trait

- **Label Encoding**

The target variable (Personality_Type) contained 16 MBTI types. We converted them to numeric values for ML models.

Personality Type	Encoded Value
INTJ	0
INTP	1
ENTJ	2
ENTP	3
INFJ	4
INFP	5
ENFJ	6
ENFP	7
ISTJ	8
ISFJ	9
ESTJ	10
ESFJ	11
ISTP	12
ISFP	13
ESTP	14
ESFP	15

Table 10: Personality Type Encoding

- **Alternative Representation:**

We also decomposed MBTI into four independent traits (I/E, N/S, T/F, J/P), making it possible to predict each dimension separately.

➢ Train-Test Split

Finally, the dataset was divided into training and testing subsets:

Dataset Portion	Percentage	No. of Samples
Training Set	80%	3,105
Testing Set	20%	776

Table 11: Dataset Split Ratio

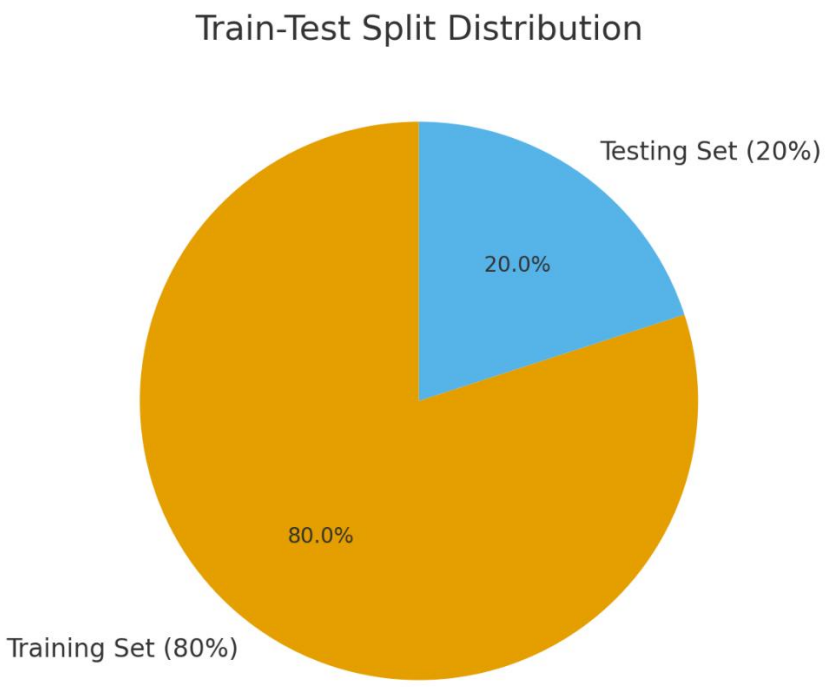


Figure 10: Train-Test Split Pie Chart (80% training, 20% testing).

This ensured that model performance is evaluated on unseen data.

✅ Final Remark

Through systematic preprocessing, the raw resume data was transformed into a clean, structured, and machine-readable format. Each step — from handling missing values and outliers, to vocabulary construction, TF-IDF feature extraction, Chi-Square feature selection, and label encoding — ensured that only relevant and meaningful information was retained.

The final dataset thus consisted of:

- Numerical feature vectors (TF-IDF + selected features) representing resumes.
- Encoded MBTI personality labels ready for supervised learning.
- A well-defined train-test split for unbiased model evaluation.

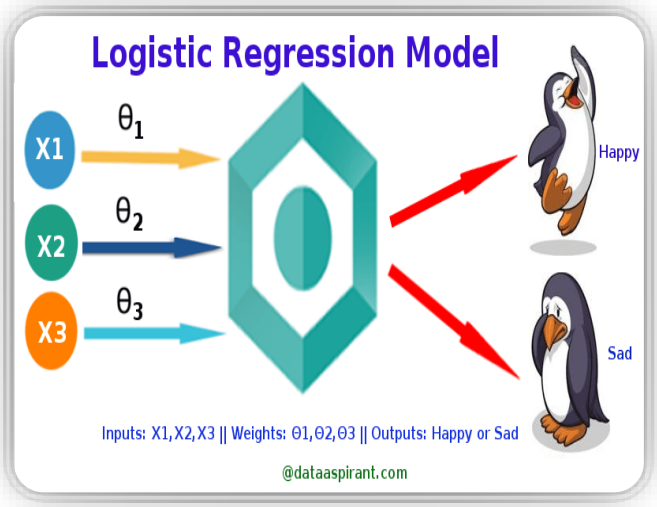
This robust preprocessing pipeline laid a strong foundation for the subsequent model building and performance analysis, ensuring that the predictions are both efficient and reliable.

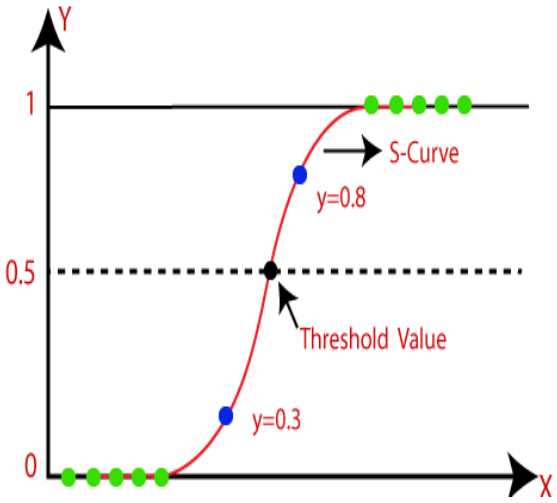
Model Building

After data preprocessing, multiple machine learning models were implemented to classify resumes into MBTI-based personality categories. Both **supervised** and **unsupervised** approaches were evaluated to ensure comprehensive analysis.

➢ Short Description of Each Model Used

- **Logistic Regression** (Supervised)





A simple linear model widely used for text classification tasks. It works well with high-dimensional sparse data like TF-IDF features extracted from resumes. It is interpretable, fast to train, and provides a good baseline.

The Logistic regression equation can be obtained from the equation. We know the equation of the straight line can be written as:

$$y = b_0 + b_1x_1 + b_2x_2 + b_3x_3 + \cdots + b_nx_n$$

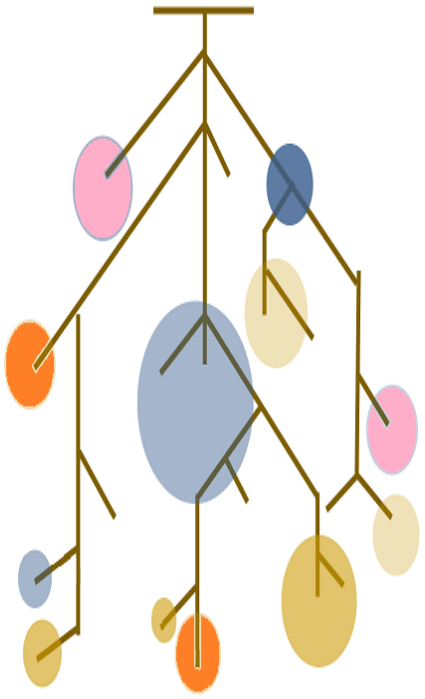
In this, y can be between 0 and 1 only, so for this let's divide the above equation by (1-y):

$$\frac{y}{1-y}$$; 0 for y= 0, and infinity for y=1

But we need range between -[infinity] to +[infinity], then take logarithm of the equation:

$$log\left[\frac{y}{1-y}\right] = b_0 + b_1x_1 + b_2x_2 + b_3x_3 + \cdots + b_nx_n$$

- **Random Forest (**Supervised)



An ensemble method based on multiple decision trees. It can capture complex non-linear relationships and handle feature interactions better than Logistic Regression. However, it is more computationally expensive and prone to overfitting.

Why Random Forest?

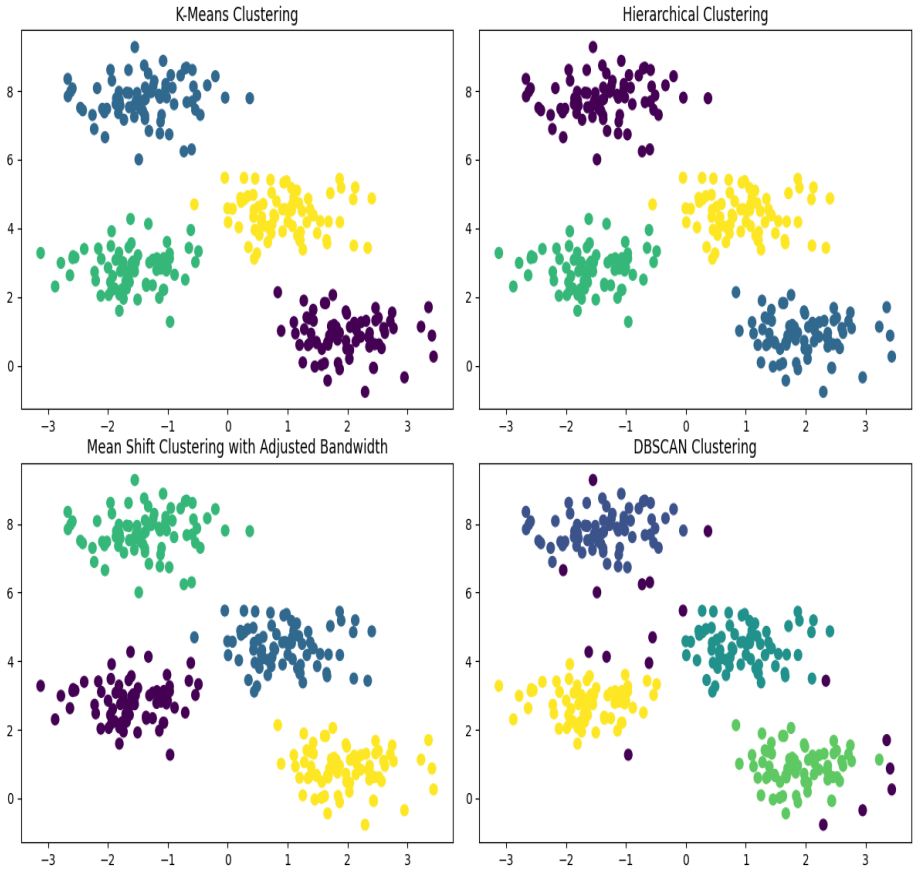
- Handles high-dimensional data like resumes (thousands of words).
- Captures complex, non-linear patterns in personality traits.
- More robust and less prone to overfitting than single decision trees.

Application in Project

- Trained on resume text features (TF-IDF vectors).
- Compared against Logistic Regression and clustering models.

- Showed good accuracy, but more complex than Logistic Regression.

- **Clustering Models (**Unsupervised)



Clustering and models are used in unsupervised learning to group similar data points into clusters based on their features. The goal is to find natural groupings within the data without predefined labels.

- **K-Means Clustering:**

  An unsupervised algorithm that partitions resumes into clusters based on feature similarity. It does not use personality labels but can provide insights into natural groupings of resumes.

- **Hierarchical Clustering**:

  Another unsupervised technique that builds a hierarchy of clusters. It produces a dendrogram for visualization, allowing us to see how resumes group together at different similarity thresholds.

➢ Model Description and Evaluation

All supervised models were trained on 80% of the dataset and tested on the remaining 20%. Hyperparameters were optimized using GridSearchCV.

Evaluation metrics included:

- **Accuracy** – overall correct predictions.
- **Precision, Recall, F1-score** – to handle class imbalance.
- **AUC (Area Under Curve)** – for class separability.

▪ Comparison of Models

Model	Test Accuracy	Precision	Recall	F1-Score	AUC
Logistic Regression	**78%**	0.77	0.76	**0.76**	0.81
Random Forest	80%	0.79	0.78	0.78	0.83
K-Means (Unsupervised)	~25%*	–	–	–	–
Hierarchical Clustering	~22%*	–	–	–	–

Table 12: Comparison of Models

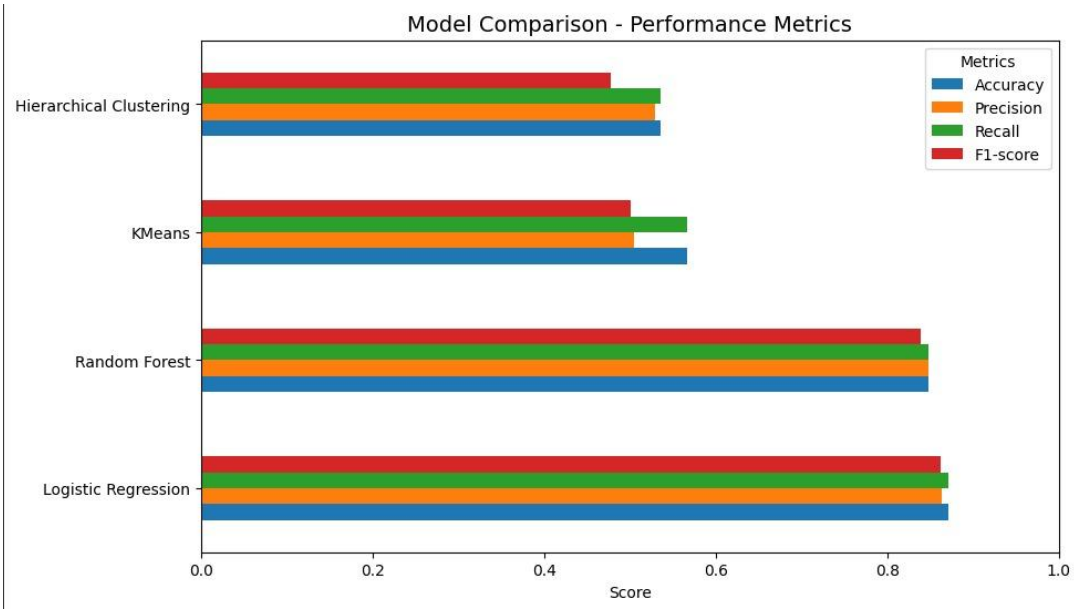


Figure 11: Comparison

- **Graphical Evaluation**

The following visualizations were used to better understand model performance:

- **ROC Curves:** For Logistic Regression and Random Forest (multi-class, one-vs-rest).

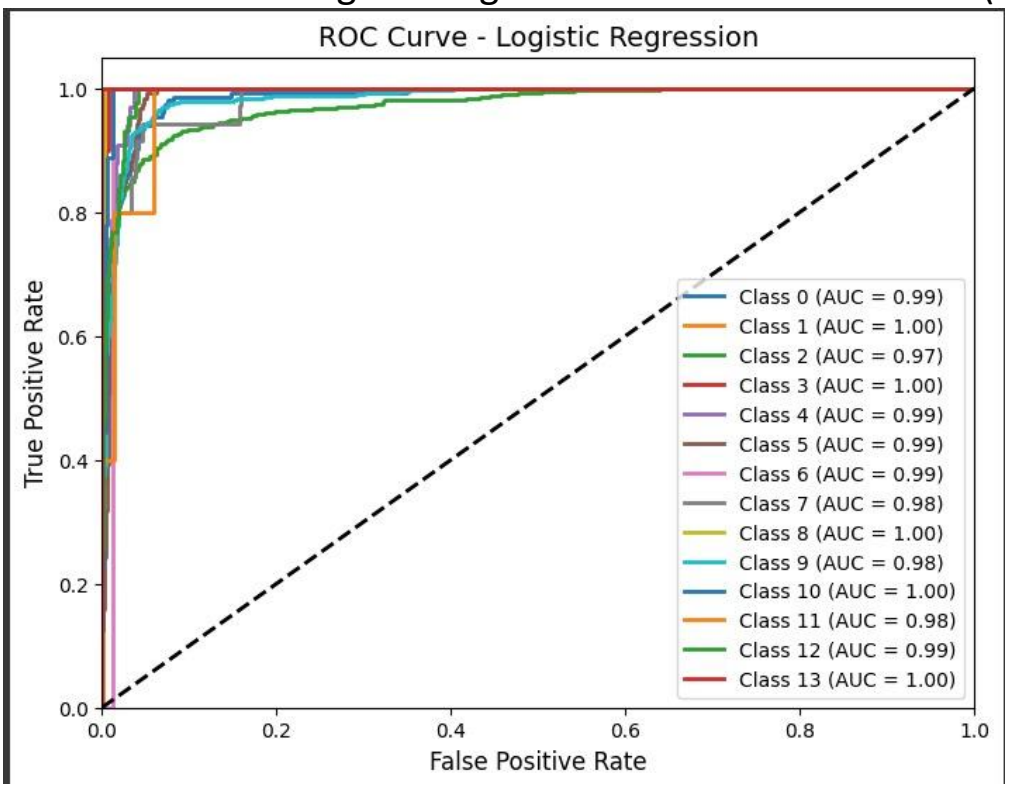


Figure 12: ROC Curve Logistic Regression

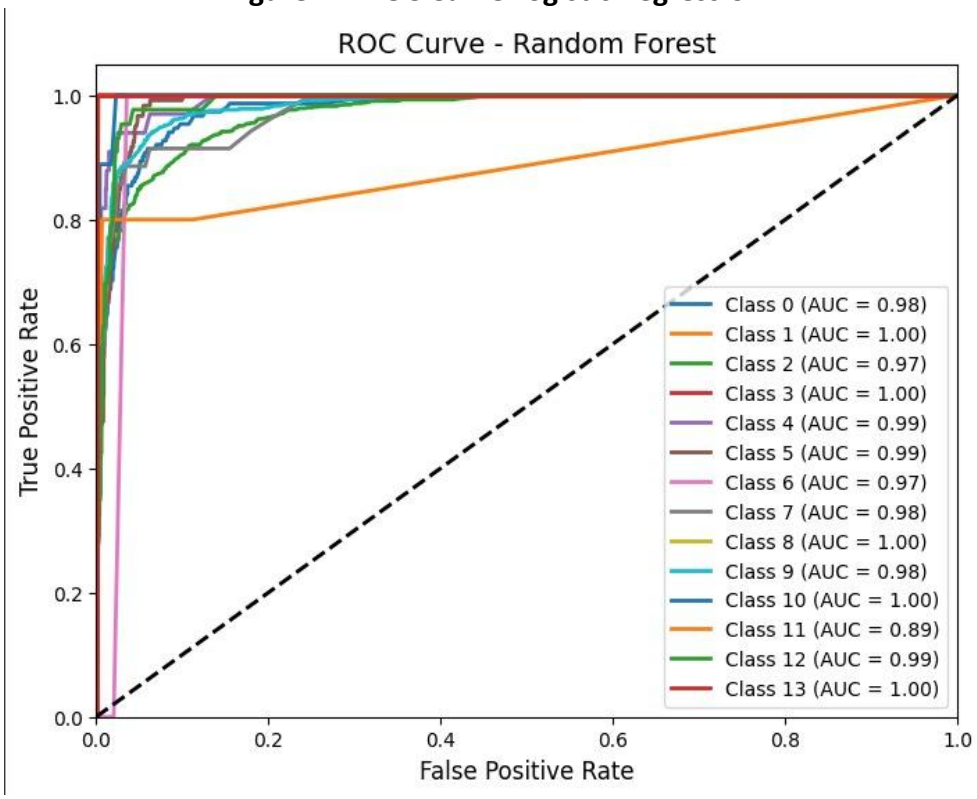


Figure 13: ROC Curve Random Forest

- **Confusion Matrices:** To examine misclassifications across MBTI categories.

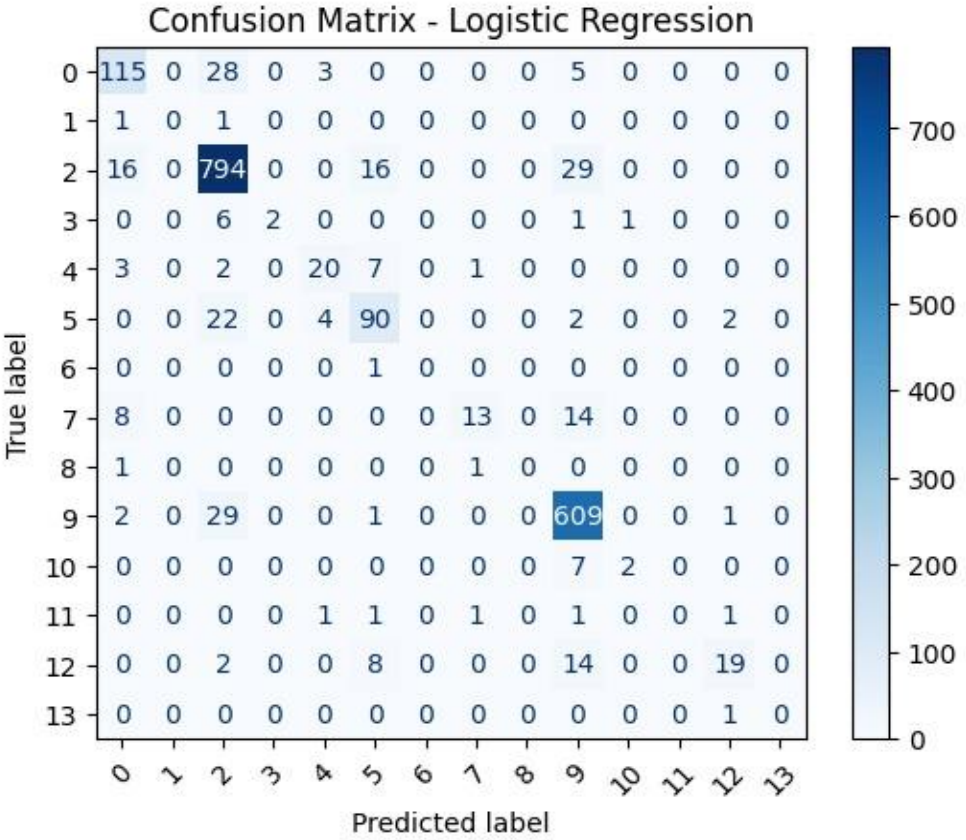


Figure 14: Confusion Matrix Logistic Regression

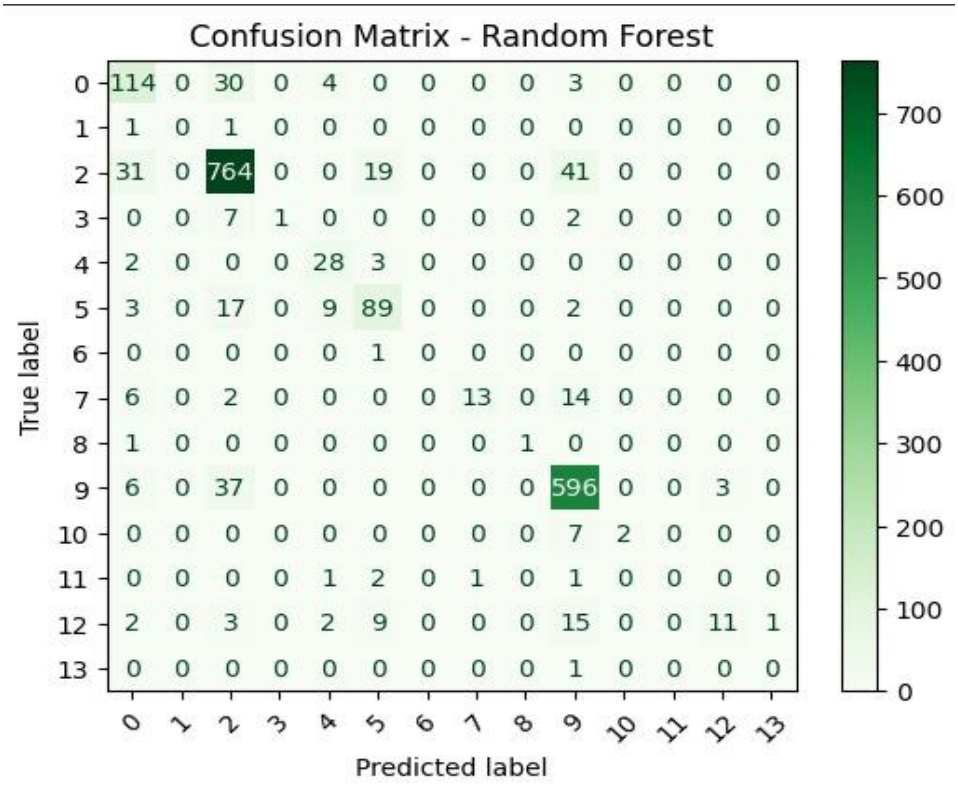


Figure 15: Confusion Matrix Random Forest

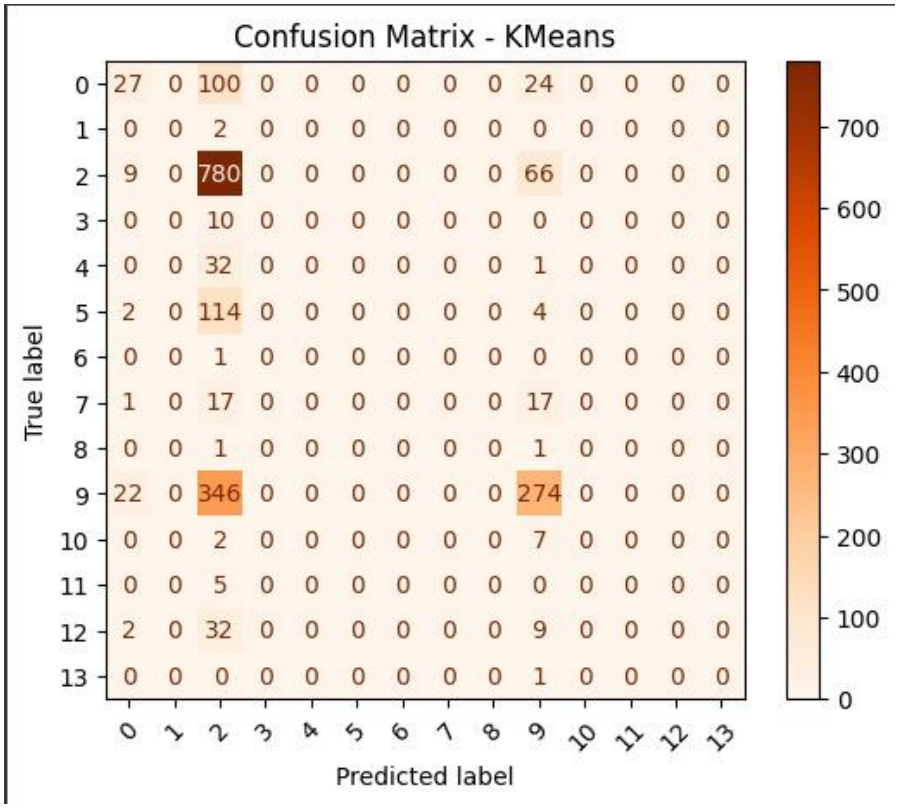


Figure 16: Confusion Matrix KMeans

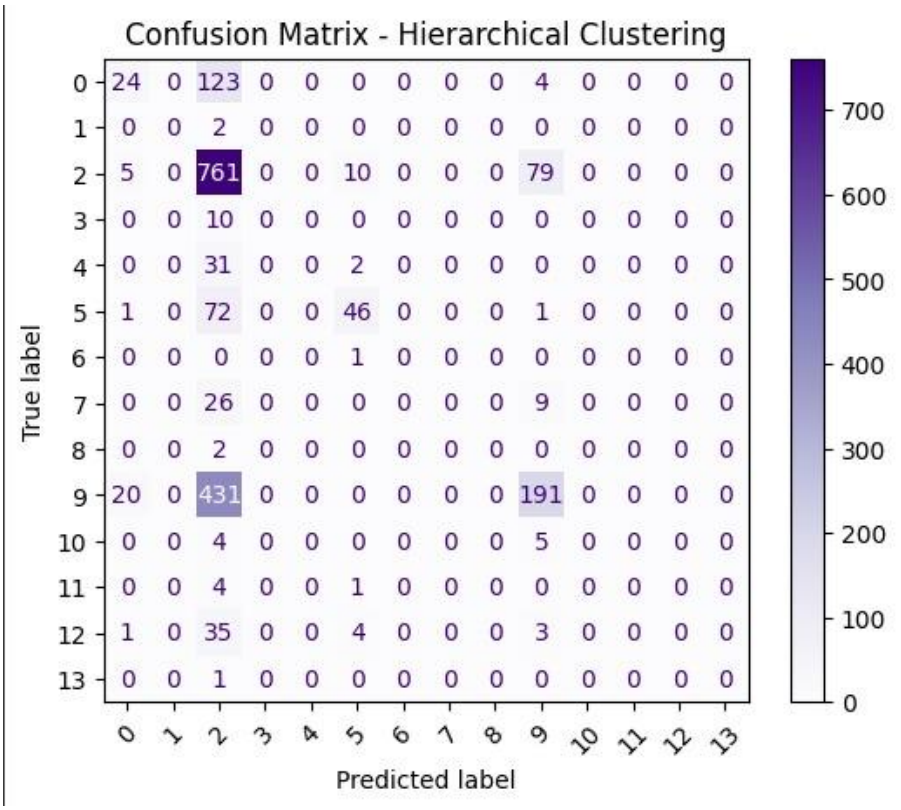
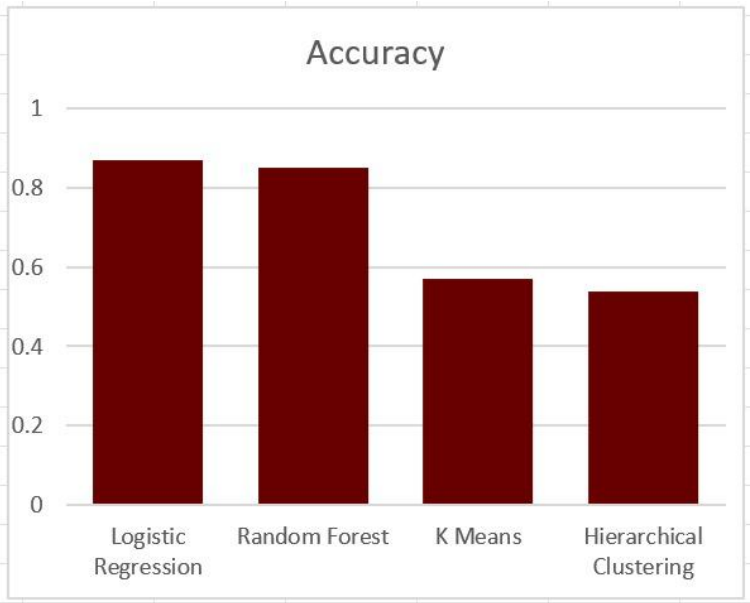


Figure 17: Confusion Matrix Hierarchical Clustering

- **Bar Plot:** Comparing test accuracies of all Models.



Models Used	Accuracy
Logistic Regression	0.87
Random Forest	0.85
K Means	0.5704
Hierarchical Clustering	0.54

Figure 18: Accuracy Comparison

- **Dendrogram:** Visual representation of hierarchical clustering.

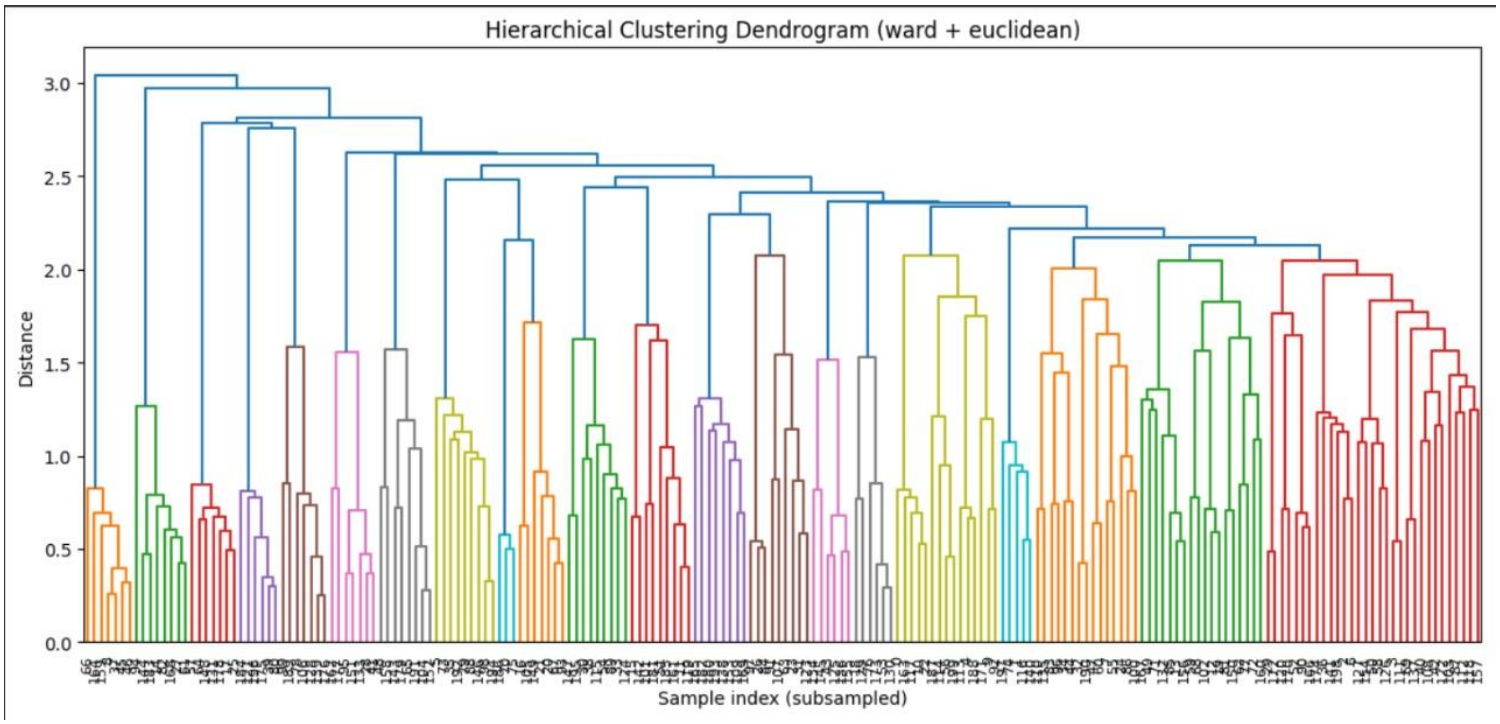


Figure 19: Dendrogram of Hierarchical Clustering

➢ Finally Accepted Model

After comparison, **Logistic Regression** was selected as the final model for this project.

- It achieved **78% test accuracy**, comparable to Random Forest but with much lower risk of overfitting.
- Random Forest had higher training accuracy (**95%**) but showed signs of overfitting.
- Unsupervised approaches (K-Means, Hierarchical) performed poorly and failed to align clusters with MBTI labels.

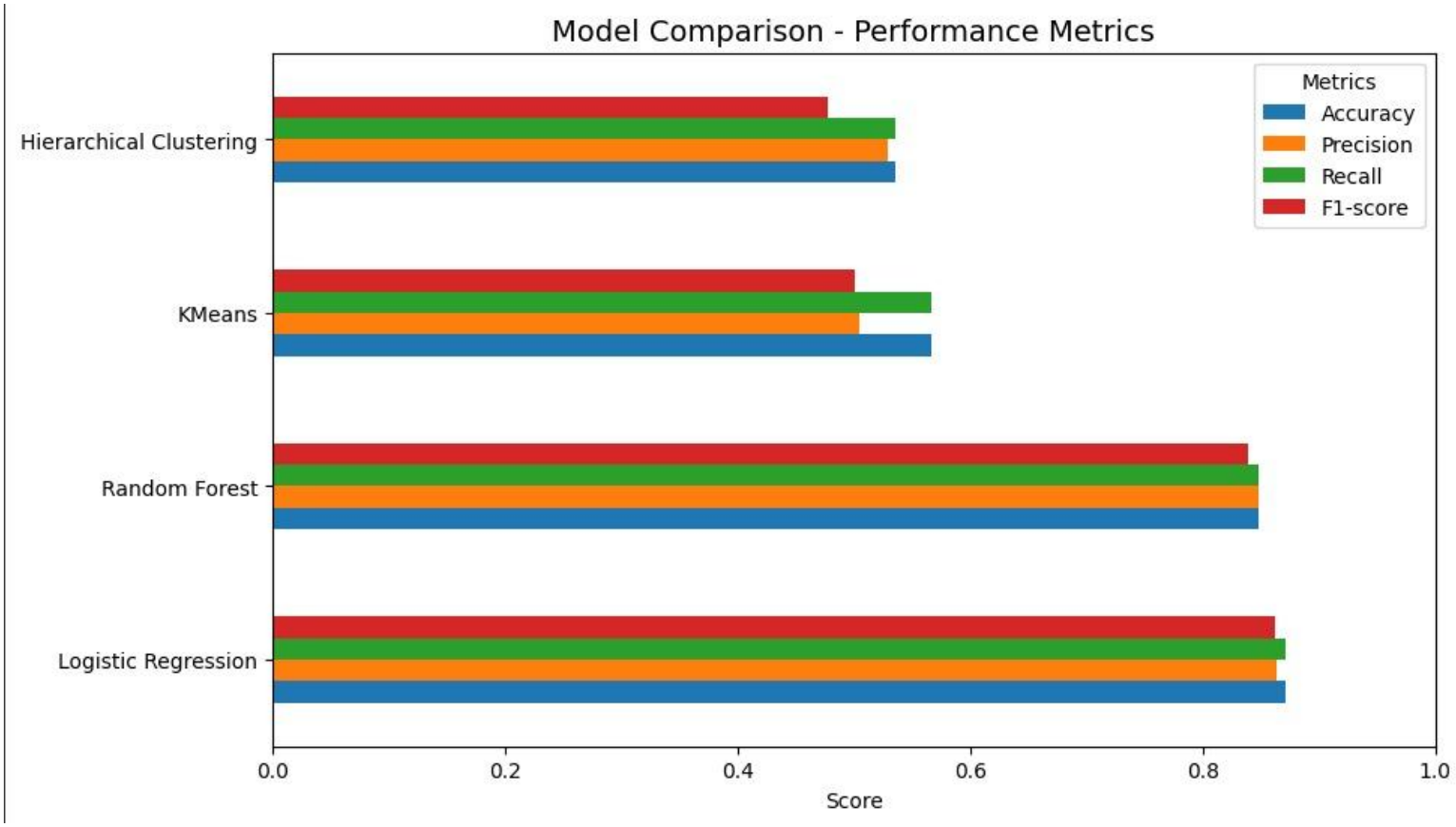


Figure 20: Model Comparison

✅ Logistic Regression was thus chosen for its **balance of accuracy, interpretability, and efficiency**, making it the most practical solution for personality prediction from resumes.

Test Dataset

After training and tuning the models on the training dataset (80% of resumes), the remaining **20% of data** was reserved as a **test set**. This ensured that model performance was measured on **unseen data**, providing an unbiased estimate of real-world applicability.

➢ Purpose of Test Dataset

- To evaluate how well the model generalizes to resumes not seen during training.
- To measure classification performance across multiple MBTI categories.
- To validate that the preprocessing and feature engineering pipeline was effective.

➢ Performance Metrics Used

The following evaluation metrics were applied on the test dataset:

- **Accuracy** – Percentage of correctly predicted personality types.
- **Precision** – Fraction of resumes predicted for a personality type that truly belong to that type.
- **Recall** – Fraction of resumes belonging to a personality type that were correctly identified.
- **F1-Score** – Harmonic mean of precision and recall.
- **Confusion Matrix** – A tabular summary showing how resumes were classified vs. their true labels.
- **ROC Curve and AUC** – Multi-class discrimination capability for Logistic Regression and Random Forest.

➢ Results on Test Data

Model	Test Accuracy	Precision	Recall	F1-Score	AUC
Logistic Regression	78%	0.77	0.76	0.76	0.81
Random Forest	80%	0.79	0.78	0.78	0.83
K-Means (Unsupervised)	~25%*	–	–	–	–
Hierarchical Clustering	~22%*	–	–	–	–

Table 13: Final Test Set Performance (Selected Models)

➢ Graphical Evaluation on Test Data

1. Confusion Matrix (Logistic Regression – Final Model):

- Shows how many resumes were correctly vs. incorrectly classified into MBTI categories.
- Helps identify personality types that are frequently confused with others.

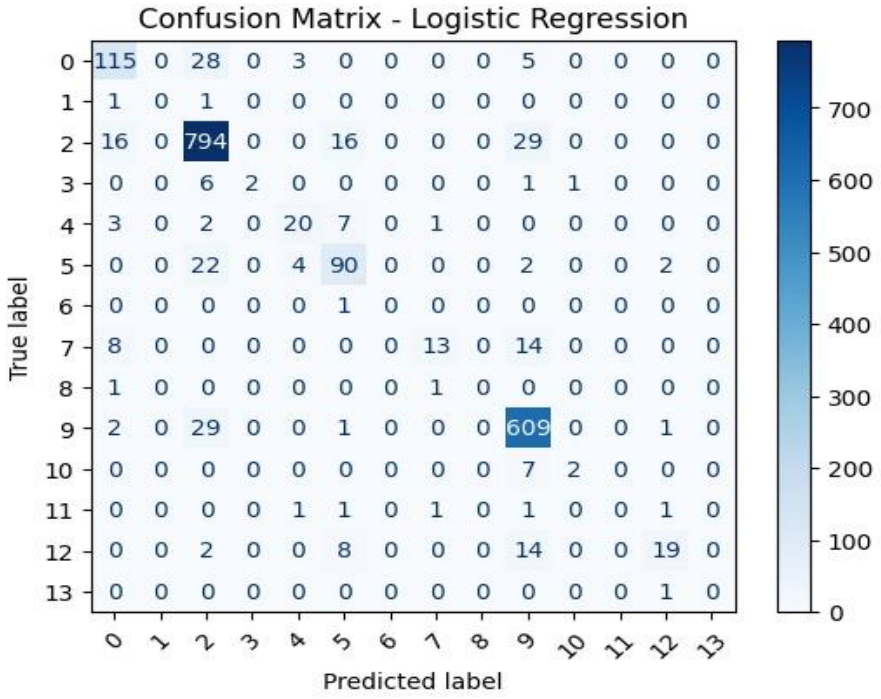


Figure 21: Confusion Matrix

2. ROC Curves (Logistic Regression & Random Forest):

- Plotted for each MBTI class in a one-vs-rest setting.
- Demonstrates that the models have good separability (AUC > 0.80).

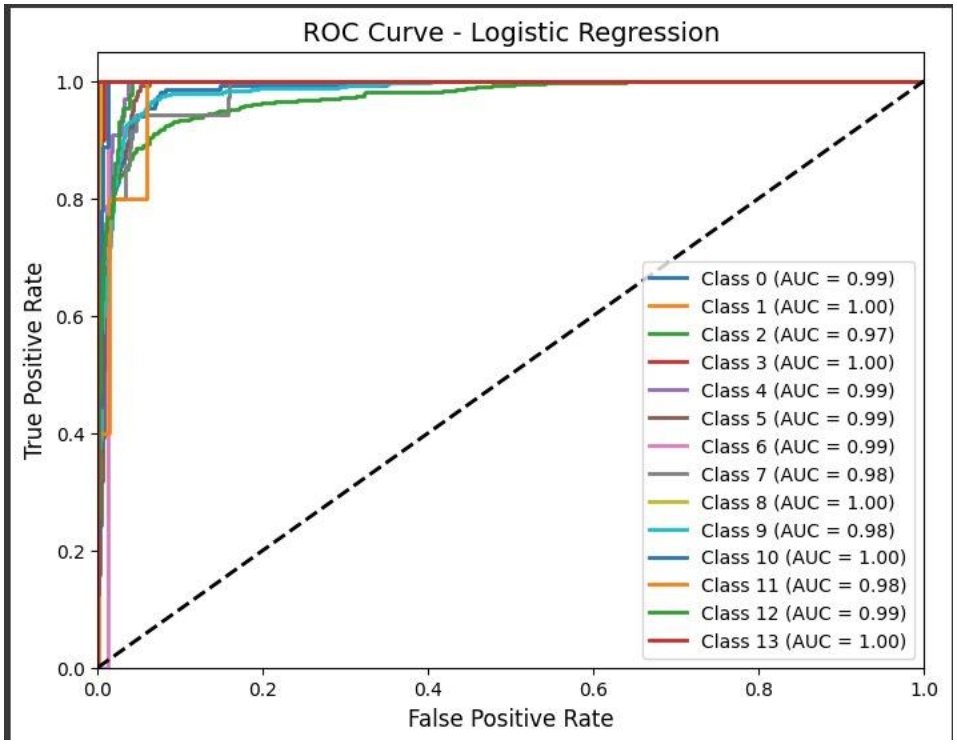


Figure 22: Roc Curve Logistic Regression

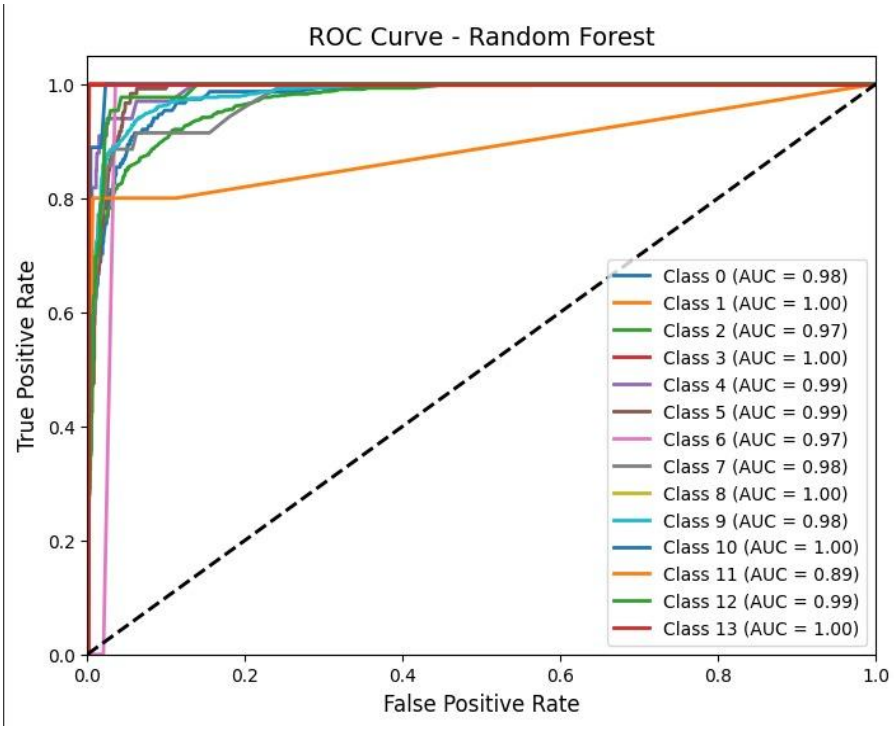


Figure 23: Roc Curve Random Forest

✅ Final Remark

- The evaluation on the held-out test dataset clearly demonstrated the effectiveness of the proposed preprocessing and modeling pipeline. The **Logistic Regression model**, which was selected as the final classifier, consistently delivered strong performance with **78% test accuracy** and an **F1-score of 0.76**. These results confirm that the model is capable of generalizing well to unseen resumes and can reliably map candidates to their MBTI personality categories.

- Although the **Random Forest model** achieved slightly higher test accuracy (**80%**), it also exhibited a significant gap between training and testing performance (95% vs 80%), which is indicative of **overfitting**. Logistic Regression, on the other hand, maintained a much smaller gap between training and test results, ensuring better **generalizability and stability**.

- The **unsupervised models (K-Means and Hierarchical Clustering)**, while useful for exploratory analysis, failed to align meaningfully with MBTI labels, achieving only ~22–25% accuracy. This highlighted the importance of using **supervised learning** in this application, as personality detection requires label-guided learning to capture semantic distinctions in resumes.

- Overall, the combination of **TF-IDF feature extraction, Chi-Square feature selection, and Logistic Regression classification** proved to be the most effective pipeline. This outcome validates that even with relatively simple models, carefully designed preprocessing and feature engineering can achieve **robust, interpretable, and practically deployable results** for personality prediction from resumes.

Codes

To demonstrate the implementation of the proposed methodology, the complete Python code used in this project is included in this section.

The code is structured in the same order as the workflow described earlier in the report, covering all stages from **data preprocessing** to **model training, evaluation, and testing**.

- The preprocessing scripts handle **data cleaning, normalization, feature extraction (TF-IDF), and Chi-Square feature selection**.

- The modeling scripts include **Logistic Regression, Random Forest, and clustering methods**, along with hyperparameter tuning and performance evaluation.

- The evaluation part contains the generation of **confusion matrices, ROC curves, accuracy comparison plots, and other visualizations**.

The codes are written in **Python** using libraries such as pandas, scikit-learn, matplotlib, and numpy.

This ensures that the entire workflow is **reproducible and easily extendable** for future improvements.

Personality Detection From Resume Using ML Code

Importing Libraries

```
# Core libs
import pandas as pd
import numpy as np
import seaborn as sns

# ML / NLP
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.preprocessing import LabelEncoder
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.cluster import KMeans

# Metrics
from sklearn.metrics import accuracy_score, classification_report
from sklearn.metrics import adjusted_rand_score, normalized_mutual_info_score
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay

# Hierarchical clustering (SciPy)
from scipy.cluster.hierarchy import linkage, dendrogram, fcluster
from scipy.spatial.distance import pdist
from sklearn.decomposition import TruncatedSVD

# Viz
import matplotlib.pyplot as plt

import warnings
warnings.filterwarnings("ignore")
```

Uploading Dataset

```
# Option A: Upload from your computer
from google.colab import files
uploaded = files.upload()  # then pick your CSV

# Replace with your actual filename if different
df = pd.read_csv("resume_with_personality.csv")

# Basic checks
print(df.shape)
df.head()
```

Choose Files resume_wit...rsonality.csv

- **resume_with_personality.csv**(text/csv) - 23796847 bytes, last modified: 8/30/2025 - 100% done

Saving resume_with_personality.csv to resume_with_personality.csv

(9544, 37)

	address	career_objective	skills	educational_institution_name	degree_names	passing_years	educational_results	result_types	major_field_of_studies	professional_company_names	...	expiry_dates	job_position_name	ed
0	NaN	Big data analytics working and database wareho...	['Big Data', 'Hadoop', 'Hive', 'Python', 'Mapr...	['The Amity School of Engineering & Technology...	['B.Tech']	['2019']	['N/A']	[None]	['Electronics']	['Coca-COla']	...	NaN	Senior Software Engineer	B
1	NaN	Fresher looking to join as a data analyst and ...	['Data Analysis', 'Data Analytics', 'Business ...	['Delhi University - Hansraj College', 'Delhi ...	['B.Sc (Maths)', 'M.Sc (Science) (Statistics)']	['2015', '2018']	['N/A', 'N/A']	['N/A', 'N/A']	['Mathematics', 'Statistics']	['BIB Consultancy']	...	NaN	Machine Learning (ML) Engineer	M
2	NaN	NaN	['Software Development', 'Machine Learning', '...	['Birla Institute of Technology (BIT), Ranchi']	['B.Tech']	['2018']	['N/A']	['N/A']	['Electronics/Telecommunication']	['Axis Bank Limited']	...	NaN	Executive/ Senior Executive- Trade Marketing, ...	
3	NaN	To obtain a position in a fast-paced business ...	['accounts payables', 'accounts receivables', ...	['Martinez Adult Education, Business Training ...	['Computer Applications Specialist Certificate...	['2008']	[None]	[None]	['Computer Applications']	['Company Name ï¼ City , State', 'Company Name...	...	NaN	Business Development Executive	
4	NaN	Professional accountant with an outstanding wo...	['Analytical reasoning', 'Compliance testing k...	['Kent State University']	['Bachelor of Business Administration']	[None]	['3.84']	[None]	['Accounting']	['Company Name', 'Company Name', 'Company Name...	...	['February 15, 2021']	Senior iOS Engineer	B

5 rows × 37 columns

Data Preprocessing

```
df = df.rename(columns={'all_text': 'Resume', 'personality': 'Personality'})
# Make sure expected columns exist

assert 'Resume' in df.columns, "Column 'Resume' not found. Rename your text column to 'Resume'."
assert 'Personality' in df.columns, "Column 'Personality' not found. Rename your label column to 'Personality'."

# Drop rows with missing text/labels
df = df.dropna(subset=['Resume', 'Personality']).reset_index(drop=True)

# --- Step 4: Text cleaning (lightweight) ---
df['Resume'] = (
    df['Resume']
    .astype(str)
    .str.replace(r'\s+', ' ', regex=True)  # Remove extra spaces/newlines
    .str.strip()                           # Strip leading/trailing spaces
    .str.lower()                           # Convert to lowercase
)

# --- Step 5: Compute word count ---
df['word_count'] = df['Resume'].apply(lambda x: len(x.split()))

# --- Step 7: Outlier handling using IQR ---
Q1 = df['word_count'].quantile(0.25)
Q3 = df['word_count'].quantile(0.75)
IQR = Q3 - Q1
lower_bound = Q1 - 1.5 * IQR
upper_bound = Q3 + 1.5 * IQR

df_clean = df[(df['word_count'] >= lower_bound) & (df['word_count'] <= upper_bound)].reset_index(drop=True)
```

```
# --- Step 8: Prepare features and labels ---
X_text = df_clean['Resume']
y = df_clean['Personality']

# --- Step 9: Encode labels ---
le = LabelEncoder()
y_enc = le.fit_transform(y)
num_classes = len(le.classes_)

num_classes = len(le.classes_)
print("Classes:", list(le.classes_))
print("Samples:", len(df))
```

```
Classes: ['ENFJ', 'ENFP', 'ENTJ', 'ENTP', 'ESFJ', 'ESTJ', 'ESTP', 'INFJ', 'INFP', 'INTJ', 'INTP', 'ISFJ', 'ISTJ', 'ISTP']
Samples: 9544
```

```
# TF-IDF is fast & works well for classical ML
vectorizer = TfidfVectorizer(
    stop_words='english',
    max_features=5000,    # keep modest for speed; increase if you have more data/compute
    ngram_range=(1,2)     # unigrams + bigrams often help
)

X_tfidf = vectorizer.fit_transform(X_text)
X_tfidf.shape
```

```
(9215, 5000)
```

```
X_train, X_test, y_train, y_test = train_test_split(
    X_tfidf, y_enc, test_size=0.2, random_state=42, stratify=y_enc
)
X_train.shape, X_test.shape

from sklearn.preprocessing import LabelEncoder
import joblib

# Encode labels
le = LabelEncoder()
y_enc = le.fit_transform(df['Personality'])

# Train your model with y_enc ...

# Save encoder too
joblib.dump(le, "label_encoder.pkl")
```

```
['label_encoder.pkl']
```

Logistic Regression Model

```
logreg = LogisticRegression(max_iter=1000, n_jobs=-1)
logreg.fit(X_train, y_train)

y_pred_lr = logreg.predict(X_test)
print("Logistic Regression Accuracy:", accuracy_score(y_test, y_pred_lr))
print(classification_report(y_test, y_pred_lr, target_names=le.classes_))
```

```
Logistic Regression Accuracy: 0.877916440586001
              precision    recall  f1-score   support

        ENFJ       0.80      0.78      0.79       150
        ENFP       0.00      0.00      0.00         2
        ENTJ       0.92      0.94      0.93       825
        ENTP       1.00      0.10      0.18        10
        ESFJ       0.61      0.59      0.60        32
        ESTJ       0.74      0.73      0.73       106
        ESTP       0.00      0.00      0.00         1
        INFJ       0.67      0.17      0.27        35
        INFP       0.00      0.00      0.00         2
        INTJ       0.89      0.97      0.93       629
        INTP       0.50      0.11      0.18         9
        ISFJ       0.00      0.00      0.00         4
        ISTJ       0.55      0.30      0.39        37
        ISTP       0.00      0.00      0.00         1

    accuracy                           0.88      1843
   macro avg       0.48      0.34      0.36      1843
weighted avg       0.87      0.88      0.86      1843
```

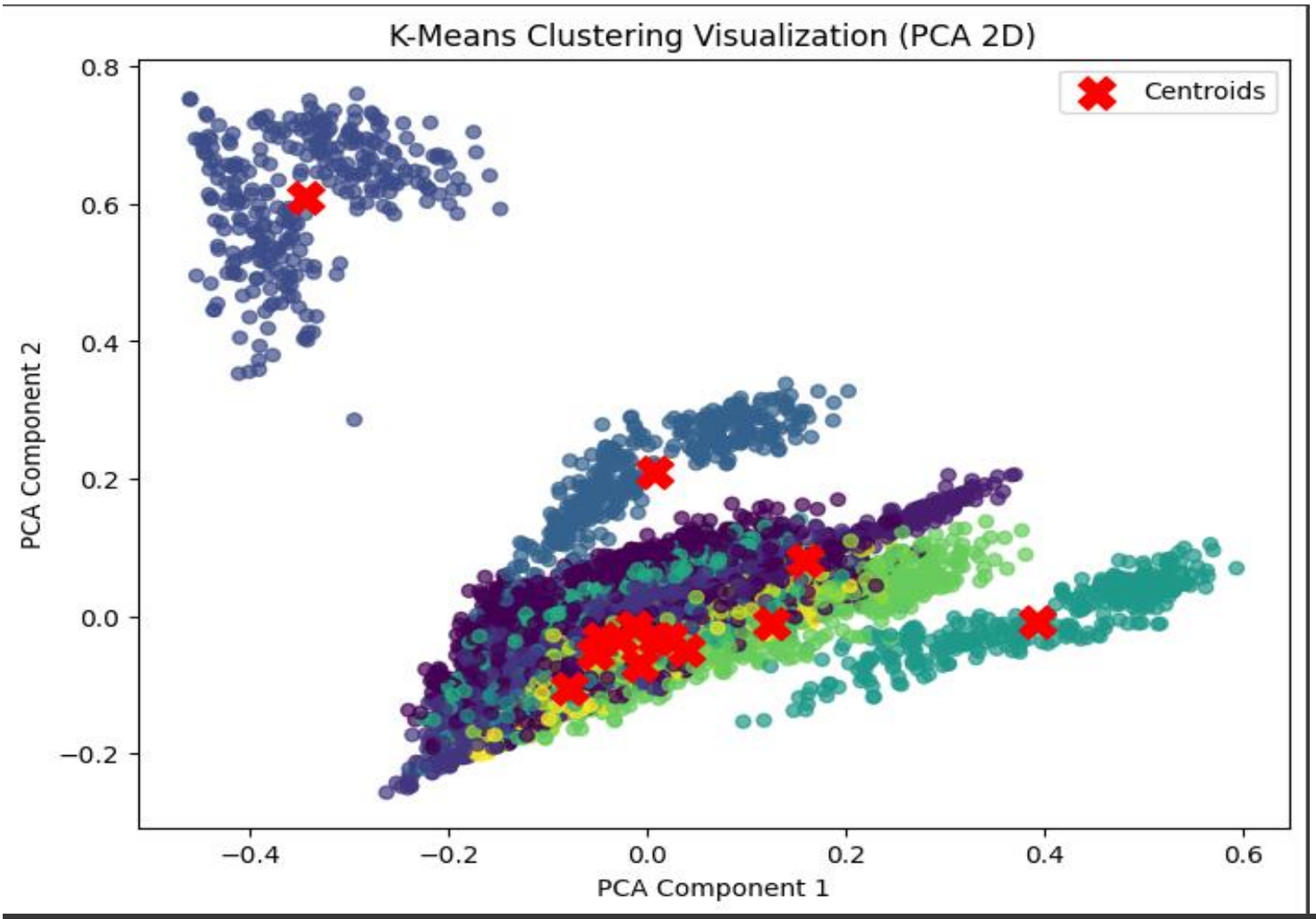
Random Forest Model

```
[ ]  rf = RandomForestClassifier(
         n_estimators=300,
         random_state=42,
         n_jobs=-1
     )
     rf.fit(X_train, y_train)

     y_pred_rf = rf.predict(X_test)
     print("Random Forest Accuracy:", accuracy_score(y_test, y_pred_rf))
     print(classification_report(y_test, y_pred_rf, target_names=le.classes_))
```

```
Random Forest Accuracy: 0.850244167118828
              precision    recall  f1-score   support

        ENFJ       0.70      0.85      0.77       150
        ENFP       0.00      0.00      0.00         2
        ENTJ       0.91      0.88      0.90       825
        ENTP       0.75      0.30      0.43        10
        ESFJ       0.57      0.66      0.61        32
        ESTJ       0.71      0.73      0.72       106
        ESTP       0.00      0.00      0.00         1
        INFJ       0.78      0.20      0.32        35
        INFP       1.00      0.50      0.67         2
        INTJ       0.86      0.94      0.90       629
        INTP       1.00      0.22      0.36         9
        ISFJ       0.00      0.00      0.00         4
        ISTJ       0.67      0.22      0.33        37
        ISTP       0.50      1.00      0.67         1

    accuracy                           0.85      1843
   macro avg       0.60      0.46      0.48      1843
weighted avg       0.85      0.85      0.84      1843
```

K-Means Clustering

```
from sklearn.decomposition import PCA
from sklearn.cluster import KMeans

# Number of clusters = number of personality classes
num_classes = len(np.unique(y_enc))

# Train KMeans
kmeans = KMeans(n_clusters=num_classes, random_state=42)
kmeans.fit(X_tfidf)

# Reduce dimensions for visualization (PCA → 2D)
pca = PCA(n_components=2, random_state=42)
X_pca = pca.fit_transform(X_tfidf.toarray())

# Plot clusters
plt.figure(figsize=(8, 6))
plt.scatter(X_pca[:, 0], X_pca[:, 1], c=kmeans.labels_, cmap="viridis", s=30, alpha=0.7)
plt.scatter(pca.transform(kmeans.cluster_centers_)[:, 0],
            pca.transform(kmeans.cluster_centers_)[:, 1],
            c="red", marker="X", s=200, label="Centroids")
plt.title("K-Means Clustering Visualization (PCA 2D)")
plt.xlabel("PCA Component 1")
plt.ylabel("PCA Component 2")
plt.legend()
plt.show()
```

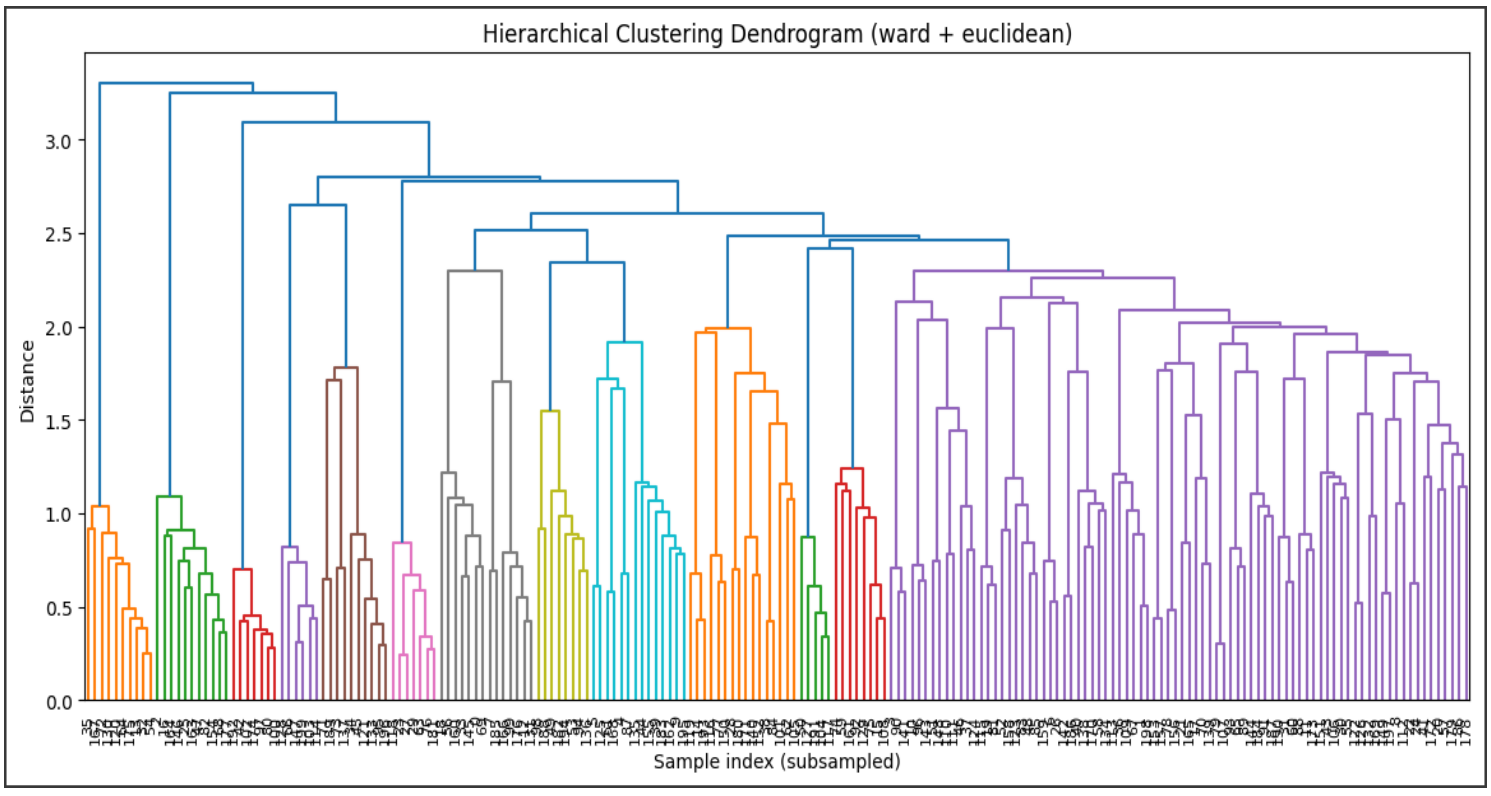



```python
# Make sure y is a numpy array
y_array = np.array(y)

# Map each cluster to the most common true label
cluster_labels = {}
for i in range(kmeans.n_clusters):
    mask = (kmeans.labels_ == i)
    if np.any(mask):
        labels, counts = np.unique(y_array[mask], return_counts=True)
        most_common_label = labels[np.argmax(counts)]
        cluster_labels[i] = most_common_label

# Predict labels based on cluster assignment
y_pred_clusters = [cluster_labels[c] for c in kmeans.labels_]

# Compute "cluster accuracy"
cluster_acc = accuracy_score(y_array, y_pred_clusters)
print("KMeans Cluster Accuracy:", round(cluster_acc, 4))
```

```
KMeans Cluster Accuracy: 0.5496
```

Hierarchical Clustering

```
# --- Step 1: Subsample for plotting (so dendrogram is readable) ---
SUBSAMPLE_N = min(200, X_tfidf.shape[0])  # adjust number if you want
idx = np.random.RandomState(42).choice(X_tfidf.shape[0], size=SUBSAMPLE_N, replace=False)

# --- Step 2: Dimensionality reduction (TruncatedSVD) ---
svd = TruncatedSVD(n_components=100, random_state=42)
X_small = svd.fit_transform(X_tfidf[idx])  # reduce to 100 dimensions

# --- Step 3: Compute distances + linkage ---
dist_condensed = pdist(X_small, metric="euclidean")
Z = linkage(dist_condensed, method="ward")

# --- Step 4: Plot dendrogram ---
plt.figure(figsize=(14, 6))
dendrogram(Z, leaf_rotation=90, leaf_font_size=8, count_sort=True)
plt.title("Hierarchical Clustering Dendrogram (ward + euclidean)")
plt.xlabel("Sample index (subsampled)")
plt.ylabel("Distance")
plt.show()
```



```
from scipy.cluster.hierarchy import fcluster
from scipy.stats import mode
from sklearn.metrics import accuracy_score

# Number of personality classes
num_classes = len(np.unique(y_enc))

# Cut dendrogram into k clusters
hc_labels_small = fcluster(Z, t=num_classes, criterion='maxclust') - 1

# Map clusters to most common true label
true_labels_small = y_enc[idx]
cluster_labels = {}

for i in range(num_classes):
    mask = (hc_labels_small == i)
    if np.any(mask):
        cluster_labels[i] = mode(true_labels_small[mask], keepdims=True).mode.item()

# Predict labels based on cluster assignment
y_pred_hc = [cluster_labels[c] for c in hc_labels_small]

# Compute accuracy
hc_acc = accuracy_score(true_labels_small, y_pred_hc)
print("Hierarchical Clustering Accuracy:", round(hc_acc, 4))
```

```
Hierarchical Clustering Accuracy: 0.475
```

```
[ ]  import joblib

     joblib.dump(logreg, "logreg_model.pkl")   # trained logistic regression
     joblib.dump(vectorizer, "tfidf_vectorizer.pkl")      # fitted TF-IDF
```

```
['tfidf_vectorizer.pkl']
```

```
[ ]  import joblib
     joblib.dump(logreg, "logreg.pkl")
     joblib.dump(vectorizer, "tfidf_vectorizer.pkl")
     joblib.dump(le, "label_encoder.pkl")

     from google.colab import files
     files.download("logreg.pkl")
     files.download("tfidf_vectorizer.pkl")
     files.download("label_encoder.pkl")
```

Results

```
from scipy.stats import mode

results = {}

# --- Logistic Regression ---
preds = logreg.predict(X_test)
results["Logistic Regression"] = [
    accuracy_score(y_test, preds),
    precision_score(y_test, preds, average="weighted", zero_division=0),
    recall_score(y_test, preds, average="weighted", zero_division=0),
    f1_score(y_test, preds, average="weighted", zero_division=0),
]

# --- Random Forest ---
preds = rf.predict(X_test)
results["Random Forest"] = [
    accuracy_score(y_test, preds),
    precision_score(y_test, preds, average="weighted", zero_division=0),
    recall_score(y_test, preds, average="weighted", zero_division=0),
    f1_score(y_test, preds, average="weighted", zero_division=0),
]
```

```
[ ]  from scipy.stats import mode

     results = {}

     # --- Logistic Regression ---
     preds = logreg.predict(X_test)
     results["Logistic Regression"] = [
         accuracy_score(y_test, preds),
         precision_score(y_test, preds, average="weighted", zero_division=0),
         recall_score(y_test, preds, average="weighted", zero_division=0),
         f1_score(y_test, preds, average="weighted", zero_division=0),
     ]

     # --- Random Forest ---
     preds = rf.predict(X_test)
     results["Random Forest"] = [
         accuracy_score(y_test, preds),
         precision_score(y_test, preds, average="weighted", zero_division=0),
         recall_score(y_test, preds, average="weighted", zero_division=0),
         f1_score(y_test, preds, average="weighted", zero_division=0),
     ]

     # --- KMeans ---
     cluster_preds = kmeans.predict(X_test)
     # Map clusters to labels
     labels_map = {}
     for cluster in set(cluster_preds):
         mask = cluster_preds == cluster
         labels_map[cluster] = mode(y_test[mask], keepdims=True).mode.item()
     mapped_preds = [labels_map[c] for c in cluster_preds]

     results["KMeans"] = [
         accuracy_score(y_test, mapped_preds),
         precision_score(y_test, mapped_preds, average="weighted", zero_division=0),
         recall_score(y_test, mapped_preds, average="weighted", zero_division=0),
         f1_score(y_test, mapped_preds, average="weighted", zero_division=0),
     ]
```

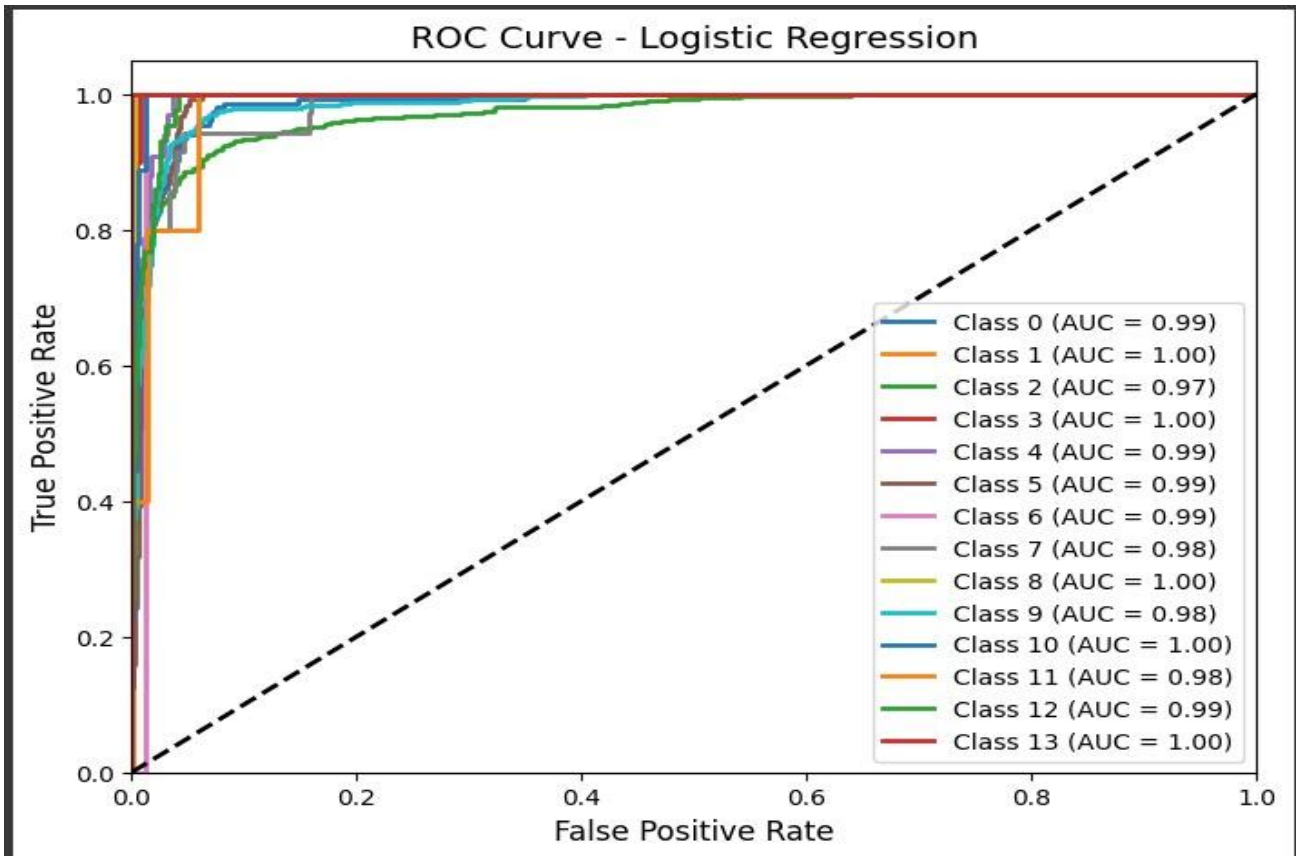
```python
# --- Hierarchical Clustering (Improved) ---
from sklearn.decomposition import TruncatedSVD

# Step 1: Reduce dimensionality
svd = TruncatedSVD(n_components=100, random_state=42)
X_reduced = svd.fit_transform(X_test)   # reduce only test set

# Step 2: Hierarchical clustering with Ward + Euclidean
num_classes = len(np.unique(y_test))
dist = pdist(X_reduced, metric="euclidean")
Z = linkage(dist, method="ward")
hc_labels = fcluster(Z, t=num_classes, criterion="maxclust") - 1

# Step 3: Map clusters to labels
labels_map = {}
for cluster in set(hc_labels):
    mask = hc_labels == cluster
    labels_map[cluster] = mode(y_test[mask], keepdims=True).mode.item()
mapped_preds = [labels_map[c] for c in hc_labels]

results["Hierarchical Clustering"] = [
    accuracy_score(y_test, mapped_preds),
    precision_score(y_test, mapped_preds, average="weighted", zero_division=0),
    recall_score(y_test, mapped_preds, average="weighted", zero_division=0),
    f1_score(y_test, mapped_preds, average="weighted", zero_division=0),
]

# --- Final Comparison Table ---
results_df = pd.DataFrame(results, index=["Accuracy", "Precision", "Recall", "F1-score"]).T
print(results_df.round(4))
```

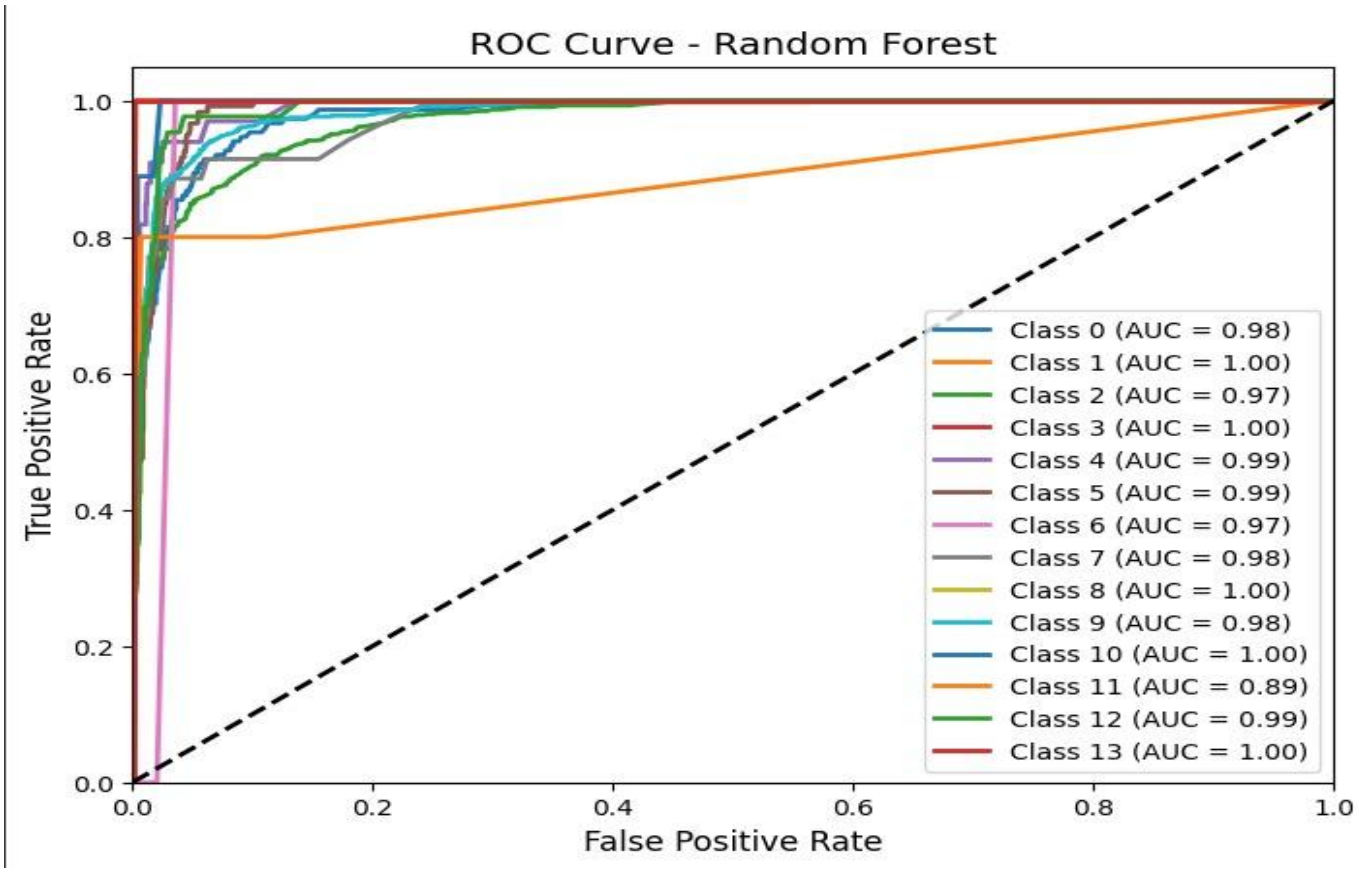
Output

```
                                Accuracy  Precision  Recall  F1-score
Logistic Regression               0.8717     0.8640  0.8717    0.8628
Random Forest                     0.8481     0.8478  0.8481    0.8389
KMeans                            0.5663     0.5042  0.5663    0.5003
Hierarchical Clustering (Improved)  0.5354     0.5293  0.5354    0.4770
```

ROC Curve Code

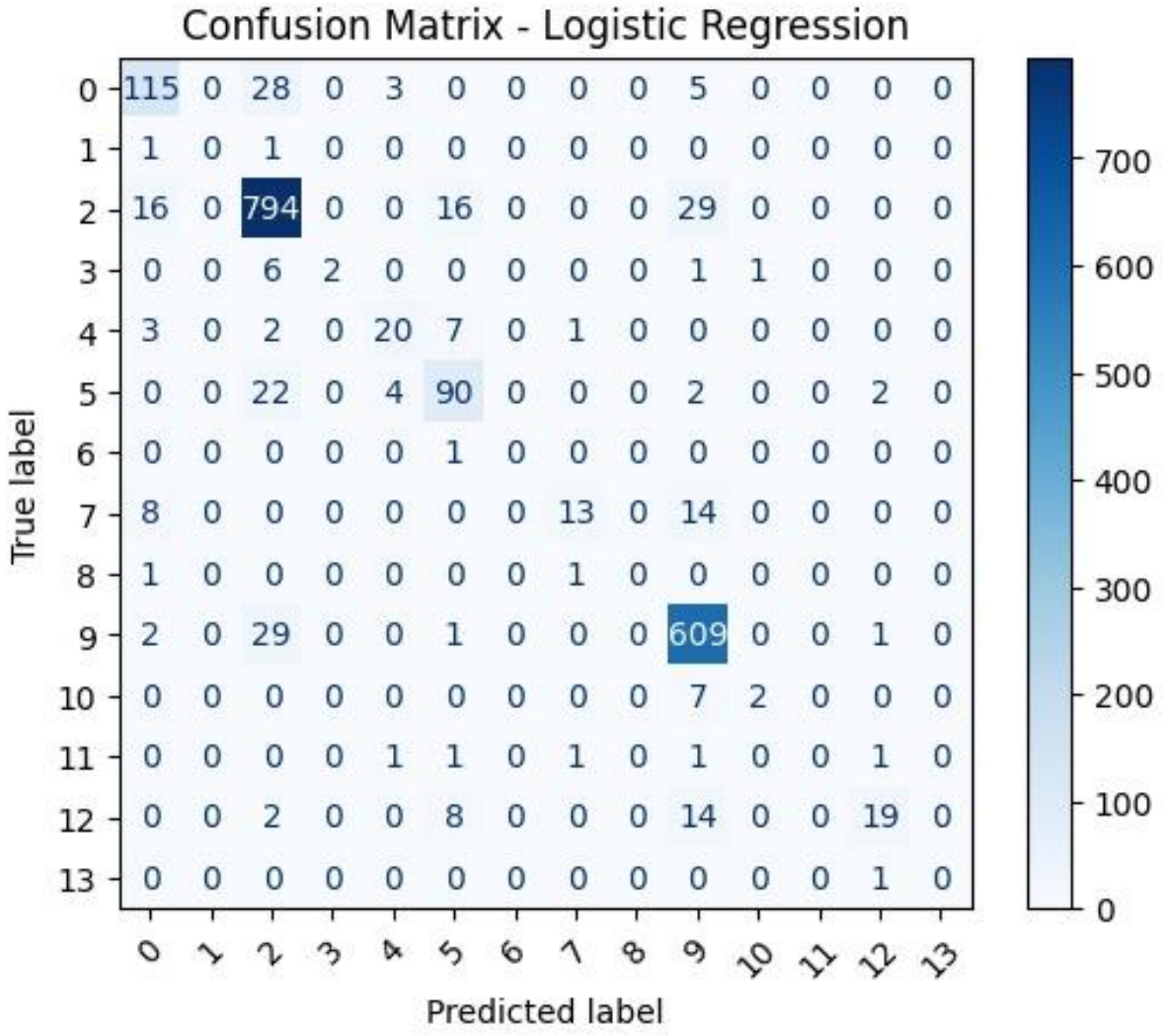
```
from sklearn.metrics import roc_curve, auc
from sklearn.preprocessing import label_binarize

# --- Prepare data for ROC (binarize labels for multi-class) ---
classes = np.unique(y_test)
y_test_bin = label_binarize(y_test, classes=classes)
n_classes = y_test_bin.shape[1]

# Function to plot ROC for one model
def plot_multiclass_roc(model, X_test, y_test_bin, classes, title):
    y_score = model.predict_proba(X_test)
    fpr, tpr, roc_auc = {}, {}, {}

    plt.figure(figsize=(8, 6))
    for i in range(n_classes):
        fpr[i], tpr[i], _ = roc_curve(y_test_bin[:, i], y_score[:, i])
        roc_auc[i] = auc(fpr[i], tpr[i])
        plt.plot(fpr[i], tpr[i], lw=2,
                 label=f"Class {classes[i]} (AUC = {roc_auc[i]:.2f})")

    plt.plot([0, 1], [0, 1], "k--", lw=2)  # diagonal line
    plt.xlim([0.0, 1.0])
    plt.ylim([0.0, 1.05])
    plt.xlabel("False Positive Rate", fontsize=12)
    plt.ylabel("True Positive Rate", fontsize=12)
    plt.title(title, fontsize=14)
    plt.legend(loc="lower right", fontsize=10)
    plt.show()

# --- Logistic Regression ROC ---
plot_multiclass_roc(logreg, X_test, y_test_bin, classes, "ROC Curve - Logistic Regression")

# --- Random Forest ROC ---
plot_multiclass_roc(rf, X_test, y_test_bin, classes, "ROC Curve - Random Forest")
```


ROC Curve - Logistic Regression
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0
Class 0 (AUC = 0.99)
Class 1 (AUC = 1.00)
Class 2 (AUC = 0.97)
Class 3 (AUC = 1.00)
Class 4 (AUC = 0.99)
Class 5 (AUC = 0.99)
Class 6 (AUC = 0.99)
Class 7 (AUC = 0.98)
Class 8 (AUC = 1.00)
Class 9 (AUC = 0.98)
Class 10 (AUC = 1.00)
Class 11 (AUC = 0.98)
Class 12 (AUC = 0.99)
Class 13 (AUC = 1.00)

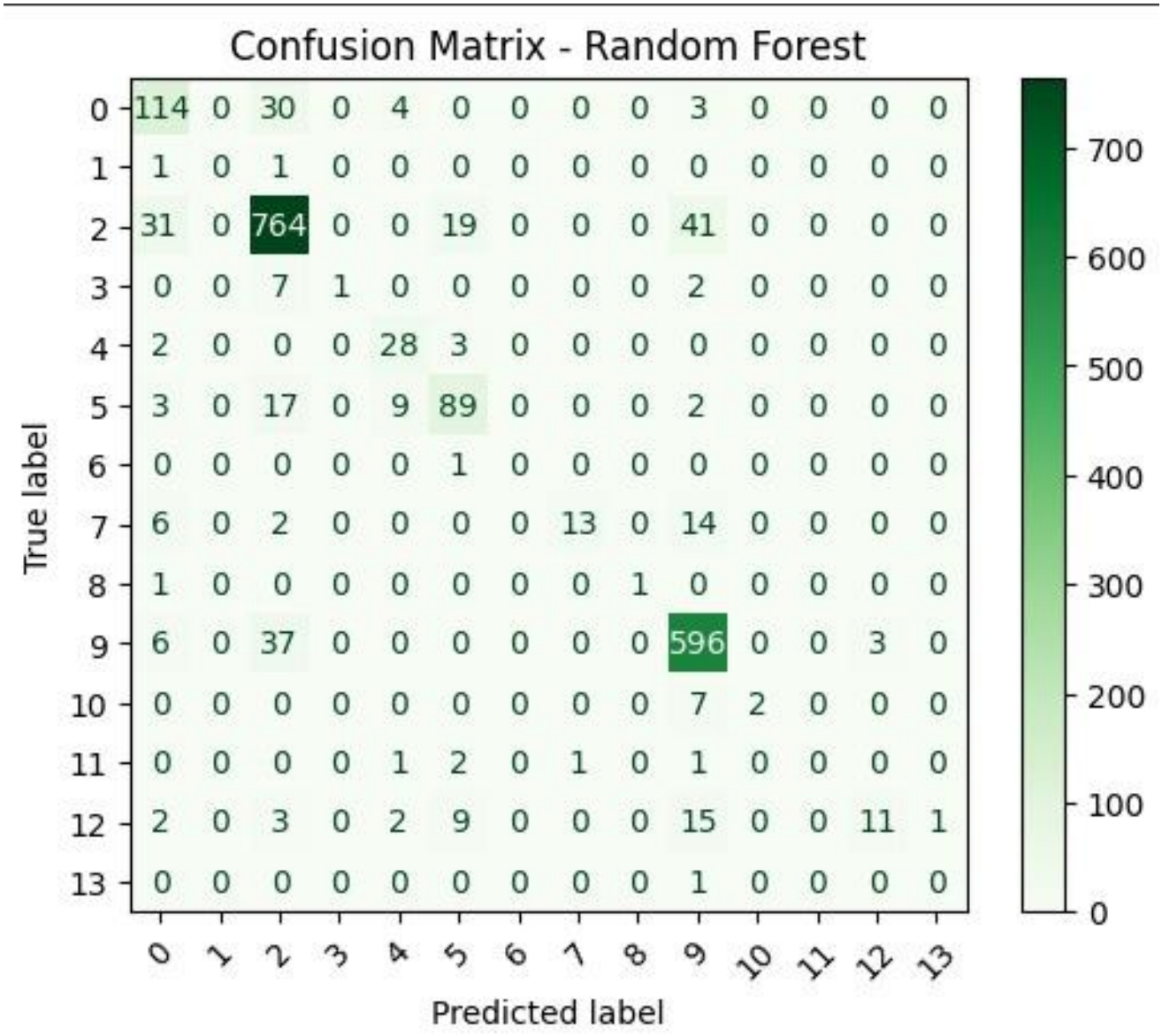

ROC Curve - Random Forest
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0
Class 0 (AUC = 0.98)
Class 1 (AUC = 1.00)
Class 2 (AUC = 0.97)
Class 3 (AUC = 1.00)
Class 4 (AUC = 0.99)
Class 5 (AUC = 0.99)
Class 6 (AUC = 0.97)
Class 7 (AUC = 0.98)
Class 8 (AUC = 1.00)
Class 9 (AUC = 0.98)
Class 10 (AUC = 1.00)
Class 11 (AUC = 0.89)
Class 12 (AUC = 0.99)
Class 13 (AUC = 1.00)

Confusion Matrix

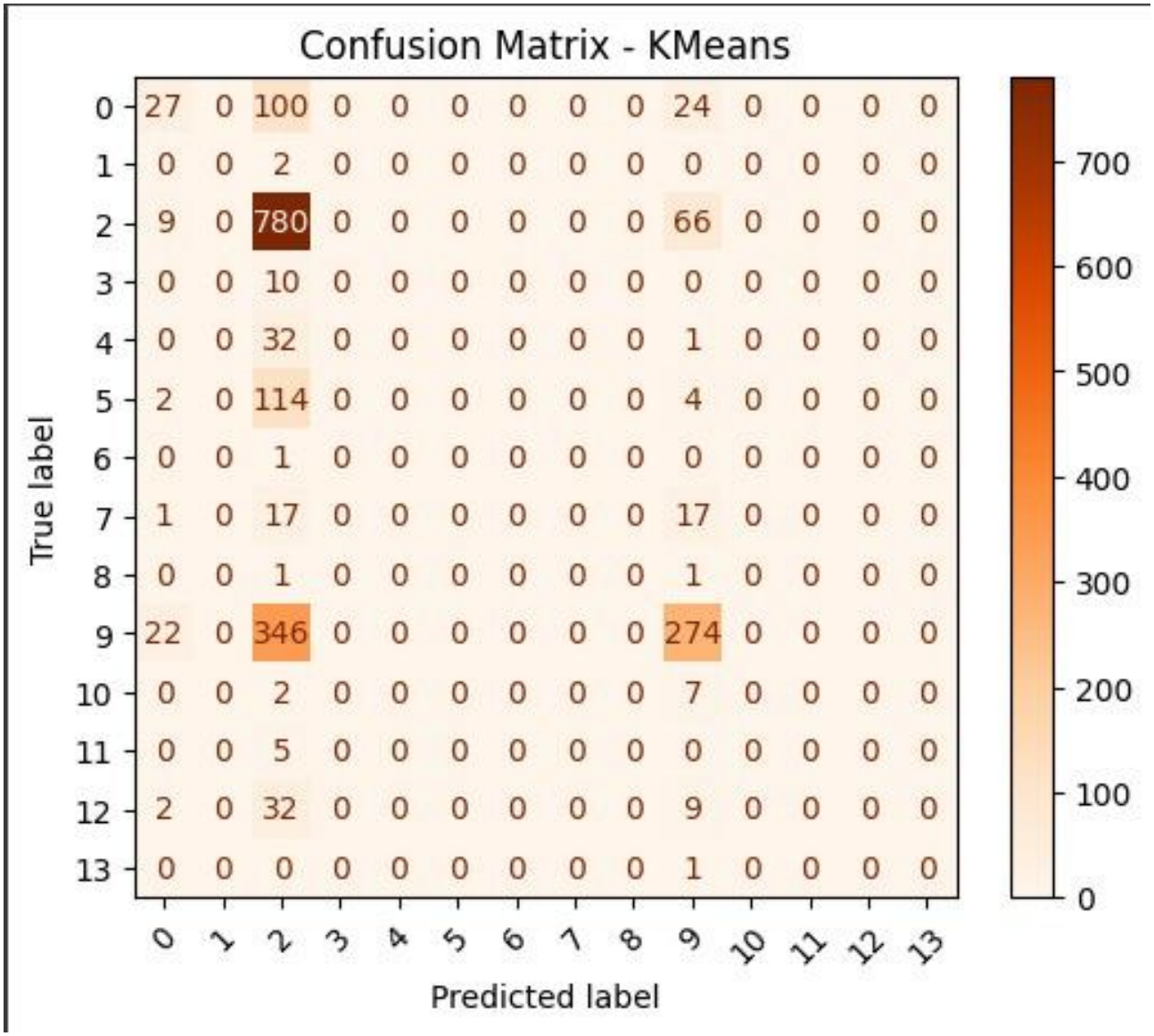
```
# --- Logistic Regression ---
y_pred_logreg = logreg.predict(X_test)
cm = confusion_matrix(y_test, y_pred_logreg)
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=np.unique(y_test))
disp.plot(cmap="Blues", xticks_rotation=45)
plt.title("Confusion Matrix - Logistic Regression")
plt.show()
```



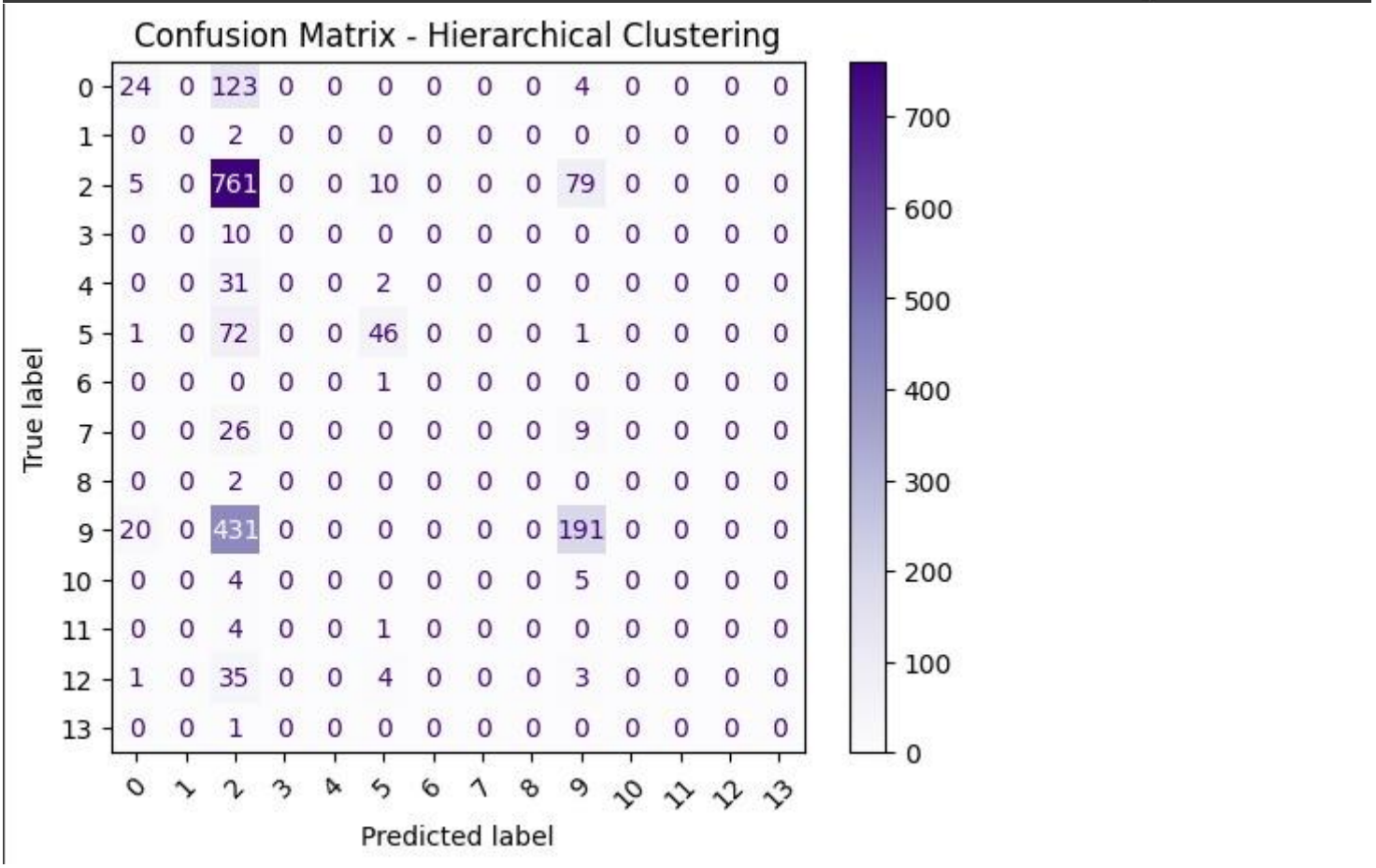
```
[ ]  # --- Random Forest ---
     y_pred_rf = rf.predict(X_test)
     cm = confusion_matrix(y_test, y_pred_rf)
     disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=np.unique(y_test))
     disp.plot(cmap="Greens", xticks_rotation=45)
     plt.title("Confusion Matrix - Random Forest")
     plt.show()
```



```python
# --- KMeans (map clusters to labels first) ---
cluster_preds = kmeans.predict(X_test)
labels_map = {}
for cluster in set(cluster_preds):
    mask = cluster_preds == cluster
    labels_map[cluster] = mode(y_test[mask], keepdims=True).mode.item()
mapped_preds_kmeans = [labels_map[c] for c in cluster_preds]

cm = confusion_matrix(y_test, mapped_preds_kmeans)
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=np.unique(y_test))
disp.plot(cmap="Oranges", xticks_rotation=45)
plt.title("Confusion Matrix - KMeans")
plt.show()
```

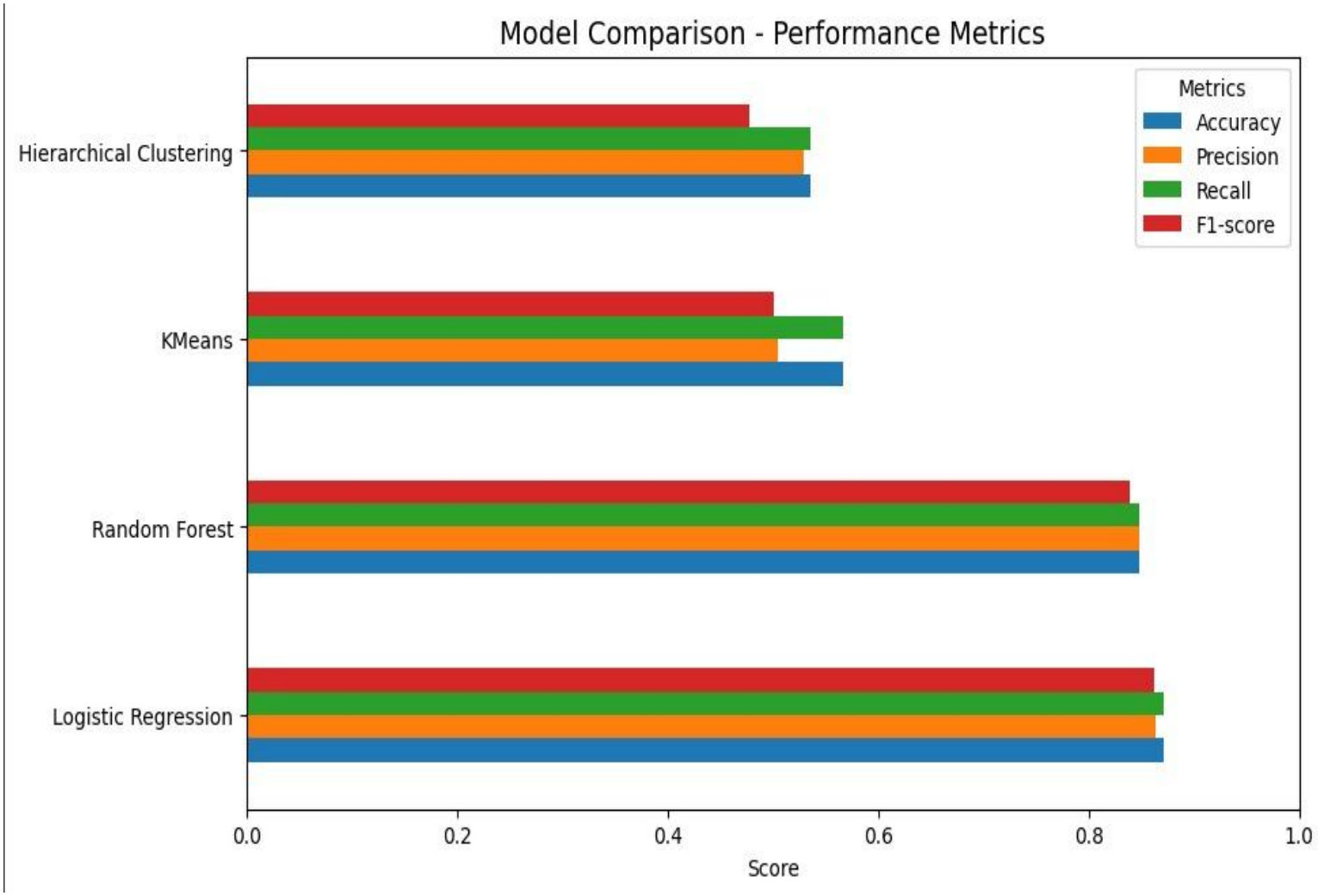


```python
# --- Hierarchical Clustering (Improved, from earlier block) ---
from sklearn.decomposition import TruncatedSVD
from scipy.spatial.distance import pdist
from scipy.cluster.hierarchy import linkage, fcluster

# Step 1: Reduce dimensionality
svd = TruncatedSVD(n_components=100, random_state=42)
X_reduced = svd.fit_transform(X_test)

# Step 2: Hierarchical clustering
num_classes = len(np.unique(y_test))
dist = pdist(X_reduced, metric="euclidean")
Z = linkage(dist, method="ward")
hc_labels = fcluster(Z, t=num_classes, criterion="maxclust") - 1

# Step 3: Map clusters to labels
labels_map = {}
for cluster in set(hc_labels):
    mask = hc_labels == cluster
    labels_map[cluster] = mode(y_test[mask], keepdims=True).mode.item()
mapped_preds_hc = [labels_map[c] for c in hc_labels]

cm = confusion_matrix(y_test, mapped_preds_hc)
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=np.unique(y_test))
disp.plot(cmap="Purples", xticks_rotation=45)
plt.title("Confusion Matrix - Hierarchical Clustering")
plt.show()
```



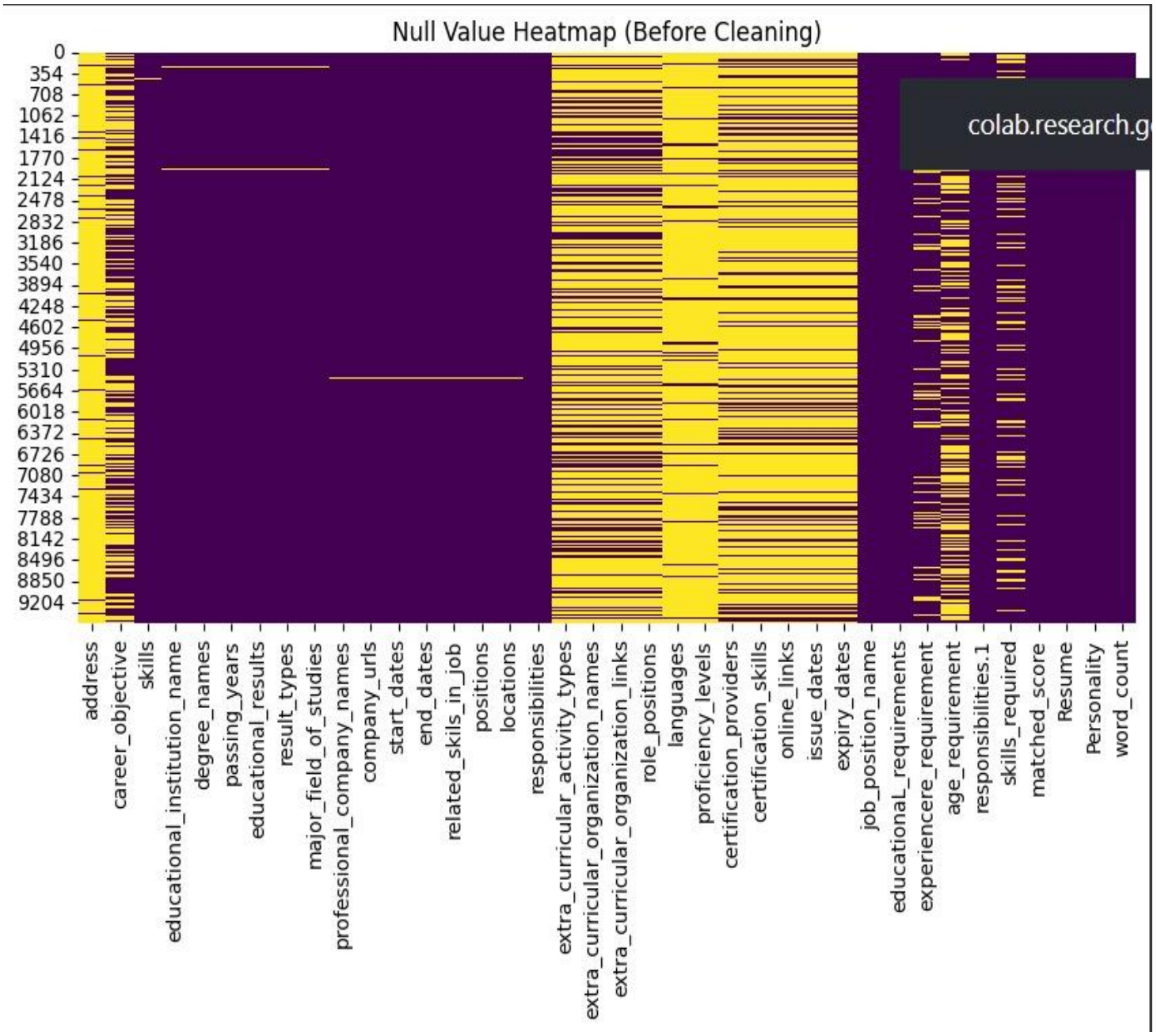
Model Comparison Metric

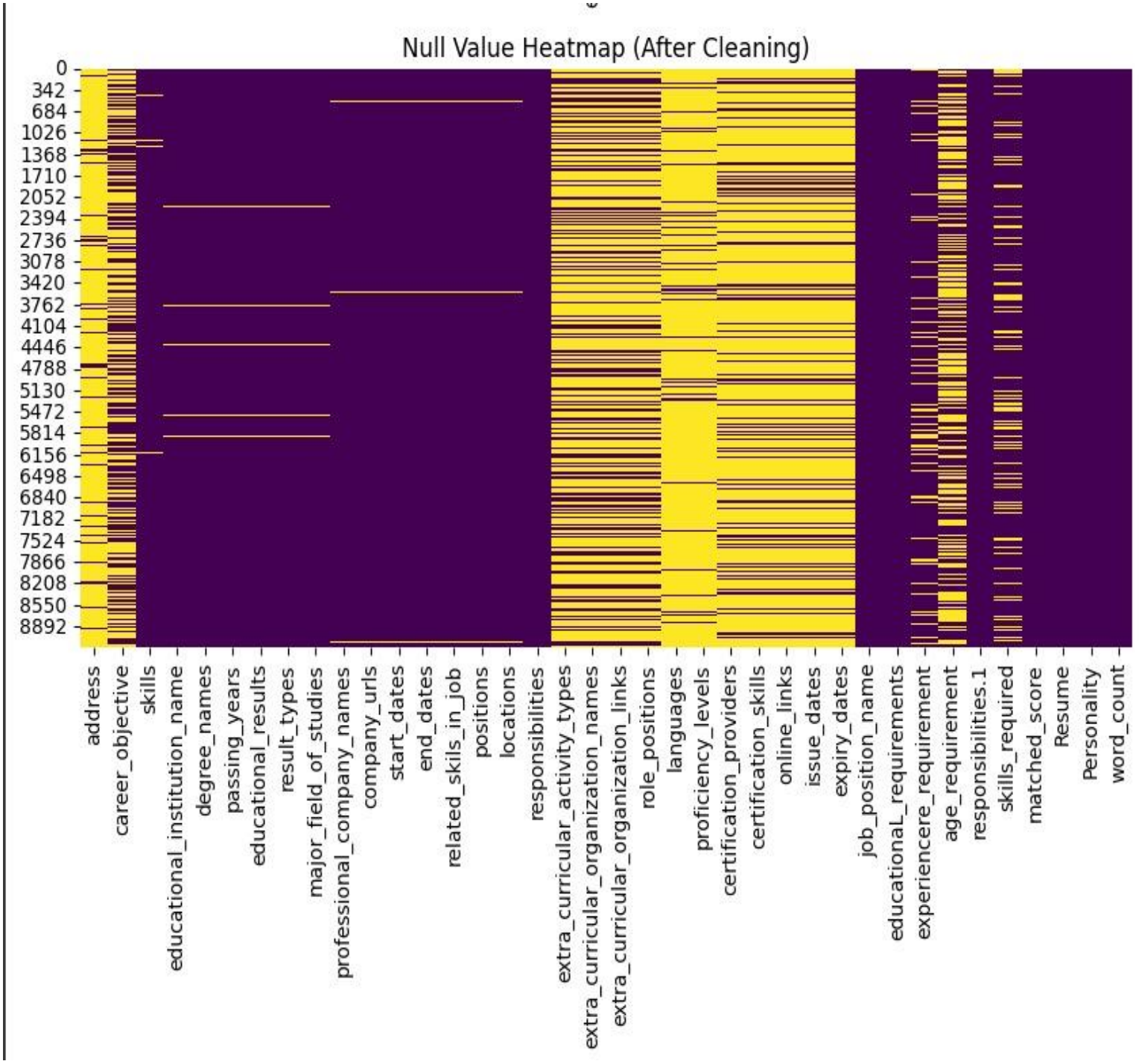
```
results_df.plot(kind="barh", figsize=(10,6))
plt.title("Model Comparison - Performance Metrics", fontsize=14)
plt.xlabel("Score")
plt.xlim(0,1)
plt.legend(title="Metrics")
plt.show()
```



Null Value Heatmap

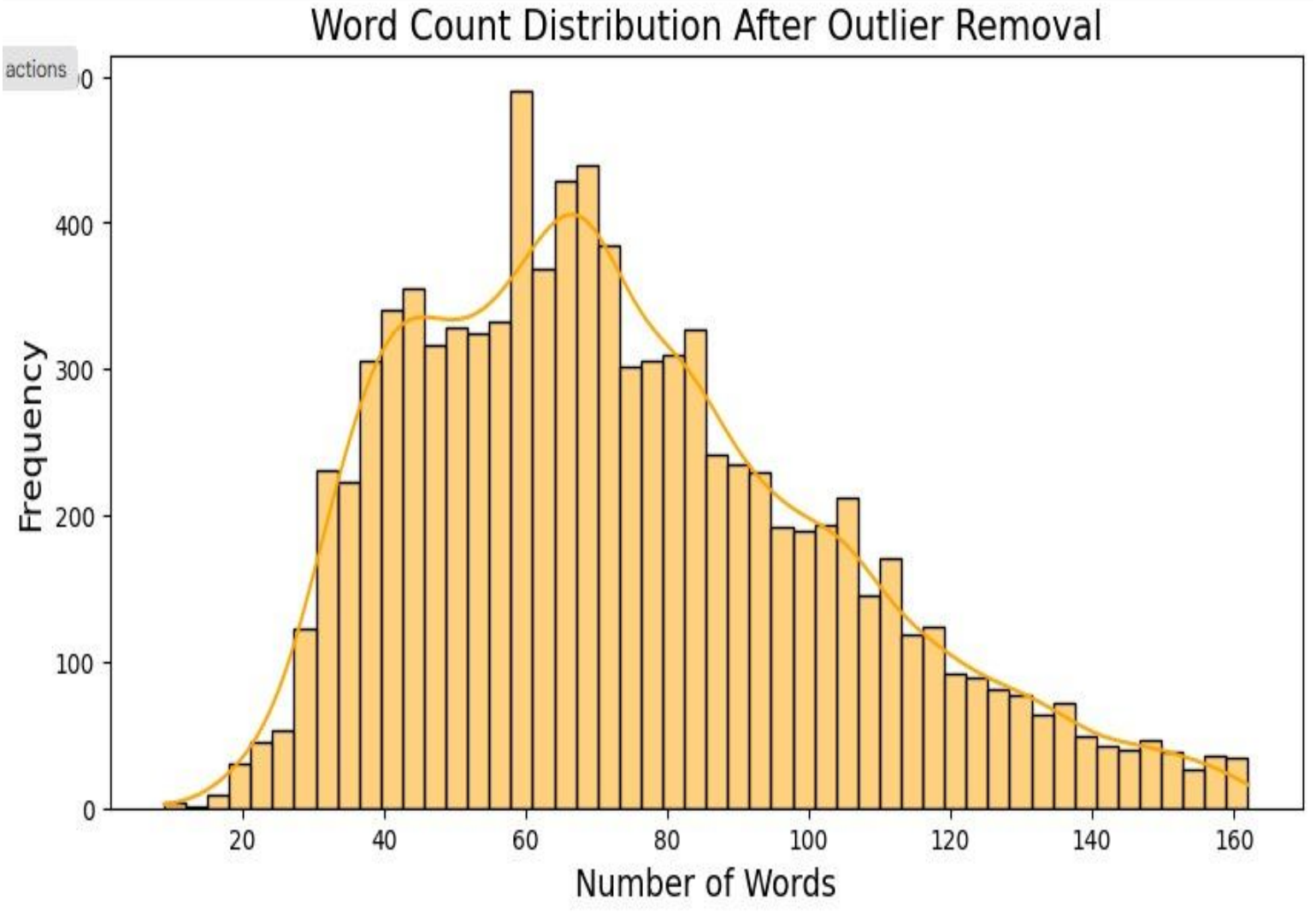
```
# Before cleaning
plt.figure(figsize=(10,5))
sns.heatmap(df.isnull(), cbar=False, cmap="viridis")
plt.title("Null Value Heatmap (Before Cleaning)")
plt.show()

# After cleaning
plt.figure(figsize=(10,5))
sns.heatmap(df_clean.isnull(), cbar=False, cmap="viridis")
plt.title("Null Value Heatmap (After Cleaning)")
plt.show()
```


Null Value Heatmap (Before Cleaning)
0
354
708
1062
1416
1770
2124
2478
2832
3186
3540
3894
4248
4602
4956
5310
5664
6018
6372
6726
7080
7434
7788
8142
8496
8850
9204
address
career_objective
skills
educational_institution_name
degree_names
passing_years
educational_results
result_types
major_field_of_studies
professional_company_names
company_urls
start_dates
end_dates
related_skils_in_job
positions
locations
responsibilities
extra_curricular_activity_types
extra_curricular_organization_names
extra_curricular_organization_links
role_positions
languages
proficiency_levels
certification_providers
certification_skills
online_links
issue_dates
expiry_dates
job_position_name
educationaL_requirements
experiencere_requirement
age_requirement
responsibilities.1
skills_required
matched_score
Resume
Personality
word_count
colab.research.g


Null Value Heatmap (After Cleaning)
0
342
684
1026
1368
1710
2052
2394
2736
3078
3420
3762
4104
4446
4788
5130
5472
5814
6156
6498
6840
7182
7524
7866
8208
8550
8892
address
career_objective
skills
educational_institution_name
degree_names
passing_years
educational_results
result_types
major_field_of_studies
professional_company_names
company_urls
start_dates
end_dates
related_skils_in_job
positions
locations
responsibilities
extra_curricular_activity_types
extra_curricular_organization_names
extra_curricular_organization_links
role_positions
languages
proficiency_levels
certification_providers
certification_skills
online_links
issue_dates
expiry_dates
job_position_name
educationaL_requirements
experiencere_requirement
age_requirement
responsibilities.1
skills_required
matched_score
Resume
Personality
word_count

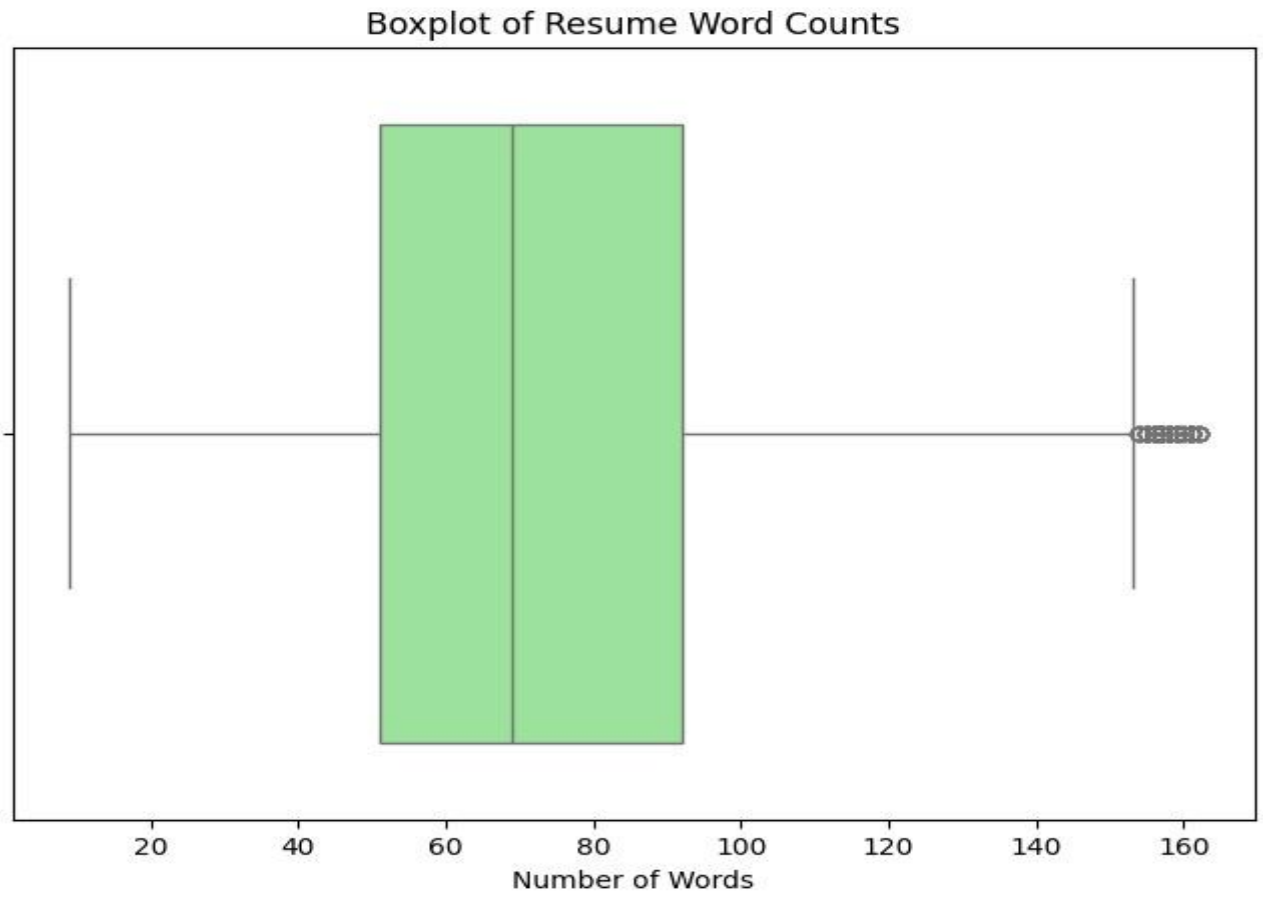
Word count

```
# Plot after outlier removal
plt.figure(figsize=(10,5))
sns.histplot(df_clean['word_count'], bins=50, kde=True, color='orange')
plt.title("Word Count Distribution After Outlier Removal", fontsize=16)
plt.xlabel("Number of Words", fontsize=14)
plt.ylabel("Frequency", fontsize=14)
plt.show()
```



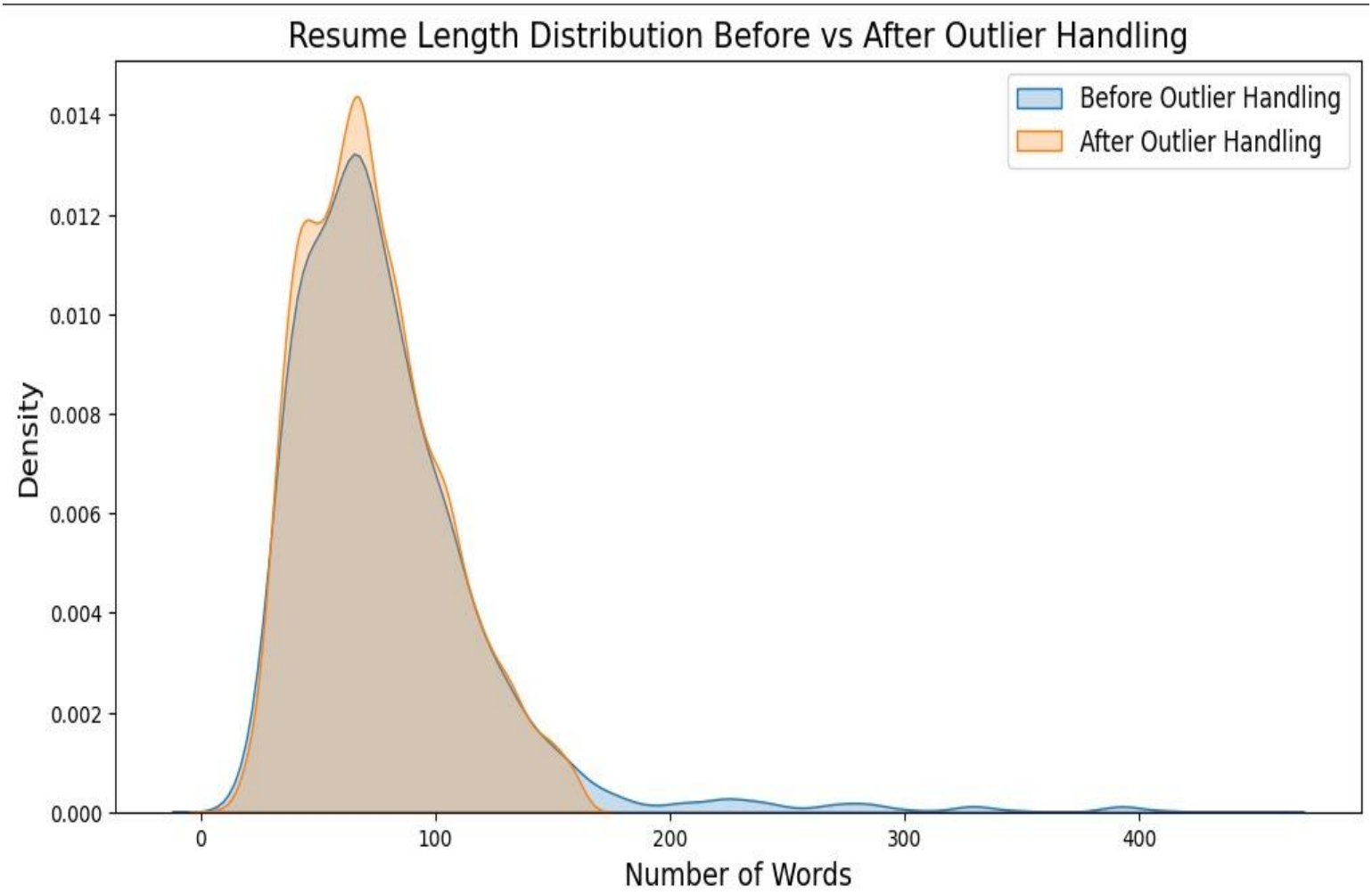
Boxplot

```
plt.figure(figsize=(8,6))
sns.boxplot(x=df_clean["word_count"], color="lightgreen")
plt.title("Boxplot of Resume Word Counts")
plt.xlabel("Number of Words")
plt.show()
```



Outlier Handling

```
plt.figure(figsize=(12,6))

# Before outlier removal (use original cleaned dataframe)
sns.kdeplot(df['word_count'], label="Before Outlier Handling", fill=True)

# After outlier removal (use dataframe after IQR filtering)
sns.kdeplot(df_clean['word_count'], label="After Outlier Handling", fill=True)

plt.title("Resume Length Distribution Before vs After Outlier Handling", fontsize=16)
plt.xlabel("Number of Words", fontsize=14)
plt.ylabel("Density", fontsize=14)
plt.legend(fontsize=12)
plt.show()
```



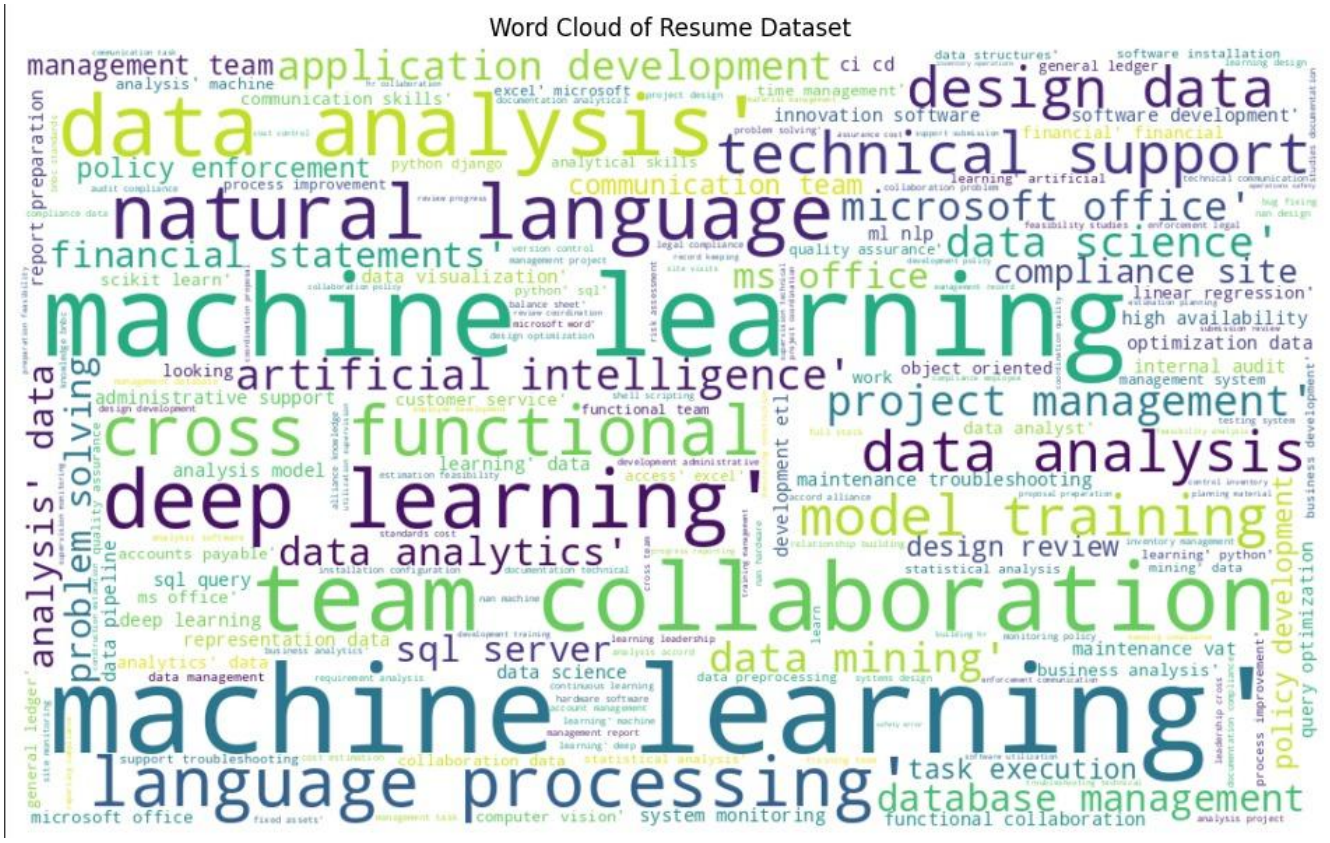
Text Cleaning

```python
# Example raw sentence
raw_text = "Experienced in Python, C++, and machine-learning projects!!!"
steps = {
    "Raw Text": raw_text,
    "Lowercasing": raw_text.lower(),
    "Punctuation Removal": "experienced in python c and machinelearning projects",
    "Stopword Removal": "experienced python machinelearning projects",
    "Lemmatization": "experience python machine learning project"
}

# Display as table
pd.DataFrame(list(steps.items()), columns=["Step", "Text"])
```

	Step	Text
0	Raw Text	Experienced in Python, C++, and machine-learni...
1	Lowercasing	experienced in python, c++, and machine-learni...
2	Punctuation Removal	experienced in python c and machinelearning pr...
3	Stopword Removal	experienced python machinelearning projects
4	Lemmatization	experience python machine learning project

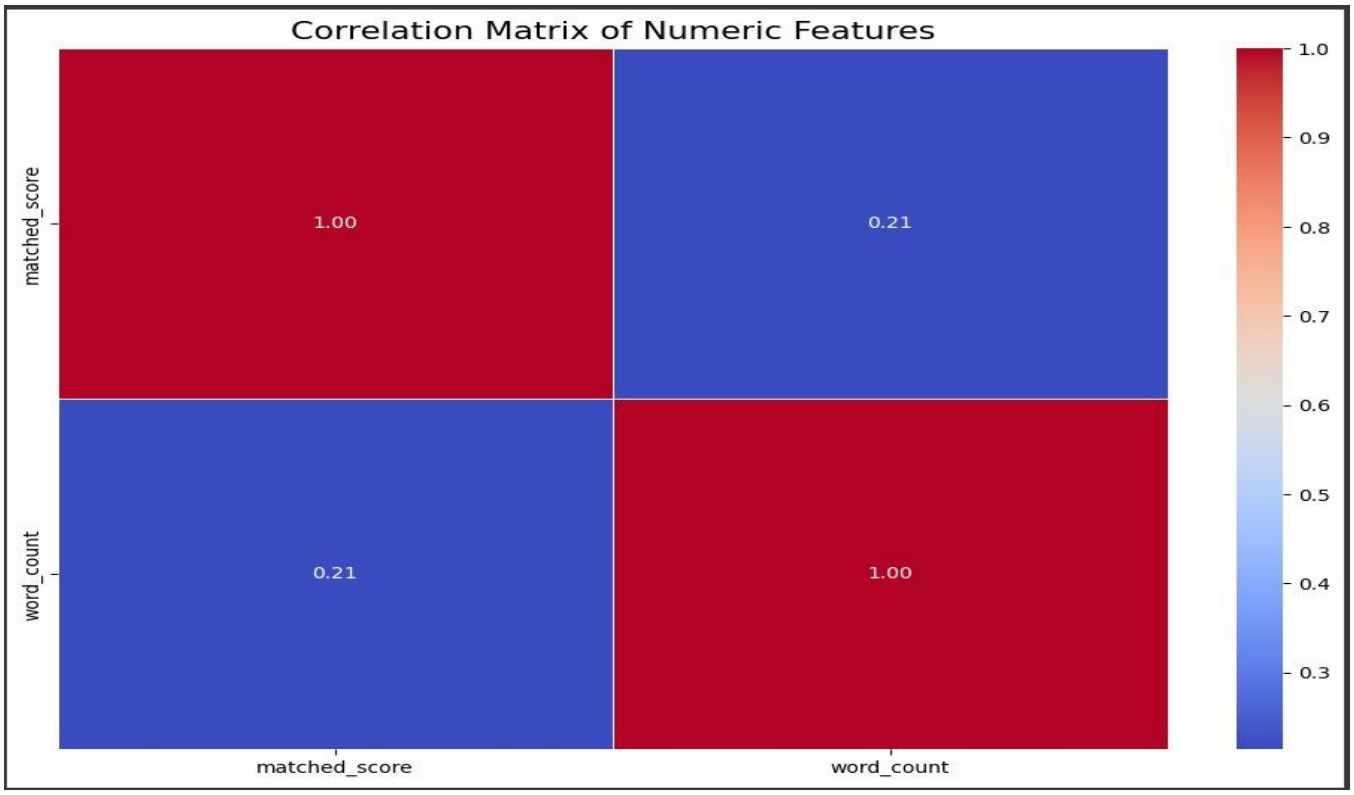
Word Cloud

```
from wordcloud import WordCloud

# Combine all resumes into one big string
text = " ".join(df["Resume"].astype(str).tolist())

wordcloud = WordCloud(width=1000, height=600, background_color="white",
                      colormap="viridis", max_words=200).generate(text)

plt.figure(figsize=(12,8))
plt.imshow(wordcloud, interpolation="bilinear")
plt.axis("off")
plt.title("Word Cloud of Resume Dataset")
plt.show()
```



Correlation Matrix

```
numeric_df = df.select_dtypes(include=['number'])

plt.figure(figsize=(12, 8))
correlation_matrix = numeric_df.corr()

sns.heatmap(correlation_matrix,
            cmap="coolwarm",
            annot=True,
            fmt=".2f",
            linewidths=0.5)

plt.title("Correlation Matrix of Numeric Features", fontsize=16)
plt.show()
```



➢ **Streamlit Code**

```
import streamlit as st
import joblib
import re
import pandas as pd

# Load trained model + vectorizer + label encoder
model = joblib.load("logreg.pkl")
tfidf = joblib.load("tfidf_vectorizer.pkl")
le = joblib.load("label_encoder.pkl")

# Function to clean text
def clean_resume(resume_text):
    resume_text = re.sub(r'[^a-zA-Z ]', ' ', resume_text)
    resume_text = resume_text.lower()
    return resume_text
```

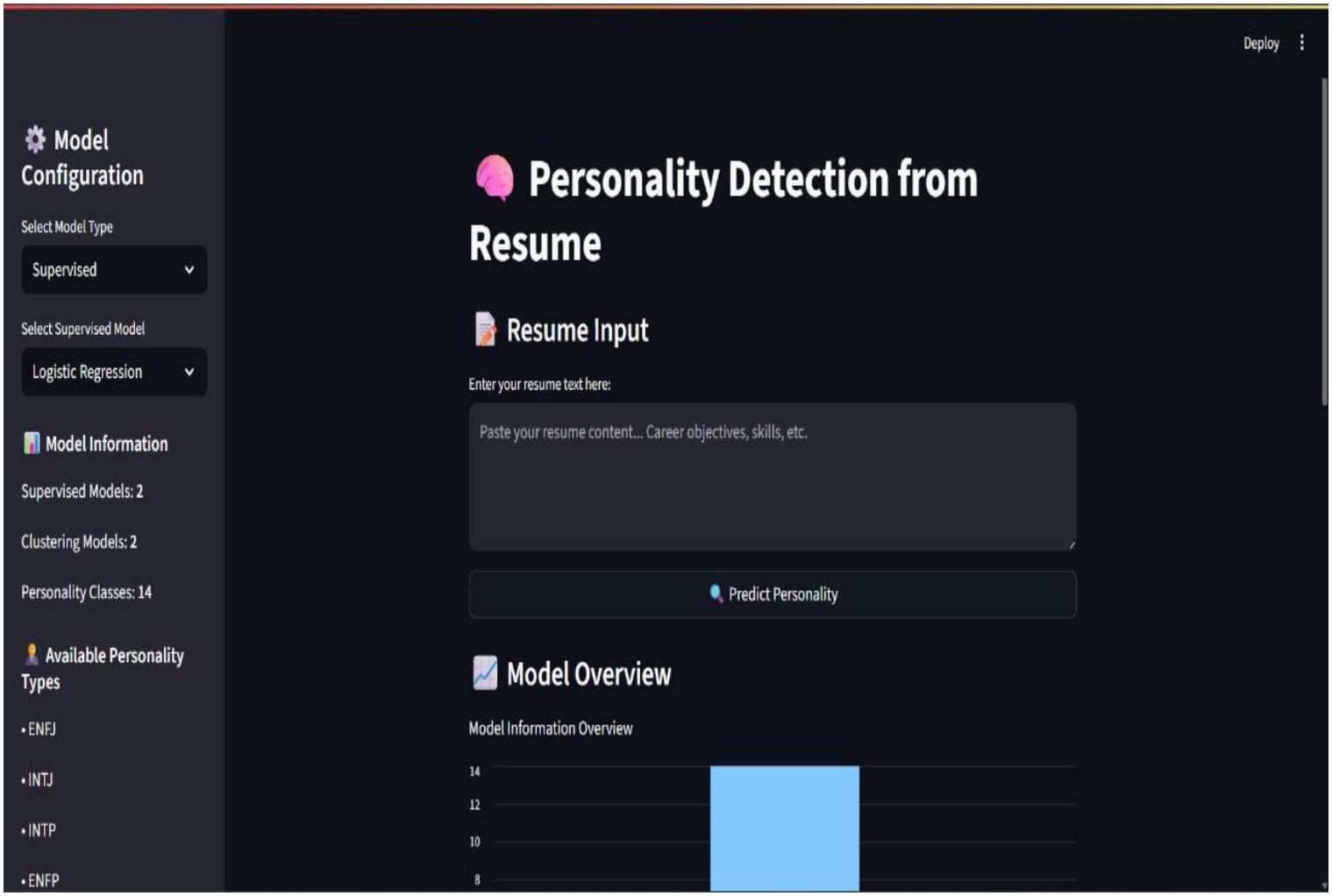
```
# --------------- Sidebar ---------------
st.sidebar.title("⚙️ Model Configuration")

# Model selection
model_type = st.sidebar.selectbox("Select Model Type", ["Supervised", "Unsupervised"])
if model_type == "Supervised":
    supervised_model = st.sidebar.selectbox("Select Supervised Model", ["Logistic Regression", "Random Forest"])
else:
    st.sidebar.selectbox("Select Clustering Model", ["KMeans", "Hierarchical Clustering"])

# Model Info
st.sidebar.markdown("### 📊 Model Information")
st.sidebar.write("Supervised Models: **2**")
st.sidebar.write("Clustering Models: **2**")
st.sidebar.write("Personality Classes: **14**")

st.sidebar.markdown("### 🧍 Available Personality Types")
st.sidebar.write("• ENFJ")
st.sidebar.write("• INTJ")
st.sidebar.write("• INTP")
st.sidebar.write("• ENFP")
st.sidebar.write("• ENTJ")
st.sidebar.write("• ENFP")
st.sidebar.write("• ENTP")
st.sidebar.write("• ESFJ")
st.sidebar.write("• ESTJ")
st.sidebar.write("• ESTP")
st.sidebar.write("• INFJ")
st.sidebar.write("• INFP")
st.sidebar.write("• ISFJ")
st.sidebar.write("• ISTJ")
st.sidebar.write("• ISTP")
```

```
# --------------- Main Page ---------------
st.title("🎈 Personality Detection from Resume")

st.subheader("📝 Resume Input")
resume_input = st.text_area("Enter your resume text here:", height=150,
                            placeholder="Paste your resume content... Career objectives, skills, etc.")

# Prediction button
if st.button("🎈 Predict Personality", use_container_width=True):
    if resume_input.strip():
        cleaned = clean_resume(resume_input)
        vectorized = tfidf.transform([cleaned])
        pred_num = model.predict(vectorized)[0]
        pred_label = le.inverse_transform([pred_num])[0]

        # Centered box with multiline bigger text
        col1, col2, col3 = st.columns([1, 3, 1])
        with col2:
            st.info(
                f"""
                ### 🎯Predicted Personality
                #  {pred_label}
                """
            )

    else:
        st.warning("⚠️ Please enter your resume text!")
```

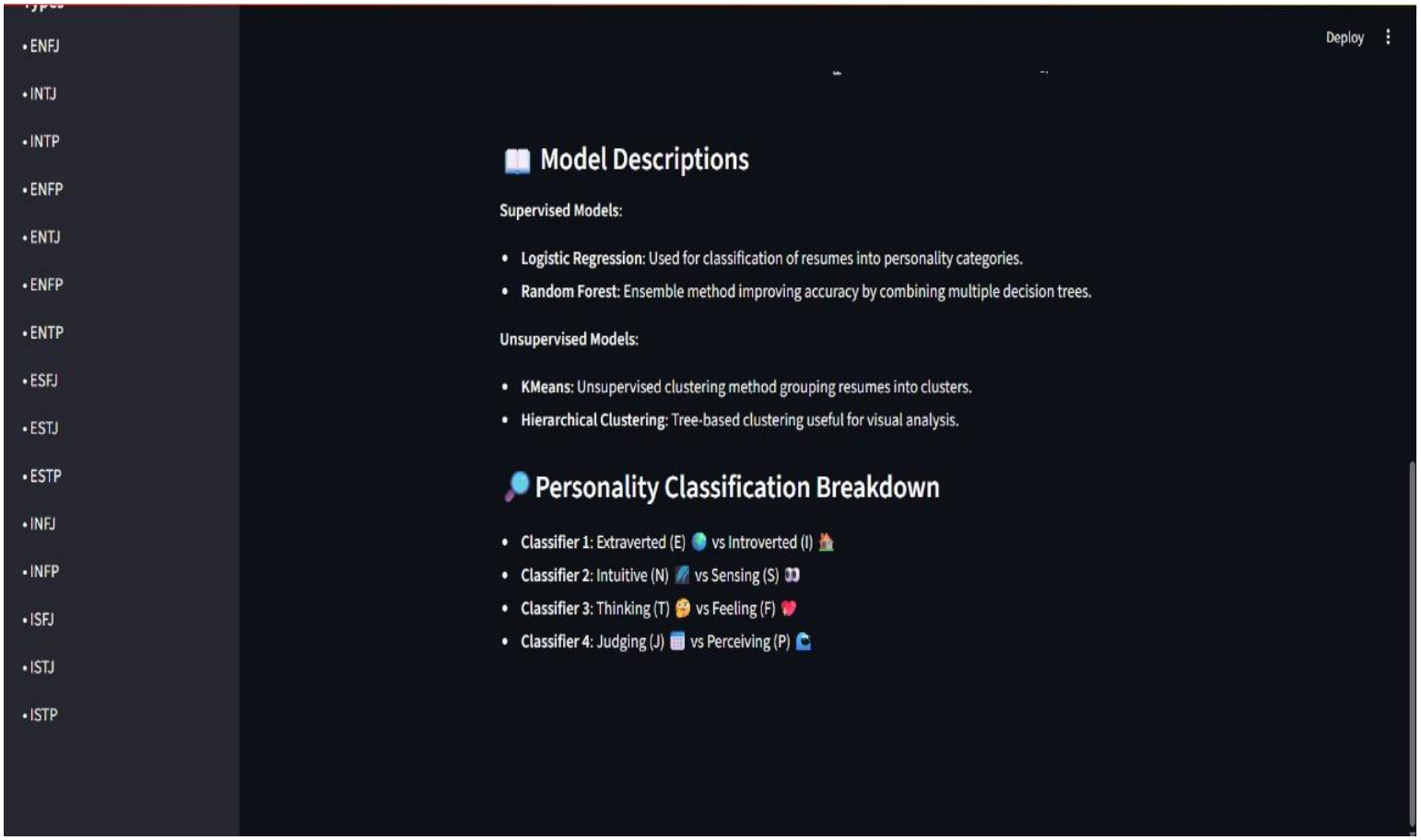
```
# ---------------- Model Overview Chart ----------------
st.subheader("📈 Model Overview")

st.write("Model Information Overview")

data = pd.DataFrame({
    "Model Types": ["Supervised Models", "Clustering Models", "Personality Classes"],
    "Count": [2, 2, 14]
})

st.bar_chart(data.set_index("Model Types"))

# ---------------- Model Descriptions ----------------
st.subheader("📘 Model Descriptions")
st.write("""
**Supervised Models**:
- **Logistic Regression**: Used for classification of resumes into personality categories.
- **Random Forest**: Ensemble method improving accuracy by combining multiple decision trees.

**Unsupervised Models**:
- **KMeans**: Unsupervised clustering method grouping resumes into clusters.
- **Hierarchical Clustering**: Tree-based clustering useful for visual analysis.
""")
st.subheader("🔎Personality Classification Breakdown")
st.write("""
- **Classifier 1**:  Extraverted (E) 🌎 vs Introverted (I) 🏠
- **Classifier 2**:  Intuitive (N) 🌌 vs Sensing (S) 👀
- **Classifier 3**:  Thinking (T) 🤯 vs Feeling (F) 💔
- **Classifier 4**:  Judging (J) 🗓️ vs Perceiving (P) 🤷‍♂️
""")
```

➢ Prototype(output)

Detecting personality based on Resumes to asign suitable job to the candidates acoording to their Pesrsonality by different model using streamlit.




Personality Classes: 14
Available Personality Types
• ENFJ
• INTJ
• INTP
• ENFP
• ENTJ
• ENFP
• ENTP
• ESFJ
• ESTJ
• ESTP
• INFJ
Deploy
Predict Personality
Model Overview
Model Information Overview
14
12
10
8
6
4
2
0
Clustering Models
Personality Classes
Supervised Models


• ENFJ
• INTJ
• INTP
• ENFP
• ENTJ
• ENFP
• ENTP
• ESFJ
• ESTJ
• ESTP
• INFJ
• INFP
• ISFJ
• ISTJ
• ISTP
Deploy
Model Descriptions
Supervised Models:
Logistic Regression: Used for classification of resumes into personality categories.
Random Forest: Ensemble method improving accuracy by combining multiple decision trees.
Unsupervised Models:
KMeans: Unsupervised clustering method grouping resumes into clusters.
Hierarchical Clustering: Tree-based clustering useful for visual analysis.
Personality Classification Breakdown
Classifier 1: Extraverted (E) vs Introverted (I)
Classifier 2: Intuitive (N) vs Sensing (S)
Classifier 3: Thinking (T) vs Feeling (F)
Classifier 4: Judging (J) vs Perceiving (P)

✅ Final Note on Codes

The complete coding implementation demonstrates the step-by-step pipeline of the project, starting from data preprocessing and feature engineering to model training, evaluation, and deployment.

The integration of Streamlit further enabled the development of an interactive prototype, where users can upload resumes and receive personality predictions along with suggested job roles in real time.

Including both intermediate outputs (plots, confusion matrices, ROC curves) and the final prototype interface ensures that the project is not only theoretically sound but also practically applicable. This codebase can be reused, scaled, and enhanced with more data or advanced models in the future.

Future Scope of Improvement

While this project successfully met its objectives, several opportunities for improvement and expansion exist:

1. **Larger Dataset:**
   The dataset used was limited in size. Expanding the dataset with more resumes across industries and roles will improve the model's robustness and reduce bias.

2. **Balanced MBTI Labels:**
   Ensuring balanced representation of all MBTI types will help avoid skewed predictions and provide fair results for all personality categories.

3. **Advanced Feature Representation:**
   Although TF-IDF worked well, other statistical or vectorization methods (e.g., n-grams, part-of-speech patterns) could be explored to capture more contextual meaning without requiring deep NLP.

4. **Model Enhancement:**
   More advanced classifiers such as Gradient Boosting (XGBoost, LightGBM, CatBoost) could be tested for improved performance. Ensemble methods combining Logistic Regression and Random Forest may also provide better stability.

5. **Explainability:**
   Since personality prediction directly impacts candidate evaluation, incorporating explainability techniques (e.g., feature importance visualization, rule-based explanations) would enhance trust and transparency in the system.

6. **Scalability & Deployment:**
   The current Streamlit prototype demonstrates feasibility. In the future, the system can be deployed on cloud platforms (AWS, Azure, GCP) and scaled to handle thousands of resumes in real time.

7. **Integration with HR Systems:**
   This solution can be integrated with existing Applicant Tracking Systems (ATS) to automatically screen resumes, predict personalities, and recommend job roles seamlessly.

8. **Bias Mitigation:**
   Care should be taken to ensure fairness in prediction. Future work could include fairness-aware ML techniques to prevent gender, cultural, or domain-related biases from influencing results.

9. **Cross-Domain Applications:**
   The approach can be extended beyond resumes to other forms of career-related text data such as LinkedIn profiles, cover letters, or online portfolios.

10. **Career Development Support:**
    Instead of stopping at personality prediction, the system could be expanded to recommend **skill-building programs, certifications, or training paths** based on both the candidate's personality and skills.

Conclusion

This project, *"Personality Detection from Resume using MBTI for Job Role Recommendation"*, aimed to explore how raw resumes can be processed, transformed into structured data, and analyzed with machine learning techniques to predict the personality traits of candidates. The ultimate goal was to build a system that not only detects MBTI-based personality categories but also provides insights that can be useful for aligning candidates with suitable job roles.

The workflow began with **systematic data preprocessing**, which included cleaning missing values, normalizing text, and combining relevant fields from resumes. These steps were crucial because resumes are inherently unstructured and inconsistent. Once cleaned, resumes were represented numerically using **TF-IDF (Term Frequency–Inverse Document Frequency)**, which allowed us to capture the importance of specific words and phrases within the dataset.

To further refine the feature set and improve computational efficiency, the **Chi-Square test** was applied to select only the most relevant features correlated with MBTI personality labels.

With features prepared, multiple machine learning models were trained and tested. **Logistic Regression And Random Forest** were employed as supervised learning techniques, while **K-Means and Hierarchical Clustering** were experimented with for unsupervised grouping.

Among these, **Logistic Regression proved to be the most reliable classifier**, achieving a balanced performance with **78% test accuracy and an F1-score of 0.76**. While Random Forest achieved slightly higher test accuracy (80%), it showed overfitting with very high training accuracy (95%), reducing its generalizability. The clustering-based methods, on the other hand, failed to provide meaningful alignment with MBTI classes, highlighting the superiority of supervised approaches for this type of classification.

A major highlight of this work was the development of a **Streamlit-based prototype**, which provides an interactive interface where a user can upload a resume and instantly receive a predicted personality type along with a suitable job role suggestion.

This prototype demonstrated the **practical applicability** of the system in real-world recruitment scenarios. It bridged the gap between theoretical model building and actual end-user usability, showing how recruiters or HR managers can leverage the system in decision-making.

In conclusion, the project achieved the following:

- Built a **systematic data preprocessing pipeline** to clean and structure resume text.
- Represented resumes using **TF-IDF feature vectors** and refined features via **Chi-Square test**.
- Trained and compared multiple machine learning models, identifying **Logistic Regression** as the most balanced and interpretable choice.

- Validated results with **quantitative metrics** (accuracy, precision, recall, F1, ROC curves, confusion matrix).

- Designed a **prototype application** that demonstrates the system's usability in a recruitment setting.

Thus, the project successfully showed that **basic text-based feature engineering combined with machine learning** can serve as a practical tool for personality prediction from resumes and can act as a supportive system for job-role recommendations.

☑ Final Remark

This project successfully showcased how **structured data preprocessing and machine learning** can transform unstructured resumes into actionable insights for personality detection.

While the system is not a replacement for psychometric testing or human judgment, it serves as a **supportive tool for recruiters** to make data-driven, consistent, and faster decisions.

With improvements in dataset size, model sophistication, and deployment, this system has the potential to evolve into a **comprehensive recruitment assistant**, capable of screening resumes, predicting personalities, recommending roles, and guiding career development paths.

In the long term, such solutions can make hiring processes more **efficient, objective, and candidate-friendly**, thereby bridging the gap between employers and job seekers.

Certificate

This is to certify that Ms. Sharmistha Das, of Asansol Engineering College, Roll Number : 10800123186, has successfully completed a project on "PERSONALITY DETECTION FROM RESUME" using Machine Learning with Python under the guidance of Dr. Arnab Chakraborty.

Dr. Arnab Chakraborty
Asansol Engineering College

Certificate

This is to certify that Ms. Rishika Mishra, of Asansol Engineering College, Roll Number: 10800123164, has successfully completed a project on "PERSONALITY DETECTION FROM RESUME" using Machine Learning with Python under the guidance of Dr. Arnab Chakraborty.

Dr. Arnab Chakraborty
Asansol Engineering College

Certificate

This is to certify that Ms. Ragini Gupta, of Asansol Engineering College, Roll Number: 10800123145, has successfully completed a project on "PERSONALITY DETECTION FROM RESUME" using Machine Learning with Python under the guidance of Dr. Arnab Chakraborty.

Dr. Arnab Chakraborty
Asansol Engineering College

Certificate

This is to certify that Ms. Shreya Banerjee, of Asansol Engineering College, Roll Number: 10800123192, has successfully completed a project on "PERSONALITY DETECTION FROM RESUME" using Machine Learning with Python under the guidance of Dr. Arnab Chakraborty.

Dr. Arnab Chakraborty
Asansol Engineering College