

# Data Cleaning and Filtering Code

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.



To address the request, we will perform the following steps:

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

Here is the Python code to achieve this:

```
python
1 import pandas as pd
2
3 # Load the dataset
4 df = pd.read_csv('stratified_history.csv')
5
6 # Filter the dataset based on the specified criteria
7 df_filtered = df[
8     (df['admission_period'] >= 2006.1) &
9     (df['admission_period'] <= 2019.2) &
10    (df['enrollment_type'] != 'Exempted') &
11    (df['situation'] != 'In Progress')
12 ]
13
14 # Display basic information about the cleaned dataset
15 print("Shape of the filtered dataset:", df_filtered.shape)
16 print("Columns in the filtered dataset:", df_filtered.columns.tolist())
17
18 # Save the filtered dataset to a new CSV file (optional)
19 df_filtered.to_csv('filtered_dataset.csv', index=False)
```

## Explanation of the Code:

1. **Loading the Dataset** : The `pd.read_csv()` function reads the CSV file into a pandas DataFrame.
2. **Filtering** :
  - The condition `(df['admission_period'] >= 2006.1) & (df['admission_period'] <= 2019.2)` ensures that only rows with `admission_period` within the specified range are retained.
  - The condition `(df['enrollment_type'] != 'Exempted')` removes rows where `enrollment_type` is labeled as "Exempted".
  - The condition `(df['situation'] != 'In Progress')` excludes rows where `situation` is "In Progress".
3. **Storing the Cleaned Data** : The filtered data is stored in a new DataFrame named `df_filtered`.

## Key Characteristics of the Cleaned Dataset:

- **Rows and Columns** : The shape of the filtered dataset (`df_filtered.shape`) indicates the number of rows and columns after filtering.
- **Columns** : The list of columns (`df_filtered.columns.tolist()`) provides an overview of the available attributes in the cleaned dataset.

This cleaned dataset (`df_filtered`) can now be used for further analysis or to answer specific analytical questions. If you

have a specific question in mind, please provide it, and I will address it using the cleaned dataset.

Determine whether a student's performance in PROGRAMMING II can be accurately estimated using their performance in PROGRAMMING I and PROGRAMMING LABORATORY I. Outline three analytical approaches and justify the optimal choice based on predictive accuracy.

 qwen-max-latest

To determine whether a student's performance in **PROGRAMMING II** can be accurately estimated using their performance in **PROGRAMMING I** and **PROGRAMMING LABORATORY I**, we need to analyze the relationship between these variables. Below are three analytical approaches that can be used, along with justifications for selecting the optimal approach based on predictive accuracy.

### 1. Correlation Analysis

**Approach:**

- Compute pairwise correlations (e.g., Pearson correlation coefficient) between:
  - Performance in **PROGRAMMING II** .
  - Performance in **PROGRAMMING I** .
  - Performance in **PROGRAMMING LABORATORY I** .
- Assess the strength and direction of the relationships.

**Justification:**

- This is a quick and straightforward method to identify linear relationships between variables.
- If strong positive or negative correlations exist, it suggests that performance in **PROGRAMMING I** and **PROGRAMMING LABORATORY I** could be good predictors of performance in **PROGRAMMING II** .

**Limitations:**

- Correlation does not imply causation.
- It only captures linear relationships and may miss non-linear patterns.
- Does not account for interactions or other confounding variables.

### 2. Multiple Linear Regression

**Approach:**

- Build a multiple linear regression model where:
  - Dependent variable: Performance in **PROGRAMMING II** .
  - Independent variables: Performance in **PROGRAMMING I** and **PROGRAMMING LABORATORY I** .
- Evaluate the model's predictive accuracy using metrics such as:
  - **R-squared** : Proportion of variance explained by the model.
  - **Root Mean Squared Error (RMSE)** : Measure of prediction error.
  - **Adjusted R-squared** : Adjusts for the number of predictors in the model.

**Justification:**

- Regression explicitly models the relationship between predictors and the target variable.
- It allows us to quantify the contribution of each predictor and test for statistical significance.
- The inclusion of multiple predictors (e.g., both **PROGRAMMING I** and **PROGRAMMING LABORATORY I** ) can capture more nuanced relationships.

**Limitations:**

- Assumes a linear relationship between predictors and the target variable.
- Sensitive to multicollinearity (high correlation between predictors).
- May overfit if additional irrelevant predictors are included.

### 3. Machine Learning Model (e.g., Random Forest or Gradient Boosting)

**Approach:**

- Train a machine learning model (e.g., Random Forest or Gradient Boosting) to predict performance in **PROGRAMMING II** using:
  - Features: Performance in **PROGRAMMING I** and **PROGRAMMING LABORATORY I** .
- Evaluate the model using cross-validation and metrics such as:
  - **Mean Absolute Error (MAE)** .

- RMSE .
- R-squared .

**Justification:**

- Machine learning models can capture non-linear relationships and interactions between predictors.
- They are robust to multicollinearity and can handle noisy data better than linear regression.
- Techniques like cross-validation ensure the model generalizes well to unseen data.

**Limitations:**

- More computationally intensive than correlation or regression.
- Less interpretable compared to regression models.
- Requires sufficient data to avoid overfitting.

### Optimal Choice Based on Predictive Accuracy

The optimal choice depends on the dataset characteristics and the complexity of the relationships between variables:

1. If the relationships are linear :

- Use **Multiple Linear Regression** because it is simple, interpretable, and effective for linear relationships.
- Example Python Code:

```
python
1 import pandas as pd
2 import statsmodels.api as sm
3
4 # Load the cleaned dataset
5 df = pd.read_csv('filtered_dataset.csv')
6
7 # Filter relevant columns
8 df_relevant = df[['PROGRAMMING_II', 'PROGRAMMING_I', 'PROGRAMMING_LAB_I']].dropna()
9
10 # Define dependent and independent variables
11 X = df_relevant[['PROGRAMMING_I', 'PROGRAMMING_LAB_I']]
12 y = df_relevant['PROGRAMMING_II']
13
14 # Add a constant to the independent variables
15 X = sm.add_constant(X)
16
17 # Fit the regression model
18 model = sm.OLS(y, X).fit()
19
20 # Print summary
21 print(model.summary())
```

2. If the relationships are non-linear or complex :

- Use a **Machine Learning Model** like Random Forest or Gradient Boosting for better predictive accuracy.
- Example Python Code:

```
python
1 from sklearn.ensemble import RandomForestRegressor
2 from sklearn.model_selection import train_test_split
3 from sklearn.metrics import mean_squared_error, r2_score
4
5 # Load the cleaned dataset
6 df = pd.read_csv('filtered_dataset.csv')
7
8 # Filter relevant columns
9 df_relevant = df[['PROGRAMMING_II', 'PROGRAMMING_I', 'PROGRAMMING_LAB_I']].dropna()
10
11 # Define features and target
12 X = df_relevant[['PROGRAMMING_I', 'PROGRAMMING_LAB_I']]
13 y = df_relevant['PROGRAMMING_II']
14
15 # Split into training and testing sets
16 X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
17
18 # Train a Random Forest model
19 model = RandomForestRegressor(random_state=42)
20 model.fit(X_train, y_train)
21
22 # Make predictions
23 y_pred = model.predict(X_test)
24
25 # Evaluate the model
26 rmse = mean_squared_error(y_test, y_pred, squared=False)
27 r2 = r2_score(y_test, y_pred)
28
29 print(f"RMSE: {rmse}")
30 print(f"R-squared: {r2}")
```

3. If interpretability is critical :

- Start with **Correlation Analysis** to understand basic relationships before proceeding to regression or machine learning.

## Conclusion

The **optimal choice** is likely **Random Forest** or **Gradient Boosting** because:

- These models can capture both linear and non-linear relationships.
- They provide high predictive accuracy and robustness.
- Cross-validation ensures reliable performance on unseen data.

However, if simplicity and interpretability are prioritized, **Multiple Linear Regression** is a strong alternative.

