

Data Cleaning and Filtering

22/03/2025

stratified_history.csv
0.1 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 requirements, we will follow these steps:

- Understand the Dataset :** The dataset contains student records with attributes such as admission period, enrollment type, situation, and other academic details.
- 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".
- Store the Cleaned Data :** Save the filtered data into a new DataFrame named df_filtered.
- Prepare for Future Analysis :** Ensure the cleaned dataset is ready for any specific analytical question.

Below is the Python code to perform these tasks:

```
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 # Filter the dataset based on the specified conditions
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 # Display the first few rows of the cleaned dataset to verify
16 print(df_filtered.head())
17
18 # Optionally, save the cleaned dataset to a new CSV file for future use
19 df_filtered.to_csv('filtered_dataset.csv', index=False)
```

Explanation of the Code:

- Loading the Dataset :** The dataset is loaded using pd.read_csv() from the provided file path.
- Filtering Conditions :**
 - (df['admission_period'] >= 2006.1) & (df['admission_period'] <= 2019.2) : Ensures only records within the specified admission period range are included.
 - (df['enrollment_type'] != 'Exempted') : Removes rows where the enrollment type is "Exempted".
 - (df['situation'] != 'In Progress') : Excludes rows where the situation is "In Progress".
- Saving the Filtered Data :** The cleaned dataset is saved to a new CSV file (filtered_dataset.csv) for future use.

Key Characteristics of the Dataset for Data-Driven Decision-Making:

- Temporal Scope :** The filtered dataset focuses on records from 2006.1 to 2019.2, making it suitable for analyzing trends over this period.
- Enrollment Type :** By excluding "Exempted" entries, the dataset ensures consistency in analyzing standard enrollment processes.
- Situation :** Removing "In Progress" entries allows for analysis of completed or resolved cases, providing clearer insights into outcomes like graduation, dropout, or reentry.

This cleaned dataset (df_filtered) is now ready for any specific analytical question. If you have a particular question in mind, please provide it, and I will derive the answer using this dataset.

Analyze student dropout trends using time-series analysis, survival modeling, and machine learning classification (decision trees, logistic regression). Select the best method based on predictive accuracy and trend stability.

qwen-max-latest

To analyze student dropout trends using the provided dataset, we will follow a structured approach that incorporates time-series analysis, survival modeling, and machine learning classification. Below is a step-by-step breakdown of the methodology, along with Python code snippets for implementation.

Step 1: Data Preparation

We already have the cleaned dataset (df_filtered) from the previous task. This dataset will be used for all analyses. Ensure the following:

- The admission_period and dropout_period columns are in datetime format.
- Create a new column time_to_dropout to represent the duration between admission and dropout (or censoring).

```
python
1 import pandas as pd
2 from datetime import datetime
3
4 # Load the cleaned dataset
5 file_path = 'filtered_dataset.csv'
```

```

6 df_filtered = pd.read_csv(file_path)
7
8 # Convert admission_period and dropout_period to numeric for calculations
9 df_filtered['admission_period'] = df_filtered['admission_period'].astype(float)
10 df_filtered['dropout_period'] = df_filtered['dropout_period'].astype(float)
11
12 # Calculate time_to_dropout (in semesters)
13 df_filtered['time_to_dropout'] = df_filtered['dropout_period'] - df_filtered['admission_period']
14
15 # Create a binary target variable: 1 for dropout, 0 for no dropout
16 df_filtered['dropout'] = (df_filtered['situation'] == 'Dropout').astype(int)
17
18 print(df_filtered[['admission_period', 'dropout_period', 'time_to_dropout', 'dropout']].head())

```

Step 2: Time-Series Analysis

Time-series analysis helps identify trends in dropout rates over time. We will group data by `admission_period` and calculate dropout rates.

```

python
1 import matplotlib.pyplot as plt
2
3 # Group by admission_period and calculate dropout rate
4 dropout_trends = df_filtered.groupby('admission_period')['dropout'].mean().reset_index()
5
6 # Plot dropout trends over time
7 plt.figure(figsize=(10, 6))
8 plt.plot(dropout_trends['admission_period'], dropout_trends['dropout'], marker='o')
9 plt.title('Dropout Rate Over Admission Periods')
10 plt.xlabel('Admission Period')
11 plt.ylabel('Dropout Rate')
12 plt.grid()
13 plt.show()

```

Step 3: Survival Modeling

Survival analysis models the time until an event occurs (e.g., dropout). We will use the Kaplan-Meier estimator and Cox proportional hazards model.

Kaplan-Meier Estimator

```

python
1 from lifelines import KaplanMeierFitter
2
3 # Fit Kaplan-Meier model
4 kmf = KaplanMeierFitter()
5 kmf.fit(durations=df_filtered['time_to_dropout'], event_observed=df_filtered['dropout'])
6
7 # Plot survival function
8 kmf.plot_survival_function()
9 plt.title('Kaplan-Meier Survival Curve')
10 plt.xlabel('Time to Dropout (Semesters)')
11 plt.ylabel('Survival Probability')
12 plt.show()

```

Cox Proportional Hazards Model

```

python
1 from lifelines import CoxPHFitter
2
3 # Select relevant features for survival modeling
4 features = ['time_to_dropout', 'dropout', 'gender', 'race', 'enrollment_type', 'school_type']
5 df_survival = pd.get_dummies(df_filtered[features], drop_first=True)
6
7 # Fit Cox model
8 cph = CoxPHFitter()
9 cph.fit(df_survival, duration_col='time_to_dropout', event_col='dropout')
10
11 # Print summary
12 print(cph.summary)

```

Step 4: Machine Learning Classification

We will use decision trees and logistic regression to predict dropout probabilities.

Preprocessing

```

python
1 from sklearn.model_selection import train_test_split
2 from sklearn.preprocessing import StandardScaler
3
4 # Encode categorical variables
5 df_ml = pd.get_dummies(df_filtered.drop(columns=['dropout']), drop_first=True)
6
7 # Define features and target
8 X = df_ml
9 y = df_filtered['dropout']
10
11 # Split into training and testing sets
12 X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
13
14 # Scale numerical features
15 scaler = StandardScaler()
16 X_train = scaler.fit_transform(X_train)
17 X_test = scaler.transform(X_test)

```

Decision Tree Classifier

```

python
1 from sklearn.tree import DecisionTreeClassifier
2 from sklearn.metrics import accuracy_score, roc_auc_score
3
4 # Train decision tree
5 dt = DecisionTreeClassifier(random_state=42)
6 dt.fit(X_train, y_train)
7
8 # Evaluate
9 y_pred_dt = dt.predict(X_test)
10 accuracy_dt = accuracy_score(y_test, y_pred_dt)
11 roc_auc_dt = roc_auc_score(y_test, dt.predict_proba(X_test)[:, 1])
12
13 print(f"Decision Tree Accuracy: {accuracy_dt:.2f}, ROC AUC: {roc_auc_dt:.2f}")

```

Logistic Regression

```

python
1 from sklearn.linear_model import LogisticRegression
2
3 # Train logistic regression
4 lr = LogisticRegression(max_iter=1000, random_state=42)
5 lr.fit(X_train, y_train)
6
7 # Evaluate

```

```
8 y_pred_lr = lr.predict(X_test)
9 accuracy_lr = accuracy_score(y_test, y_pred_lr)
10 roc_auc_lr = roc_auc_score(y_test, lr.predict_proba(X_test)[:, 1])
11
12 print(f"Logistic Regression Accuracy: {accuracy_lr:.2f}, ROC AUC: {roc_auc_lr:.2f}")
```

Step 5: Compare Methods

Compare the performance of the three methods based on predictive accuracy and trend stability:

- **Time-Series Analysis** : Provides insights into dropout trends but does not predict individual outcomes.
- **Survival Modeling** : Captures time-to-event dynamics and identifies risk factors but assumes proportional hazards.
- **Machine Learning Classification** :
 - Decision Trees: High interpretability but prone to overfitting.
 - Logistic Regression: Simple and interpretable, performs well with linear relationships.

Select Best Method

Based on ROC AUC scores and interpretability, choose the method with the highest predictive accuracy while maintaining trend stability.

Final Answer

The best method for analyzing student dropout trends depends on the specific goals:

- For **trend identification** , time-series analysis and survival modeling are most suitable.
- For **predictive accuracy** , machine learning classification (e.g., logistic regression) performs better.

If predictive accuracy is prioritized, **logistic regression** or **decision trees** would likely be the best choice. However, if understanding temporal patterns and risk factors is critical, **survival modeling** provides deeper insights.

Boxed Final Answer:

Use logistic regression for predictive accuracy and survival modeling for trend stability.

