

Data Cleaning and Analysis

06/05/2025

stratified_history.csv
8.1 MB

Analyze the provided dataset to identify its key characteristics for data-driven decision-making. Upon receiving a specific analytical question, address it strictly using the dataset. Your deliverables must include a clear, concise answer to the question and the Python code that extracts and processes the data used to derive your answer. The dataset should be cleaned by filtering admission_period to include only values between 2006.1 and 2019.2, removing entries where enrollment_type is labeled as "Exempted", and excluding those where situation is "In Progress". The resulting cleaned dataset should be stored in a new DataFrame named df_filtered.

qwen-max-latest

To address the task, I will first load the dataset and clean it according to the specified criteria. Then, I will analyze the cleaned dataset to identify its key characteristics for data-driven decision-making. Finally, I will provide Python code that performs these operations.

Key Characteristics of the Dataset

The dataset contains historical academic records of students, including details about their admission, enrollment, performance, demographics, and outcomes. Key fields in the dataset include:

- Admission Period** (admission_period) : Indicates when the student was admitted.
- Enrollment Type** (enrollment_type) : Describes how the student enrolled (e.g., Entrance Exam, Unified Selection System).
- Situation** (situation) : The current status of the student (e.g., Graduate, Dropout).
- Demographics** : Includes gender, race, and schooling background.
- Academic Performance** : Fields like GPA, course grades, and credit hours.
- Course Details** : Information about specific courses taken by the student.

Cleaning Criteria

- Filter admission_period to include only values between 2006.1 and 2019.2.
- Remove entries where enrollment_type is labeled as "Exempted".
- Exclude entries where situation is "In Progress".

Python Code for Data Cleaning

Below is the Python code to clean the dataset and store the result in a new DataFrame named df_filtered .

```
python
1 import pandas as pd
2
3 # Load the dataset
4 file_path = 'stratified_history.csv'
5 df = pd.read_csv(file_path)
6
7 # Display initial dataset info
8 print("Initial Dataset Shape:", df.shape)
9
10 # Cleaning Step 1: Filter admission_period between 2006.1 and 2019.2
11 df_filtered = df[(df['admission_period'] >= 2006.1) & (df['admission_period'] <= 2019.2)]
12
13 # Cleaning Step 2: Remove rows where enrollment_type is "Exempted"
14 df_filtered = df_filtered[df_filtered['enrollment_type'] != 'Exempted']
15
16 # Cleaning Step 3: Exclude rows where situation is "In Progress"
17 df_filtered = df_filtered[df_filtered['situation'] != 'In Progress']
18
19 # Display cleaned dataset info
20 print("Filtered Dataset Shape:", df_filtered.shape)
21
22 # Save the cleaned dataset to a new CSV file (optional)
23 df_filtered.to_csv('filtered_dataset.csv', index=False)
```

Explanation of the Code

- Loading the Dataset** : The dataset is loaded using pd.read_csv .
- Filtering Admission Period** : Rows are filtered based on the admission_period column to include only values between 2006.1 and 2019.2.
- Removing Exempted Enrollments** : Entries with enrollment_type labeled as "Exempted" are removed.
- Excluding In Progress Situations** : Rows where situation is "In Progress" are excluded.
- Saving the Cleaned Dataset** : The cleaned dataset is optionally saved to a new CSV file for further use.

Key Characteristics of the Cleaned Dataset

After cleaning, the dataset will contain:

- Only students admitted between 2006.1 and 2019.2.
- Students who enrolled through standard processes (excluding "Exempted").
- Students whose academic situations are resolved (excluding "In Progress").

This cleaned dataset can now be used for data-driven decision-making, such as analyzing dropout rates, graduation trends, or demographic influences on academic performance.

Let me know if you have a specific analytical question to address using this cleaned dataset!

Evaluate the academic performance (e.g., average grade, improvement rate) of the bottom 10% of students in mathematics sector courses. Propose three analytical methods and select the best option.



qwen-max-latest

To evaluate the academic performance of the bottom 10% of students in mathematics sector courses, we will follow a structured approach. The process involves identifying the relevant data, defining metrics for performance evaluation, and proposing analytical methods to assess the bottom 10%. Finally, we will select the best method based on its robustness, interpretability, and alignment with the dataset.

Step 1: Data Preparation

We will filter the dataset to focus on mathematics sector courses. These are identified by the `Academic Unit` column containing "Mathematics". Additionally, we will:

- Exclude rows where `situation` is "In Progress" or `enrollment_type` is "Exempted".
- Filter `admission_period` between 2006.1 and 2019.2.

The cleaned dataset (`df_filtered`) will be used for further analysis.

Step 2: Metrics for Academic Performance

We define two key metrics to evaluate performance:

- Average Grade** : The mean grade (`final_grade`) across all mathematics courses for each student.
- Improvement Rate** : The rate of improvement in grades over time, calculated as:

$$\text{Improvement Rate} = \frac{\text{Final Grade} - \text{Initial Grade}}{\text{Time Period (semesters)}}$$

Step 3: Identifying the Bottom 10%

The bottom 10% of students will be identified based on their average grade in mathematics courses. This ensures we focus on the lowest-performing students.

Step 4: Analytical Methods

We propose three analytical methods to evaluate the academic performance of the bottom 10%:

Method 1: Simple Average Grade Analysis

- Calculate the average grade of the bottom 10% of students.
- Compare this average grade to the overall average grade of all students in mathematics courses.
- Strengths: Simple and interpretable.
- Limitations: Ignores temporal trends and individual improvement.

Method 2: Improvement Rate Analysis

- Calculate the improvement rate for each student in the bottom 10%.
- Analyze the distribution of improvement rates and identify patterns (e.g., whether most students show improvement or stagnation).
- Strengths: Captures temporal trends in performance.
- Limitations: Requires sufficient longitudinal data for each student.

Method 3: Regression-Based Analysis

- Fit a linear regression model to predict grades based on factors such as `time_period`, `gender`, `race`, and `schooling_background`.
- Use the model to identify predictors of poor performance among the bottom 10%.
- Strengths: Provides insights into contributing factors for low performance.
- Limitations: More complex and requires careful interpretation.

Step 5: Selecting the Best Method

The best method depends on the goals of the analysis:

- If simplicity and interpretability are prioritized, **Method 1** (Simple Average Grade Analysis) is the best choice.
- If understanding temporal trends is important, **Method 2** (Improvement Rate Analysis) is more suitable.
- If identifying underlying causes of poor performance is critical, **Method 3** (Regression-Based Analysis) is ideal.

For this task, we prioritize simplicity and interpretability while still capturing meaningful insights. Therefore, **Method 1** is selected as the best option.

Python Code Implementation

Below is the Python code to implement Method 1:

```
python
1 import pandas as pd
2
3 # Load and clean the dataset
4 file_path = 'stratified_history.csv'
5 df = pd.read_csv(file_path)
6
7 # Cleaning steps
8 df_filtered = df[
9     (df['admission_period'] >= 2006.1) &
10    (df['admission_period'] <= 2019.2) &
11    (df['enrollment_type'] != 'Exempted') &
12    (df['situation'] != 'In Progress')
13 ]
14
15 # Filter for mathematics courses
16 math_courses = df_filtered[df_filtered['academic_unit'].str.contains('Mathematics', na=False)]
17
18 # Group by student and calculate average grade
```

```
19 student_avg_grade = math_courses.groupby('student_id')['final_grade'].mean().reset_index()
20 student_avg_grade.rename(columns={'final_grade': 'avg_grade'}, inplace=True)
21
22 # Identify the bottom 10%
23 bottom_10_percent = student_avg_grade['avg_grade'].quantile(0.1)
24 bottom_students = student_avg_grade[student_avg_grade['avg_grade'] <= bottom_10_percent]
25
26 # Calculate the average grade of the bottom 10%
27 bottom_avg_grade = bottom_students['avg_grade'].mean()
28
29 # Compare with overall average grade
30 overall_avg_grade = student_avg_grade['avg_grade'].mean()
31
32 print(f"Average Grade of Bottom 10%: {bottom_avg_grade:.2f}")
33 print(f"Overall Average Grade: {overall_avg_grade:.2f}")
```

Expected Output

- 1. **Average Grade of Bottom 10%** : A numerical value representing the mean grade of the lowest-performing students.
- 2. **Overall Average Grade** : A numerical value representing the mean grade of all students in mathematics courses.

This output will provide a clear comparison of the academic performance of the bottom 10% relative to the overall student population.

Conclusion

By implementing **Method 1** , we can effectively evaluate the academic performance of the bottom 10% of students in mathematics courses. This approach is straightforward, interpretable, and provides actionable insights for targeted interventions to improve performance.

