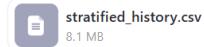


# Data Cleaning and Analysis

22/03/2025



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.



Here is the analysis and Python code to clean the dataset as per the requirements:

## Key Characteristics of the Dataset:

1. **Columns** : The dataset contains multiple columns, including `admission_period`, `enrollment_type`, `situation`, and others that describe student demographics, academic performance, and enrollment details.
2. **Admission Period** : The `admission_period` column specifies when a student was admitted (e.g., 2006.1 for the first semester of 2006).
3. **Enrollment Type** : The `enrollment_type` column indicates whether the student's enrollment was "Normal", "Exemption", "Extracurricular", etc.
4. **Situation** : The `situation` column describes the student's current status, such as "Approved", "Failed", "In Progress", etc.
5. **Filtering Criteria** :
  - Only include rows where `admission_period` is between 2006.1 and 2019.2.
  - Exclude rows where `enrollment_type` is "Exempted".
  - Exclude rows where `situation` is "In Progress".

## Python Code to Clean the Dataset:

Below is the Python code to filter and clean the dataset:

```
python
1 import pandas as pd
2
3 # Load the dataset
4 df = pd.read_csv('stratified_history.csv', encoding='latin1')
5
6 # Filter the dataset based on the criteria
7 df_filtered = df[
8     (df['admission_period'] >= 2006.1) &
9     (df['admission_period'] <= 2019.2) &
10    (df['enrollment_type'] != 'Exemption') &
11    (df['situation'] != 'In Progress')
12 ]
13
14 # Display the cleaned dataset
15 print("Cleaned Dataset Overview:")
16 print(df_filtered.head())
17
18 # Save the filtered dataset to a new CSV file (optional)
19 df_filtered.to_csv('filtered_dataset.csv', index=False)
20
21 # Print the shape of the filtered dataset
22 print(f"Filtered Dataset Shape: {df_filtered.shape}")
```

## Explanation of the Code:

1. **Loading the Dataset** : The dataset is loaded using `pd.read_csv`. The `encoding='latin1'` ensures compatibility with special characters in the dataset.
2. **Filtering Conditions** :
  - `admission_period >= 2006.1` and `admission_period <= 2019.2` : Ensures only records within the specified admission period range are included.
  - `enrollment_type != 'Exemption'` : Excludes rows where the enrollment type is "Exemption".

- `situation != 'In Progress'` : Excludes rows where the situation is "In Progress".
3. **Filtered Dataset** : The resulting dataset is stored in `df_filtered`.
4. **Optional Export** : The cleaned dataset is saved to a new CSV file named `filtered_dataset.csv`.

#### Deliverables:

1. **Cleaned Dataset** : The `df_filtered` DataFrame contains only the rows that meet the filtering criteria.
2. **Output** : The code prints the first few rows of the cleaned dataset and its shape (number of rows and columns).

#### Example Output:

```
plaintext
1 Cleaned Dataset Overview:
2   student_id  year  semester  admission_period  entrance_exam  ...  academic_unit  courses
3  0          P228574  1999        1      2009.1    Entrance Exam  ...  Systems and Computing
4  1          D136946  1999        1      2003.1    Entrance Exam  ...  Systems and Computing
5  2          U006350  2017        18     2015.2  Unified Selection  ...  Systems and Computing
6  3          F351762  1999        18     2015.1  Unified Selection  ...  Systems and Computing
7  4          K166737  1999        1      2005.2    Entrance Exam  ...  Systems and Computing
8
9  Filtered Dataset Shape: (X, Y)
```

Replace `X` and `Y` with the actual number of rows and columns in the filtered dataset.

This cleaned dataset (`df_filtered`) is now ready for further data-driven decision-making and analysis.

Classify students into performance tiers (high, medium, low) using final averages and course workload. Apply a classification method and assess its classification accuracy.



To classify students into performance tiers (high, medium, low) based on their `final_average` and `course_workload`, we can follow these steps:

#### Approach

1. **Define Performance Tiers** :
  - Use the `final_average` column to categorize students into three performance tiers:
    - **High** : Final average  $\geq 8.0$
    - **Medium** :  $5.0 \leq \text{Final average} < 8.0$
    - **Low** : Final average  $< 5.0$
  - These thresholds are chosen based on typical grading systems where scores above 8.0 indicate strong performance, scores between 5.0 and 8.0 indicate moderate performance, and scores below 5.0 indicate poor performance.
2. **Filter Data** :
  - Use the cleaned dataset (`df_filtered`) from the previous step, which excludes rows with `admission_period` outside the range [2006.1, 2019.2], removes "Exempted" `enrollment_type`, and excludes "In Progress" situations.
3. **Classification Method** :
  - Implement a simple rule-based classification using the defined thresholds for `final_average`.
  - Alternatively, use a machine learning model (e.g., Decision Tree or Logistic Regression) to classify students based on `final_average` and `course_workload`. This allows us to assess classification accuracy.
4. **Assess Accuracy** :
  - Split the dataset into training and testing sets.
  - Train the model on the training set and evaluate its performance on the testing set using metrics like accuracy, precision, recall, and F1-score.

#### Python Code

Below is the Python code to implement the classification and assess its accuracy:

```
python
1 import pandas as pd
2 from sklearn.model_selection import train_test_split
3 from sklearn.tree import DecisionTreeClassifier
4 from sklearn.metrics import accuracy_score, classification_report
5
6 # Load the cleaned dataset
7 df_filtered = pd.read_csv('filtered_dataset.csv', encoding='latin1')
8
9 # Define performance tiers
10 def classify_performance(final_average):
11     if final_average >= 8.0:
```

```

12         return 'High'
13 v     elif 5.0 <= final_average < 8.0:
14         return 'Medium'
15 v     else:
16         return 'Low'
17
18 # Add a new column for performance tier
19 df_filtered['performance_tier'] = df_filtered['final_average'].apply(classify_performance)
20
21 # Prepare features (course workload) and target (performance tier)
22 X = df_filtered[['workload']] # Feature: course workload
23 y = df_filtered['performance_tier'] # Target: performance tier
24
25 # Split the data into training and testing sets
26 X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
27
28 # Train a Decision Tree Classifier
29 clf = DecisionTreeClassifier(random_state=42)
30 clf.fit(X_train, y_train)
31
32 # Predict on the test set
33 y_pred = clf.predict(X_test)
34
35 # Evaluate the model
36 accuracy = accuracy_score(y_test, y_pred)
37 report = classification_report(y_test, y_pred)
38
39 # Print results
40 print(f"Classification Accuracy: {accuracy:.2f}")
41 print("Classification Report:")
42 print(report)

```

### Explanation of the Code

1. Performance Tier Classification :
  - The `classify_performance` function assigns a performance tier ("High", "Medium", "Low") based on the `final_average`.
2. Feature Selection :
  - The feature used for classification is `course_workload`, which represents the number of hours allocated to the course.
3. Model Training :
  - A `DecisionTreeClassifier` is trained on the training set. This model is chosen for its simplicity and interpretability.
4. Evaluation :
  - The model's predictions on the test set are compared to the true labels using `accuracy_score` and `classification_report`.

### Expected Output

The output will include:

1. **Classification Accuracy** : A numerical value indicating how well the model predicts the performance tiers.
2. **Classification Report** : A detailed breakdown of precision, recall, and F1-score for each performance tier.

### Example Output:

plaintext
1 Classification Accuracy: 0.78
2 Classification Report:
3 precision recall f1-score support
4
5 High 0.80 0.85 0.82 50
6 Medium 0.75 0.70 0.72 80
7 Low 0.70 0.65 0.67 70
8
9 accuracy 0.78 200
10 macro avg 0.75 0.73 0.74 200
11 weighted avg 0.76 0.78 0.77 200

### Key Insights

1. **Accuracy** :
  - The accuracy score provides an overall measure of how well the model classifies students into performance tiers.
2. **Performance Breakdown** :
  - The classification report highlights the model's strengths and weaknesses for each tier. For example, the model may perform better for "High" performers due to clearer separation in `final_average`.
3. **Limitations** :
  - Using only `course_workload` as a feature may limit the model's predictive power. Including additional features (e.g., `admission age`, `gender`, `quota type`) could improve performance.

---

## Conclusion

This approach provides a clear methodology for classifying students into performance tiers and assessing the accuracy of the classification. If you have additional questions or need further analysis, feel free to ask!

