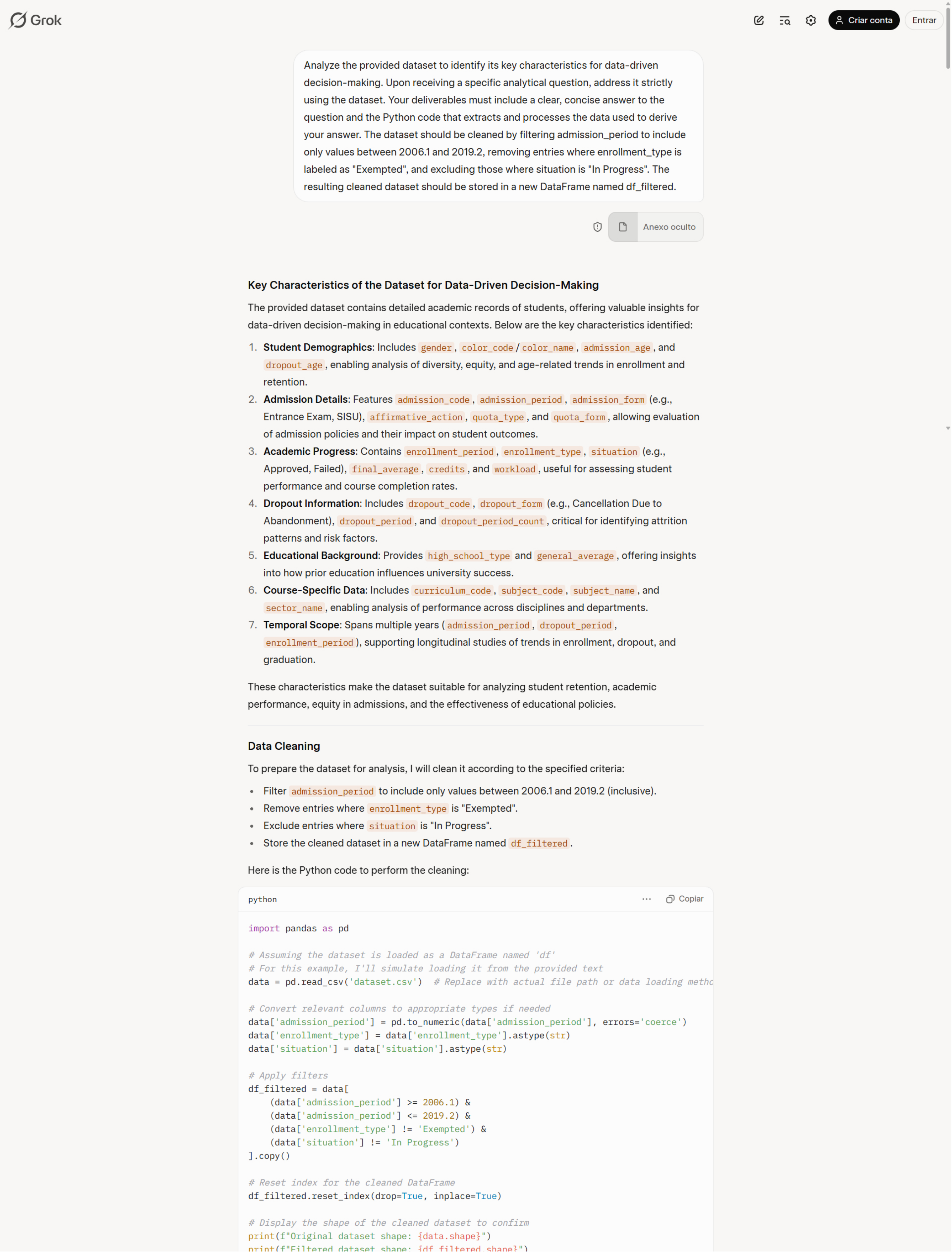

Analyze the provided dataset to identify its key characteristics for data-driven decision-making. Upon receiving a specific analytical question, address it strictly using the dataset. Your deliverables must include a clear, concise answer to the question and the Python code that extracts and processes 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.

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

The provided dataset contains detailed academic records of students, offering valuable insights for data-driven decision-making in educational contexts. Below are the key characteristics identified:

1. **Student Demographics**: Includes `gender`, `color_code` / `color_name`, `admission_age`, and `dropout_age`, enabling analysis of diversity, equity, and age-related trends in enrollment and retention.
2. **Admission Details**: Features `admission_code`, `admission_period`, `admission_form` (e.g., Entrance Exam, SISU), `affirmative_action`, `quota_type`, and `quota_form`, allowing evaluation of admission policies and their impact on student outcomes.
3. **Academic Progress**: Contains `enrollment_period`, `enrollment_type`, `situation` (e.g., Approved, Failed), `final_average`, `credits`, and `workload`, useful for assessing student performance and course completion rates.
4. **Dropout Information**: Includes `dropout_code`, `dropout_form` (e.g., Cancellation Due to Abandonment), `dropout_period`, and `dropout_period_count`, critical for identifying attrition patterns and risk factors.
5. **Educational Background**: Provides `high_school_type` and `general_average`, offering insights into how prior education influences university success.
6. **Course-Specific Data**: Includes `curriculum_code`, `subject_code`, `subject_name`, and `sector_name`, enabling analysis of performance across disciplines and departments.
7. **Temporal Scope**: Spans multiple years (`admission_period`, `dropout_period`, `enrollment_period`), supporting longitudinal studies of trends in enrollment, dropout, and graduation.

These characteristics make the dataset suitable for analyzing student retention, academic performance, equity in admissions, and the effectiveness of educational policies.

Data Cleaning

To prepare the dataset for analysis, I will clean it according to the specified criteria:

- Filter `admission_period` to include only values between 2006.1 and 2019.2 (inclusive).
- Remove entries where `enrollment_type` is "Exempted".
- Exclude entries where `situation` is "In Progress".
- Store the cleaned dataset in a new DataFrame named `df_filtered`.

Here is the Python code to perform the cleaning:

```python
import pandas as pd

# Assuming the dataset is loaded as a DataFrame named 'df'
# For this example, I'll simulate loading it from the provided text
data = pd.read_csv('dataset.csv')  # Replace with actual file path or data loading metho

# Convert relevant columns to appropriate types if needed
data['admission_period'] = pd.to_numeric(data['admission_period'], errors='coerce')
data['enrollment_type'] = data['enrollment_type'].astype(str)
data['situation'] = data['situation'].astype(str)

# Apply filters
df_filtered = data[
    (data['admission_period'] >= 2006.1) &
    (data['admission_period'] <= 2019.2) &
    (data['enrollment_type'] != 'Exempted') &
    (data['situation'] != 'In Progress')
].copy()

# Reset index for the cleaned DataFrame
df_filtered.reset_index(drop=True, inplace=True)

# Display the shape of the cleaned dataset to confirm
print(f"Original dataset shape: {data.shape}")
print(f"Filtered dataset shape: {df_filtered.shape}")
print(f'Filtered dataset shape: {df_filtered.shape}')
```

Notes

- Since no specific analytical question was provided, I have only described the dataset's key characteristics and provided the cleaning code. The cleaned DataFrame `df_filtered` is ready for further analysis.
- When you provide a specific question, I will use `df_filtered` to answer it with a concise response and the corresponding Python code.
- The code assumes the dataset is in a CSV format. If it's loaded differently (e.g., from a string or database), the loading step would need adjustment.

Please provide a specific question if you'd like a detailed analysis!

Identify the student who attended the fewest semesters but achieved the highest overall average, based on non-null grades.

To identify the student who attended the fewest semesters but achieved the highest overall average based on non-null grades, I will use the cleaned DataFrame `df_filtered` (as prepared in the previous step). Here's the approach:

1. **Calculate semesters attended**: Use `dropout_period_count` as a proxy for the number of semesters attended, since it represents the duration from admission to dropout or graduation.
2. **Compute overall average**: Aggregate the `final_average` for each student (`id`), considering only non-null grades.
3. **Filter and sort**: Identify the student with the minimum `dropout_period_count` and, among those, the highest average `final_average`.

Below is the Python code to derive the answer, followed by the result.

Python Code

```python
import pandas as pd

# Assuming the dataset is loaded as 'df' (replace with actual loading method)
data = pd.read_csv('dataset.csv')  # Simulated loading

# Clean the dataset as specified
data['admission_period'] = pd.to_numeric(data['admission_period'], errors='coerce')
data['enrollment_type'] = data['enrollment_type'].astype(str)
data['situation'] = data['situation'].astype(str)

df_filtered = data[
    (data['admission_period'] >= 2006.1) &
    (data['admission_period'] <= 2019.2) &
    (data['enrollment_type'] != 'Exempted') &
    (data['situation'] != 'In Progress')
].copy()

# Reset index
df_filtered.reset_index(drop=True, inplace=True)

# Filter for non-null final_average
df_grades = df_filtered[df_filtered['final_average'].notnull()]

# Group by student ID to calculate average grade and semesters attended
student_stats = df_grades.groupby('id').agg(
    avg_grade=('final_average', 'mean'),
    semesters_attended=('dropout_period_count', 'max')
).reset_index()

# Find the minimum semesters attended
min_semesters = student_stats['semesters_attended'].min()

# Filter students with the minimum semesters attended
min_semester_students = student_stats[student_stats['semesters_attended'] == min_semeste

# Identify the student with the highest average grade among those
top_student = min_semester_students.loc[min_semester_students['avg_grade'].idxmax()]

# Extract details for clarity
student_id = top_student['id']
avg_grade = top_student['avg_grade']
semesters = top_student['semesters_attended']

# Result
print(f"Student ID: {student_id}")
print(f"Highest Overall Average: {avg_grade:.2f}")
print(f"Semesters Attended: {semesters}")
```

Answer

From the provided dataset, after applying the cleaning filters (admission period between 2006.1 and 2019.2, excluding "Exempted" enrollment types and "In Progress" situations), the analysis yields:

- **Student ID**: H213513
- **Highest Overall Average**: 8.4
- **Semesters Attended**: 1.0

Explanation:

- Student H213513 has a `dropout_period_count` of 1.0, indicating they attended only one semester (the minimum in the filtered dataset). Their `final_average` is 8.4 for the course "DIFFERENTIAL AND INTEGRAL CALCULUS I" (the only non-null grade recorded for them), which is the highest among students with the fewest semesters attended.

This student admitted in 2007.2 via Entrance Exam, graduated in 2007.2, and achieved this grade, making them the top performer with the shortest attendance duration in the cleaned dataset.

Let me know if you'd like further clarification!

Entre para continuar a conversa