Dataset 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 analyze the dataset and address a specific analytical question, we first need to clean the data as per the requirements. Here's how we approach the task:

Steps for Data Cleaning:

1. **Filter `admission_period`** : Include only rows where `admission_period` is between `2006.1` and `2019.2` .
2. **Remove "Exempted" entries** : Exclude rows where `enrollment_type` is labeled as "Exempted".
3. **Exclude "In Progress" situations** : Remove rows where `situation` is "In Progress".
4. **Store the cleaned data** : Save the resulting dataset in a new DataFrame named `df_filtered` .

Example Analytical Question:

What is the distribution of students by `dropout_code` in the cleaned dataset?

This question will help us understand the reasons for student dropouts within the filtered dataset.

Python Code Implementation

python

```python
import pandas as pd

# Load the dataset
df = pd.read_csv('stratified_history.csv')

# Step 1: Filter admission_period to include only values between 2006.1 and 2019.2
df_filtered = df[(df['admission_period'] >= 2006.1) & (df['admission_period'] <= 2019.2)]

# Step 2: Remove entries where enrollment_type is labeled as "Exempted"
df_filtered = df_filtered[df_filtered['enrollment_type'] != 'Exempted']

# Step 3: Exclude rows where situation is "In Progress"
df_filtered = df_filtered[df_filtered['situation'] != 'In Progress']

# Display the cleaned dataset
print("Cleaned Dataset Overview:")
print(df_filtered.head())

# Analytical Question: Distribution of students by dropout_code
dropout_distribution = df_filtered['dropout_code'].value_counts()

# Display the results
print("\nDistribution of Students by Dropout Code:")
print(dropout_distribution)

# Optional: Save the cleaned dataset to a new CSV file
df_filtered.to_csv('cleaned_dataset.csv', index=False)
```

Explanation of the Code:

1. **Loading the Dataset** : The dataset is loaded using `pd.read_csv()` .
2. **Filtering `admission_period`** : A boolean mask is applied to filter rows where `admission_period` falls within the specified range.
3. **Removing "Exempted" Entries** : Rows with `enrollment_type` equal to "Exempted" are excluded.
4. **Excluding "In Progress" Situations** : Rows with `situation` equal to "In Progress" are removed.
5. **Analyzing Dropout Codes** : The `value_counts()` method is used to count the occurrences of each `dropout_code` in the cleaned dataset.
6. **Saving the Cleaned Dataset** : The cleaned dataset is optionally saved to a new CSV file for future use.

Sample Output:

Cleaned Dataset Overview:

The first few rows of the cleaned dataset will be displayed, showing only the filtered data.

Distribution of Students by Dropout Code:

This output will show the count of students grouped by their `dropout_code` . For example:

```
1     500
4     300
12    150
21    80
...
Name: dropout_code, dtype: int64
```

Final Answer:

The distribution of students by `dropout_code` in the cleaned dataset is as follows:

- **Dropout Code 1** : X students
- **Dropout Code 4** : Y students
- **Dropout Code 12** : Z students
- ...

(Replace X, Y, Z with actual counts derived from the dataset.)

This analysis provides insights into the primary reasons for student dropouts within the specified period, enabling data-driven decision-making.

Using advanced analysis of characteristics from previously graduated students, develop hypothetical student profiles that are associated with a higher probability of graduation. Propose three approaches (e.g., cluster analysis, generative models, predictive modeling) and select the best option.

qwen-max-latest

To develop hypothetical student profiles associated with a higher probability of graduation, we can leverage the cleaned dataset (`df_filtered`) and apply advanced analytical techniques. Below, I outline three potential approaches, followed by a recommendation for the best option.

Approach 1: Cluster Analysis

Objective : Group students into clusters based on shared characteristics, then identify which clusters are associated with higher graduation rates.

Steps :

1. **Feature Selection** : Select relevant features such as `admission_period`, `enrollment_type`, `situation`, `dropout_code`, `race`, `school_type`, `gender`, `age_at_admission`, etc.
2. **Preprocessing** : Normalize numerical features (e.g., age, admission score) and encode categorical variables (e.g., race, gender).
3. **Clustering Algorithm** : Use algorithms like K-Means, DBSCAN, or Hierarchical Clustering to group students.
4. **Cluster Profiling** : Analyze each cluster's characteristics and calculate the graduation rate within each cluster.
5. **Interpretation** : Identify clusters with the highest graduation rates and extract their defining characteristics.

Pros :

- Provides interpretable groups of students.
- Highlights patterns in the data without requiring labeled outcomes.

Cons :

- Does not explicitly predict graduation but rather identifies correlations.
- May require extensive feature engineering and tuning.

Approach 2: Generative Models

Objective : Use generative models to simulate hypothetical student profiles that align with historical graduation trends.

Steps :

1. **Data Preparation** : Focus on features predictive of graduation (e.g., demographics, academic performance, enrollment type).
2. **Model Selection** : Train generative models such as Gaussian Mixture Models (GMM), Variational Autoencoders (VAE), or Bayesian Networks.
3. **Profile Generation** : Sample from the trained model to generate synthetic student profiles.
4. **Validation** : Evaluate generated profiles against historical data to ensure they align with observed graduation patterns.

Pros :

- Can generate realistic and diverse student profiles.
- Useful for simulating "what-if" scenarios.

Cons :

- Complex to implement and interpret.
- Requires large amounts of high-quality data for accurate modeling.

Approach 3: Predictive Modeling

Objective : Build a supervised machine learning model to predict the likelihood of graduation and identify key factors contributing to success.

Steps :

1. **Target Variable** : Define the target variable (`situation` or `graduated` flag) to indicate whether a student graduated.
2. **Feature Engineering** : Create features such as:
   - Academic performance metrics (e.g., GPA, dropout codes).
   - Demographic information (e.g., race, gender, school type).
   - Enrollment details (e.g., admission period, enrollment type).
3. **Model Training** : Train models like Logistic Regression, Random Forest, Gradient Boosting (e.g., XGBoost, LightGBM), or Neural Networks.
4. **Feature Importance** : Extract feature importance scores to identify key predictors of graduation.
5. **Hypothetical Profiles** : Use the model to create hypothetical profiles by varying input features and observing changes in predicted graduation probabilities.

Pros :

- Directly predicts graduation likelihood.
- Provides actionable insights through feature importance analysis.
- Highly interpretable and scalable.

Cons :

- Requires labeled data (e.g., graduation status).
- Risk of overfitting if the dataset is small or noisy.

Recommendation: Predictive Modeling

Predictive modeling is the most suitable approach for this task because:

1. It directly addresses the goal of identifying student profiles associated with higher graduation probabilities.
2. It provides interpretable insights into the factors influencing graduation.
3. It is widely used in educational data mining and has proven effectiveness in similar contexts.

Implementation Plan for Predictive Modeling

Step 1: Data Preparation

```python
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report

# Load the cleaned dataset
df_filtered = pd.read_csv('cleaned_dataset.csv')

# Define target variable
df_filtered['graduated'] = df_filtered['situation'].apply(lambda x: 1 if x == 'Graduate' else 0)

# Feature selection
features = [
    'admission_period', 'enrollment_type', 'race', 'school_type',
    'gender', 'age_at_admission', 'dropout_code'
]
X = df_filtered[features]
y = df_filtered['graduated']

# Split data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

Step 2: Preprocessing and Model Training

```python
# Preprocessing pipeline
```

```python
numeric_features = ['admission_period', 'age_at_admission']
categorical_features = ['enrollment_type', 'race', 'school_type', 'gender', 'dropout_code']

preprocessor = ColumnTransformer(
    transformers=[
        ('num', StandardScaler(), numeric_features),
        ('cat', OneHotEncoder(handle_unknown='ignore'), categorical_features)
    ]
)

# Model pipeline
pipeline = Pipeline([
    ('preprocessor', preprocessor),
    ('classifier', RandomForestClassifier(random_state=42))
])

# Train the model
pipeline.fit(X_train, y_train)

# Evaluate the model
y_pred = pipeline.predict(X_test)
print(classification_report(y_test, y_pred))
```

Step 3: Feature Importance and Hypothetical Profiles

python

```python
# Extract feature importance
feature_names = numeric_features + list(pipeline.named_steps['preprocessor'].transformers_[1][1]
importances = pipeline.named_steps['classifier'].feature_importances_

# Create a DataFrame for feature importance
importance_df = pd.DataFrame({'Feature': feature_names, 'Importance': importances})
importance_df = importance_df.sort_values(by='Importance', ascending=False)
print(importance_df)

# Generate hypothetical profiles
hypothetical_profiles = pd.DataFrame({
    'admission_period': [2017.1, 2018.2],
    'enrollment_type': ['Regular', 'Unified Selection System (SISU)'],
    'race': ['White', 'Brown'],
    'school_type': ['Only Public School', 'Only Private School'],
    'gender': ['Male', 'Female'],
    'age_at_admission': [18, 20],
    'dropout_code': [1, 0]
})

# Predict graduation probabilities
probabilities = pipeline.predict_proba(hypothetical_profiles)
hypothetical_profiles['Graduation Probability'] = probabilities[:, 1]
print(hypothetical_profiles)
```

Conclusion

Using predictive modeling, we can identify key factors influencing graduation and generate hypothetical student profiles associated with higher graduation probabilities. This approach is actionable, interpretable, and aligned with the goal of data-driven decision-making.

Final Answer : Predictive modeling is the best approach to develop hypothetical student profiles associated with higher graduation probabilities.

Qwen

Potencializado por Open WebUI