Project Documentation: Data Preprocessing for Job Role Suggestion

1. Introduction

This Jupyter Notebook documents the data preprocessing steps performed on the roo_data.csv dataset. The overall goal of the project is to build a machine learning model that suggests suitable job roles for students based on their academic background, skills, and personality traits. This specific notebook focuses exclusively on preparing the raw data, making it suitable for input into various classification algorithms. The key tasks involved are data loading, exploration, cleaning, transformation (encoding and scaling), and feature engineering.

2. Setup and Data Loading

(Corresponds to Cell 1)

- **Objective:** To set up the Python environment and load the dataset into memory.
- **Libraries:** I began by ensuring all necessary libraries were installed using pip. Then, I imported the core libraries:
    - pandas for data manipulation (DataFrames).
    - numpy for numerical operations.
    - seaborn and matplotlib.pyplot for data visualization.
    - sklearn.preprocessing (specifically LabelEncoder, StandardScaler) for data transformation.
    - sklearn.model_selection (train_test_split) for splitting the data later.
    - google.colab.files for uploading the data file in the Google Colab environment.
- **Data Loading:** I used the files.upload() function to upload the roo_data.csv file. The uploaded file was then read into a pandas DataFrame named df using pd.read_csv().
- **Initial Verification:** To confirm successful loading, I printed the shape of the DataFrame (df.shape) and displayed the first 5 rows using df.head(). This initial check showed the dataset contains 20,000 entries (students) and 39 features.

3. Exploratory Data Analysis (EDA)

(Corresponds to Cell 2)

- **Objective:** To understand the structure, data types, potential issues (like missing values), and basic statistical properties of the dataset.
- **Dataset Overview:**
    - I re-confirmed the shape: (20000, 39).
    - I listed all column names using df.columns.tolist().
    - I checked the data types using df.dtypes.value_counts(). This revealed a mix of data types: 14 numerical (int64) columns and 25 categorical (object, i.e., text) columns. This mix necessitates encoding for the categorical features.
- **Missing Value Analysis:**
    - A critical step was checking for missing data using df.isnull().sum(). The analysis showed **"No missing values found!"**. This simplifies the process, as no

imputation techniques (like filling missing values with mean, median, or mode) are required.

- **Numerical Statistics:**
    - I used `df.describe()` on the numerical columns identified earlier. This provided summary statistics (count, mean, standard deviation, min, max, quartiles) for each numerical feature. This helped understand the scale and distribution – for example, academic percentages range roughly 60-94, while ratings are typically 1-9, and hackathons 0-6. The differing scales highlighted the need for feature scaling later.

4. Data Preprocessing and Feature Engineering

(Corresponds to Cells 3, 4, 5)

- **Objective:** To transform the raw, mixed-type data into a fully numerical format suitable for machine learning algorithms, and to create potentially useful new features. I created a copy of the original DataFrame (`df_processed = df.copy()`) to avoid modifying the raw data.
- **4.1. Encoding Categorical Variables (Cell 3):**
    - **Rationale:** Machine learning models generally require numerical input. Therefore, all columns containing text data needed conversion.
    - **Binary Encoding:** For columns with simple 'yes'/'no' answers (like `'self-learning capability?'`, `'Introvert'`, etc.), I mapped 'yes' to 1 and 'no' to 0 directly using a dictionary and the `.map()` function. This is a straightforward way to represent binary choices numerically.
    - **Ordinal Encoding:** For categorical features possessing a natural order (e.g., `'reading and writing skills': ['poor', 'medium', 'excellent']`), I used `LabelEncoder` from scikit-learn. This assigns numerical values (0, 1, 2...) while attempting to preserve the inherent ranking (though careful interpretation is needed, here `excellent` got 0, `medium` 1, `poor` 2 based on alphabetical order during fit, which might need adjustment depending on the model). I printed the mapping for clarity. Features like `'Gentle or Tuff behaviour?'` and `'hard/smart worker'` were also treated as ordinal/binary here.
    - **Nominal (Label) Encoding:** For other categorical columns with no inherent order but relatively few unique values (like `'Management or Technical'`, `'Salary/work'`), I also applied `LabelEncoder`. This assigns a unique integer to each category. I saved the encoders in case the mapping needs to be reversed later. The number of unique categories encoded was printed. `'Interested Type of Books'` was also label encoded despite having 31 categories.
    - **Nominal (One-Hot) Encoding:** For categorical features with a moderate number of unique values where numerical ordering would be meaningless (like `'Interested subjects'`), I applied One-Hot Encoding using `pd.get_dummies()`. This creates new binary (0 or 1) columns for each unique category, preventing the model from assuming any ordinal relationship between subjects. This increased

the number of features. I applied this to 'Interested subjects' and intended to apply it to 'interested career area ' and 'Type of company want to settle in?' but they were commented out or removed in the final execution shown.
    - **Target Variable Encoding:** Finally, the target variable, 'Suggested Job Role', was encoded using LabelEncoder. This converts the job titles into numerical labels that the model will predict. I stored this specific encoder (target_encoder) separately, as it's crucial for interpreting the model's predictions later. The mapping from job titles to numbers was implicitly created, and the list of unique job roles was printed.
- **4.2. Scaling Numerical Features (Cell 4):**
    - **Rationale:** Numerical features had varying scales (e.g., percentages 60-94 vs. ratings 1-9). Algorithms sensitive to feature scales (like SVM, KNN, or those using gradient descent) can be biased towards features with larger ranges. Standardization rescales features to have a mean of 0 and a standard deviation of 1.
    - **Implementation:** I used StandardScaler from scikit-learn. I identified the numerical columns, fitted the scaler on them (scaler.fit_transform()), and replaced the original columns with their scaled versions.
- **4.3. Feature Engineering (Cell 4):**
    - **Rationale:** To potentially provide the model with more informative inputs by combining related existing features.
    - **Implementation:** I created three new features:
        - academic_performance_score: Calculated as the mean of all columns containing 'percentage' in their name. This gives a single measure of overall academic strength.
        - technical_skills_score: Calculated as the mean of 'coding skills rating', 'Logical quotient rating', and 'hackathons'. This aggregates technical aptitude indicators.
        - soft_skills_score: Calculated as the mean of 'public speaking points', 'Communication skills', and 'reading and writing skills'. This aggregates communication and interpersonal skill indicators. (Note: The code only included 2 metrics eventually, as 'Communication skills' was likely scaled numeric, not ordinal encoded).
- **4.4. Final Cleanup (Cell 5):**
    - **Rationale:** To remove columns that were deemed unnecessary or redundant after encoding/feature engineering.
    - **Implementation:** I dropped columns like 'certifications', 'workshops', etc., using df_processed.drop().
    - **Verification:** I printed the shapes of the original and processed DataFrames to see the change (39 to 46 columns due to one-hot encoding). I confirmed again that there were no missing values (df_processed.isnull().sum().sum()) and displayed the head of the final, fully numerical df_processed DataFrame.

5. Preparing for Machine Learning Model Training

(Corresponds to Cell 6)

- **Objective:** To split the processed data into training and testing sets for model development and evaluation.
- **Train-Test Split:**
    - **Rationale:** It's essential to train the model on one portion of the data and test its performance on a separate, unseen portion to get an unbiased estimate of how it will perform on new data.
    - **Implementation:** I used the `train_test_split` function from scikit-learn. I designated `df_processed` excluding the target column as `X` (features) and the `'Suggested Job Role'` column as `y` (target). I specified a `test_size` of 0.2 (meaning 20% of the data for testing, 80% for training) and set a `random_state` for reproducibility.
- **Stratification:**
    - **Rationale:** Since this is a classification problem, it's important that the proportion of each job role (class) is the same in both the training and testing sets. This is especially important if some job roles are much less frequent than others (class imbalance). Stratification ensures this proportional representation.
    - **Implementation:** I achieved this by setting the `stratify=y` parameter within `train_test_split`.
- **Verification:** I printed the shapes of the resulting `X_train`, `X_test`, `y_train`, and `y_test` sets. I also printed the distribution (counts and percentages) of each job role in the original data, the training data, and the testing data to confirm that the stratification worked correctly. The distributions were indeed nearly identical across the sets.
- **Saving Data:** The fully processed DataFrame (`df_processed`) was saved to a new CSV file (`processed_student_job_data.csv`) for potential future use without repeating the preprocessing steps. A download link was generated using `files.download()`.

6. Data Visualization

(Corresponds to Cells 7 & 8 - Note: Cell 7 from previous response seems missing in provided notebook, but Cell 8 corresponds to the Histograms, and Cell 17 in the notebook corresponds to the Box plots, Cell 14 to the Bar chart)

- **Objective:** To visually explore distributions and relationships in the processed data.
- **Distribution of Engineered Features (Cell 16 in notebook):**
    - I created histograms using `seaborn.histplot` for the three engineered features (`academic_performance_score`, `technical_skills_score`, `soft_skills_score`). This helps visualize their distributions after scaling (they appear roughly normally distributed around zero, as expected from standardization).
- **Scores vs. Role Preference (Cell 17 in notebook):**
    - I created box plots using `seaborn.boxplot` to compare the distributions of the engineered scores between students interested in 'Management' vs. 'Technical' roles (using the original, unencoded `'Management or Technical'` column alongside

the processed scores). This visual comparison can indicate if certain scores differ significantly between these two broad career interests. The plots suggest subtle differences, potentially useful for the prediction model.

- **Job Role Distribution (Cell 14 in notebook):**
    - I generated a bar chart showing the frequency of each suggested job role in the dataset using `matplotlib.pyplot.bar`. I used the `target_encoder` to display the actual job names on the x-axis instead of the numerical labels. This visualization clearly shows the number of students associated with each role, highlighting that 'Network Security Administrator' is the most frequent suggestion, while others like 'Programmer Analyst' are less common. This confirms the mild class imbalance observed during the train-test split analysis.

7. Conclusion

This notebook successfully performed the necessary preprocessing on the `roo_data.csv` dataset. The data was loaded, explored, and cleaned. Categorical features were appropriately encoded using binary, ordinal, and one-hot methods. Numerical features were standardized to a common scale. Three new features summarizing academic, technical, and soft skills were engineered. Finally, the data was split into stratified training and testing sets (`X_train`, `y_train`, `X_test`, `y_test`). The resulting datasets are now entirely numerical and ready for use in training and evaluating various machine learning classification models for the job role suggestion task.