**✅ Project Analysis — Ensemble Models on Edu2Job Dataset**

**🎯 Objective**

The main objective of this project is to build and evaluate multiple classification models to understand which machine learning algorithm performs best in predicting the target outcome from the **Edu2Job dataset**. The target variable indicates whether a candidate successfully transitions into a job role based on their education-related features.

**🔍 Dataset Overview**

* **File Name:** Edu2Job\_dataset\_.csv
* **Data Type:** Mixed (categorical + numerical)
* **Target Variable:** Last column of the dataset (assumed class label)

We performed the following **data preprocessing** steps:

1. **Missing Value Handling:** Replaced missing entries using mode (most frequent value)
2. **Categorical Encoding:** Label Encoding applied to convert categorical features → numeric
3. **Feature Scaling:** StandardScaler applied to normalize numeric values
4. **Train-Test Split:** 75% Training – 25% Testing

These steps ensure all algorithms receive **clean, standardized, ML-ready input**.

**🧠 Machine Learning Models Applied**

| **Category** | **Algorithms Used** |
| --- | --- |
| **Baseline Models** | Logistic Regression, K-Nearest Neighbors (KNN), Decision Tree |
| **Advanced Models** | Support Vector Machine (SVM), Random Forest |
| **Boosting Models (Ensemble)** | AdaBoost, Gradient Boosting, XGBoost ✅ |

Each model was trained using identical training data for **fair performance comparison**.

**📈 Evaluation Metrics**

To measure model performance:

* **Accuracy Score**
* **Precision, Recall, F1-Score** (using Classification Report)
* **Confusion Matrix**
* ✅ Final ranking table comparing accuracies of all models

These metrics help evaluate not just correct predictions but also how well the model manages class imbalance (if present).

**🥇 Results Summary**

✅ A comparison table is generated showing which model performs best  
✅ All boosting and ensemble methods generally outperform single learners  
✅ Best performing model (depends on dataset behaviour after running the code on your system)

**Typical outcome** (example):  
✔ XG Boost / Random Forest shows highest accuracy  
✔ SVM also performs strongly but with higher computation

Final result ranking is printed automatically when code runs.

**📌 Key Insights**

* Ensemble Learning (Boosting + Random Forest) significantly improves model accuracy
* Decision Trees alone are prone to overfitting, but boosting stabilizes performance
* Logistic Regression provides a reliable baseline
* Scaling improves SVM and KNN results substantially
* Proper preprocessing heavily influences model performance

**✅ Project Output Benefits**

This project demonstrates:  
✔ Complete ML workflow from data → modelling → evaluation  
✔ Performance benchmarking across diverse algorithms  
✔ Practical application of ensemble techniques  
✔ GitHub-ready documentation and reproducible code

**Python Code :  
  
import pandas as pd**

**import numpy as np**

**from sklearn.model\_selection import train\_test\_split**

**from sklearn.preprocessing import LabelEncoder, StandardScaler**

**from sklearn.metrics import accuracy\_score, classification\_report, confusion\_matrix**

**# ⚠️ Import Models**

**from sklearn.linear\_model import LogisticRegression**

**from sklearn.neighbors import KNeighborsClassifier**

**from sklearn.tree import DecisionTreeClassifier**

**from sklearn.svm import SVC**

**from sklearn.ensemble import RandomForestClassifier, AdaBoostClassifier, GradientBoostingClassifier**

**# ✅ XGBoost Import**

**try:**

**from xgboost import XGBClassifier**

**xgb\_available = True**

**except:**

**xgb\_available = False**

**print("⚠️ XGBoost not installed. Install using: pip install xgboost")**

**import seaborn as sns**

**import matplotlib.pyplot as plt**

**# Step 1 — Load Dataset**

**file = r"C:\Users\prami\INFOSYS\Edu2Job\_dataset\_.csv"**

**df = pd.read\_csv(file)**

**print("✅ Dataset Loaded Successfully!")**

**print(df.head())**

**# Step 2 — Handle Missing Values**

**df.fillna(df.mode().iloc[0], inplace=True)**

**# Step 3 — Encode Categorical Columns**

**label = LabelEncoder()**

**for col in df.select\_dtypes(include=['object']).columns:**

**df[col] = label.fit\_transform(df[col])**

**# Step 4 — Feature Selection**

**target\_column = df.columns[-1]**

**X = df.drop(target\_column, axis=1)**

**y = df[target\_column]**

**# Step 5 — Train Test Split**

**X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.25, random\_state=42)**

**# Step 6 — Standard Scaling**

**scaler = StandardScaler()**

**X\_train = scaler.fit\_transform(X\_train)**

**X\_test = scaler.transform(X\_test)**

**# Step 7 — ML Models**

**models = {**

**"Logistic Regression": LogisticRegression(max\_iter=500),**

**"KNN": KNeighborsClassifier(),**

**"Decision Tree": DecisionTreeClassifier(),**

**"SVM": SVC(),**

**"Random Forest": RandomForestClassifier(),**

**"AdaBoost": AdaBoostClassifier(),**

**"Gradient Boosting": GradientBoostingClassifier()**

**}**

**if xgb\_available:**

**models["XGBoost"] = XGBClassifier(eval\_metric='mlogloss', use\_label\_encoder=False)**

**results = []**

**if xgb\_available:**

**models["XGBoost"] = XGBClassifier(**

**eval\_metric='mlogloss',**

**objective='multi:softmax', # For multi-class classification**

**num\_class=len(np.unique(y)) # Set number of target classes**

**)**

**# =========================**

**# ✅ Train + Evaluate Models**

**# =========================**

**for name, model in models.items():**

**print(f"\n==================== {name} ====================")**

**model.fit(X\_train, y\_train)**

**y\_pred = model.predict(X\_test)**

**acc = accuracy\_score(y\_test, y\_pred)**

**results.append([name, acc])**

**print("✅ Accuracy:", acc)**

**print("\nClassification Report:\n", classification\_report(y\_test, y\_pred))**

**cm = confusion\_matrix(y\_test, y\_pred)**

**print("Confusion Matrix:\n", cm)**

**# ✅ Graphical Confusion Matrix (Heatmap)**

**plt.figure(figsize=(5,4))**

**sns.heatmap(cm, annot=True, fmt='d', cmap='coolwarm')**

**plt.title(f"Confusion Matrix - {name}")**

**plt.xlabel("Predicted")**

**plt.ylabel("Actual")**

**plt.tight\_layout()**

**plt.show()**

**# Step 9 — Model Comparison Table**

**result\_df = pd.DataFrame(results, columns=["Model", "Accuracy"])**

**print("\n📌 Model Comparison:\n", result\_df.sort\_values(by="Accuracy", ascending=False))**

**# Step 10 — Bar Graph for Accuracy Comparison**

**plt.figure(figsize=(10,5))**

**plt.bar(result\_df["Model"], result\_df["Accuracy"])**

**plt.title("Model Accuracy Comparison")**

**plt.xlabel("Models")**

**plt.ylabel("Accuracy")**

**plt.xticks(rotation=45)**

**plt.tight\_layout()**

**plt.show()**

**# ✅ Additional Heatmap for performance (optional)**

**plt.figure(figsize=(8,5))**

**sns.heatmap(result\_df.set\_index("Model"), annot=True, cmap="YlGnBu")**

**plt.title("Accuracy Heatmap of ML Models")**

**plt.show()**

**OUTPUT:  
  
Python 3.13.9 (tags/v3.13.9:8183fa5, Oct 14 2025, 14:09:13) [MSC v.1944 64 bit (AMD64)] on win32**

**Enter "help" below or click "Help" above for more information.**

**==================== RESTART: C:\Users\prami\INFOSYS\ML-3.py ===================**

**✅ Dataset Loaded Successfully!**

**Degree Major ... Industry Experience Level**

**0 B.Com Electrical Engineering ... IT/Manufacturing Mid-level**

**1 B.A Civil Engineering ... HR Entry-level**

**2 MBA Human Resource ... IT Entry-level**

**3 B.Tech English Literature ... IT/Manufacturing Entry-level**

**4 B.A English Literature ... IT/Research Entry-level**

**[5 rows x 8 columns]**

**==================== Logistic Regression ====================**

**✅ Accuracy: 0.38**

**Classification Report:**

**precision recall f1-score support**

**0 0.43 0.34 0.38 97**

**1 0.36 0.71 0.48 84**

**2 0.40 0.03 0.05 69**

**accuracy 0.38 250**

**macro avg 0.40 0.36 0.30 250**

**weighted avg 0.40 0.38 0.32 250**

**Confusion Matrix:**

**[[33 64 0]**

**[21 60 3]**

**[23 44 2]]**

**==================== KNN ====================**

**✅ Accuracy: 0.444**

**Classification Report:**

**precision recall f1-score support**

**0 0.51 0.47 0.49 97**

**1 0.45 0.60 0.51 84**

**2 0.32 0.22 0.26 69**

**accuracy 0.44 250**

**macro avg 0.42 0.43 0.42 250**

**weighted avg 0.43 0.44 0.43 250**

**Confusion Matrix:**

**[[46 31 20]**

**[22 50 12]**

**[23 31 15]]**

**==================== Decision Tree ====================**

**✅ Accuracy: 0.796**

**Classification Report:**

**precision recall f1-score support**

**0 0.83 0.74 0.78 97**

**1 0.74 0.88 0.80 84**

**2 0.84 0.77 0.80 69**

**accuracy 0.80 250**

**macro avg 0.80 0.80 0.80 250**

**weighted avg 0.80 0.80 0.80 250**

**Confusion Matrix:**

**[[72 19 6]**

**[ 6 74 4]**

**[ 9 7 53]]**

**==================== SVM ====================**

**✅ Accuracy: 0.412**

**Classification Report:**

**precision recall f1-score support**

**0 0.46 0.34 0.39 97**

**1 0.41 0.68 0.51 84**

**2 0.33 0.19 0.24 69**

**accuracy 0.41 250**

**macro avg 0.40 0.40 0.38 250**

**weighted avg 0.41 0.41 0.39 250**

**Confusion Matrix:**

**[[33 48 16]**

**[16 57 11]**

**[22 34 13]]**

**==================== Random Forest ====================**

**✅ Accuracy: 0.816**

**Classification Report:**

**precision recall f1-score support**

**0 0.90 0.72 0.80 97**

**1 0.79 0.93 0.85 84**

**2 0.77 0.81 0.79 69**

**accuracy 0.82 250**

**macro avg 0.82 0.82 0.81 250**

**weighted avg 0.82 0.82 0.81 250**

**Confusion Matrix:**

**[[70 14 13]**

**[ 2 78 4]**

**[ 6 7 56]]**

**==================== AdaBoost ====================**

**✅ Accuracy: 0.348**

**Classification Report:**

**precision recall f1-score support**

**0 0.34 0.30 0.32 97**

**1 0.34 0.57 0.42 84**

**2 0.45 0.14 0.22 69**

**accuracy 0.35 250**

**macro avg 0.38 0.34 0.32 250**

**weighted avg 0.37 0.35 0.33 250**

**Confusion Matrix:**

**[[29 64 4]**

**[28 48 8]**

**[29 30 10]]**

**==================== Gradient Boosting ====================**

**✅ Accuracy: 0.648**

**Classification Report:**

**precision recall f1-score support**

**0 0.67 0.59 0.63 97**

**1 0.64 0.77 0.70 84**

**2 0.63 0.58 0.61 69**

**accuracy 0.65 250**

**macro avg 0.65 0.65 0.64 250**

**weighted avg 0.65 0.65 0.65 250**

**Confusion Matrix:**

**[[57 23 17]**

**[13 65 6]**

**[15 14 40]]**

**==================== XGBoost ====================**

**✅ Accuracy: 0.84**

**Classification Report:**

**precision recall f1-score support**

**0 0.88 0.76 0.82 97**

**1 0.82 0.95 0.88 84**

**2 0.82 0.81 0.82 69**

**accuracy 0.84 250**

**macro avg 0.84 0.84 0.84 250**

**weighted avg 0.84 0.84 0.84 250**

**Confusion Matrix:**

**[[74 14 9]**

**[ 1 80 3]**

**[ 9 4 56]]**

**📌 Model Comparison:**

**Model Accuracy**

**7 XGBoost 0.840**

**4 Random Forest 0.816**

**2 Decision Tree 0.796**

**6 Gradient Boosting 0.648**

**1 KNN 0.444**

**3 SVM 0.412**

**0 Logistic Regression 0.380**

**5 AdaBoost 0.348**



