STUDENT-PERFORMANCE ANALYZER

CODE:

```
pip install pandas numpy matplotlib seaborn scikit-learn
```

```
import requests
import pandas as pd
import urllib3

# Disable SSL verification warning
urllib3.disable_warnings(urllib3.exceptions.InsecureRequestWarning)

# URLs for current and historical quiz data
current_quiz_url_1 = "https://jsonkeeper.com/b/LLQT"
current_quiz_url_2 = "https://api.jsonserve.com/rJvd7g"
historical_quiz_url = "https://api.jsonserve.com/XgAgFJ"

# Mock data for testing (taken as an example)
current_quiz_data_1 = {
    "quiz": [
        {"topic": "Physics", "userScore": 20, "difficulty": "Hard"},
        {"topic": "Chemistry", "userScore": 50, "difficulty": "Medium"},
        {"topic": "Biology", "userScore": 80, "difficulty": "Easy"}
    ]
}
current_quiz_data_2 = {
    "quiz": [
        {"topic": "Physics", "userScore": 40, "difficulty": "Hard"},
        {"topic": "Chemistry", "userScore": 60, "difficulty": "Medium"},
        {"topic": "Biology", "userScore": 85, "difficulty": "Easy"}
    ]
}
historical_quiz_data = [
    {"quiz_id": 1, "topic": "Physics", "score": 80},
    {"quiz_id": 2, "topic": "Chemistry", "score": 45},
    {"quiz_id": 3, "topic": "Biology", "score": 70},
]

# Use live data if needed
# current_quiz_data_1 = requests.get(current_quiz_url_1, verify=False).json()
```

```python
# current_quiz_data_2 = requests.get(current_quiz_url_2).json()
# historical_quiz_data = requests.get(historical_quiz_url).json()

# Combining the current quiz data
current_quiz_data = {**current_quiz_data_1, **current_quiz_data_2}

# Creating DataFrames
if 'quiz' in current_quiz_data and current_quiz_data['quiz']:
    current_quiz_df = pd.DataFrame(current_quiz_data['quiz'])
else:
    current_quiz_df = pd.DataFrame(columns=['topic', 'userScore', 'score',
'quiz_score'])

historical_quiz_df = pd.DataFrame(historical_quiz_data)

# Checking for score column
score_column = None
for col in ['userScore', 'score', 'quiz_score']:
    if col in current_quiz_df.columns:
        score_column = col
        break

# Calculating average score by topic
if score_column and not current_quiz_df.empty:
    avg_score_by_topic = current_quiz_df.groupby('topic')[score_column].mean()
else:
    avg_score_by_topic = pd.Series(dtype=float)

# Including historical scores in analysis
if not historical_quiz_df.empty:
    historical_avg = historical_quiz_df.groupby('topic')['score'].mean()
    avg_score_by_topic = avg_score_by_topic.combine(historical_avg, max,
fill_value=0)

# Identifying weak areas and give recommendations
if not avg_score_by_topic.empty:
    weak_area = avg_score_by_topic.idxmin()
    recommendation = f"Focus more on the topic: {weak_area}."
    print(recommendation)
else:
    print("No quiz data available to provide a recommendation.")

# Persona analysis
```

```python
if not avg_score_by_topic.empty:
    if avg_score_by_topic.mean() > 70:
        persona = "Balanced Learner"
    elif avg_score_by_topic.min() < 40:
        persona = "Struggler"
    else:
        persona = "Improving Learner"
    print(f"Persona: {persona}")
else:
    print("Insufficient data to determine a persona.")

# Strengths and weaknesses
strengths = avg_score_by_topic[avg_score_by_topic >= 70].index.tolist()
weaknesses = avg_score_by_topic[avg_score_by_topic < 40].index.tolist()

print("\nStrengths:")
for topic in strengths:
    print(f"- {topic}: Excellent performance!")

print("\nWeaknesses:")
for topic in weaknesses:
    print(f"- {topic}: Needs improvement.")

# Add creative labels
print("\nCreative Labels:")
for topic, score in avg_score_by_topic.items():
    if score >= 80:
        label = "Master"
    elif 60 <= score < 80:
        label = "Competent"
    elif 40 <= score < 60:
        label = "Intermediate"
    else:
        label = "Beginner"
    print(f"{topic}: {label} - Avg Score: {score:.2f}")
```

```python
import matplotlib.pyplot as plt

avg_score_by_topic.plot(kind='bar', color='skyblue', title="Average Scores by
Topic")
```

```
plt.xlabel("Topic")
plt.ylabel("Average Score")
plt.show()
```