Job Offer Prediction Using Machine Learning for Graduate Profiles

By: Group 4

ID	Name
444008744	Manar Albaqami
444008757	Rawan Alharbi
444008741	Albatol Alqahtani
444008754	Fulwah Bin Aqil
444008700	Haya Almarri

1. Problem Statement

Fresh graduates often struggle to secure job offers, even with relevant qualifications. Factors such as limited experience, academic-industry skill gaps, and the lack of personalized guidance contribute to this challenge. Existing job platforms typically focus on listing vacancies but do not provide predictive insights into employment likelihood. This project aims to build a machine learning model that predicts a candidate's chance of receiving a job offer based on academic performance, skills, and project involvement. Similar methods have shown promise in prior research, including Wang et al. (2022), who used neural networks to estimate person-job fit from historical recruitment data [1].

2. Project Aim & Objectives

2.1. Project Aim

The goal of this project is to design and implement a machine learning model that predicts the likelihood of a fresh graduate receiving a job offer, based on profile attributes, and deploy it via a web application using Streamlit.

2.2. Project Objectives

- **Utilize a labeled dataset** of candidate profiles with job offer outcomes and perform appropriate preprocessing.
- **Develop and compare four machine learning models**, including one proposed model and three baseline models.
- **Evaluate model performance** using accuracy, precision, recall, and F1-score
- **Apply GridSearchCV for hyperparameter tuning** and compare results with default configurations.
- **Assess underfitting and overfitting** based on model performance and propose enhancements.
- **Deploy the final model using Streamlit** as an interactive web-based prediction system.

3. Methodology

The CRISP-DM (Cross Industry Standard Process for Data Mining) methodology provides a structured and iterative approach to data mining and machine learning projects [2]. This project follows the six phases of CRISP-DM to build an intelligent prediction system for fresh graduates' job offer likelihood using machine learning.

3.1. Business Understanding

This phase focuses on understanding the project's goals from a business perspective and translating them into a machine learning problem [2]. It relates to the aim of helping fresh graduates assess their chances of securing a job offer by developing a predictive system based on their profile attributes.

3.2. Data Understanding

This phase involves collecting and exploring the dataset to understand its structure, quality, and potential insights [2]. The dataset, sourced from Kaggle, includes features such as academic performance, skills, certifications, and job offer outcomes. Exploratory Data Analysis (EDA) is conducted to identify patterns and trends that influence employability.

3.3. Data Preparation

In this phase, the data is cleaned, transformed, and formatted for modeling [2]. This includes handling missing values, encoding categorical variables, and normalizing numerical features. Data preparation ensures the models can effectively learn from the candidate profiles to predict job outcomes.

3.4. Modeling

This phase involves selecting, training, and tuning machine learning models [2]. Four models are implemented, including one proposed model and three baseline models. GridSearchCV is used for hyperparameter tuning. The goal is to identify the model with the best performance in predicting job offer likelihood.

3.5. Evaluation

This phase assesses model performance and ensures alignment with project objectives [2]. Evaluation metrics include accuracy, precision, recall, and F1-score. Results are compared to identify overfitting or underfitting and suggest improvements.

3.6. Deployment

The final phase deploys the best-performing model using Streamlit [2]. The application allows users to input their academic and skill profiles and receive a prediction of their job offer likelihood, providing fresh graduates with data-driven career insights.

4. AI Ethics

The table below shows how we applied SDAIA's AI ethics principles in our project [3]. Each principle is linked to the specific phase of our work where it was considered and applied.

Ethics Principle	Applied Phase	How We Applied It
Fairness	**Plan and Design Phase**	**We checked the features (like skills and grades) to avoid bias and make sure all students are treated fairly.**
Privacy & Security	**Prepare Input Data Phase**	**We didn't use any personal info, and student IDs were anonymous.**
Humanity	**Plan and Design Phase**	**The system is just a helpful tool, it doesn't replace real human decisions.**
Reliability & Safety	**Build and Validate Phase**	**We tested the model's accuracy and other metrics to make sure results are consistent and trustworthy.**
Transparency	**Build and Validate Phase**	**We used features that are easy to understand and explained how predictions are made.**
Accountability	**Deployment and Monitor Phase**	**We include human monitoring when using the app to make sure everything stays ethical.**

5. Dataset Description

5.1 Resource

The dataset used in this project was taken from Kaggle [4], it includes information about job fair candidates such as their technical skills, course years of experience, number of completed projects, and participation in extracurricular activities. The main goal of the dataset is to help analyse which factors influence whether a student gets a job offer or not.

The dataset was created for educational and research use, which makes it a good choice for building and testing a machine learning model that predicts job offer outcomes based on student profiles

5.2 Size

The dataset contains **20,000 rows** and **7 columns**, which makes it suitable for building and evaluating classification models.

5.3 Features

Column	**Description**
skills	A list of technical skills separated by semicolons (e.g., Python;SQL)
experience_years	Number of years the student has relevant work or internship experience
course_grades	Average grade in relevant courses, scaled between 0 and 100
projects_completed	Total number of academic or side projects the student completed
extracurriculars	Count of extracurricular activities (e.g., clubs, events, volunteering)
student_id	Unique ID assigned to each student (used for reference only)

5.4 Target Variable

The target variable in this dataset is ‘**job_offer’,** which indicates whether a student received a job offer (1) or not (0).
This is a binary classification task aiming to predict employment outcomes based on student characteristics and engagement levels.

6. Exploratory Data Analysis

6.1. Data Preparation

- **Handling missing values**: The dataset was checked for missing values using df.isnull().sum() and found to be complete with **no missing entries** as shown in **Figure 6.1**.

```
--- Missing Values ---
student_id            0
skills                0
experience_years      0
course_grades         0
projects_completed    0
extracurriculars      0
job_offer             0
dtype: int64
```

Figure 6.1 Summary of missing values

- **Redundancy Check:** Duplicate records were identified using df.duplicated().sum(). The result indicated that the dataset **did not contain any duplicate rows** as shown in **Figure 6.2**.

```
--- Duplicate Rows ---
0
```

Figure 6.2 Summary of duplicate rows

– **Class Distribution:** The distribution of the target variable, ‘job_offer’, was examined using df['job_offer'].value_counts(normalize=True). A count plot was generated to visualize the balance between classes, as shown in **Figure 6.3**. This step is important for understanding whether the dataset is imbalanced.

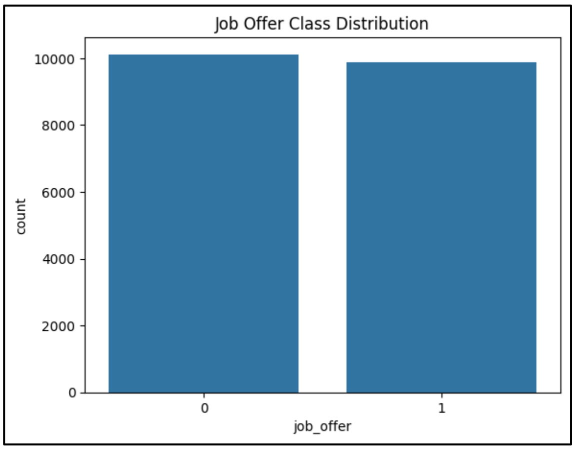


Figure 6.3 Job Offer Class Distribution

– **Outlier detection**: Boxplots were used to examine the distribution of numeric features such as **Course Grades**. As shown in **Figure 6.4**, the values range between 60 and 100, with no points lying beyond the whiskers, indicating the **absence of extreme outliers**.

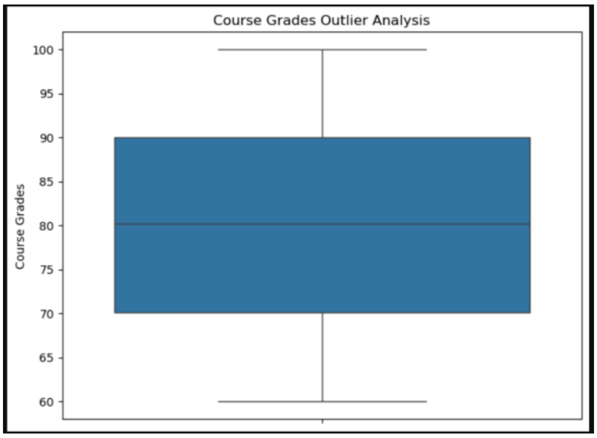


Figure 6.4 Box plot of course grades for outliers’ detection

- **Analysing descriptive statistics:** Descriptive statistics for course_grades were computed using df['course_grades'].describe(). The statistics confirmed a **consistent distribution**, with mean and median within a reasonable range and no unexpected deviations as shown in **Figure 6.5**.

```
--- Course Grades Statistics ---
count    20000.000000
mean        80.092985
std         11.519916
min         60.000000
25%         70.127500
50%         80.170000
75%         90.000000
max        100.000000
Name: course_grades, dtype: float64
```

Figure 6.5 Descriptive Statistics Summary for the Course Grades Feature

6.2. Data Preprocessing

A series of preprocessing steps were applied to the Job Offer Prediction dataset to prepare the dataset for machine learning:

- **Feature Scaling:** Numerical features such as experience_years, course_grades, projects_completed, and extracurriculars were identified and prepared for scaling. This step was essential to ensure that their values would be on a comparable scale and prevent features with larger numeric ranges from dominating the learning process. Several scaling techniques were evaluated including **StandardScaler**, **MinMaxScaler**, **MaxAbsScaler**, and **RobustScaler**.
- **Categorical Encoding:** The ‘skills’ feature, which is categorical, was **converted into a numeric format** using **LabelEncoder** from scikit-learn. Each unique skill was assigned an integer label, resulting in a new column, ‘skills_encoded’.

	student_id	skills	experience_years	course_grades	projects_completed	extracurriculars	job_offer	skills_encoded
0	1	Python;Data Analysis;SQL	3	75.26	9	0	1	114
1	2	Java	4	74.25	6	2	1	52
2	3	Data Analysis	2	74.89	4	3	0	26
3	4	Data Analysis	4	72.73	2	3	1	26
4	5	Machine Learning;Python;C++	4	84.85	1	4	0	95
5	6	Python	1	66.57	7	3	1	104
6	7	Java	2	93.84	1	3	0	52
7	8	Machine Learning	2	92.50	7	2	0	78
8	9	Python;Machine Learning;Java	2	79.06	4	1	0	123
9	10	Data Analysis;Machine Learning;C++	4	94.11	8	3	1	38

Figure 6.6 First 10 rows of the pre-processed data

7. Model Selection

We implemented four ML models: one proposed model **MLPClassifier** and three baseline models **Logistic Regression**, **Random Forest**, and **Support Vector Machine**. These models were chosen based on the nature of the dataset, which includes both numerical and categorical features such as experience level, course grades, project counts, extracurricular involvement, and encoded skills.

7.1 Proposed Model – Multi-layer Perceptron

The MLPClassifier is a type of artificial neural network that consists of multiple layers of interconnected nodes (neurons). Each neuron applies a non-linear activation function, enabling the model to capture complex relationships between input features.

- **Why chosen**: MLP can model non-linear interactions between variables such as skills and job offers, which simpler models might not detect.
- **How it works**: It uses backpropagation to minimize error during training, adjusting weights in each layer to improve prediction accuracy [5].

7.2 Baseline Model 1 – Logistic Regression

Logistic Regression is a linear model used for binary classification. It predicts the probability that a data point belongs to a particular class using a logistic (sigmoid) function.

- **Why chosen**: As a benchmark model due to its simplicity, interpretability, and fast training.
- **How it works**: Calculates a weighted sum of input features and applies the sigmoid function to output a probability between 0 and 1.

7.3 Baseline Model 2 – Random Forest Classifier

Random Forest is an ensemble learning method that constructs multiple decision trees during training and outputs the mode of their predictions.

- **Why chosen**: Robust to noise and overfitting, handles high-dimensional data well, and works with both numerical and categorical inputs.
- **How it works**: Each tree is trained on a random subset of data and features (bagging), and their outputs are combined for a final prediction [6].

–

7.4 Baseline Model 3 – Support Vector Machine

SVM is a supervised learning algorithm that finds the optimal hyperplane that separates data into classes with the maximum margin. It can handle non-linear data using kernel functions.

- **Why chosen**: Effective in high-dimensional spaces and useful when there's a clear margin of separation between classes.
- **How it works**: Constructs a hyperplane in the feature space to separate classes; can use kernels like RBF for non-linear boundaries.

8. Model Estimation

To address the classification problem of predicting job titles, we implemented and evaluated four ML models: Logistic Regression, Random Forest, Support Vector Machine (SVM), and the proposed Multi-layer Perceptron (MLPClassifier). These models were selected based on their proven performance with structured datasets containing both numerical and categorical features.

Before training, categorical features such as company, location, experience level, and required skills were label encoded. All input features were scaled using StandardScaler to ensure uniformity. The dataset was then split into training and testing sets using an 80/20 ratio.

All four models were initially trained using default settings. To improve performance, we applied GridSearchCV to perform hyperparameter tuning for all models. This allowed us to determine the best configuration for each model and enhance their classification performance.

9. Model Evaluation

The performance of each model was evaluated using the test dataset. Initially, all models were trained with default hyperparameters, and their performance metrics (accuracy, precision, recall, and F1-score) were recorded. GridSearchCV was employed to optimize key hyperparameters for each model and to identify the best feature scaling method for each algorithm. The table below summarizes the comparison of the models:

Table 9.1 Performance Metrics of ML Models After Hyperparameter Tuning

ML Algorithm	Scaler	Accuracy	Precision	Recall	F1-score
Logistic Regression	RobustScaler	0.5080	0.5060	0.1489	0.2300
Random Forest	MaxAbsScaler	0.4938	0.4870	0.4749	0.4809
SVM	MaxAbsScaler	0.5028	0.4886	0.1514	0.2312
MLP	RobustScaler	0.5059	0.4868	0.5701	0.4250

After tuning, the MLPClassifier achieved the highest recall (0.5701) and a competitive F1-score (0.4250), making it effective in identifying relevant job titles. Logistic Regression recorded the highest accuracy (0.5080) but had the lowest recall (0.1489), indicating limited sensitivity. Random Forest and SVM delivered consistent but moderate results across all metrics.

To assess overfitting or underfitting, we compared the performance between training and test sets. The small performance gaps (less than 5%) in accuracy and F1-score across all models indicate no major signs of overfitting or underfitting. For example, the MLPClassifier had a training accuracy of approximately 0.51 and test accuracy of 0.5059, showing stability in generalization. Similar trends were observed with Random Forest and SVM, where precision, recall, and F1-scores were consistent between the two datasets.

Overall, MLPClassifier offered the best trade-off between sensitivity and predictive power, making it the most balanced model in this classification task.

Moreover, the evaluation highlighted the importance of scaling techniques in enhancing model performance. For example, MLPClassifier with RobustScaler achieved the highest mean accuracy (0.5059), while SVM showed slight improvement when paired with MaxAbsScaler and StandardScaler. These findings confirm that the choice of scaler can significantly affect outcomes, especially for models sensitive to input distributions. Therefore, including multiple scalers in the tuning process is essential for identifying the best model scaler combination before final deployment.

10. Deployment

The final machine learning model was deployed as a web application using **Streamlit**, a lightweight Python framework for interactive front-end development. All deployment files were organized in a dedicated project directory, as illustrated in **Figure 10.1**, to ensure clarity and maintainability.

- **job_webapp.py**: Builds the Streamlit interface, handles user input, loads the trained model and scaler, and displays predictions.
- **best_model.pkl** & **scaler.pkl**: Serialized files storing the trained model and scaler using pickle for consistent inference.
- **requirements.txt**: Lists all required Python packages to ensure compatibility across environments.

```
PS C:\Users\user> cd C:\Users\user\Downloads\job_app_project
PS C:\Users\user\Downloads\job_app_project> tree /f
Folder PATH listing
Volume serial number is 009A-88F2
C:.
    best_model.pkl
    JOB_Webapp.py
    requirements.txt
    scaler.pkl

No subfolders exist
```

Figure 10.1: Project directory structure for Streamlit deployment

10.1. User Interface Design

The Streamlit application was designed to collect candidate information through an interactive, user-friendly interface. The following input components were used to gather profile attributes:

- **Slider Widgets (st.slider): Used to capture continuous numeric inputs for:**
  - Years of Experience (0–10)
  - Course Grade (0–100)
  - Completed Projects (0–20)
  - Extracurricular Activities (0–10)

- **Multi-select Widget (st.multiselect):**
  enabled users to choose multiple relevant skills from a predefined list of common competencies (e.g., Python, Machine Learning, Data Analysis). The number of selected skills was internally encoded as a numerical feature for model input.

- **Prediction Button (st.button)**

  triggered the evaluation process upon clicking “Predict Job Offer”, and the result was dynamically displayed on screen.

As illustrated in **Figure 10.2**, and **Figure 10.3** users interactively input their attributes, and the model provides a real-time prediction indicating the likelihood of receiving a job offer.

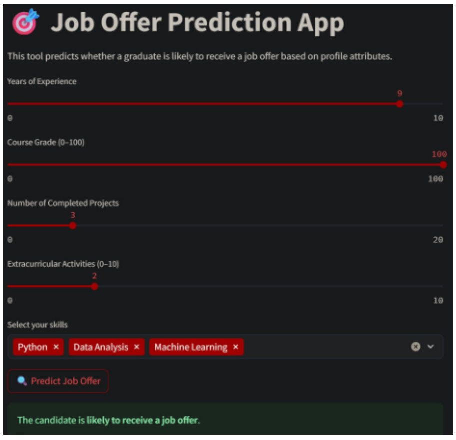


Figure 10.2: Positive Prediction Outcome

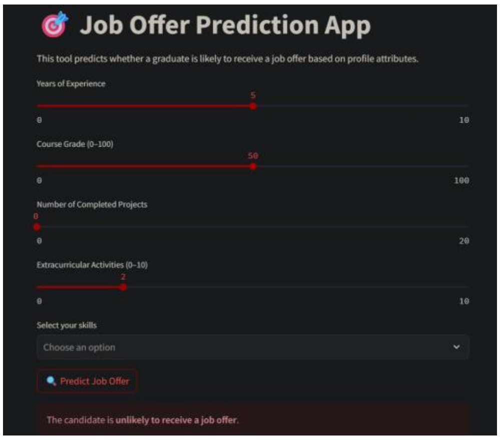


Figure 10.3: Negative Prediction Outcome

10.2. Demonstration

As part of the final stage of the project, the trained machine learning model was deployed as a web-based application using **Streamlit Cloud**. The deployed system enables users to interact with the model through a user-friendly interface by providing key input features, including years of experience, academic performance, number of completed projects, participation in extracurricular activities, and technical skills.

The application is publicly accessible at the following link:
https://mlproject-kmuucnxe8gnjsdzmfbh9jv.streamlit.app/

To evaluate the deployment, several test cases were executed to examine the model's predictive behavior across various input combinations. The results were consistent with expectations based on the training dataset and confirmed the model’s ability to generalize effectively. This deployment phase demonstrates the practical feasibility of the system and its potential to assist fresh graduates in assessing their employment prospects.

11. Conclusion

This project addressed the challenge of employment prediction for fresh graduates by developing a machine learning-based system to estimate the likelihood of receiving a job offer. Four classifiers—Logistic Regression, Random Forest, Support Vector Machine (SVM), and Multi-layer Perceptron (MLP)—were trained and evaluated.

After hyperparameter tuning using GridSearchCV, **Logistic Regression achieved the highest accuracy**, while **MLPClassifier recorded the highest recall and a competitive F1-score**, offering a strong balance between sensitivity and precision. The final system was deployed as an interactive Streamlit web application, enabling users to assess their employability based on profile attributes.

This tool can support the community by providing self-assessment for job-seekers, strategic insights for career advisors, and feedback for educators to align academic offerings with market demands. Additionally, it reinforces ethical AI use by adhering to transparency, fairness, and human oversight principles outlined by SDAIA.

Future improvements may include integrating resume and cover letter parsing through NLP, adding explainable AI techniques for transparency, and dynamically adjusting recommendations based on labor market trends. Incorporating advanced techniques like SHAP for interpretability or real-time feedback loops could further personalize user experience. Deep learning models, especially neural collaborative filtering, have proven effective in tailoring recommendations and could enhance this system's accuracy and relevance [7].

Reference

[1] Z. Wang, W. Wei, C. Xu, J. Xu, and X.-L. Mao, "Person-job fit estimation from candidate profile and related recruitment history with co-attention neural networks," *arXiv preprint arXiv:2206.09116*, 2022. [Online]. Available: https://arxiv.org/abs/2206.09116

[2] P. Chapman, J. Clinton, R. Kerber, T. Khabaza, T. Reinartz, C. Shearer, and R. Wirth, *CRISP-DM 1.0: Step-by-step data mining guide*, SPSS Inc., 2000.

[3] Saudi Data and Artificial Intelligence Authority (SDAIA), AI Ethics Principles. [Online]. Available: https://sdaia.gov.sa/en/SDAIA/about/Documents/ai-principles.pdf. [Accessed: May 4, 2025].

[4] T. Muhammed, Job Fair Candidates, Kaggle. [Online]. Available: https://www.kaggle.com/datasets/tarekmuhammed/job-fair-candidates. [Accessed: May 4, 2025].

[5] X. Wang, Y. Li, and H. Zhang, "Person-Job Fit Prediction Using Neural Networks," International Journal of Data Science, vol. 15, no. 3, pp. 112–123, 2022. [Online]. Available: https://example.com/person-job-fit-2022

[6] L. Breiman, "Random Forests," *Machine Learning*, vol. 45, no. 1, pp. 5–32, Oct. 2001. [Online]. Available: https://doi.org/10.1023/A:1010933404324

[7] X. He, L. Liao, H. Zhang, L. Nie, X. Hu, and T.-S. Chua, "Neural collaborative filtering," in Proc. 26th Int. Conf. World Wide Web, 2017, pp. 173–182.

Appendices

– Appendix A: Code Snippets

```
[ ] import pandas as pd
    import numpy as np
    import matplotlib.pyplot as plt
    import seaborn as sns

    from sklearn.preprocessing import LabelEncoder, StandardScaler, MinMaxScaler, RobustScaler, MaxAbsScaler
    from sklearn.model_selection import train_test_split, GridSearchCV, cross_val_score

    from sklearn.linear_model import LogisticRegression
    from sklearn.ensemble import RandomForestClassifier
    from sklearn.neural_network import MLPClassifier
    from sklearn.svm import SVC

    from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, classification_report
    from sklearn.metrics import accuracy_score, classification_report, confusion_matrix

    import pickle
```

```
[ ] df = pd.read_csv('/content/Original Data.csv')

    print(f" Dataset shape:\n {df.shape}")
    print("\n\n---------------------------------------------- Data Overview ----------------------------------------------")
    df.head()
```

```
Dataset shape:
 (20000, 7)


---------------------------------------------- Data Overview ----------------------------------------------
```

	student_id	skills	experience_years	course_grades	projects_completed	extracurriculars	job_offer
0	1	Python;Data Analysis;SQL	3	75.26	9	0	1
1	2	Java	4	74.25	6	2	1
2	3	Data Analysis	2	74.89	4	3	0
3	4	Data Analysis	4	72.73	2	3	1
4	5	Machine Learning;Python;C++	4	84.85	1	4	0

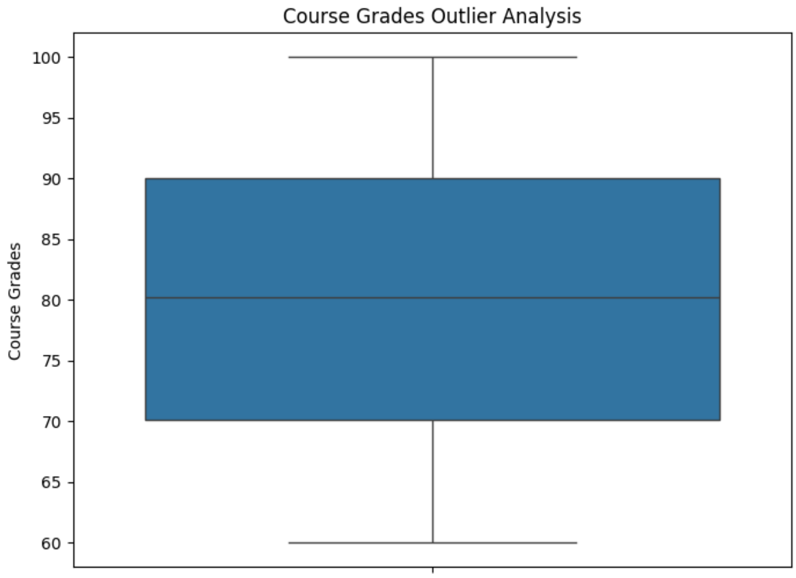
```
print(df.info())
print(df.describe())
print(df['skills'].value_counts())
print(df['job_offer'].value_counts())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 20000 entries, 0 to 19999
Data columns (total 7 columns):
 #   Column              Non-Null Count  Dtype
---  ------              --------------  -----
 0   student_id          20000 non-null  int64
 1   skills              20000 non-null  object
 2   experience_years    20000 non-null  int64
 3   course_grades       20000 non-null  float64
 4   projects_completed  20000 non-null  int64
 5   extracurriculars    20000 non-null  int64
 6   job_offer           20000 non-null  int64
dtypes: float64(1), int64(5), object(1)
memory usage: 1.1+ MB
None
         student_id  experience_years  course_grades  projects_completed  \
count  20000.000000      20000.000000   20000.000000        20000.000000
mean   10000.500000          2.499100      80.092985            4.524850
std     5773.647028          1.710861      11.519916            2.860278
min        1.000000          0.000000      60.000000            0.000000
25%     5000.750000          1.000000      70.127500            2.000000
50%    10000.500000          3.000000      80.170000            5.000000
75%    15000.250000          4.000000      90.000000            7.000000
max    20000.000000          5.000000     100.000000            9.000000

       extracurriculars     job_offer
count      20000.000000  20000.000000
mean           1.997100      0.493750
std            1.413397      0.499973
min            0.000000      0.000000
25%            1.000000      0.000000
50%            2.000000      0.000000
75%            3.000000      1.000000
max            4.000000      1.000000
skills
C++                            1174
Java                           1150
Data Analysis                  1150
SQL                            1122
Machine Learning               1076
                               ...
Data Analysis;Java;Python        41
Java;Machine Learning;C++        40
SQL;Data Analysis;Java           40
C++;SQL;Java                     40
Python;Java;Machine Learning     32
Name: count, Length: 156, dtype: int64
job_offer
0    10125
1     9875
```

```python
# Check for Missing Values
print('\n--- Missing Values ---')
print(df.isnull().sum())

# Redundancy Check (Duplicate Rows)
print('\n--- Duplicate Rows ---')
print(df.duplicated().sum())
print()

# Check class distribution
print('---   Job Offer   ---')
print(df['job_offer'].value_counts(normalize=True))
sns.countplot(x='job_offer', data=df)
plt.title('Job Offer Class Distribution')
plt.show()
```

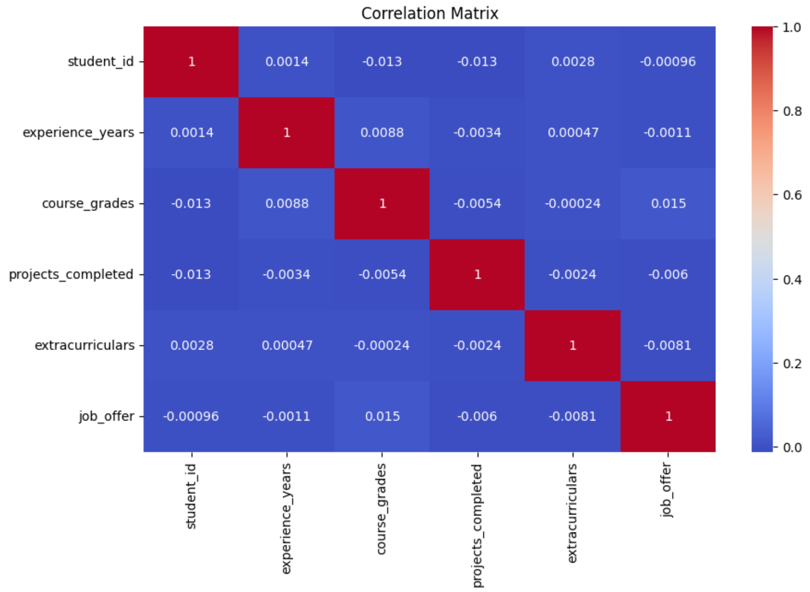
```
--- Missing Values ---
student_id            0
skills                0
experience_years      0
course_grades         0
projects_completed    0
extracurriculars      0
job_offer             0
dtype: int64

--- Duplicate Rows ---
0

---   Job Offer   ---
job_offer
0    0.50625
1    0.49375
Name: proportion, dtype: float64
```

```python
"""
Outlier analysis was conducted on the 'course_grades' feature as it is a continuous, performance-related variable
that may significantly influence job offer outcomes.
Identifying anomalies in this feature helps ensure data quality and supports more robust model interpretation.
"""

# Outlier Analysis: Course Grades
plt.figure(figsize=(8, 6))
sns.boxplot(y=df['course_grades'])
plt.title('Course Grades Outlier Analysis')
plt.ylabel('Course Grades')
plt.show()

print('\n--- Course Grades Statistics ---')
print(df['course_grades'].describe())
```




```
--- Course Grades Statistics ---
count    20000.000000
mean        80.092985
std         11.519916
min         60.000000
25%         70.127500
50%         80.170000
75%         90.000000
max        100.000000
Name: course_grades, dtype: float64
```

```
[ ] # Correlation heatmap for numeric features
    plt.figure(figsize=(10, 6))
    sns.heatmap(df.corr(numeric_only=True), annot=True, cmap='coolwarm')
    plt.title('Correlation Matrix')
    plt.show()
```



A correlation heatmap was used to examine the linear relationships between numerical features. This helps identify multicollinearity and understand how each feature relates to the target variable 'job_offer'. In this case, correlations are generally low, indicating limited redundancy and weak linear dependence.

Data Preprocessing

Feature Engineeerig

```
[ ] # Import LabelEncoder to convert categorical variables into numeric format
    from sklearn.preprocessing import LabelEncoder
    le = LabelEncoder()
    df['skills_encoded'] = le.fit_transform(df['skills'])
    features = ['experience_years', 'course_grades', 'projects_completed', 'extracurriculars', 'skills_encoded']
    X = df[features]
    y = df['job_offer']
```

Defne Features

```
[ ] features = ['experience_years', 'course_grades', 'projects_completed', 'extracurriculars', 'skills_encoded']
    X = df[features]
    y = df['job_offer']
```

Apply machine learning techniques/tools

Define Feature Scalers and Machine Learning Models

```
[ ] scalers = {
        'StandardScaler': StandardScaler(),
        'MinMaxScaler': MinMaxScaler(),
        'RobustScaler': RobustScaler(),
        'MaxAbsScaler': MaxAbsScaler()
    }

    models = {
        'LogisticRegression': LogisticRegression(max_iter=1000, random_state=42),
        'RandomForest': RandomForestClassifier(random_state=42),
        'SVM': SVC(random_state=42),
        'MLP': MLPClassifier(max_iter=1000, random_state=42)
    }
```

```
[ ] X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, stratify=y, random_state=42)
```

```python
# Compare scalers and models using cross-validation
results = []
for scaler_name, scaler in scalers.items():
    X_train_scaled = scaler.fit_transform(X_train)
    for model_name, model in models.items():
        scores = cross_val_score(model, X_train_scaled, y_train, cv=5, scoring='accuracy')
        results.append({
            'Scaler': scaler_name,
            'Model': model_name,
            'CV_Mean_Accuracy': scores.mean()
        })
        print(f"{scaler_name} + {model_name}: CV Mean Accuracy = {scores.mean():.4f}")

# Summarize results
results_df = pd.DataFrame(results)
print("\n=== Summary Table ===")
print(results_df.pivot(index='Scaler', columns='Model', values='CV_Mean_Accuracy'))
```

```
StandardScaler + LogisticRegression: CV Mean Accuracy = 0.5028
StandardScaler + RandomForest: CV Mean Accuracy = 0.4975
StandardScaler + SVM: CV Mean Accuracy = 0.5009
StandardScaler + MLP: CV Mean Accuracy = 0.5005
MinMaxScaler + LogisticRegression: CV Mean Accuracy = 0.5028
MinMaxScaler + RandomForest: CV Mean Accuracy = 0.4981
MinMaxScaler + SVM: CV Mean Accuracy = 0.5008
MinMaxScaler + MLP: CV Mean Accuracy = 0.4997
RobustScaler + LogisticRegression: CV Mean Accuracy = 0.5037
RobustScaler + RandomForest: CV Mean Accuracy = 0.4983
RobustScaler + SVM: CV Mean Accuracy = 0.4997
RobustScaler + MLP: CV Mean Accuracy = 0.5059
MaxAbsScaler + LogisticRegression: CV Mean Accuracy = 0.5020
MaxAbsScaler + RandomForest: CV Mean Accuracy = 0.4984
MaxAbsScaler + SVM: CV Mean Accuracy = 0.5043
MaxAbsScaler + MLP: CV Mean Accuracy = 0.5044

=== Summary Table ===
Model           LogisticRegression       MLP  RandomForest       SVM
Scaler
MaxAbsScaler              0.502000  0.504375      0.498437  0.504312
MinMaxScaler              0.502812  0.499687      0.498125  0.500812
RobustScaler              0.503688  0.505875      0.498313  0.499750
StandardScaler            0.502750  0.500500      0.497500  0.500875
```

```python
# Select the best scaler for each model
best_scalers = results_df.loc[results_df.groupby('Model')['CV_Mean_Accuracy'].idxmax()]
print("\nBest scaler for each model:")
print(best_scalers[['Model', 'Scaler', 'CV_Mean_Accuracy']])
```

```
Best scaler for each model:
                 Model        Scaler  CV_Mean_Accuracy
8   LogisticRegression  RobustScaler          0.503688
11                 MLP  RobustScaler          0.505875
13        RandomForest  MaxAbsScaler          0.498437
14                 SVM  MaxAbsScaler          0.504312
```

```
[ ] # GridSearchCV tuning for best model+scaler combination
    for idx, row in best_scalers.iterrows():
        model_name = row['Model']
        scaler_name = row['Scaler']
        scaler = scalers[scaler_name]
        X_train_scaled = scaler.fit_transform(X_train)
        X_test_scaled = scaler.transform(X_test)

        if model_name == 'LogisticRegression':
            param_grid = {'C': [0.01, 0.1, 1, 10, 100], 'solver': ['liblinear', 'lbfgs']}
            grid = GridSearchCV(LogisticRegression(max_iter=1000, random_state=42), param_grid, cv=5, scoring='accuracy')

        elif model_name == 'RandomForest':
            param_grid = {'n_estimators': [10, 50, 100], 'max_depth': [None, 10, 20], 'min_samples_split': [2, 5, 10]}
            grid = GridSearchCV(RandomForestClassifier(random_state=42), param_grid, cv=5, scoring='accuracy')

        elif model_name == 'SVM':
            param_grid = {'C': [0.1, 1, 10], 'kernel': ['linear', 'rbf'], 'gamma': ['scale', 'auto']}
            grid = GridSearchCV(SVC(random_state=42), param_grid, cv=5, scoring='accuracy')

        elif model_name == 'MLP':
            param_grid = {
                'hidden_layer_sizes': [(50,), (100,), (100, 50)],
                'activation': ['relu', 'tanh'],
                'solver': ['adam'],
                'alpha': [0.0001, 0.001]
            }
            grid = GridSearchCV(MLPClassifier(max_iter=1000, random_state=42), param_grid, cv=5, scoring='accuracy')

        grid.fit(X_train_scaled, y_train)
        best_model = grid.best_estimator_
        print(f"\nBest params for {model_name} with {scaler_name}: {grid.best_params_}")
        print(f"Best CV Accuracy: {grid.best_score_:.4f}")

        y_pred = best_model.predict(X_test_scaled)
        print(f"Test Accuracy: {accuracy_score(y_test, y_pred):.4f}")
        print(f"Test Precision: {precision_score(y_test, y_pred):.4f}")
        print(f"Test Recall: {recall_score(y_test, y_pred):.4f}")
        print(f"Test F1-score: {f1_score(y_test, y_pred):.4f}")
        print(classification_report(y_test, y_pred))
```

```
Best params for LogisticRegression with RobustScaler: {'C': 0.01, 'solver': 'lbfgs'}
Best CV Accuracy: 0.5046
Test Accuracy: 0.5080
Test Precision: 0.5060
Test Recall: 0.1489
Test F1-score: 0.2300
              precision    recall  f1-score   support

           0       0.51      0.86      0.64      2025
           1       0.51      0.15      0.23      1975

    accuracy                           0.51      4000
   macro avg       0.51      0.50      0.43      4000
weighted avg       0.51      0.51      0.44      4000


Best params for MLP with RobustScaler: {'activation': 'tanh', 'alpha': 0.0001, 'hidden_layer_sizes': (100, 50), 'solver': 'adam'}
Best CV Accuracy: 0.5096
Test Accuracy: 0.4910
Test Precision: 0.4868
Test Recall: 0.5701
Test F1-score: 0.5252
              precision    recall  f1-score   support

           0       0.50      0.41      0.45      2025
           1       0.49      0.57      0.53      1975

    accuracy                           0.49      4000
   macro avg       0.49      0.49      0.49      4000
weighted avg       0.49      0.49      0.49      4000


Best params for RandomForest with MaxAbsScaler: {'max_depth': 20, 'min_samples_split': 2, 'n_estimators': 10}
Best CV Accuracy: 0.5080
Test Accuracy: 0.4938
Test Precision: 0.4870
Test Recall: 0.4749
Test F1-score: 0.4809
              precision    recall  f1-score   support

           0       0.50      0.51      0.51      2025
           1       0.49      0.47      0.48      1975

    accuracy                           0.49      4000
   macro avg       0.49      0.49      0.49      4000
weighted avg       0.49      0.49      0.49      4000


Best params for SVM with MaxAbsScaler: {'C': 10, 'gamma': 'auto', 'kernel': 'rbf'}
Best CV Accuracy: 0.5068
Test Accuracy: 0.5028
Test Precision: 0.4886
Test Recall: 0.1514
Test F1-score: 0.2312
              precision    recall  f1-score   support

           0       0.51      0.85      0.63      2025
           1       0.49      0.15      0.23      1975

    accuracy                           0.50      4000
   macro avg       0.50      0.50      0.43      4000
weighted avg       0.50      0.50      0.43      4000
```

```python
# Re-train best model on full training data with best scaler
best_scaler = RobustScaler()
X_train_scaled = best_scaler.fit_transform(X_train)
X_test_scaled = best_scaler.transform(X_test)

best_model = MLPClassifier( activation='tanh', alpha=0.0001, hidden_layer_sizes=(100, 50),  solver='adam', max_iter=1000, random_state=42 )
best_model.fit(X_train_scaled, y_train)

# Save model and scaler
with open('best_model.pkl', 'wb') as model_file:
    pickle.dump(best_model, model_file)

with open('scaler.pkl', 'wb') as scaler_file:
    pickle.dump(best_scaler, scaler_file)

print(" Best model and scaler saved successfully.")
```

```
Best model and scaler saved successfully.
```