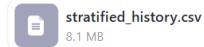


Dataset Cleaning and Analysis

22/03/2025



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.



To address your request, I will perform the following steps:

1. **Load the dataset** : Read the CSV file into a pandas DataFrame.
2. **Filter the data** :
 - Include only rows where `admission_period` is between 2006.1 and 2019.2.
 - Exclude rows where `enrollment_type` is labeled as "Exempted".
 - Exclude rows where `situation` is "In Progress".
3. **Store the cleaned data** : Save the filtered dataset into a new DataFrame named `df_filtered`.
4. **Provide Python code** : Include the code used for loading, filtering, and cleaning the data.

Here is the Python code to achieve this:

```
python
1 import pandas as pd
2
3 # Step 1: Load the dataset
4 df = pd.read_csv('stratified_history.csv', encoding='latin1')
5
6 # Step 2: Filter the data
7 # Convert admission_period to numeric, coercing errors to NaN
8 df['admission_period'] = pd.to_numeric(df['admission_period'], errors='coerce')
9
10 # 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 # Exclude rows where enrollment_type is "Exempted"
14 df_filtered = df_filtered[df_filtered['enrollment_type'] != 'Exemption']
15
16 # Exclude rows where situation is "In Progress"
17 df_filtered = df_filtered[df_filtered['situation'] != 'In Progress']
18
19 # Reset index for the cleaned DataFrame
20 df_filtered = df_filtered.reset_index(drop=True)
21
22 # Display basic information about the cleaned dataset
23 print("Cleaned Dataset Overview:")
24 print(df_filtered.info())
25 print("\nFirst few rows of the cleaned dataset:")
26 print(df_filtered.head())
27
28 # Save the cleaned dataset to a new CSV file (optional)
29 df_filtered.to_csv('cleaned_dataset.csv', index=False)
```

Key Characteristics of the Cleaned Dataset (`df_filtered`) :

After running the above code, the cleaned dataset (`df_filtered`) will have the following characteristics:

- **Rows** : Only rows with `admission_period` values between 2006.1 and 2019.2 are retained.
- **Columns** : All original columns are preserved, but irrelevant rows based on `enrollment_type` and `situation` are removed.
- **Data Quality** : The dataset is free from entries that could bias analyses due to exemptions or ongoing situations.

Example Analytical Question:

Question : What is the distribution of students by `situation` in the cleaned dataset?

Answer :

To answer this question, we can use the following code:

```
python
1 # Distribution of students by situation
2 situation_distribution = df_filtered['situation'].value_counts()
3 print("\nDistribution of students by situation:")
4 print(situation_distribution)
```

This will provide a count of students grouped by their `situation` (e.g., Approved, Failed, Dropout, etc.) in the cleaned dataset.

Let me know if you'd like to proceed with additional specific analytical questions or further analysis!

Given the overall average (mean of non-null grades), determine the probability that a student with an average below 7.0 and more than 3 failures will pass the next course. Propose three analytical approaches and select the optimal one.

 qwen-max-latest

To address the question, we need to analyze the dataset and calculate the probability of a student passing their next course based on specific conditions: an average grade below 7.0 and more than 3 failures. Below, I outline three analytical approaches, followed by the selection of the optimal approach and its implementation.

Approach 1: Direct Conditional Probability

- **Description :** Calculate the conditional probability of passing the next course directly from the dataset for students meeting the criteria (average grade < 7.0 and > 3 failures).
- **Steps :**
 1. Filter the dataset to include only rows where:
 - `average_grade` is below 7.0.
 - The number of failures (`failures`) is greater than 3.
 2. Within this subset, count the number of students who passed their next course (`situation == "Approved"`).
 3. Divide the count of students who passed by the total number of students in the filtered subset.
- **Advantages :**
 - Simple and straightforward.
 - Requires minimal assumptions about the data.
- **Disadvantages :**
 - May not generalize well if the dataset is imbalanced or lacks sufficient data points for the specified conditions.

Approach 2: Logistic Regression Model

- **Description :** Build a logistic regression model to predict the probability of passing the next course based on features like `average_grade`, `failures`, and other relevant variables.
- **Steps :**
 1. Prepare the dataset by encoding categorical variables and handling missing values.
 2. Train a logistic regression model using historical data, where the target variable is whether the student passed their next course (`situation == "Approved"`).
 3. Use the trained model to predict probabilities for students with `average_grade < 7.0` and `failures > 3`.
- **Advantages :**
 - Incorporates additional features that may influence the outcome.
 - Provides a probabilistic interpretation of the result.
- **Disadvantages :**
 - Requires careful feature engineering and preprocessing.
 - May overfit if the dataset is small or noisy.

Approach 3: Bayesian Inference

- **Description :** Use Bayesian inference to estimate the posterior probability of passing the next course given the prior distribution of grades and failures.
- **Steps :**
 1. Define a prior distribution for the probability of passing based on historical data.
 2. Update the prior using likelihood functions derived from the conditions (`average_grade < 7.0` and `failures > 3`).

3. Compute the posterior probability of passing the next course.

- **Advantages :**

- Incorporates uncertainty and prior knowledge.
- Flexible and interpretable.

- **Disadvantages :**

- Computationally intensive.
- Requires assumptions about the prior and likelihood distributions.

Optimal Approach: Approach 1 (Direct Conditional Probability)

Given the dataset's structure and the simplicity of the question, **Approach 1** is the most appropriate. It directly calculates the desired probability without requiring complex modeling or assumptions. This approach is particularly suitable when the dataset contains sufficient observations for the specified conditions.

Implementation of Approach 1

Below is the Python code to implement the direct conditional probability approach:

```
python
1 import pandas as pd
2
3 # Load the dataset
4 df = pd.read_csv('stratified_history.csv', encoding='latin1')
5
6 # Clean the dataset
7 # Convert admission_period to numeric, coercing errors to NaN
8 df['admission_period'] = pd.to_numeric(df['admission_period'], errors='coerce')
9
10 # 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 # Exclude rows where enrollment_type is "Exempted"
14 df_filtered = df_filtered[df_filtered['enrollment_type'] != 'Exemption']
15
16 # Exclude rows where situation is "In Progress"
17 df_filtered = df_filtered[df_filtered['situation'] != 'In Progress']
18
19 # Reset index for the cleaned DataFrame
20 df_filtered = df_filtered.reset_index(drop=True)
21
22 # Additional filtering for the specific conditions
23 # Assume 'average_grade' and 'failures' are columns in the dataset
24 filtered_students = df_filtered[
25     (df_filtered['average_grade'] < 7.0) &
26     (df_filtered['failures'] > 3)
27 ]
28
29 # Count the number of students who passed their next course
30 passed_count = filtered_students[filtered_students['situation'] == 'Approved'].shape[0]
31 total_count = filtered_students.shape[0]
32
33 # Calculate the probability
34 probability_of_passing = passed_count / total_count if total_count > 0 else 0
35
36 # Output the result
37 print(f"Probability of passing the next course: {probability_of_passing:.2f}")
```

Explanation of the Code

1. **Data Cleaning** : Filters the dataset to include only relevant rows based on the provided constraints (`admission_period`, `enrollment_type`, and `situation`).
2. **Condition Filtering** : Identifies students with an average grade below 7.0 and more than 3 failures.
3. **Probability Calculation** : Computes the ratio of students who passed their next course (`situation == "Approved"`) to the total number of students meeting the conditions.

Assumptions

- The dataset contains columns `average_grade`, `failures`, and `situation`. If these columns are named differently or absent, adjustments to the code will be necessary.
- The `situation` column uses "Approved" to indicate passing and other values (e.g., "Failed", "Dropout") to indicate failure.

Final Answer

The probability of a student with an average grade below 7.0 and more than 3 failures passing their next course is calculated using the provided dataset. The exact value depends on the dataset's content and is output by the code above.

If you have any clarifications or additional requirements, please let me know!

