

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.



qwen-max-latest

To begin, let me summarize the key characteristics of the dataset for data-driven decision-making:

- Dataset Content** : The dataset contains student records with various attributes such as admission period, enrollment type, situation, demographics, academic performance metrics, and more.
- Granularity** : Each row represents a specific enrollment record tied to a unique identifier (`id`), capturing details about the student's academic journey, including admission, dropout, and performance in specific subjects.
- Relevant Columns** :
 - `admission_period` , `enrollment_type` , and `situation` are critical for filtering and analyzing the data.
 - Other columns like `gender` , `color_name` , `high_school_type` , `final_average` , and `subject_name` provide additional context for deeper insights.

Cleaning and Filtering the Dataset

The cleaning process involves:

- Filtering `admission_period` to include only values between 2006.1 and 2019.2.
- Removing rows where `enrollment_type` is labeled as "Exemption".
- Excluding rows where `situation` is "In Progress".

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

```
python
1 import pandas as pd
2
3 # Load the dataset
4 df = pd.read_csv('stratified_history.csv')
5
6 # Step 1: Filter admission_period to include only values between 2006.1 and 2019.2
7 df_filtered = df[(df['admission_period'] >= 2006.1) & (df['admission_period'] <= 2019.2)]
8
9 # Step 2: Remove entries where enrollment_type is "Exemption"
10 df_filtered = df_filtered[df_filtered['enrollment_type'] != 'Exemption']
11
12 # Step 3: Exclude rows where situation is "In Progress"
13 df_filtered = df_filtered[df_filtered['situation'] != 'In Progress']
14
15 # Display the first few rows of the cleaned dataset
16 print(df_filtered.head())
```

Explanation of the Code

- Loading the Dataset** : The dataset is loaded using `pd.read_csv` .
- Filtering `admission_period`** : A logical condition is applied to retain only rows where `admission_period` falls within the range [2006.1, 2019.2].
- Removing "Exemption" Entries** : Rows where `enrollment_type` equals "Exemption" are excluded.
- Excluding "In Progress" Situations** : Rows where `situation` equals "In Progress" are removed.
- Resulting DataFrame** : The cleaned dataset is stored in `df_filtered` .

Next Steps

Once the dataset is cleaned, you can proceed to analyze it based on a specific analytical question. Please provide the question, and I will address it using the cleaned dataset (`df_filtered`) and provide the corresponding Python code.

Identify the key variables that differentiate between passed and failed students by evaluating feature importance. Propose three analytical approaches (e.g., decision tree importance, random forest, logistic regression coefficients) and select the optimal method.



qwen-max-latest

To identify the key variables that differentiate between passed and failed students, we can use feature importance analysis.

Below, I outline three analytical approaches to evaluate feature importance and propose a method to select the optimal approach.

Step 1: Preprocessing the Cleaned Dataset

Before applying any analytical methods, ensure the dataset (`df_filtered`) is appropriately preprocessed:

1. **Target Variable** : Create a binary column `passed` where:
 - `passed = 1` if `final_situation` is "Approved".
 - `passed = 0` otherwise (e.g., "Failed", "Failed Due to Absences").
2. **Feature Selection** : Exclude non-numeric columns (e.g., `id`, `subject_name`) and encode categorical variables (e.g., `gender`, `color_name`) using one-hot encoding or label encoding.
3. **Handling Missing Values** : Impute missing values in numeric columns (e.g., mean or median) and drop rows with excessive missing data.

Here is the Python code for preprocessing:

```
python
1 import pandas as pd
2
3 # Define target variable
4 df_filtered['passed'] = df_filtered['final_situation'].apply(lambda x: 1 if x == 'Approved' else 0)
5
6 # Drop irrelevant columns
7 drop_columns = ['id', 'subject_name', 'final_situation']
8 df_features = df_filtered.drop(columns=drop_columns, errors='ignore')
9
10 # Encode categorical variables
11 df_features = pd.get_dummies(df_features, drop_first=True)
12
13 # Handle missing values
14 df_features.fillna(df_features.median(), inplace=True)
```

Step 2: Analytical Approaches

Approach 1: Decision Tree Feature Importance

Decision trees provide a straightforward way to assess feature importance by measuring how much each feature contributes to reducing impurity (e.g., Gini index or entropy).

Code Implementation:

```
python
1 from sklearn.tree import DecisionTreeClassifier
2
3 # Train a decision tree classifier
4 X = df_features.drop(columns=['passed'])
5 y = df_features['passed']
6 dt_model = DecisionTreeClassifier(random_state=42)
7 dt_model.fit(X, y)
8
9 # Extract feature importance
10 dt_importance = pd.Series(dt_model.feature_importances_, index=X.columns).sort_values(ascending=False)
11 print("Decision Tree Feature Importance:")
12 print(dt_importance.head())
```

Approach 2: Random Forest Feature Importance

Random forests aggregate feature importance across multiple decision trees, providing more robust results.

Code Implementation:

```
python
1 from sklearn.ensemble import RandomForestClassifier
2
3 # Train a random forest classifier
4 rf_model = RandomForestClassifier(random_state=42, n_estimators=100)
5 rf_model.fit(X, y)
6
7 # Extract feature importance
8 rf_importance = pd.Series(rf_model.feature_importances_, index=X.columns).sort_values(ascending=False)
9 print("Random Forest Feature Importance:")
10 print(rf_importance.head())
```

Approach 3: Logistic Regression Coefficients

Logistic regression coefficients indicate the relationship between each feature and the log-odds of passing. Features with higher absolute coefficients are more influential.

Code Implementation:

```
python
1 from sklearn.linear_model import LogisticRegression
2
3 # Train a logistic regression model
4 logreg_model = LogisticRegression(max_iter=1000, random_state=42)
5 logreg_model.fit(X, y)
6
7 # Extract coefficients
8 logreg_coefficients = pd.Series(logreg_model.coef_[0], index=X.columns).sort_values(ascending=False)
9 print("Logistic Regression Coefficients:")
10 print(logreg_coefficients.head())
```

Step 3: Selecting the Optimal Method

To determine the optimal method, consider the following criteria:

1. **Interpretability** : Logistic regression provides interpretable coefficients but assumes linearity.
2. **Robustness** : Random forests handle non-linear relationships and interactions better than decision trees.
3. **Performance** : Evaluate models using cross-validation metrics (e.g., accuracy, F1-score).

Code for Model Evaluation:

```
python
1 from sklearn.model_selection import cross_val_score
2
3 # Evaluate models using cross-validation
4 models = {
5     'Decision Tree': DecisionTreeClassifier(random_state=42),
6     'Random Forest': RandomForestClassifier(random_state=42, n_estimators=100),
7     'Logistic Regression': LogisticRegression(max_iter=1000, random_state=42)
8 }
9
10 for name, model in models.items():
11     scores = cross_val_score(model, X, y, cv=5, scoring='accuracy')
12     print(f"{name} Accuracy: {scores.mean():.4f} ± {scores.std():.4f}")
```

Step 4: Final Recommendation

Based on the evaluation:

- If interpretability is critical, choose **Logistic Regression** .
- If robustness and handling non-linear relationships are priorities, choose **Random Forest** .
- Use **Decision Tree** only for simplicity or initial exploration.

For most real-world scenarios, **Random Forest** is likely the optimal choice due to its balance of performance and robustness.

Final Answer

The key variables that differentiate between passed and failed students can be identified using **Random Forest Feature Importance** , which balances interpretability and robustness. The top features will depend on the dataset and can be extracted using the provided code.

